A 1394 Bus Interface for the Chidi Media Processor

By
Yuan-Min Liu

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

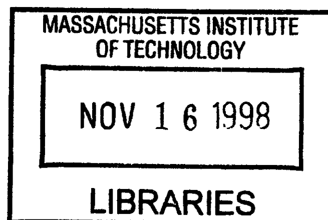
August 7, 1998
[September, 1998]

© Copyright 1998 Massachusetts Institute of Technology. All rights reserved.

Author ______________________________
Department of Electrical Engineering and Computer Science
August 7, 1998

Certified by ______________________________
V. Michael Bove, Jr.
Thesis Supervisor

Accepted by ______________________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses


MASSACHUSETTS INSTITUTE OF TECHNOLOGY
NOV 16 1998
LIBRARIES

Eng.

A 1394 Bus Interface for the Chidi Media Processor
By
Yuan-Min Liu

Submitted to the
Department of Electrical Engineering and Computer Science

August 7, 1998

in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Chidi reconfigurable media processor requires a low-cost, scalable data link for media and sensor I/O. This paper describes a 1394 Serial Bus interface for the Chidi processor, which allows for routing of data between network nodes and supports capture of DV camera output. The scope of the research covers design and implementation from the 1394 software drivers down to the physical layer. Off-the-shelf chips are used to implement the 1394 physical and link layers. Logic implemented on a Field Programmable Gate Array facilitates transfer of data from Chidi's general purpose processor and system memory to the link layer and vice versa.

This research was funded by the Digital Life Consortium at the MIT Media Laboratory

Thesis Supervisor: Dr. V. Michael Bove, Jr.
Title: Principal Research Scientist, MIT Media Laboratory

Acknowledgements

This project would not have been possible without the guidance and support of Dr. V. Michael Bove, Jr. and John Watlington. I've probably learned as much from them in the past year than in my previous four years at MIT and I sincerely thank them for the opportunity.

I would also like to acknowledge the help of the entire Chidi team – Dr. Tom Nwodoh, Mark C. Lee, Ken Kung, and Chris McEniry, Peggy Chen, Josh Stults, Chris Yang, and Peter Yang. Special thanks to Mark, without whom I wouldn't have been able to stomach dinner at the Kendall Food Court night after night.

Table of Contents

1. Background ... 6
1.1 Chidi ... 6
1.2 1394 Serial Bus ... 7
1.2.1 1394 Bus Protocol ... 8
1.2.2 Why 1394? ... 11
2. Purpose ... 13
3. Design ... 15
3.1 Hardware ... 15
3.1.1 Physical Layer ... 15
3.1.2 Link Layer ... 16
3.1.3 1394 FIFO Buffer ... 18
3.2 Processor-to-Link Interface Logic on SAG ... 19
3.2.1 Transfer Function Implementation ... 20
3.2.2 FPGA Development ... 22
3.2.3 Data Transmitter/Receiver Sub-Module ... 25
3.2.4 Link Interface Sub-Module ... 30
3.2.5 Hardware Registers Sub-Module ... 37
3.2.6 Bus Mastership Requesting Logic Sub-Module ... 40
3.2.7 Busctrl Sub-Module ... 41
3.3 Software ... 47
4. Conclusion ... 52
4.1 Future Work ... 52
Appendix A – TSB12C01A Link Layer Controller Addressing ... 54
Appendix B – 1394 Bus Initialization ... 55
Appendix C – Address Generator Implementation ... 56
Appendix D – Altera FPGA Optimization ... 57
Appendix E – Communication with 1394-Compliant Cameras ... 58

List of Figures

Figure 1 : Chidi Top Level Diagram 6
Figure 2 : Typical 1394 Bus Cycle 8
Figure 3 : 1394 Bus Protocol Layers 9
Figure 4 : Data Strobe Encoding 10
Figure 5 : Typical Chidi 1394 Network 12
Figure 6 : 1394 Interface Block Diagram 15
Figure 7 : TSB12C01A Link Layer Controller 18
Figure 8 : 1394 FIFO Buffer 19
Figure 9 : Processor-to-Link Interface Block Diagram 20
Figure 10 : Altera FLEX 10K Internal Architecture 22
Figure 11 : Altera FLEX 10K Logic Element Structure 23
Figure 12 : GPP Read from READ-1394 FIFO Timing Diagrams 26
Figure 13 : GPP Write to WRITE-1394 FIFO Timing Diagrams 27
Figure 14 : Write from memory to WRITE-1394 FIFO Timing Diagrams 28
Figure 15 : Read to memory from READ-1394 FIFO Timing Diagrams 29
Figure 16 : Link Interface Block Diagram 31
Figure 17 : Link FSM State Diagram 33
Figure 18 : Link Register Read and Write Timing Diagrams 34
Figure 19 : Link ATF/ITF Write Timing Diagram 35
Figure 20 : Link GRF Read Timing Diagram 36
Figure 21 : Advance FIFO Example Case 36
Figure 22 : Busctrl Module Block Diagram 42
Figure 23 : Busctrl Pipeline Registers 44
Figure 24 : Busctrl State Machine Diagram 46
Figure 25 : 1394 Software Hierarchy 47
Figure 26 : FW_SEND_QUEUE example 48
Figure 27 : FW_SEND_QUEUE_MEM example 49

List of Tables

Table 1 : Data Transmitter/Receiver Signal Descriptions 25
Table 2 : Link Interface Signal Descriptions 30
Table 3 : Link Layer FIFO Access Addresses 32
Table 4 : Hardware Register Addresses 37
Table 5 : Control Register Field Descriptions 38
Table 6 : Interrupt Register Field Descriptions 39
Table 7 : Busctrl Module Signal Descriptions 43
Table 8 : TSB12C01A Internal Registers 54
Table 9 : TSB12C01A Internal FIFOs 54
Table 10 : Camera Registers (taken from 1394-based Digital Camera Specification) 58
Table 11 : Camera Video Modes (taken from 1394-based Digital Camera Specification) 58

1. Background

The Object-based Media group at the MIT Media Lab is currently developing Chidi, a reconfigurable media processor which resides on a PCI card. [4] A typical use of Chidi involves inputting video data over a host PCI bus, performing computation on the data, and outputting the results to the PCI bus. The Floe operating system [5] provides 'resource managing' to allow a network of individually configured Chidi nodes to work in parallel.

1.1 Chidi

In Chidi, both the general purpose processor (GPP), a PowerPC 604e processor, and reconfigurable processor (RP) are used to perform computations. The GPP performs general computations for user applications, whereas the RP can be configured and used to perform specialized operations for which the GPP is not well suited. Resource management is handled by the Floe operating system running on the GPP. It schedules tasks for both the GPP and the RP and sets up the Stream Address Generator. Figure 1 shows a top-level block diagram of Chidi.

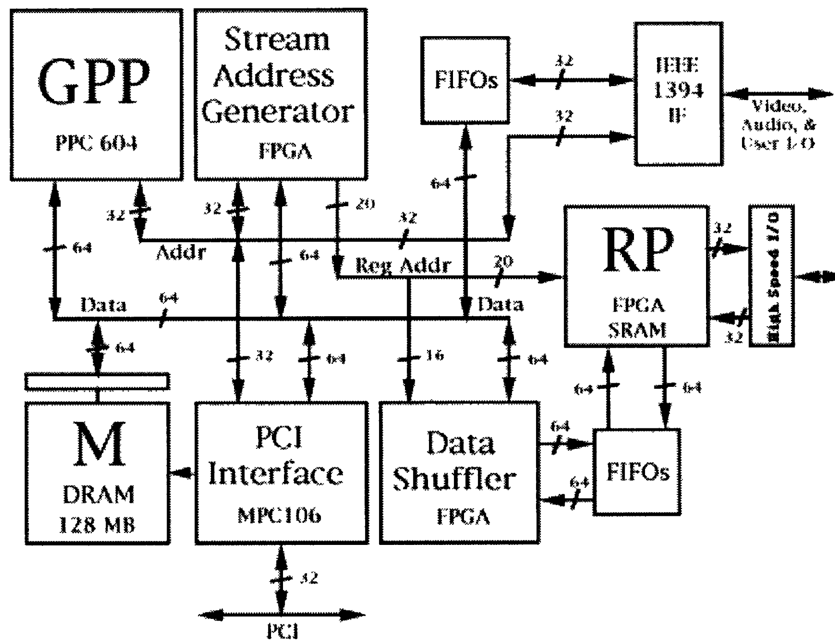


Figure 1 : Chidi Top Level Diagram

The RP is an FPGA, with ~100K gates, which can be configured as necessary to perform specialized operations on streams of data, as necessary. Examples of functions that the RP could

implement are transforms, texture mappings, and image remappings. The Data Shuffler block builds the data out of memory into streams that the RP can use.

Both the GPP and RP share a single block of DRAM memory, controlled by the MPC106 and accessible over the PPC data bus. The MPC106 also provides a PPC/PCI Bridge for communication with the host system.

The Stream Address Generator emulates a PowerPC processor to control the flow of streams of data between memory and the RP and also between memory and the 1394 I/O interface.

The 1394 I/O interface, which is the focus of this paper, provides a data link for media I/O. Among the 1394-compliant devices that are currently available or soon-to-be available are digital video cameras and camcorders and digital video disc players.

In addition to the obvious use as a link to digital media input and output, the 1394 interface has potentially a second use in networks of Chidi nodes operating under the Floe operating system. Floe supports parallelism for a network of specialised processors. In a network of PCI-based Chidi card, the initial communication path between nodes would be across the PCI bus of the host machine and then the local network to the machines hosting the other Chidis. However, the 1394 interface provides an alternative communication link between Chidi nodes.

1.2 1394 Serial Bus

As mentioned above, Chidi requires a direct interface to media and sensor I/O and a communication link between cards bypassing the local network. Goals of designing the interface are that the solution be low-cost, scalable, and provide bandwidth for real-time video input. The IEEE 1394 Serial Bus provides a moderate speed at a low cost, and is becoming widely accepted as the networking solution for digital media.

The 1394 serial bus provides up to 400 Mb/s bandwidth over a non-cyclic point-to-point network. In the cable environment, the physical layer of 1394 bus nodes consists of a repeater and multiple ports, so nodes can be chained together in a tree topology. The tree is limited to 63 nodes and the maximum node-to-node cable length is 4.5 meters. The physical cable consists of two shielded twisted pairs, and a power and ground pair. The cable power is provided to all the physical layers in the network, so that even when a device is powered down, its physical layer can still act as a repeater.

1394 supports both asynchronous and isochronous data transfers. The most important difference between the two types of transfers is that isochronous transfers are guaranteed a certain bus bandwidth, while asynchronous transfers must gain control of the bus through arbitration, and as a result, do not have guaranteed bandwidth. How does this work? First, the bus time is divided up into 125 us cycles. Then, each isochronous channel reserves the bandwidth that it needs, in terms of the fraction of the 125 us cycle that it needs, with a node that acts as the isochronous resource manager. Then, at the start of each cycle, the isochronous channels transmit over the bus, using only the bandwidth already allocated to them. After all of the isochronous transfers have been completed, the nodes can arbitrate for use of the remaining time to send asynchronous packets.

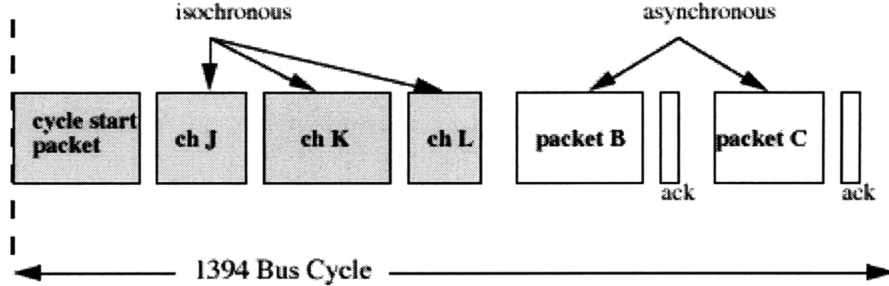


Figure 2 : Typical 1394 Bus Cycle

1.2.1 1394 Bus Protocol

The 1394 Serial Bus protocol is divided into three stacked layers: the transaction layer, link layer, and physical layer. The transaction layer defines a request-response protocol to perform the read, write, and lock bus operations. The link layer provides an acknowledged datagram service to the transaction layer for packet transmission and reception. The physical layer provides serial bus arbitration and turns the logical symbols provided by the link layer into the electrical signals on the serial bus cable. In addition, a Serial Bus Management module provides control of bus resources.

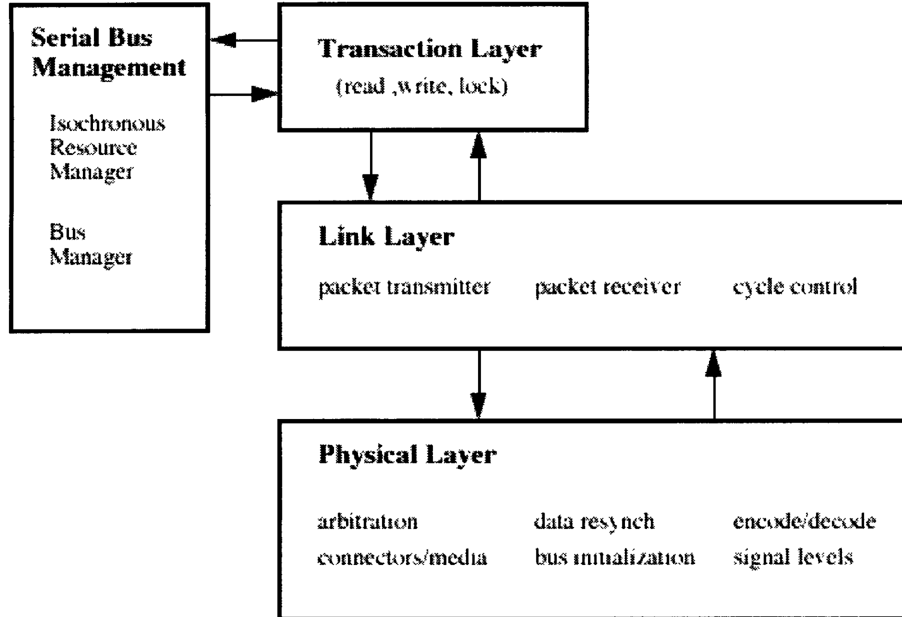


Figure 3 : 1394 Bus Protocol Layers

Transaction Layer

The transaction layer defines three different transaction types for data transfer: read, write, and lock. In a read transaction, a requester asks for data at an address within a remote node, and the remote node responds with data to the requester. In a write, a requester sends data to an address within one or more remote nodes. In a lock transaction, a requester sends data to a remote node to be processed with data at a specific address within the remote node and then transferred back to the requester.

Link Layer

The link layer provides the packet delivery service for the transaction layer. In the delivery of a packet, called a subaction, the link layer requests the physical layer to gain control of the bus. When the physical layer responds with control of the bus, the link layer provides it the packet to be transferred. An acknowledgement is returned to the link layer that indicates the response of the packet receiver.

