

Philips Semiconductors
Enhanced Full Duplex A/V Link Layer
IEEE 1394 Reference Design Kit
version 2.2.1
User's Manual

Copyright © 2000 Philips Semiconductors. All rights reserved.

Philips Semiconductors Enhanced Full Duplex A/V Link Layer IEEE 1394 Reference Design Kit version 2.2.1

User's Manual

November 2000

Copyright © 2000, Philips Semiconductors. All rights reserved.

The contents of this document may not be copied nor duplicated in any form, in whole or in part, without prior written consent from Philips.

Philips provides the information and data included in this document for your benefit, but it is not possible for us to entirely verify and test all of this information in all circumstances, particularly information relating to non-Philips manufactured products. Philips makes no warranties or representations relating to the quality, content, or adequacy of this information. Every effort has been made to ensure the accuracy of this manual; however, Philips assumes no responsibility for any errors or omissions in this document. Philips shall not be liable for any errors or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual or the examples herein. Philips assumes no responsibility for any: damage or loss resulting from the use of this manual; loss or claims by third parties which may arise through the use of this RDK; loss or claims by third parties which may arise through the use of this RDK; and for any damage or loss caused by deletion of data as a result of malfunction or repair. The information in this document is subject to change without notice. Product and Company names are trademarks or registered trademarks of their respective owners.

Table of Contents

1	INTRODUCTION	1
1.1	Philips Semiconductors Full Duplex RDK.....	1
1.2	About this Manual	1
1.3	Writing Conventions	2
1.4	Additional References.....	2
2	INSTALLATION	3
2.1	Unpacking.....	3
2.2	Minimum Requirements	3
2.3	Recommended Additional Equipment.....	4
2.4	Software Installation.....	4
2.5	Hardware Installation.....	4
2.5.1	Hardware Settings	5
2.6	Verifying the Installation.....	5
2.6.1	Step One: Verify Embedded Code.....	5
2.6.2	Step Two: Verify Monitor Program.....	6
3	MONITOR BASICS	7
3.1	Introduction.....	7
3.1.1	Asynchronous	7
3.1.2	Isochronous	7
3.2	Starting the Monitor.....	8
3.3	Program Navigation.....	9
3.3.1	Entering Data.....	9
3.4	Communication Settings	9

3.5	Event Viewer.....	9
3.6	Loading HEX Files	10
3.7	Quitting.....	11
4	MONITOR SECTIONS	13
4.1	Isochronous Services	13
4.1.1	Transmit Configuration.....	13
4.1.2	Receive Configuration.....	14
4.2	Asynchronous Services	17
4.2.1	Quadlet/Block Read Request	20
4.2.2	Quadlet/Block Write Request.....	21
4.2.3	Quadlet Lock Request	22
4.3	PHY Registers	23
4.4	Bus Management Menu	24
4.4.1	Low-Level Bus Functions	24
4.4.1.1	Bus Reset	24
4.4.1.2	Display Self-IDs	24
4.4.1.3	Show Bus Topology	24
4.4.1.4	Send PHY Config Packet.....	24
4.4.1.5	Send Link-On Packet.....	25
4.4.2	Serial Bus Management	25
4.4.2.1	CSR Quick Summary	25
4.4.2.2	Read CSR Register	28
4.4.2.3	Write to CSR Register.....	28
4.4.2.4	Lock CSR Register	28
4.4.2.5	Configuration ROM.....	28
4.5	Link Registers	28
4.6	Camcorder Operations	29
4.7	L40 Menu.....	31
4.7.1	Indirect Address Quadlet.....	31
4.7.2	FIFO Size Registers Block.....	31
4.8	Extended PHY Packets	32
4.8.1	Ping Packet.....	32
4.8.2	Remote Access and Remote Reply Packets	33
4.8.3	Remote Command and Remote Confirmation Packets.....	34
4.8.4	Resume Packet	35