Physical Layer

The physical layer has three primary functions: transmission and reception of data bits, arbitration, and provision for the electrical and mechanical interface. Data is transmitted over the serial bus using data-strobe encoding. Figure 4 shows an example of data-strobe encoding. NRZ data is sent over the Data signal. The Strb signal changes state only whenever two consecutive Data bits are the same. Thus, a clock signal that transitions each bit period can be derived from the exclusive-or of Data and Strb. Data-strobe encoding is used to improve the skew tolerance of the information transferred across the serial bus.

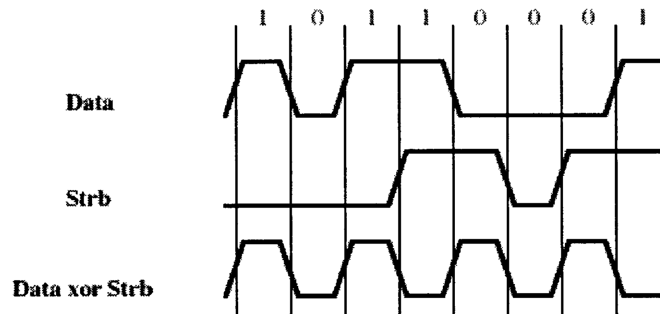


Figure 4 : Data Strobe Encoding

Arbitration

When nodes compete for bus access, the node closest to the root always wins arbitration, since the root receives the closest node's request first and grants to the first node from whom it sees a request. However, the serial bus ensures that access opportunities are split evenly among competing nodes. The fairness protocol can be explained with the following rules:

1) A node will observe a minimum period of bus idleness, called a subaction gap, before attempting bus arbitration.
2) Once granted the serial bus, a node may not arbitrate for the bus again until it sees a longer gap, called an arbitration reset gap.

The result of these rules is that eventually, every active node will get a chance to transmit over the bus once before an arbitration reset gap. Only after each node has had an opportunity to win the bus will the stay idle long enough for an arbitration reset gap.

Isochronous arbitration works under a similar premise. A cycle start packet is sent at the beginning of each isochronous cycle by a node which acts as a cycle master. After a cycle start packet is seen, nodes sending isochronous data wait only for isochronous gaps, which are shorter than asynchronous gaps. So, all isochronous packets get sent before aysnchronous packets. Since isochronous channels are only allocated as bandwidth is available, this arbitration scheme ensures that a packet can be sent on each isochronous channel for every cycle.

Serial Bus Management

The functions of Serial Bus Management include bus management and isochronous resource management. In a 1394 network, at most one node acts as a bus manager and at most one node acts as an isochronous resource manager. The bus manager keeps information regarding the topology of the network and the speed of all the nodes in the network. The information can be used to optimize bus performance. The isochronous resource manager provides a central location to track the use of isochronous resources by implementing two registers, BANDWIDTH_AVAILABLE and CHANNELS_AVAILABLE. A node allocates to itself isochronous bandwidth and channels by modifying the registers via read and write transactions.

1.2.2 Why 1394?

With its simple connectors and cables, the 1394 serial bus provides a low-cost interconnect for Chidi nodes. 1394 also provides guaranteed bandwidth to time-critical applications, so that even as the number of nodes in the network grows, we can ensure that data reaches it destination, as long as the total bus bandwidth hasn't been exceeded.

Because of its low costs and support for isochronous transfer, 1394 has been chosen by the Digital Video Consortium as the standard bus technology for digital video applications. For example, digital video camera producers, such as Sony, have added 1394 ports to their designs. As digital video devices become equipped with 1394 capabilities, it seems natural to use 1394 as the media interface for Chidi. In addition, since 1394 appears that it will be a consistent technology for the future, revisions of the Chidi board won't have to design for a different bus technology. Figure 5 shows a typical use of a 1394 bus by a Chidi network.

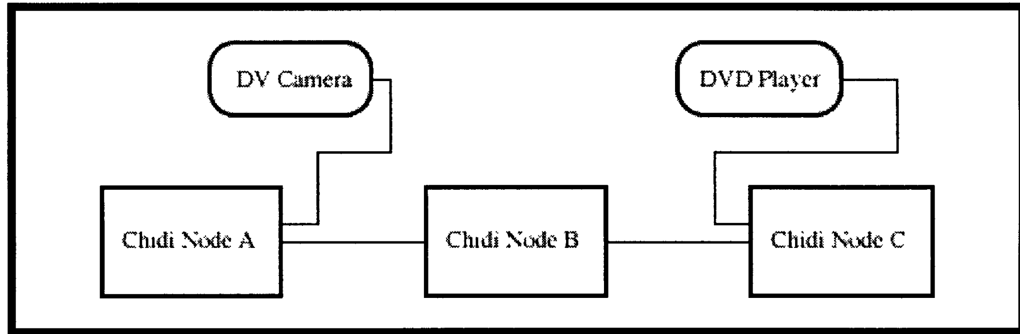


Figure 5 : Typical Chidi 1394 Network

Of course, there are limitations to using 1394. For example, the maximum cable length, hence the point-to-point distance of network devices, is 4.5 meters. This is an obvious problem if Chidi host machines are not sitting close together. Long-distance 1394 technology is currently under development to link devices whose placement exceeds network constraints. [3]

Possible 1394 solutions already exist on the marketplace, such as Adaptec's FireCard AHA-8940 1394 PCI Host Adapter. The AHA-8940 is a PCI card which allows a host machine to connect to a 1394 bus. Such existing adapters would certainly provide the connection to digital I/O that we are interested in, but ultimately isn't what we are looking for. First, one adapter would be needed for each Chidi card, meaning two PCI cards would be necessary to hook up each node to a 1394 network. Second, support for the adapter is currently restricted to Windows and Macintosh.

2. Purpose

One of the principal uses of the Chidi processor is to process streams of video data through its reconfigurable processor sub-unit. An important issue is where the video data comes from. Data can be made available to Chidi via the PCI bus of its host machine. Presumably, however, this data is "old". What if we want to use Chidi to process "current" data? A direct interface to a 1394 bus network provides an additional source for digital multimedia input, including camcorder and camera output. The ability to bring in video direct from digital camcorder and camera devices solves the problem to providing "current" data.

The first obvious use of a 1394 bus interface is a gateway to digital media input and output -- a 1394 Bus Interface provides Chidi with a direct link to a network of 1394 devices, which can include digital video cameras, digital video recorders, and countless other types of digital media devices. However, for our purposes, the 1394 network provides an additional use. The 1394 technology can also be used to link Chidi processors which would allow Chidi-to-Chidi communication at a higher rate than over the local network of the host machine.

The two main goals in coupling 1394 bus technology with Chidi are to provide Chidi with both digital camera input and node-to-node communication at a high rate. In order to facilitate digital camera input, the following 1394 transfer modes must be supported:

- *asynchronous no-data/quadlet packet send from GPP*
- *asynchronous no-data/quadlet packet receive to GPP*
- *isochronous block receive to memory*

As discussed earlier, status and control commands are sent to digital cameras via asynchronous no-data and asynchronous quadlet packets. Replies from the cameras are provided in the same format, while data packets are sent out by the camera as isochronous data block packets. For node-to-node communication, these additional transfer modes are necessary:

- *asynchronous block send from memory*
- *asynchronous block receive to memory*

The functionality used to provide digital camera commands would be shared for sending and receiving status and control commands in node-to-node communication. Block transfers support the transfer of large buffers of data.

This paper discusses the design and implementation of an IEEE1394 Serial Bus Interface for the Chidi Media Processor. The purpose of which is to facilitate digital media input and Chidi-to-Chidi communication, by integrating the functions specified above into the Chidi architecture. The scope of the research covers design of the interface from the 1394 physical layer up to the low-level driver libraries to be used by software applications.

3. Design

The design of the 1394 Bus Interface is divided into hardware and software. The software runs on the General Purpose Processor and implements the 1394 Serial Bus transaction and serial bus management layers, providing higher level applications with interface control. The hardware implements the link Serial Bus link and physical layers and interfaces the link layer to the PowerPC Bus and by doing so, the General Purpose Processor. The hardware and software together provide the support for the necessary functions as described in the previous section.

3.1 Hardware

The 1394 interface starts with three 1394 cable ports which connect into the Physical Layer, where functions such as bus arbitration, data synchronization, and data repeating are taken care of. Above the Physical Layer is the Link Layer which provides the 1394 packet delivery service. FIFO modules provide buffering between the Link Layer and the PPC Data Bus, Chidi's main data bus provides a path to memory and both the GPP and RP. The Processor-to-Link Interface Logic controls interfacing of the FIFOs with the PPC Bus and the Link Layer. Figure 6 shows a block diagram for the 1394 interface.

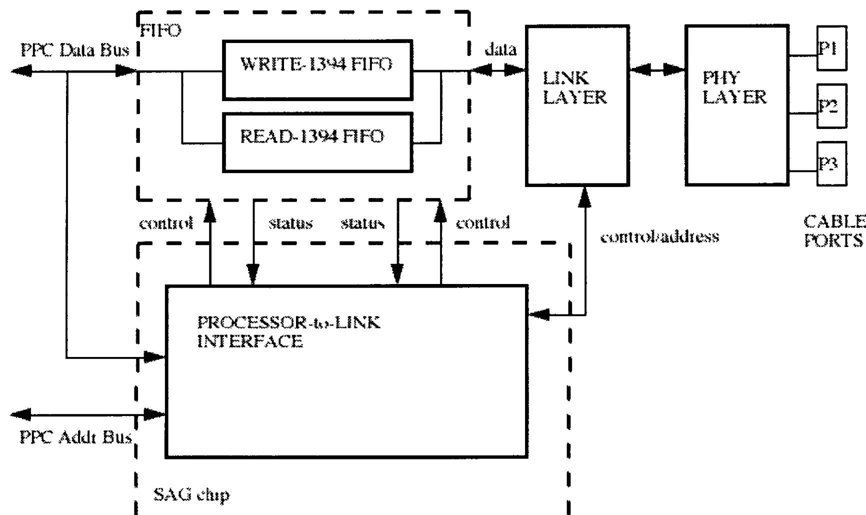


Figure 6 : 1394 Interface Block Diagram

3.1.1 Physical Layer

The physical layer provides the actual physical interface between bus nodes and also is responsible for several functions, including bus arbitration, data synchronization, data encoding

and decoding, and data repeating, Data strobe encoding is used for data bits transmitted over the bus in order to improve skew tolerance. Three 1394 connectors provide ports for point-to-point links with other bus nodes.

The aforementioned physical layer functions are implemented with an off-the-shelf 1394 physical layer transceiver, Texas Instruments' TSB21LV03. The TSB21LV03 provides the highest currently available 1394 bus speed at 200 Mb/s. Other physical layer chips are also available, but the TSB21LV03 is used because it is the fastest and also provides three bus ports.

The TSB21LV03 implements all of the necessary analog physical layer functions for a 1394 network node. It provides a data interface to link layer controllers, such as the one used in this implementation which will be introduced in the following section. Data to be transmitted over the 1394 bus is fed to the TSB21LV03 through this interface. In addition, it allows access to internal control and status registers on the physical layer chip. Status registers provide information, such as the physical ID assigned to the node and the state of the bus lines. The control registers indicate to the physical layer how to handle a bus-reset and is also used to alter its arbitration fairness values.

Control of various bus functions pertaining to bus configuration and arbitration are provided by accessing the TSB21LV03 internal registers. Bus resets are initiated by setting a particular register bit. Other bits are also used to set a node as a possible bus manager and to force a node as the bus root. The gap count used by the physical layer in bus arbitration is also stored in a write-able register. (Decreasing the gap count value lets a 1394 bus node take control of the bus sooner.)

3.1.2 Link Layer

The link layer provides a data packet delivery service for the two types of 1394 packets: asynchronous and isochronous. In asynchronous packets, a variable amount of data with command information is transferred to an explicit address and an acknowledge is returned. Isochronous packets contain a variable amount of data transferred over channels at regular intervals with no acknowledge.

Asynchronous packets carry 64 bit addresses, of which 16 bits identify the destination node and the remaining 48 bits identify an address within the node. Except during broadcasts, an acknowledge packet is sent upon reception of an asynchronous packet by the destination. Instead of being sent to an explicit address, isochronous packets are broadcast over the entire bus and identified by a channel_ID, from which bus devices decide whether or not the packet should be read. See Appendix E for a summary of packet formats.

In a Chidi network, both asynchronous and isochronous transfers are used. Command and control communication utilizes asynchronous packets, since they are not sent every cycle (i.e. they would be sent 'as necessary'). Data transfers can be sent asynchronously or isochronously depending on the size and rate of the transfer block. For example, if a node were sending just an 8 byte block, an asynchronous packet would suffice. However, a continuous stream of data between nodes or from a device, such as a DV camera would be better suited to isochronous transfer. On a network with few nodes, there is less competition for bus bandwidth, in which case isochronous capabilities might not be necessary at all, but in order to scale to larger networks, isochronous transfers become a necessity. Therefore, it is important that both forms of communication are supported in the design.

There are various link layer chips commercially available. Some are relatively powerful, but aren't applicable to Chidi. For example TI's PCI-to-1394 link chip isn't applicable to Chidi, since our design interfaces with Chidi's local bus and not the PCI bus. Therefore, two less powerful link-layer controllers were considered, the TSB21LV31 and TSB12C01A. The TSB21LV31 is "tailored and optimized" for use as a peripheral link-layer controller. It provides separate 16-bit asynchronous and 8-bit isochronous interfaces, which allow data and control packets to be separated. The TSB12C01A provides a simpler host interface with just one 32-bit data interface used for both asynchronous and isochronous communication. In order to use both the asynchronous and isochronous ports on the TSB21LV31, the control signals necessary would double and separate FIFO blocks would be needed for each interface, taking up at least twice as much board area. With pins and board space, not to mention power – a 64 x 32 x 2 FIFO draws 2.5 watts – at a premium, the simpler TSB12C01A was chosen to implement the 1394 link-layer.

The TSB12C01A interfaces directly to a set of TI physical layer chips, including the TSB21LV03 which is being used to implement the physical layer in this system. Figure 7 shows a block diagram of the link to physical layer interface. A 4-bit wide data bus and 2-bit control bus run between the link layer and physical layer chips. The physical layer provides a 50 Mhz clock (SCLK) to the link layer. SCLK is used as the clock rate between the two devices.

Internally, the TSB12C01A contains three variable length 32-bit wide FIFOs, an asynchronous transmit FIFO (ATF), isochronous transmit FIFO(ITF), and general receive FIFO (GRF). The ATF is used to hold data to be transmitted asynchronously, while the ITF contains data to be transmitted isochronously. The GRF provides buffering for packets received, regardless of transfer type, over the 1394 bus. The total FIFO size in the TSB12C01A is 509 32-bit words, which is distributed among the three FIFOs.

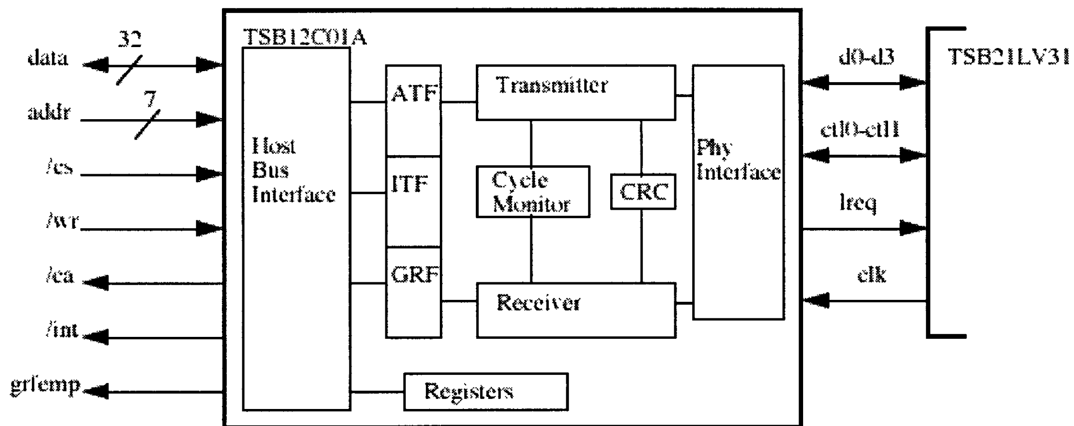


Figure 7 : TSB12C01A Link Layer Controller

The TSB12C01A also provides 1394 cycle master capabilities and takes care of error detection. When configured as cycle master, the chip generates cycle-start packets based on its internal clock. (In order to support isochronous communication, exactly one node in a 1394 bus network broadcasts a cycle-start packet every 125 us to the rest of the network which indicates the beginning of a new isochronous cycle.) In addition, the TSB12C01A generates and transmits a 32-bit cyclic redundancy check (CRC) based on the header and data of each packet it transmits. Received packets are checked against the CRCs provided with them.

A host bus interface allows the TSB12C01A to connect to the rest of the system. The interface consists of a 32-bit data bus, an 8-bit address bus and handshaking signals, which run at a maximum clock rate of 33 MHz. (see Figure 7). The address bus specifies either an internal register location or an internal FIFO location for data to be written to or read from. The internal registers direct the operation of the TSB12C01A. See appendix B for an overview of the internal register and their functions. The TSB12C01A is interrupt driven and also provides a dedicated output signal that indicates when there is data in the GRF.

3.1.3 1394 FIFO Buffer

The link layer bus is 32 bits wide and runs at 33 MHz, while the Chidi main data bus is 64 bits wide at 66 MHz. Two 64 x 64 bit FIFOs provide buffering for moving data between the busses. The WRITE-1394 FIFO is used for data going into the link layer and the READ-1394 FIFO buffers data headed out of the link layer for the main bus. The FIFOs are implemented

using two 64 x 36 x 2 bidirectional FIFOs, with one connecting to the lower 32 bits of the 64-bit PPC bus, and one for the upper 32 bits. This allows the 64 bit words from the main bus to be easily broken up into 32 bits to fit the link data interface. Mailbox registers on the FIFO chips provide bypass paths to and from the link layer.

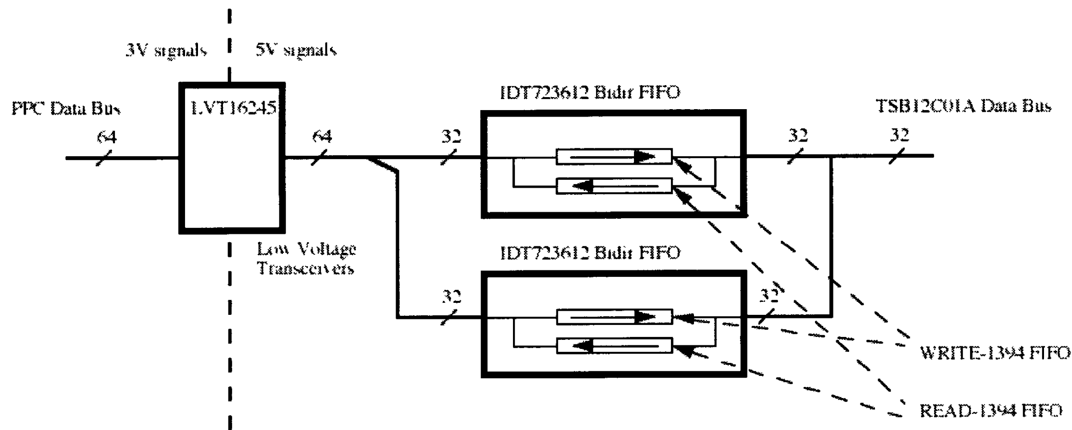


Figure 8 : 1394 FIFO Buffer

A set of LVT16245 low-voltage transceivers provide buffering between the FIFOs' 5 volt data pins and the 3 volt PPC bus. The buffering keeps the FIFOs from possibly driving 5 volts to the MPC106 and PPC604e (GPP), which are 3 volt devices.

3.2 Processor-to-Link Interface Logic on SAG

The goal of the Processor-to-Link Interface Logic (PLI) is to abstract the 1394 link layer from the GPP so that it has the appearance of a generic local bus slave. To that end, the PLI performs the following functions: 1) control the interface between FIFO side B and the link layer, 2) control the interface between FIFO side A and the PPC data bus, and 3) arbitrate for the PPC Bus as both a bus master and a bus slave. As mentioned above, the PLI logic is housed on the SAG FPGA, along with other logic modules essential to Chidi, but unrelated to the 1394 interface

The PLI logic is divided into sub-modules, as described in Figure 9. The Link Interface sub-module controls the FIFO to link layer interface. The Data Transmitter/Receiver (DTR) sub-module handles the FIFO to PPC data bus interface. The Busctrl sub-module handles PPC bus arbitration. In addition, a 1394 Hardware Registers sub-module provides the GPP with control of the Processor-to-Link Interface. The Bus Mastership Requesting Logic sub-module determines when the PLI needs control of the PPC Bus.

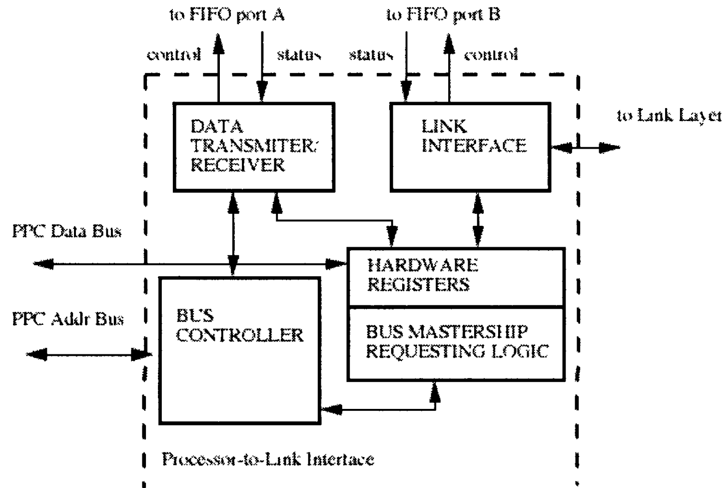


Figure 9 : Processor-to-Link Interface Block Diagram

3.2.1 Transfer Function Implementation

Now that the basic structure of the interface has been established, we can discuss implementation of the five necessary transfer functions. Again, those functions are:

1) asynchronous no-data/quadlet packet send from GPP
2) asynchronous no-data/quadlet packet receive to GPP
3) isochronous block receive to memory
4) isochronous block send from memory
5) asynchronous block send from memory
6) asynchronous block receive to memory

Asynchronous no-data/quadlet packet send from GPP

In order for the GPP to send an asynchronous no-data/quadlet packet, the GPP first writes packet information (packet length and packet type) to the Control Register in the 1394 Hardware Registers sub-module. Once the Control Register has been set, the GPP is then allowed to write data to the WRITE-1394 FIFO. After the GPP has written all of the quadlets in the packet to the FIFO, it must wait until a ready_next_packet interrupt from the PLI, before transmitting another packet.

Asynchronous block send from memory / Isochronous block send from memory

To have data buffers in memory transmitted through the 1394 interface, the GPP writes information to the PLI describing the transfer, which the PLI uses to transfer the data from the

memory to the WRITE-1394 FIFO. First, the GPP sets the Input Start Register (buffer starting address), Input Stop Register (buffer ending address), and Memory Packet Length Register (buffer transmit packet size) in the 1394 Hardware Registers sub-module. With this information, the PLI transmits the buffer in multiple packets of the size specified in the Memory Packet Length Register. Interrupts are signalled to the GPP to write packet headers and when the entire buffer has been transferred.

Asynchronous no-data/quadlet packet receive to GPP / Isochronous block receive to memory / Asynchronous block receive to memory

Data received into the READ-1394 FIFO is automatically written by the PLI into a specifically allocated buffer in memory. The location of the buffer is set in two hardware registers: Output Start Register (buffer starting address) and Output Stop Register (buffer ending address). In addition, a third register, Output Stop Half Register provides the address at the midpoint of the buffer. Interrupts are generated when data has filled the buffer up to the value in Output Stop Half and when data has reached Output Stop. These interrupts signal to the GPP to determine the contents of the buffer. For example, when the receive_buffer_half_full interrupt has been generated, the GPP looks at the first section of the buffer, while the PLI continues to fill up the second half of the buffer. Similarly, when the receive_buffer_full interrupt is generated, the GPP can read the second section of the buffer, while the PLI continues writing from the beginning again. By scanning through the buffer, the GPP reads in the packets which were meant to be received by the processor, and keeps track of the location of the received block data.

An additional interrupt, received_data_waiting, is signalled after 1394 packet reception has ceased for a certain amount of time and data has been received in the buffer since the last receive_buffer_half_full or receive_buffer_full interrupts. This alerts the processor of received data that should be read, since a half full or full interrupt is not likely to occur until later.

Accessing Link Layer Registers

The FIFO buffers provide mailbox registers which bypass the actual FIFO memory. These bypass paths to and from the PPC bus and the Link Layer are used for accessing the Link Layer's internal registers. Using the bypass paths separates configuration accesses from data accesses and also allows register accesses to not have to wait for the FIFO buffers to clear.

3.2.2 FPGA Development

This section discusses the characteristics of the specific FPGA used to implement the Processor-to-Link Interface Logic and also covers the development process used in programming the FPGA. In following sections, as the Interface Logic is presented, design optimization techniques specific to the FPGA used will also be mentioned.

The firmware providing the interface between the General Purpose Processor and the commercially available 1394 link layer and physical layer chips is implemented on an FPGA named the Stream Address Generator (SAG). In addition to housing the aforementioned interface, the SAG also generates addresses for input and output streams to and from the Reconfigurable Processor (RP). Although the 1394 Interface section of the SAG takes up only ~700 logic cells, an Altera 10K50 FPGA (2880 useable logic cells) was selected for this implementation, with the bulk of its resources being used up for Chidi stream address generation.

Altera FLEX10K Devices

Altera's FLEX10K family of programmable logic devices provide integrated embedded arrays and logic arrays in a single device. The embedded array is used to implement memory

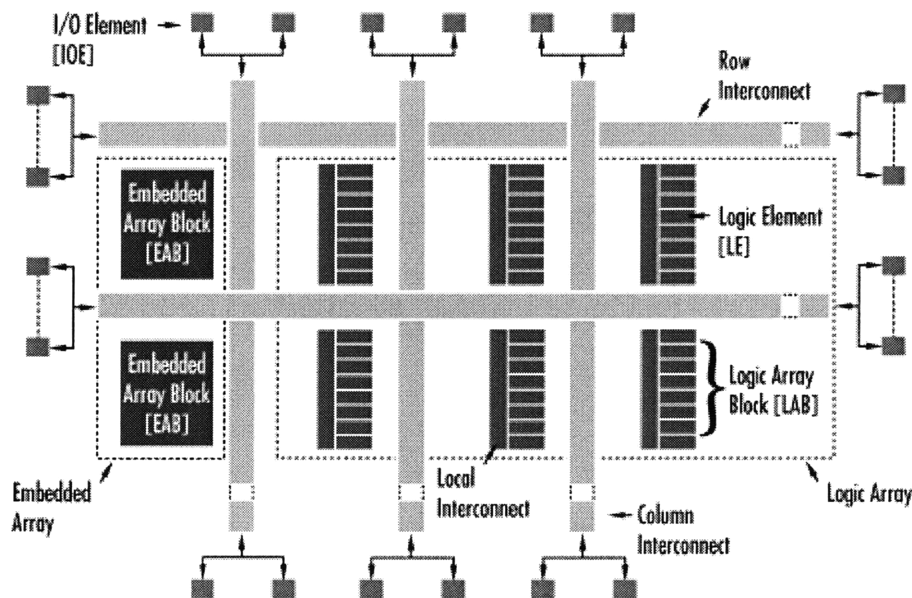


Figure 10 : Altera FLEX 10K Internal Architecture

functions or complex logic, while the logic array are used for general gate logic. Typical gate size of devices in the 10K family range from 10,000 to 250,000. The number of available I/O pins in 10K devices ranges from 134 to 470 pins.

The embedded array is made up of a series of embedded array blocks (EABs), which can be configured as memory (RAM, ROM) or complex logic functions, such as multipliers or microcontrollers. EABs can be used individually or together to create larger designs. Similarly, the logic array is comprised of logic array blocks (LABs), which themselves are made up of eight logic elements (LEs). Figure 10 shows a diagram of the basic internal architecture of the 10K devices. Each LE contains a four-input LUT, which can quickly compute any function of four variables, and a flip-flop with a synchronous enable. In combinational functions, the LUT output bypasses the flip-flop, which can be programmed for D, T, JK, and SR operation, and drives the output of the LE. Figure 11 gives the FLEX 10K Logic Element structure.

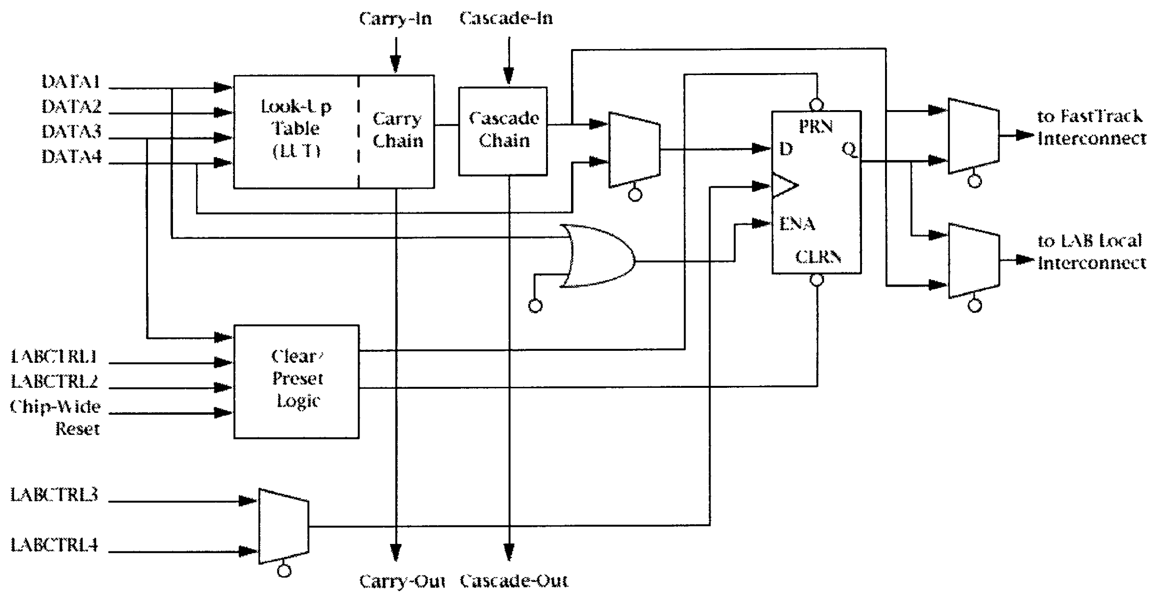


Figure 11 : Altera FLEX 10K Logic Element Structure

Development Process

The FPGA development process used for this project is made up of three main steps: VHDL development, synthesis compilation with Synopsys Design Compiler, and project compilation using Altera Maxplus2.

VHDL Development

The first main step in the FPGA development process is to create a description of the design using VHDL (VHSIC Hardware Description Language), a language for describing digital systems. The VHDL code is then analyzed and simulated, in order to test for code functionality. Since, at this stage, the design has yet to be synthesized into gates, this simulation reveals only functional behavior and provides no insight into performance or size of the design.

Synthesis Compilation using Synopsys Design Compiler

After the VHDL description has been functionally tested, the next step is to synthesize the design for Altera FLEX10K devices with the Synopsys Design Compiler. Altera design libraries are used to create architecture-specific optimization and mapping. The design is then presented as an EDIF netlist file which is read by Maxplus2.

Project Compilation using Altera Maxplus2

In Maxplus2, the design specified by the EDIF file created by Design Compiler is compiled for the specific device being used, in this case, FLEX 10K50. The Maxplus2 compiler places and routes the design into actual FLEX 10K elements. Once a project has been compiled, device timing performance can be analyzed. In the case that the analysis returns unsatisfactory results, constraints can be provided to the compiler, in order to increase performance. The principal methods to increase speed of compiled projects in Maxplus2 included setting global timing constraints, setting logic cliques, and explicitly placing logic and pin assignments.

Synthesis compilation also can be done using the Altera Maxplus2 software, thus removing the hassle of moving the design between different programs. However, Synopsys Design Compiler is used since it is the better synthesis compiler.

3.2.3 Data Transmitter/Receiver Sub-Module

The Data Transmitter/Receiver provides controls for the PPC Bus side of the FIFO buffer. It reacts to PPC bus arbitration signals and input signals which indicate the type of transaction on the current PPC bus data tenure. It supports four main actions: GPP read from READ-1394 FIFO, GPP write to WRITE-1394 FIFO, read from READ-1394 FIFO to memory, and write to WRITE-1394 FIFO from memory. Table 1 lists the Data Transmitter/Receiver inputs and outputs.

Table 1 : Data Transmitter/Receiver Signal Descriptions

Signal Name	Direction	Description
Bus Controller Interface		
Read	Input	Read – Indicates that current data bus tenure is a processor read from 1394 FIFOs
Write	Input	Write – Indicates that current data bus tenure is a processor write to 1394 FIFOs
Bypass	Input	Bypass – If Read or Write are asserted, indicates that current transaction is through FIFO bypass path
Memread	Input	Memory Read – Indicates that current data bus tenure or address bus tenure is a mastered read from 1394 FIFOs to memory
Memwrite	Input	Memory Write – Indicates that current data bus tenure is a mastered write from memory to 1394 FIFOs
Burst	Input	Burst – Indicates that current transaction is a burst transfer
Burst_addr	Input	Burst Address Tenure – Indicates that the transaction currently with the address tenure (i.e. the next data transaction) is a burst. Used for reads to memory only.
/dbg	Input	Data Bus Grant – Indicates PPC Bus arbiter has granted mastership of the data bus
/dbglb	Input	Data Bus Grant for Local Bus Slave – Indicates PPC Bus arbiter has left /ta to be asserted by local bus slave
/dbb_604	Input	Data Bus Busy – Indicates that PPC604 (GPP) is driving the data bus
/ta_in	Input	Transfer Acknowledge In – Signals the end of a mastered transaction
/ta_out	Output	Transfer Acknowledge Out – Signals the end of a transaction for which the module is a slave
FIFO Interface		
Fwregoeab	Output	1394 Register AtoB Output Enable – Enables A to B data flow through 1394 FIFO registered buffers
/Fwregoeba	Output	1394 Register BtoA Output Enable – Enables B to A data flow through 1394 FIFO registered buffers
Fwfifomba	Output	1394 FIFO Mailbox Select Port A – Asserted selects the 1394 FIFO mailbox registers
Fwfifowra	Output	1394 FIFO Write/Read Port A – Asserted selects 1394 FIFO write operation; negated selects read operation
/fwfifocsa	Output	1394 FIFO Chip Select Port A – Asserted enables 1394 FIFO read or write on rising clock edge
/fwfifombf2	Input	1394 FIFO Mailbox Flag 2 – Indicates valid data in READ-1394 FIFO mailbox register

GPP Read from READ-1394 FIFO

The Data Transmitter/Receiver supports a processor read from 1394-READ FIFO which can be used to read FIFO Mailbox Register 2. This transaction is used to read the contents of link layer registers. (A processor read from the actual FIFO port isn't supported since all received packets are transferred to memory).

When both the read and bypass inputs are asserted, the read_bypass state machine is activated. It waits for /dbglb to be asserted then it waits for the /fwfifombf flag to be asserted. (The Link Interface module provides the link layer register data to the mailbox register.) Once the mailbox flag is asserted, /fwfifocsa is set to read the data out of the bypass path and onto the bus, and /ta_out is asserted, signalling that the transaction is comple

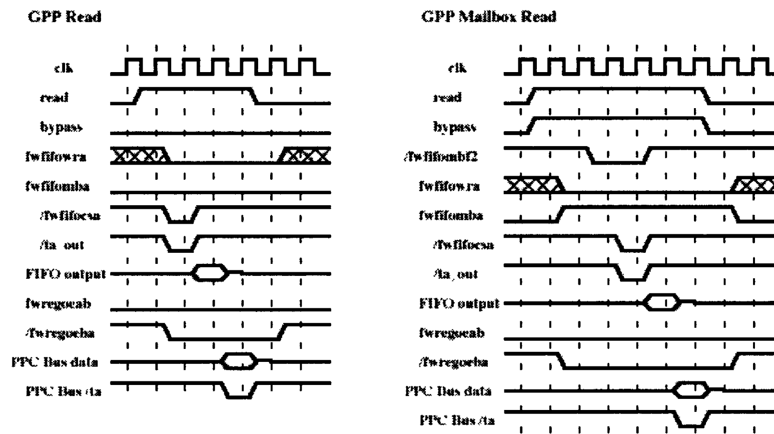


Figure 12 : GPP Read from READ-1394 FIFO Timing Diagrams

GPP Write to WRITE-1394 FIFO

When the write input is asserted, the write state machine jumps to the wait_dbb state where it waits for the /dbb_604 signal to be asserted. After /dbb_604 is asserted, which means the processor is driving valid data on the PPC Bus, the /fwfifocsa signal is asserted along with fwfifowra. In addition, /ta_out is asserted completing the transaction.

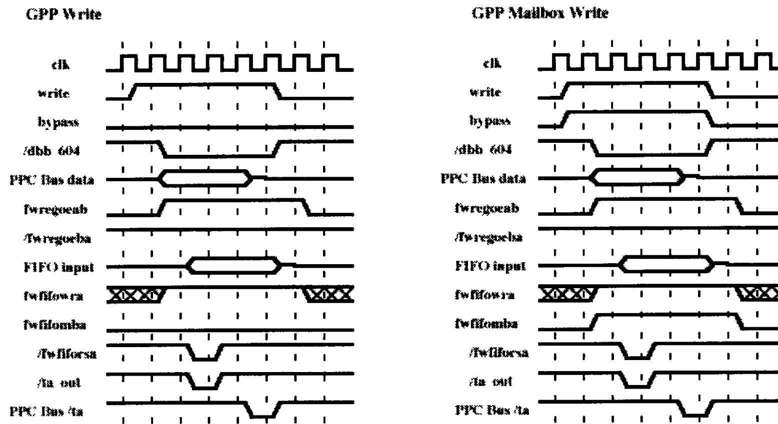


Figure 13 : GPP Write to WRITE-1394 FIFO Timing Diagrams

Processor writes to a link layer register via FIFO Mailbox Register 1 are made by asserting the bypass input. If the bypass input is '1', then fwfifomba is asserted during the transaction enabling the write to the mailbox register.

Write to WRITE-1394 FIFO from memory

The memwrite input signals that the current data tenure is a write of data from memory to the WRITE-1394 FIFO. If burst is not also asserted, then the memwrite signal moves the write_mem state machine from the wmem_idle to the wmem_write_single state, where it waits for /ta_in, which means that the data from memory has been placed on the bus, to be asserted before returning to idle. If burst is asserted, then the write_mem state machine initially transitions to wmem_write_burst1. On successive assertions of /ta_in, it moves to wmem_write_burst2, wmem_write_burst3, and wmem_write_burst4 and back to wmem_idle. In both cases, when /ta_in is asserted and the write_mem state does not equal wmem_idle, /fwfifocsa is asserted, writing the data into the FIFO.

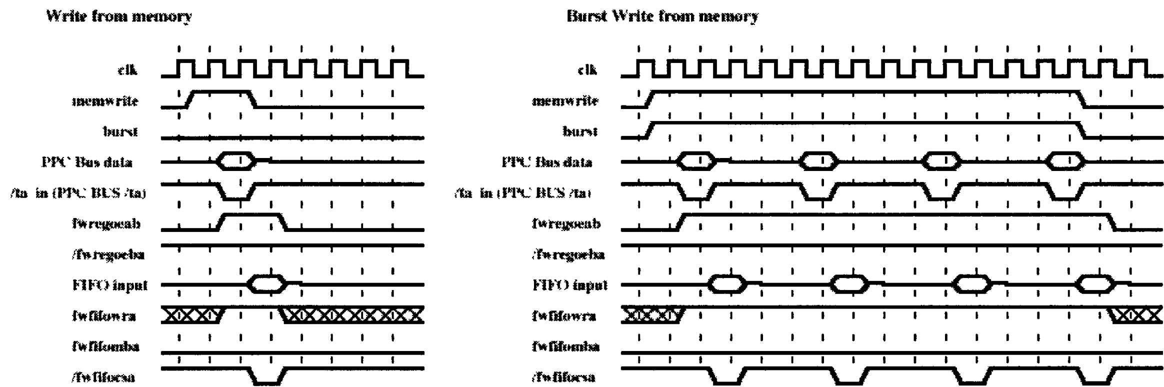


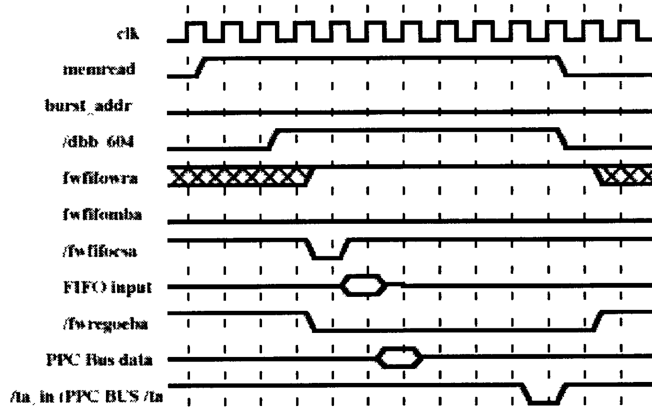
Figure 14 : Write from memory to WRITE-1394 FIFO Timing Diagrams

Read from READ-1394 FIFO to memory

The asserted memread input indicates that data in the READ-1394 FIFO is to be transferred to system memory. When the Data Transmitter/Receiver receives an asserted memread input and deasserted burst input, it asserts /fwfifocsa when /dbb_604 goes high. When burst is asserted, /fwfifocsa is also asserted three more times to provide data for the succeeding three words in the burst. The data read out of the FIFO is valid only for one clock cycle, so the assertion of /fwfifocsa is timed to provide data that is valid when latched by the memory controller.

This function differs slightly from the previous three in that memread is actually asserted when the address tenure has a 1394 read-to-memory or when the data-tenure has a 1394 read-to-memory. The reason for this is to provide the memread signal earlier, so that the /dbb_604 signal is caught by the Data Transmitter/Receiver. Because of register delays between the PPC Bus interface and this sub-module introduced for higher performance, the rising edge of /dbb_604 would be seen before memread if memread were asserted solely during the data tenure.

Read to memory



Burst Read to memory

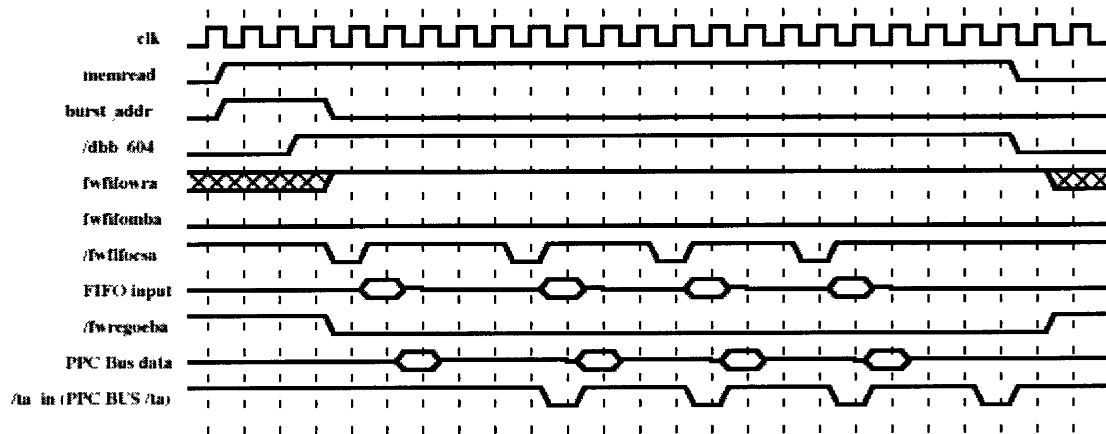


Figure 15 : Read to memory from READ-1394 FIFO Timing Diagrams

3.2.4 Link Interface Sub-Module

The Link Interface provides the interface between the FIFOs and the link layer. It controls the flow of data from the WRITE-1394 FIFO and the Write Mailbox Register to the link layer and from the link layer to the READ-1394 FIFO and the Read Mailbox Register. The Link Interface responds to the FIFO and link layer status signals by asserting control signals to move data between the FIFOs and link layer.

Table 2 : Link Interface Signal Descriptions

Signal Name	Direction	Description
PLI Logic Interface		
Reg_read	Input	Link Register Read – Indicates that a link layer register read is the current data transaction
Reg_write	Input	Link Register Write – Indicates that a link layer register write is the current data transaction
Reg_addr_in <5..0>	Input	Link Register Address Input – The address of the link layer register to be accessed. Valid when reg_read or reg_write are asserted.
Data_length <8..0>	Input	Packet Data Length – Length in quadlets of the next packet to be written from FIFO to link layer. Should be provided before the first quadlet is transferred.
Async	Input	Packet Async/Isoch – When asserted, indicates that the next packet to be written from FIFO to link layer is asynchronous. Negated indicates that packet is isochronous
Next_packet	Output	Next Packet - Indicates that the data_length input for the next packet may now be provided
FIFO Interface		
/Fwfifocsb0	Output	1394 FIFO Chip Select Port B Low – Must be asserted to enable low FIFO Port B access on rising clock edge
/Fwfifocsb1	Output	1394 FIFO Chip Select Port B High – Must be asserted to enable high FIFO Port B access on rising clock edge
Fwfifowrb	Output	1394 FIFO Write/Read Port B – Asserted selects FIFO Port B write operation; negated selects FIFO Port B read operation
Fwfifombb	Output	1394 FIFO Mailbox Select Port B – Asserted select FIFO Port B mailbox registers
Fwfifoenb	Output	1394 FIFO Enable Port B – Asserted enables FIFO Port B read or write on rising clock edge
/Fwfifoefb	Input	1394 FIFO Empty Flag Port B – Asserted indicates that WRITE-1394 FIFO is empty
/fwfifoafb	Input	1394 FIFO Almost Full Flag Port B – Asserted indicates that READ-1394 FIFO is almost full
/fwfifombf1	Input	1394 FIFO Mailbox Flag 1 – Asserted indicates that WRITE-1394 FIFO mailbox register has received valid data
Link Layer interface		
/fwcs	Output	Link Cycle Start – Enables access to Link Layer configuration registers or internal FIFO
/fwwr	Output	Link Read/Write Enable – Asserted (low) selects a Link Layer write operation; deasserted (high) selects a Link Layer read operation
/fwca	Input	Link Cycle Acknowledge – Asserted indicates that Link Layer access is complete
Llc_addr <5..0>	Output	Link Address – Link Layer address bus addresses internal FIFOs and configuration registers
Fwgrfemp	Input	Link GRF Empty – Asserted (high) indicates that the Link Layer GRF is empty
Fwclk	Input	Link Layer Host Bus Clock – 33.33 MHz Link Layer Host Interface bus clock.

The four main parts of the Link Interface are the Link FSM, words left counter, register address latch, and address selector. Figure 16 shows a block diagram of the Link Interface sub-module.

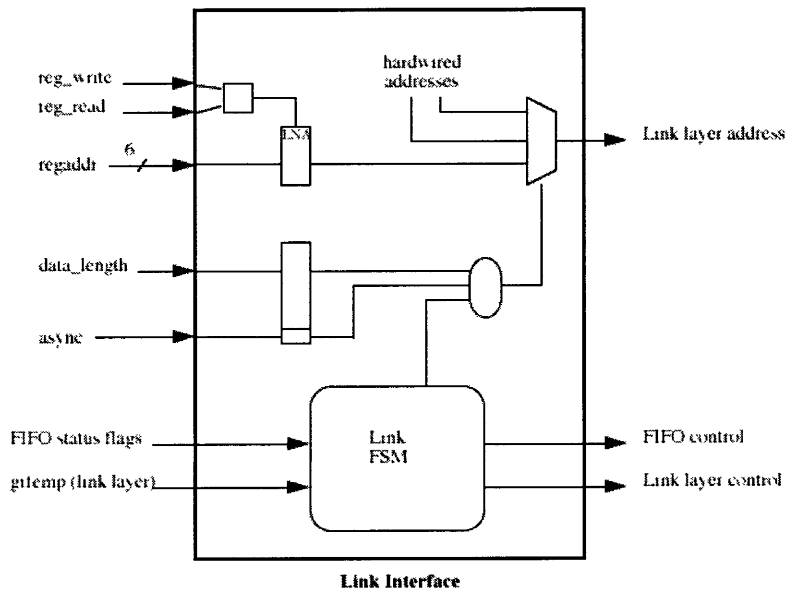


Figure 16 : Link Interface Block Diagram

Register Address Latch

The register address latch latches the link address to be accessed when a register read or write is requested. It is enabled when either of the reg_read or reg_write inputs are asserted. The latch output is passed on to the address selector.

Words_left_counter

The words left counter counts down the number of quadlets left to be transferred from the FIFO to the link layer controller in the packet currently being written. The counter is loaded with the data_length input at the start of each packet. At the same time the data_length input is loaded into the counter, the async input bit is latched. The counter decrements as each quadlet is read from the FIFOs and written to the link layer. The value kept by the counter and the latched async bit are used by the address selector to determine the link layer address to be written to.

Address selector

The address selector outputs the six bit address to the link layer during reads and writes. It selects from among a set of hardwired addresses and the output of the register address latch. The latched register address is selected during link register reads and writes. One of the hardwired addresses is used for reads from the GRF or writes to the ATF or ITF. Table 3 lists and describes the set of link FIFO addresses for the TSB12C01A Link Layer Controller.

Table 3 : Link Layer FIFO Access Addresses

Address	Name	Description
0x80	ATF_first	Location to write first quadlet of an asynchronous packet
0x84	ATF_continue	Location to write second to next-to-last quadlets of an asynchronous packet
0x8C	ATF_update	Location to write last quadlet of an asynchronous packet
0x90	ITF_first	Location to write first quadlet of an isochronous packet
0x94	ITF_continue	Location to write second to next-to-last-quadlets of an isochronous packet
0x9C	ITF_update	Location to write last quadlet of an isochronous packet
0xC0	GRF_data	Location to read received remote data

Link FSM

The Link FSM performs five different sequences of operations: read_reg, write_reg, read, write, and advance_fifo. Figure 17 shows the state diagram for the Link FSM.

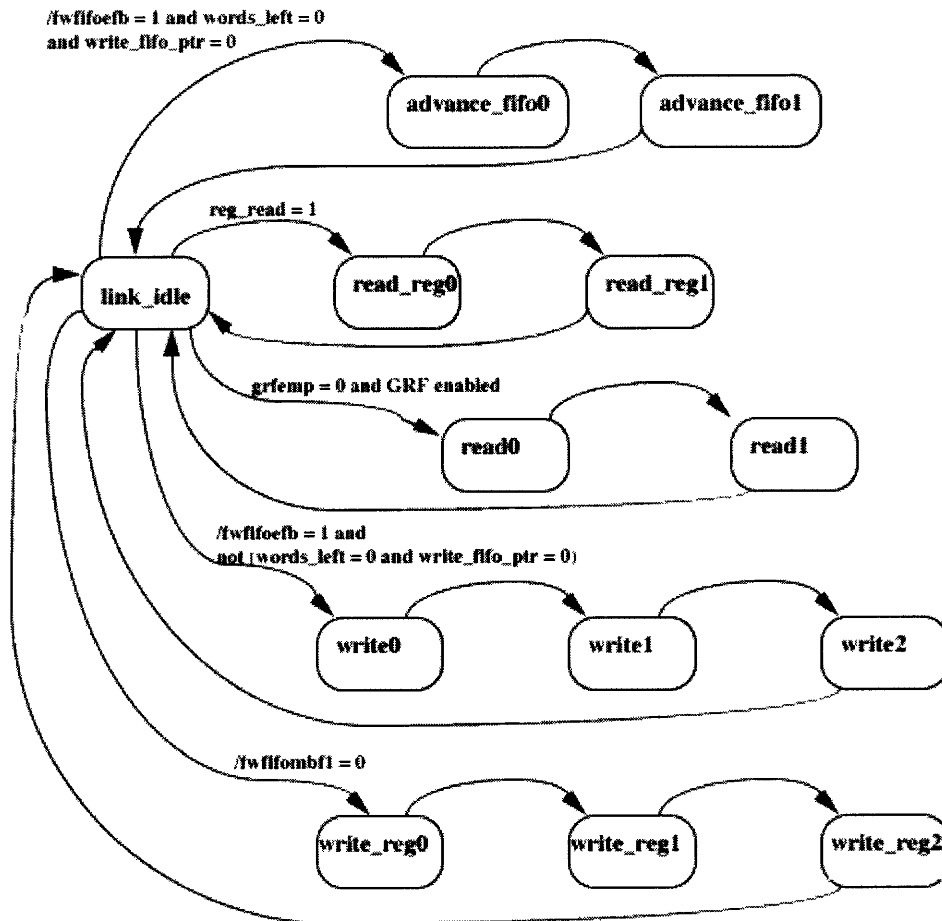


Figure 17 : Link FSM State Diagram

Read_reg

If the Link FSM receives and asserted reg_read input from the higher level PLI logic, then it initiates the read_reg sequence. It reads from the link layer using the address provided in the register address latch and writes the returned value into the FIFO mailbox registers.

Write_reg

On a link layer register write, the reg_write input is asserted along with the destination register address so that the address can be latched. Sometime afterwards, the write_reg sequence is triggered by the FIFO mailbox 1 flag being asserted, which indicates that data to be written to a link register is available in the FIFO mailbox register. After this signal has been asserted, the data is read out of the FIFO mailbox and provided to the link layer controller with the latched register address.

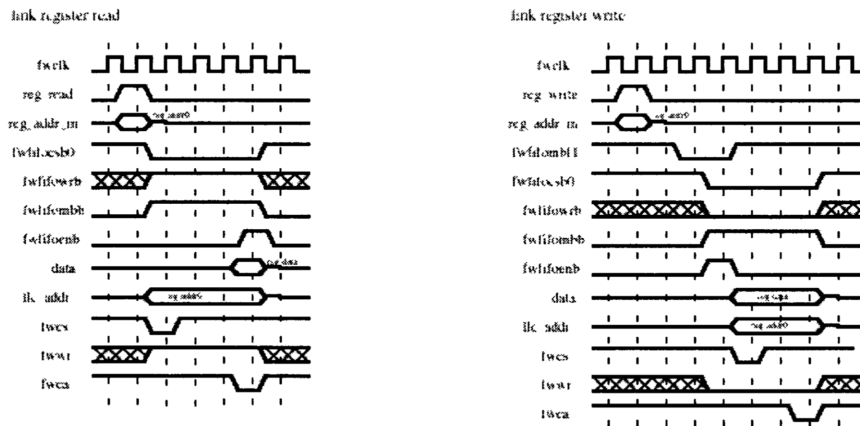


Figure 18 : Link Register Read and Write Timing Diagrams

Write

If data is present in the WRITE-1394 FIFO, the Link Interface will write the next available 32 bit word into the link layer by reading from the WRITE-1394 FIFO and writing to the link layer controller. Writes to the link layer controller must be accompanied with an address, specifying an internal location. When writing to a transmit FIFO (ATF or ITF), different addresses are used to indicate the position of the quadlet being written within the packet currently being transferred. The Link Interface determines the address it selects by looking at the words_left_counter and the async bit. These reveal whether a quadlet is first, middle, or last in a packet and whether the packet is asynchronous or isochronous. Hence the link address can be derived (see table 3).

Quadlets are written alternately out of the high FIFO then from the low FIFO. The write_fifo pointer keeps track of the current 32-bit FIFO being read from. It resets to the high FIFO, then after a quadlet has been read out of the high fifo, it switches to the low FIFO. After the low FIFO has been read, it returns to the high FIFO and so forth.

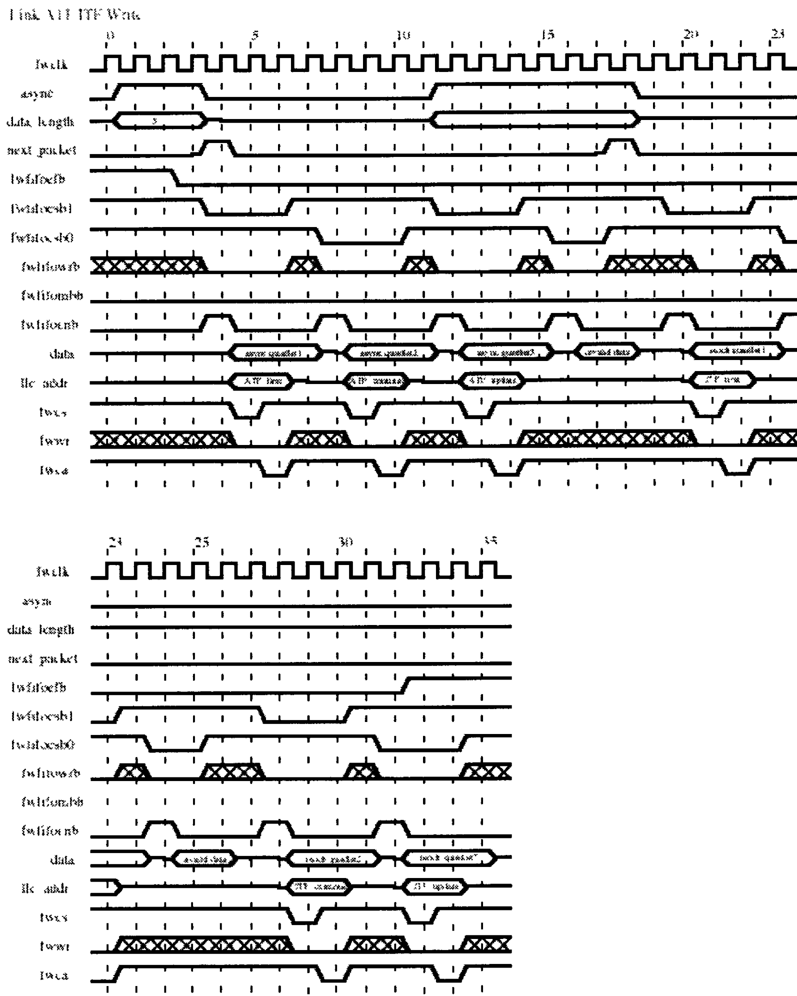


Figure 19 : Link ATF/ITF Write Timing Diagram

Read

When the grfemp signal from the link layer is deasserted (GRF is not empty) to indicate that remote data has been received into the link layer, the link FSM begins the read sequence if the READ-1394 FIFO is not alread almost full. It reads at the GRF address and writes the received quadlet into the READ-1394 FIFO. The link layer alternates between writing the received quadlet to the high FIFO and the low FIFO.

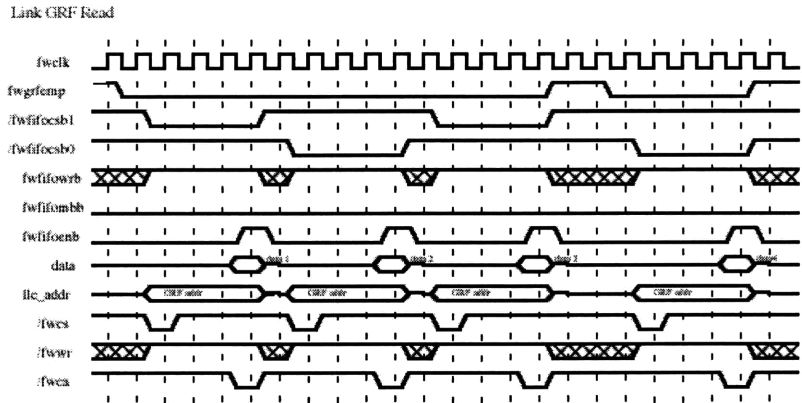


Figure 20 : Link GRF Read Timing Diagram

Advance_fifo

The advance_fifo sequence is used to restart the write_fifo pointer back to the high FIFO by discarding the next word in the low FIFO. This is necessary when a packet made up of an odd number of quadlets has just been written. Figure 21 shows an example of when it is necessary to advance the FIFO.

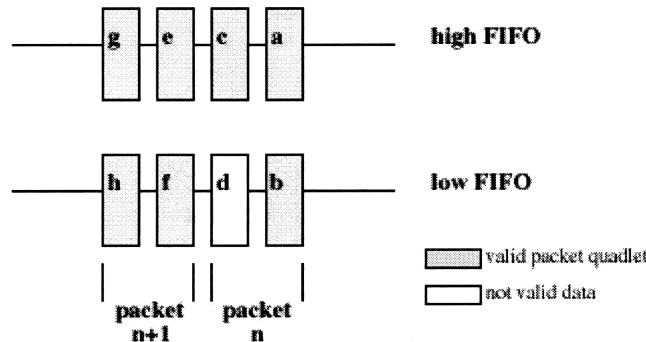


Figure 21 : Advance FIFO Example Case

After packet *n* has been written to the link layer, the write_fifo pointer points at quadlet *d*. However, quadlet *d* is just used for padding and is not to be written to the link, so it must be discarded and the write_fifo pointer should be set to point at quadlet *e*. This case is detected when the words_left_counter indicates the end of a packet and the pointer points at the low FIFO. The quadlet is discarded by reading from the FIFO without initiating a write transaction to the link layer.

3.2.5 Hardware Registers Sub-Module

A set of registers located on the SAG provide the processor with control of the 1394 Bus Interface. Table 4 lists the registers along with their addresses as seen from the GPP. Note that the registers are addressed using one hot encoding. Since the address space was large compared to the space needed, one hot encoding was utilized to decrease address decoding times.

Table 4 : Hardware Register Addresses

Chidi Memory Space Address	Hardware Register
0x6000_0008	Control Register
0x6000_0010	Input Start Register
0x6000_0020	Input Stop Register
0x6000_0040	Output Start Register
0x6000_0080	Output Stop Half Register
0x6000_0100	Output Stop Full Register
0x6000_0200	Memory Packet Length Register
0x6000_0400	Interrupt Status Register
0x6000_0800	Interrupt Mask Register
0x6000_1000 to 0x6000_1080	Link Layer Registers

Control Register

The Control Register holds information regarding the next packet to be read out of the WRITE-1394 FIFO to the Link Layer. It contains the data length in quadlets of the packet and also a bit indicating whether the packet is asynchronous or isochronous. Before the Link Interface begins reading the packet out of the FIFO and into the Link Layer, it reads the value in the control register. As discussed earlier, it needs this info in order to determine the packet type and also when the packet begins and ends, because these effect the address provided to the Link Layer controller on writes. The control register holds information regarding only one packet, thus there can be at most two packets in the WRITE-1394 FIFO at any given time – one in the process of being transferred to the Link Layer and the other for whom the control register holds valid info. This characteristic could be modified to allow a greater number of packets by adding more control registers. For each extra control register added, one more packet can fit into the FIFO.

Table 5 : Control Register Field Descriptions

Bit(s)	Field Name	Description
0 to 8	Data_length	Length in quadlets of the next packet to be read out of the WRITE-1394 FIFO
9	Async	If asserted, next packet to be read out of the WRITE-1394 FIFO is asynchronous. If not asserted, then isochronous
10	Writing_header	Bit should be set when writing to data_length and async fields prior to writing the header quadlet of a packet during a memory buffer transfer
11	GRF_read_enable	Bit must be set to enable Link Interface reading of Link Layer GRF

Input Start Register and Input Stop Register

These registers are used when the GPP requests the PLI to transfer a block of memory over the 1394 bus. The Input Start Register holds the memory address of the first word in the buffer to be transferred, while the Input Stop Register contains the address of the word *after* the last word in the buffer to be transferred.

An internal address counter, called the Input Counter, points to the address of the next word to be transferred. Whenever the address specified in Input Counter does not equal the value in the Input Stop Register, the PLI will move the 64-bit word addressed by Input Counter from memory to the WRITE-1394 FIFO and increment Input Counter. When Input Counter equals Input Counter Stop, the input_counter_done interrupt bit is set. Writes to *either* the Input Stop Register or the Input Start Register locations load the Input Counter with the data that is written. So, in order to effect a block transfer, first, the Input Stop Register is written and then the Input Start Register is written.

Memory Packet Length Register

The Memory Packet Length Register is utilized when the SAG is writing a data buffer in memory over the 1394 bus. The GPP writes the packet data length that should be used for the packets that the memory buffer will be broken up into. So, if the register contains the value *mem_packet_length* , then after *mem_packet_length* quadlets have been written from memory to the WRITE-1394 FIFO, the PLI sets the write_header interrupt. This interrupt signals to the GPP that a complete packet has been written into the WRITE-1394 FIFO and that a header needs to be written for the start of the next packet.

Output Start Register, Output Stop Half Register, and Output Stop Full Register

The Output Start Register, Output Stop Half Register, and Output Stop Full Registers are used to set the location of the 1394-Receive Buffer in memory and also to alert the GPP when new data has been placed in the buffer. The Output Start Register holds the first address in the buffer. The Output Stop Half Register and Output Stop Full Registers contain the addresses of the words immediately following the halfway point and final word of the buffer, respectively.

Output Counter, an internal counter, keeps track of the current memory address pointer, i.e. where the next word received will be written to. When the address in Output Counter equals the value in the Output Counter Stop Half Register, the receive_buffer_half_full interrupt in the Interrupt Status Register is set. Likewise, when Output Counter reaches the value in the Output Counter Stop Full register, the receive_buffer_full interrupt is set.

Interrupt Status Register and Interrupt Mask Register

The Interrupt Status Register allows the PLI to alert the GPP to different situations. When the GPP receives an interrupt from the SAG, it can read the Interrupt Status Register to determine what the cause of the interrupt was. The Interrupt Mask Register provides a manner to mask out possible interrupts that may not need to be signalled to the GPP at a particular time. Table 6 lists the bit values for both the Interrupt Status Register and Interrupt Mask Register.

In order for an interrupt to be generated, both an Interrupt Status bit *and* its corresponding Interrupt Mask bit must be set. Reads from the Interrupt Status Register return the values in the register, while writes to the Interrupt Status Register can be used to clear interrupts.

Table 6 : Interrupt Register Field Descriptions

Bit	Field Name	Description
0	Ready_next_packet	Interface ready for next packet parameters to be written into Control Register and packet quadlets to be written to WRITE-1394 FIFO
1	Input_done	Done transferring memory buffer to WRITE-1394 FIFO
2	Receive_buffer_half_full	Data in Memory Receive Buffer has reached halfway address
3	Receive_buffer_full	Data in Memory Receive Buffer has reached last address
4	Write_header	Processor must write header for memory buffer packet transfer
5	Link_interrupt	An interrupt has been signalled by the Link Layer
6	Received_data_waiting	Data has been received into the Memory Receive Buffer
7	Write_FIFO_full	WRITE-1394 FIFO is full

3.2.6 Bus Mastership Requesting Logic Sub-Module

The Bus Mastership Requesting Logic sub-module makes the requests to the Bus Controller sub-module to gain mastership of the PPC Bus. The sub-module makes both write and read requests.

Write Request

A write request is signalled when there is data to be written from the memory to the WRITE-1394 FIFO. (Note: a 1394 write request is actually a request to read from PPC Bus memory.) The write request is made whenever the following three conditions are satisfied: 1) the value in the Input Counter does not equal the value in the Input Stop Register; 2) a total of mem_packet_length quadlets have yet to be written for the currently-transmitting packet; and 3) the WRITE-1394 FIFO is not almost-full. Whenever an aligned four double-word write can be made, a burst write request will be signalled rather than a single write request.

Read Request

Read requests are asserted to the Bus Controller when there is data in the READ-1394 FIFO. In order to make the best use of PPC bandwidth, the design attempts burst reads as much as possible. When less than four double-words of data are in the READ-1394 FIFO, the read_single_wait timer is enabled. When the read_single_wait timer reaches a certain value, a request for a single read will be made. The rationale for implementing the timer is that by waiting a little longer for more data to fill up the FIFO, the more efficient burst read can be made. However, once we've waited past a certain point, then the data should just be transferred out in single reads.

It should be noted that in a system consisiting of solely a 1394 interface, making single read transactions would not be an issue. However, in the case of Chidi, there are multiple resources competing for PPC bus bandwidth, so efficiency becomes a concern.

3.2.7 Busctrl Sub-Module

The Busctrl Module provides an interface to Chidi's main address and data busses, which can act as both a bus master and slave. The main bus on Chidi conforms to the PowerPC bus protocol and is arbitrated by the MPC106. Bus master capabilities are necessary for when the SAG fulfills DMA responsibilities for reading and writing data to memory. The SAG acts as a bus slave during 1394 and register transactions requested by the General Purpose Processor.

PowerPC 604 Bus Protocol

The 604 bus provides processors with access to resources that may share the bus, such as memory, caches, and I/O devices. The basic transfer protocol supports the transfer of any number of 32- or 64- bit continuous bytes within an aligned double word to any address in a 32-bit address range over a 32-bit address bus and a 64-bit data bus. Burst transfers are also supported. The separate address and data busses support limited pipelining and split-bus transactions, allowing the different masters to have control of the address and data bus at the same time. Arbitration for bus control is normally handled by an external arbiter.

Memory accesses are made up of an address and data tenure. The address and data tenures are each comprised of bus arbitration, transfer, and termination phases. Pipelining allows the address tenure of a bus transaction to begin before the data tenure of the previous transaction finishes.

Address Bus Tenure

During address bus arbitration, a device requests control of the address bus using the /BR signal. Assertion of the /BG signal from the arbiter indicates ownership of the address bus has been granted, marking the completion of the arbitration phase. Once granted control of the address bus, the device can begin the address transfer phase, in which the physical address and transfer attributes are passed to the slave device. The transfer attribute signals indicate among other things, transfer size and transaction type. The transfer phase is marked by the assertion of the /TS (transfer start) signal by the controlling device. Address bus tenure termination occurs when /AACK is asserted to the master device.

Data Bus Tenure

The data bus arbitration phase begins with the assertion of /TS. The assertion of /TS is an implied data bus request. The arbiter grants the data bus with the /DBG (data bus grant) signal to

the requesting device. This signals the start of the transfer phase. Normal termination of the data bus tenure occurs when the /TA (transfer acknowledge) signal is asserted by the slave. On write transactions, data must be provided until /TA is asserted. On read transactions, data is returned when /TA is asserted. Burst transfers are terminated with /TA asserted for four bus cycles.

Bus Controller Functionality

The SAG acts as both a 604 bus master and a slave to the GPP. In order to perform reads and writes to memory, as described above in the SAG functionality, the SAG must act as a bus master. It provides the memory addresses over the address bus and also drives the data bus on

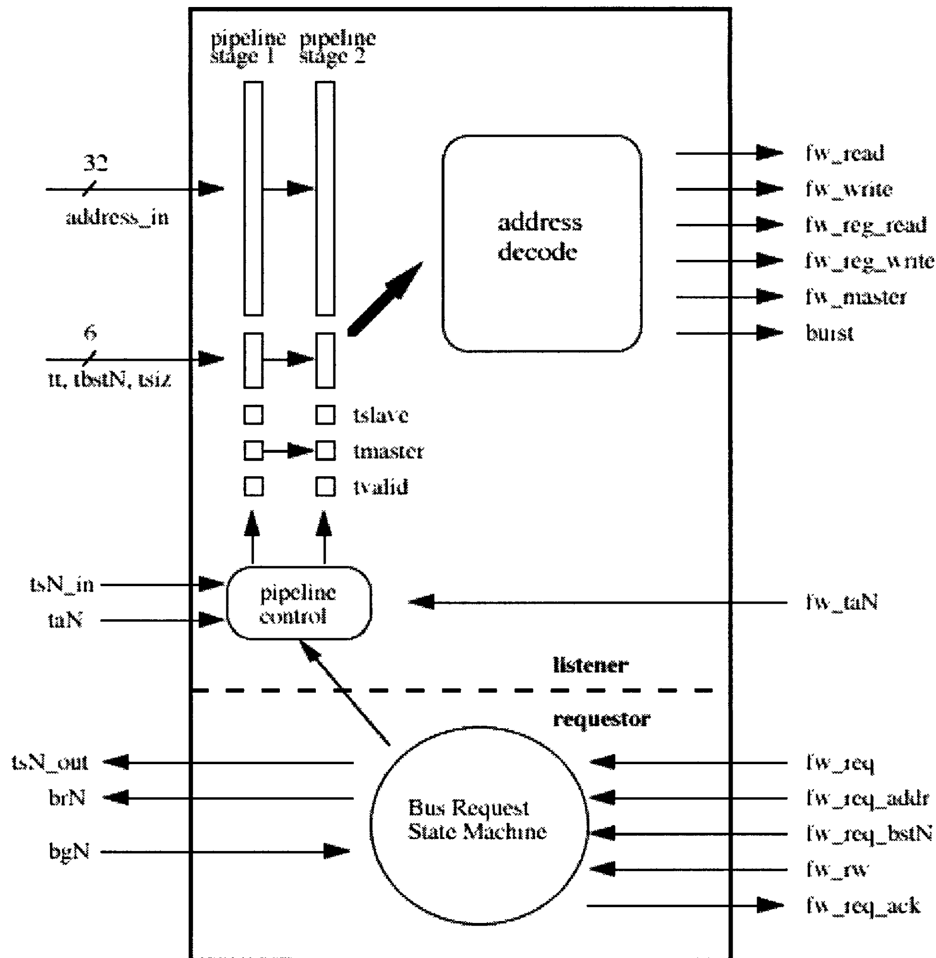


Figure 22 : Busctrl Module Block Diagram

writes and reads from the data bus on reads. The SAG acts as a slave in the cases where the 604

writes to the WRITE-1394 FIFO or accesses registers on both the SAG and Link Layer. Hence, the bus has two potential masters, the GPP and the SAG, with the MPC106 provides the external bus arbitration.

In addition to the 1394 interface logic, other modules of the SAG also use the Busctrl module to access Chidi's main data bus. Thus, the Busctrl Module designed by the author provides a general PowerPC bus interface, which is shared by both the 1394 section of the SAG and the non-1394 section of the SAG. The Busctrl logic is divided up into two main parts: a listener and a requestor. Figure 22 shows a block diagram of the Busctrl logic structure.

On the PPC Bus side, the Busctrl module connects to a set of bus arbitration, transfer, and transfer attribute signals. On the SAG side, it outputs a set of transaction description signals and takes in a set of bus request signals. Table 7 provides a list of signals and their descriptions.

Table 7 : Busctrl Module Signal Descriptions

Signal Name	Direction	Description
/TS_in	Input	Transfer Start In – Indicates that the GPP has initiated a bus transaction
ADDR_in <31..0>	Input	Address Bus In – Represents the physical address of the data to be transferred
/TBST_in	Input	Transfer Burst In – Indicates whether a transaction is a burst transfer
TT_in <4..0>	Input	Transfer Type In – Indicates the type of transfer in progress. "01010" for read; "01000" for write.
TSIZ_in <2..0>	Input	Transfer Size In – Indicates data transfer size of current transaction
/TA_in	Input	Transfer Acknowledge In – Indicates that a single beat transfer completed or that a data beat in a burst transfer completed
/BR	Output	Bus Request – Indicates the SAG is requesting mastership of the address bus
/BG	Input	Bus Grant – Indicates that the SAG has been granted mastership of the address bus
/AACK	Input	Address Acknowledge – Indicates that the address phase of a transaction is complete
/TS_out	Output	Transfer Start Out – Indicates that the SAG has initiated a bus transaction
ADDR_out <31..0>	Output	Address Bus Out – Represents the physical address of the data to be transferred by the SAG
/TBST_out	Output	Transfer Burst Out – Indicates whether a SAG-mastered transaction is a burst transfer.
TT_out <4..0>	Output	Transfer Type Out – Indicates the type of transfer in progress mastered by the SAG
TSIZ_out <2..0>	Output	Transfer Size Out – Indicates the data transfer size of the current transaction mastered by the SAG
/TA_out	Output	Transfer Acknowledge Out – Indicates that the SAG has succesfully completed a transfer for which it is a slave
fw_read	Output	1394 Read – Indicates the current data tenure is a GPP read from 1394-READ FIFO
Fw_write	Output	1394 Write – Indicates the current data tenure is a GPP write to 1394-WRITE FIFO
Fw_reg_read	Output	1394 Register Read – Indicates the current data tenure is a GPP read from a 1394 register
Fw_reg_write	Output	1394 Register Write – Indicates the current data tenure is a GPP write to a 1394 register
Burst	Output	Burst – Indicates that the current data tenure is a burst transfer
Fw_master_read	Output	1394 Master Read – Indicates that the current data tenure is a SAG mastered read transaction from 1394-READ FIFO to memory

Fw_master_write	Output	1394 Master Write – Indicates that the current data tenure is a SAG mastered write transaction from memory to 1394-WRITE FIFO
Regaddr <19..0>	Output	Register Address – Address bits used to select specific registers????
/Fw_ta_in	Input	1394 Transfer Acknowledge – Indicates that the 1394 Interface has successfully completed a data transfer
Fw_req	Input	1394 Bus Mastership Request – Indicates the 1394 Interface requests mastership of the PPC bus
Fw_req_addr <31..0>	Input	1394 Request Address – Indicates the address for the requested transfer
/Fw_req_bst	Input	1394 Request Burst – Indicates whether the requested transaction is a burst transfer
Fw_rw	Input	1394 Request Read/Write – Indicates whethere the request transfer is a read or a write.
Fw_req_ack	Output	1394 Request Acknowledge – Indicates that the requested transfer has been placed on the bus

Listener

In the listener, two sets of pipelined registers store the address and transaction control signals from the two most recent address bus tenures. The one level of pipelining is necessary since the address and data busses are split. The basic principle of operation follows: 1)the first pipeline stage is loaded when a new address tenure is begun; and 2) the second pipeline stage is loaded with the values in the first pipeline stage when a data tenure ends. Thus, the values stored in the second pipeline stage are valid for the current data tenure, while the values in the first pipeline stage will be valid on the next data tenure. The contents of the registers in the second pipeline stage are decoded into signals which describe the destination and transaction type of the current transaction.

The listener pipeline is made up of two stages of pipelined registers, containing information describing the current data transfers, and a block of logic that controls the flow of through the pipeline. The first stage of the pipeline holds the information for the transaction that has the current address tenure, and hence, the next data tenure. The second stage of the pipeline describes the transaction that has the current data tenure. Since our system is limited to one level of pipelining, these two stages are sufficient. Figure 23 shows a detailed diagram of the pipeline registers.

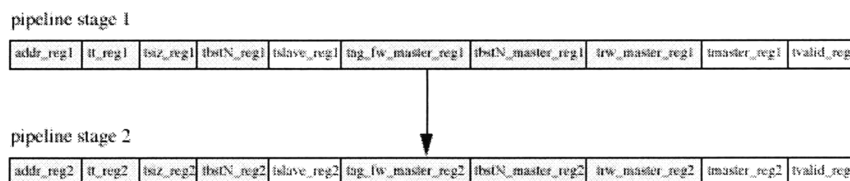


Figure 23 : Busctrl Pipeline Registers

Each pipeline stage holds the following signals addr_reg, tt_reg, tsiz_reg, tbstN_reg, tag_fw_reg, tbstN_master_reg, trw_master_reg, tslave_reg, tmaster_reg, and tvalid_reg. When set to 1, tvalid_reg indicates that the register contents are valid. So, if tvalid_reg1 is set and tvalid_reg2 is cleared, then the registers in pipeline stage 1 are valid and the registers in pipeline stage 2 are invalid. The valid bits are necessary to keep invalid data, which would be present during idle periods on the PPC bus, from being decoded into unwanted commands. When set to 1, tslave indicates that the register contents represent a transaction for which the SAG is a slave. The tmaster bit indicates if the transaction is one in which the SAG masters the bus.

The addr, tt, tsiz, and tbstN bits hold transaction signals loaded off the PPC bus. They hold the same meanings for the transactions in the pipeline stages that the bus signal meanings hold for transactions on the bus. For example, if tt_reg2 holds "01010", then the current data tenure is a read. (see previous Signals section). The addr, tt, tsiz, and tbstN registers are only valid when the tslave bit in the same pipeline stage is asserted.

The tag_fw, tbstN_master, trw_master bits describe the transactions when the SAG is the master. Tag_fw distinguishes between a transaction involving the 1394 interface and one involving a different SAG component. TbstN_master indicates if the transaction is burst-y, while trw_master is set to 1 on a read and 0 on a write.

The pipeline control logic controls the flow through the pipeline stages by monitoring /TA and /TS signals. When /TS is asserted from an external source, addr_reg1, tt_reg1, tsiz_reg1, and tbstN_reg1 are loaded with the addr, tt, tsiz, and tbsN inputs. If the addr inputs address a SAG location, then tslave_reg1 is set. When the SAG initiates a transcation, tmaster_reg1 is set and tag_fw_reg1, tbstN_reg1, and trw_master_reg1 are loaded with the values for the transaction. In both cases, tvalid_reg1 is set.

When the current data tenure transaction, described by the second stage, is a single beat read or write, the assertion of /TA causes the values in stage two to be loaded with the register contents in stage one and the registers in stage one to be cleared. When the transaction is a burst, four /TA's are counted before the shift occurs. In addition, when the stage two registers are invalid (tvalid_reg2 = 0), the values in the stage one registers are moved up to stage two.

The address decode logic decodes the transaction information in the pipeline stage two registers into the output signals fw_write, fw_read, bypass, burst, fw_master_read, and fw_master_write.

Requestor

The requestor side of the Busctrl module provides the mechanism that allows the SAG to make requests for ownership of the bus. The Bus Request State Machine takes request signals from SAG modules and asserts the bus request signal to the MPC106 Bus Arbitrator and an acknowledge signal to the requesting module. When it receives a bus grant, a placeholder is inserted into the listener's first pipeline stage, which marks the data tenure for the requested transaction. The placeholder does not contain values based on the actual transaction, it is up to the requesting module to keep track of the transaction that it requested.

The Bus Request State Machine has three states: br_idle, start_addr_tenure, and addr_tenure. Figure 24 shows the state diagram for the Bus Request State Machine. When the state machine is in the br_idle state and fw_req and /BG input are asserted, the state machine moves to the start_addr_tenure state. In start_addr_tenure, the requestor sends a signal to the Pipeline Control Logic telling it that a transaction mastered by the SAG is starting. It also drives the bus transaction signals addr, tt, tsiz, and tbstN and asserts the /TS signal for one cycle. On the next clock cycle, the state machine transitions to the addr_tenure state. In the addr_tenure state, the requestor continues to drive the bus signals until /AACK is returned by the MPC106. When /AACK is received, the state machine returns to br_idle.

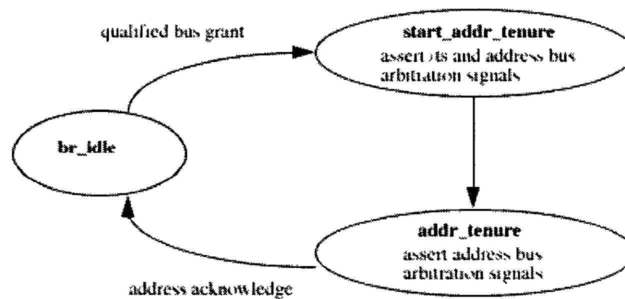


Figure 24 : Busctrl State Machine Diagram

3.3 Software

The 1394 Interface software running on the GPP is divided into two main layers, as show in Figure 25. The bottom layer communicates directly with the hardware via the PPC Bus. The higher layer takes care of serial bus management and also provides an interface to the applications programmer.

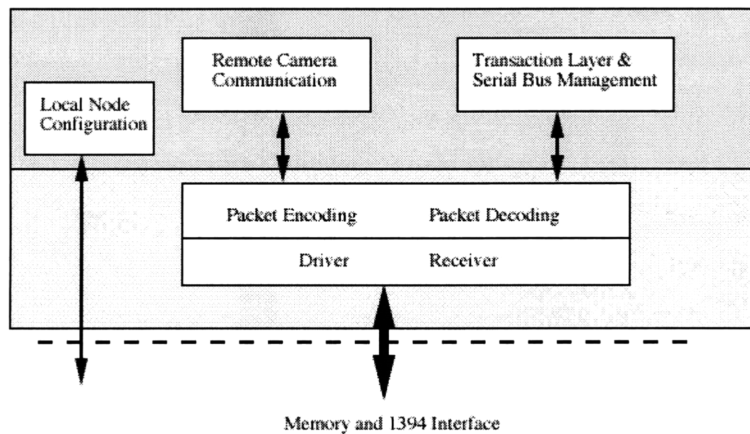


Figure 25 : 1394 Software Hierarchy

3.3.1 Low-level Software Layer

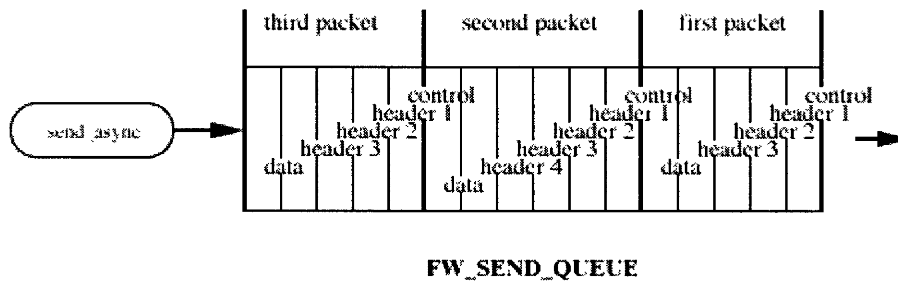
The bottom layer is made up of two main pieces of code, the Driver and the Receiver. The Driver handles the writing of outgoing packets to the 1394 interface, and also setting up the PLI to transmit a buffer to memory. The receiver scans the 1394 Receive Buffer in memory as necessary and passes information about received packets up to the higher level. In addition, packet encoding libraries take packet information (transaction type, destination, etc.) to create correctly formatted link layer inputs and packet decoding functions convert received link layer outputs into readable results.

Driver

There are three actions that the GPP can undertake in regards to the 1394 Interface: 1) transmit a 1394 packet, 2) request that a buffer of data residing in memory be transmitted over the 1394 bus, and 3) access a register residing on the SAG or the Link Layer. The Driver provides the mechanism for both transmit types, using queues to receive packet information from higher level applications. Packet encoding functions are used to place to-be-transmitted packets and quadlet packet descriptors in the FW_SEND_QUEUE, while the parameters for memory transfers are

lined up in the FW_SEND_QUEUE_MEM queue. Register accesses are typically made directly by higher level programs, with queues being unnecessary.

The send_async function takes data and transmit information and places a correctly formatted link layer input along with packet information into FW_SEND_QUEUE. The ready_next_packet interrupt signalled by the PLI on the SAG causes the first packet in FW_SEND_QUEUE to be written to the 1394 FIFOs. First the control quadlet is written to the control register on the SAG. Then the packet itself is written into the FIFOs. Figure 26 shows an example of the contents of FW_SEND_QUEUE. In the example, the first and third packets are four quadlet (three quadlets header and one quadlet data) packets, while the second packet has five quadlets. In all three cases, the actual packet quadlets are preceded by a control quadlet to be written to the PLI control register. The Driver uses the value in the control quadlet to determine the length of the packet.



FW_SEND_QUEUE

Figure 26 : FW_SEND_QUEUE example

The send_async_from_mem function takes the data location and transmit information and places a correctly formatted header along with packet information into FW_SEND_QUEUE_MEM. Any size buffer can be requested to be transmitted. The Data Transmitter breaks the buffer up into as many packets as are necessary, based on the GPP specified packet length in the PLI Memory Packet Length Register. At the beginning of each packet, the Transmitter will use the write_header interrupt to signal the GPP to write the transmit header to the 1394 link. The Data Transmitter signals that it is finished transmitting the entire buffer by signalling the input_counter_done interrupt.

Figure 27 shows example contents of the FW_SEND_QUEUE_MEM buffer. As with the FW_SEND_QUEUE FIFO, the control quadlet and header are write into the queue. However, rather than data, the memory start and stop addresses of the buffer to be transmitted, and the packet length the buffer should be broken up into are provided.

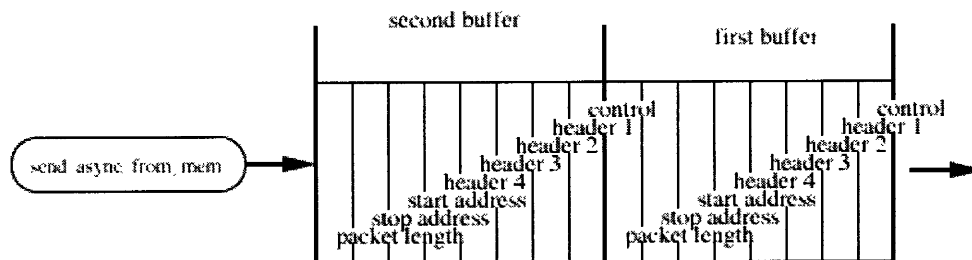


Figure 27 : FW_SEND_QUEUE_MEM example

Receiver

The Receiver scans the 1394 Receive Buffer in memory, where received 1394 packets are written to by the Processor-to-Link Interface on the SAG. The location of the 1394 Receive Buffer is set by the Output Start Register and Output Stop Register located on the PLI module on the SAG.

When the Output Counter address pointer reaches the value in the Output Stop Half Register (i.e. buffer is half full), the receive_buffer_half_full interrupt is triggered to the GPP. The Receiver then starts scanning through the first half of memory. By reading packet headers, the Receiver determines whether received packets should remain in memory or be read into the processor. Initially, the GPP only knows where the first packet is located. However, decoding its header reveals the packet length and hence the start of the next packet. Decoding the next packet's header leads to the third packet's location and so on.

While the GPP is handling the interrupt, the received packets can continue to be written into the second half of the buffer. When the Output Stop Register value is reached, the GPP will get the receive_buffer_full interrupt and will service it by scanning through the last half of the buffer. At this point, the PLI will begin writing over the first half again. Important data in the first half will have been read or moved to another part of memory by this time.

Configuration and status packets are read into the processor and placed in the FW_RECV_QUEUE. The locations of remaining packets in the memory buffer are stored in a data_locations structure. Depending on the application and the actual packet contents themselves,

these packets could be moved to another memory buffer or passed on to a different part of the system. After the processor has completed scanning the buffer, the contents of FW_RECV_QUEUE are decoded and passed up to the transaction layer.

Packet Encoding and Decoding

The Packet Encoding and Decoding Library contains functions for header encoding and decoding and packet formatting. It abstracts the 1394 packet formats from the higher-level programmer. Send_async takes quadlet data and transfer information and produces a correctly formatted packet. Send_async_from_mem takes the location of a memory buffer and transfer information and sets up a multiple packet transfer to be handled by the Data Transmitter. The various decode functions take header quadlets and return the values in certain header fields.

3.3.2 Transaction/Management Layer

The Transaction/Management Layer includes libraries which handle Serial Bus Management, Local Node Configuration, and Remote Camera Management.

Serial Bus Management

Each Chidi node has the ability to perform certain serial bus management responsibilities. The actions include various bus management functions, isochronous resource management, and node control.

Each node is isochronous resource manager capable. Serial Bus Management registers are implemented in software and accessible through the transaction layer. The nodes are not bus manager capable, because for the purposes of this project, the Bus Manager Capable characteristics, such as the SPEED MAP and TOPOLOGY MAP registers are not worth implementing. All nodes will are the same speed and the networks topology will likely be very simple. Of course, that functionality can certainly be added in the future.

In a network with multiple Chidi nodes, each Chidi will be isochronous resource manager capable. After a bus reset, each node reads its received self-ID packets and determines the physical ID of the isochronous resource manager. Nodes that do not recognize themselves as the isochronous resource manager will limit themselves to being only Isochronous Capable. Only the node which is selected isochronous resource manager will enable its cycle master and isochronous resource manager capabilities.

Received and decoded configuration packets (see last sub-section) are dealt with in Serial Bus Management. If permissible, write transactions modify the addressed registers and responses are returned to the source nodes. Read requests are answered with read response back to the source nodes.

Local Node Configuration

The Local Node Configuration library provides functions for configuring the Processor-to-Link Interface, Link Layer and Physical Layer and for retrieving status information from said modules.

Remote Camera Management

The Remote Camera Management library includes functions for remote camera control. These functions have asynchronous read request or write request packets sent to remote camera nodes by calling the send_async function in the Packet Decoding library.

According to the 1394-based Digital Camera Specification, Rev. 1.04, all cameras have a basic set of command and status registers accessible through asynchronous transactions. [2] By reading from the register locations, software can learn the state of a camera and through writing the registers, the camera can be controlled.

4. Conclusion

At the time of this writing, the design of all the Processor-to-Link sub-modules have been debugged. The Busctrl, Link Interface, and Data Transmitter have all been tested and are functional, while the Hardware Registers sub-module has been tested on a smaller scale (i.e. implemented with 4-bit input and output counters rather than 12-bit counters). An integrated design without the interrupt registers discussed earlier has also been tested. In the next two weeks, the author expects to have the entire full-scale Processor-to-Link logic integrated and tested.

4.1 Future Work

A working 1394 hardware interface for Chidi has been designed that will allow the Chidi software designer to integrate use of the 1394 network into designs without knowing the intricacies of the 1394 protocol. However, the work that has actually been completed only forms the backbone of the entire interface that is necessary to do this. The areas that need to be addressed in the future are software testing, physical layer debugging, integration with Chidi Address Generator, and analysis of 1394 network capacity and efficiency.

The main software functions have been written and tested minimally in a UNIX environment. On-board software testing will have to be completed by continuing researchers. However, the author is confident that debugging of the software design will generally be just a matter of being able to use the hardware correctly.

As of yet, an actual 1394 Bus network has not been made to function.When connected via standard 1394 cables, the physical layers of Chidi nodes and the DV Camera used for testing do not appear to recognize one another. Communication over the 1394 cable media is made using low voltage differential signaling (LVDS), which makes debugging more difficult than typical on-board testing. An attempt to address this problem will be made in the remaining term of research.

Once a functioning SAG Address Generator sub-module design has been completed, steps must be taken to implement the 1394 interface using the Address Generator counters and bus mastership logic rather than the analogous pieces described earlier. These changes make more efficient use of SAG logic resources, since the PLI module won't need to implement its own counters. Minimal firmware design changes are necessary to complete this migration.

After these various aspects have been addressed, an analysis of 1394 network capacity and efficiency and also Interface effectiveness can provide valuable information, such as the optimal packet transfer size and receive buffer size. In addition, determining how use of the 1394 network affects the efficiency of Chidi's reconfigurable processor can show how to best make use of the Interface. For example, it could be revealed that the current implementation interrupts the General Purpose Processor too frequently in order to deal with other computations at the same time. These findings would be used to improve upon this design in future versions of the Chidi media processor.

Appendix A – TSB12C01A Link Layer Controller Addressing

During read and write accesses to the TSB12C01A, the host-bus interface address bus is used to address the TSB12C01A's internal configuration registers and transmit/receive FIFOs.

Internal Registers

Table 8 lists the internal registers and their addresses.

Table 8 : TSB12C01A Internal Registers

Address	Register Name	Description
0x00	Version Register	Allows software to be written that supports multiple versions of link layer controllers
0x04	Node Address	Control which packets are accepted/rejected
0x08	Control Register	Dictates basic operation of the link layer controller
0x0C	Interrupt Register	Works in tandem with Interrupt Mask Register to inform host of state changes
0x10	Interrupt Mask Register	See above
0x14	Cycle-Timer Register	Contains counters for cycle timing
0x18	Isoch Port Number Register	Controls which isochronous channels are received by this node
0x20	Diagnostics Register	Allows for monitoring and control of diagnostic features
0x24	Phy-Chip Access Register	Allows access to physical layer chip registers
0x30	ATF Status Register	Allows for monitoring and control of the ATF
0x34	ITF Status Register	Allows for monitoring and control of the ITF
0x3C	GRF Status Register	Allows for monitoring and control of the GRF

Internal FIFOs

The TSB12C01A assigns different addresses for different FIFO accesses. Table 9 shows how the FIFOs are mapped to addresses.

Table 9 : TSB12C01A Internal FIFOs

Address	Name	Description
0x80	ATF_first	Location to write first quadlet of an asynchronous packet
0x84	ATF_continue	Location to write second to next-to-last quadlets of an asynchronous packet
0x8C	ATF_update	Location to write last quadlet of an asynchronous packet
0x90	ITF_first	Location to write first quadlet of an isochronous packet
0x94	ITF_continue	Location to write second to next-to-last-quadlets of an isochronous packet
0x9C	ITF_update	Location to write last quadlet of an isochronous packet
0xC0	GRF_data	Location to read received remote data

Appendix B – 1394 Bus Initialization

The initialization routine for the 1394 Bus Interface is as follows:

a) Write 1394 Receive Buffer parameters to Output Start, Output Stop Half, and Output Stop Full registers.

b) Enable the link layer receiver and set the receive self-ID packets option on. These are fields in the link layer control register.

c) If Chidi node is to be root/isochronous resource manager, set the root hold-off bit in physical layer registers. This instructs the node to try to become root node on the next bus reset.

d) Initiate a bus reset by setting the initiate bus reset bit in the physical layer registers. Steps a and b can be combined into the same register write.

e) Set the Revision A enable bit in the link layer control register. This enables the GRFEMP output signal from the link layer controller.

f) Enable reading of the GRF by the Link Interface sub-module by setting the appropriate bit in the PLI control register.

g) Self-ID packets will be transferred to the 1394 Receive Buffer in memory. Read self-ID packets to determine network devices.

h) If root node, enable cycle master capabilities on link layer controller. If not the root node, use IDs to determine root node.

Appendix C – Address Generator Implementation

An alternate implementation for the 1394 Bus Interface takes advantage of the address generation capabilities of the Address Generator module on the SAG. The Address Generator takes a memory access pattern written to it by the general purpose processor and provides DMA transfers of data using the specified pattern. When the transfer is completed, the Address Generator signals an interrupt to the processor.

In the Address Generator implementation of the 1394 Bus Interface, the input and output counters in the Hardware Registers sub-module are replaced with two dedicated channels in the address generator.

Specific documentation of the Address Generator can be found at http://chidi.www.media.mit.edu/projects/chidi/index.html.

Appendix D – Altera FPGA Optimization

A major constraint in designing the Processor-to-Link Interface was that the logic had to perform at a clock speed of 66.66 MHz. The registered performance optimization techniques used can be divided into two categories: place and route assignments, and logic replication.

Place and route assignments used with the fitting software (Altera MaxPlus2) included clique setting and explicit logic cell assignment. A clique is a group of logic cells and registers which the compiler attempts to place close together. By selecting cliques intelligently, register-to-register paths can be minimized. Explicit logic cell assignment goes a step further than cliques, and removes the compilers freedom to place logic itself.

In many cases, duplication of certain registers can significantly speed up performance. The first case where register duplication is applicable is when a register is driving two cells located in different areas such that by making the delay from the register to one cell manageable, the delay to the second cell becomes to large. In this situation, creating a functionally equivalent register that drives the second cell is a possible solution.

The second case is based on the fact that signal delays increase as register fan-out increases. The less logic cells that a register signal is driving, the faster that signal can reach the logic cells. As in the first case, additional registers that are identical in function, but are named differently will decrease signal delays.

In addition, registers can be inserted into long paths in order to decrease the actual register-to-register delay. However, this can have a significant impact on design functionality, since adding a register adds a clock cycle delay.

Appendix E – Communication with 1394-Compliant Cameras

A camera that is compliant with the 1394-based Digital Camera Specification, Rev. 1.04, is a passive device that initiates no actions of its own. In order to have the camera perform any action, a node(s) acting as the camera controller must access the camera's control registers. Table 10 below provides a short description of the standard camera registers. All registers are offset from a base 1394 Serial Bus address specified in the camera's configuration ROM address.

Table 10 : Camera Registers (taken from 1394-based Digital Camera Specification [2])

Offset	Register Name	Description
0x000	Camera initialize register	Sets camera to initial (factory-setting) state
0x100	Inquiry register for video format	Specifies the availability of video formats (e.g. VGA non compressed format)
0x180	Inquiry register for video mode	Specifies the availability of video modes (e.g. 640 x 480 YUV [4:2:2])
0x200	Inquiry register for video frame rate	Specifies the availability of video frame rates (e.g. 15 fps)
0x400	Inquiry register for basic function	Specifies basic function availability
0x404	Inquiry register for feature presence	Specifies feature availability (e.g. brightness control)
0x500	Inquiry register for feature elements	Specifies capabilities of available features
0x600	Status and control register for camera	Provides control of camera functions
0x800	Status and control register for features	Provides control of available features

The table below shows possible video transfer modes. Transfer rates of 2 lines per packet require a 200 Mb/s data rate. 4 lines per packet requires 400 Mb/s.

Table 11 : Camera Video Modes (taken from 1394-based Digital Camera Specification [2])

Mode	Video Format	60fps	30fps	15fps	7.5fps	3.75fps
Mode_0	160x120 YUV(4:4:4) 24bit/pixel		1/2H 80p 60q	1/4H 40p 30q	1/8H 20p 15q	
Mode_1	320X240 YUV(4:2:2) 16bit/pixel		1H 320p 160q	1/2H 160p 80q	1/4H 80p 40q	1/8H 40p 20q
Mode_2	640x480 YUV(4:1:1) 12bit/pixel		2H 1280p 480q	1H 640p 240q	1/2H 320p 120q	1/4H 160p 60q
Mode_3	640x480 YUV(4:2:2) 16bit/pixel		2H 1280p 640q	1H 640p 320q	1/2H 320p 160q	1/4H 160p 80q
Mode_4	640x480 RGB 24bit/pixel		2H 1280p 960q	1H 640p 480q	1/2H 320p 240q	1/4H 160p 120q
Mode_5	640x480 Y (Mono) 8bit/pixel	4H 2560p 640q	2H 1280p 320q	1H 640p 160q	1/2H 320p 80q	1/4H 160p 40q

H: Line/Packet p: pixel/packet q: quadlet/packet

References

[1] IEEE Computer Society, "IEEE Standard for a High Performance Serial Bus", IEEE Std 1394-1995, Institute of Electrical Engineering and Electronic Engineers, New York, 1996.

[2] Camera Working Group of the 1394 Trade Association, "1394-based Digital Camera Specification", Version 1.04, 1394 Trade Association, Austin TX, 1996.

[3] Xiaolin Lu, Peng-Gang Zhang, and Walter Chen, "An Internet gateway for long-distance 1394 home network", DSPS Research and Development Center, Texas Instruments Incorporated, 1997.

[4] "Chidi: The Flexible Media Processor," MIT Media Lab, Information and Entertainment Group, http://chidi.www.media.mit.edu/projects/chidi/index.html, 1997.

[5] John A. Watlington and V.Michael Bove, Jr. "A System for Parallel Media Processing", MIT Media Lab, April 1, 1997

[6] Peter Johansson "Software Design for IEEE 1394 Peripherals", Presentation outline, http://www.1394ta.org/ ,1997.

[7] "FLEX 10K: Embedded Programmable Logic Family," *Altera Data Book 1996*, Altera Corporation, Version 2, June 1996.

[8] "MPC106 PCI Bridge/Memory Controller Technical Summary," Motorola, Rev. 1, August 1996.

[9] "PowerPC Microprocessor Family: The Bus Interface for 32-bit Microprocessors," IBM and Motorola, Rev. 0, March 1997.