5	EMBEDDED SOFTWARE	36
5.1	Introduction.....	36
5.2	BSP Architecture Notes	37
5.2.1	The Current Platform.....	37
5.2.2	Threads.....	38
5.2.3	Semaphores	38
5.2.4	Events.....	38
5.2.5	Memory Management	39
5.2.6	Interrupts	39
5.2.7	Errors.....	39
5.2.8	Watermarks	39
5.2.9	Porting to a new Platform.....	40
5.2.10	Porting to an Operating System.....	40
5.2.11	Porting to Another 1394 Link	40
5.3	API Overview.....	40
5.3.1	Asynch Module	42
5.3.1.1	Overview.....	42
5.3.1.2	AsynchRequestParams structure	42
5.3.1.3	Implementation Details.....	43
5.3.1.4	Porting to a Preemptive Multitasking System.....	45
5.3.1.5	AsynchTxRequest.....	46
5.4	Application Overview.....	47
5.4.1	Main	47
5.4.2	Monitor Support	47
5.4.3	1394 Reference Implementation.....	47
5.4.4	Example Code	47
6	SERIAL INTERFACE DOCUMENTATION	48
6.1	Overview.....	48
6.2	Commands and Responses	48
6.2.1	Register access	48
6.2.2	Asynchronous communications	49
6.2.3	Isochronous communications	49
6.2.4	Misc. 1394 functions.....	50
6.2.5	Debug & Verification commands	51
6.3	Events.....	51
6.4	Downloading.....	52
7	SOFTWARE COMPLIANCE.....	53

7.1	Overview.....	53
7.2	Cycle Master.....	53
7.3	Isochronous Resource Manager (IRM).....	53
7.4	Bus Manager.....	53
7.5	ISO/IEC 61883 and AV/C.....	54
8	CUSTOMER SUPPORT	55
9	TROUBLESHOOTING.....	56
9.1	General Problems	56
9.2	Serial communications	56
9.3	Link and PHY Register Windows	56
9.4	Asynchronous Transactions	56
9.5	Bus Resets.....	57
9.6	Loading HEX Files	57
10	ERROR CODES	58
10.1	Overview.....	58
10.1.1	PENDING (blink code 1,2).....	58
10.1.2	E_ERROR (blink code 1,3).....	59
10.1.3	E_REQ_MALLOC_TIMEOUT (blink code 1,4).....	59
10.1.4	E_ASYTXREQ (blink code 1,5).....	59
10.1.5	E_ASYTXRESP (blink code 1,6).....	59
10.1.6	E_NO_QUADLETS_AVAILABLE (blink code 1,7).....	59
10.1.7	E_ROUTER_TABLE_FULL (blink code 1,8).....	59
10.1.8	E_LOOP_DETECTED (blink code 2,1).....	60
10.1.9	E_BUS_RESET (blink code 2,2).....	60
10.1.10	E_RESP_MANGLED (blink code 2,3).....	60
10.1.11	E_CONF_MANGLED (blink code 2,4).....	61
10.1.12	E_CONF_NO_RESPONSE (blink code 2,5).....	61
10.1.13	E_CONF_ERROR (blink code 2,6).....	61
10.1.14	E_CONF_TIMEOUT (blink code 2,7).....	61
10.1.15	E_REQ_MANGLED (blink code 2,8).....	61
10.1.16	E_PACKET_MANGLED (blink code 3,1).....	61
10.1.17	E_UNSUPPORTED_TCODE (blink code 3,2).....	62
10.1.18	E_TIMEOUT (blink code 3,3).....	62

10.1.19	E_BR_NO_IDVALID (blink code 3,4)	62
10.1.20	E_BR_NO_HEADER (blink code 3,5)	62
10.1.21	E_BR_MULTIPLE_HEADERS (blink code 3,6)	62
10.1.22	E_BR_SIDQAV_NOT_SEEN (blink code 3,7)	62
10.1.23	E_BR_PACKET_TIMEOUT (blink code 3,8)	63
10.1.24	E_BR_ACK_DATA_ERROR (blink code 4,1)	63
10.1.25	E_BR_INVALID_PACKET (blink code 4,2)	63
10.1.26	E_INVALID_TOPO_MAP (blink code 4,3)	63
10.1.27	E_INVALID_NODE_TREE (blink code 4,4)	63
10.1.28	E_CABLE_LOOP (blink code 4,5)	63
10.1.29	E_BR_CYCLE_START (blink code 4,6)	64
10.1.30	E_TOO_MANY_BUS_RESETS (blink code 4,7)	64
10.1.31	E_INTERNAL (blink code 4,8)	64
10.1.32	E_EVENT_Q_FULL (blink code 5,1)	64
10.1.33	E_INVALID_EVENT_ID (blink code 5,2)	64
10.1.34	E_PHY_TIMEOUT (blink code 5,3)	64
10.1.35	E_LINKPHY_FATAL_INTERRUPT (blink code 5,4)	64
10.1.36	E_ASY_FATAL_INTERRUPT (blink code 5,5)	64
10.1.37	IRX_FATAL_INTERRUPT (blink code 5,6)	65
10.1.38	ITX_FATAL_INTERRUPT (blink code 5,7)	65
10.1.39	E_RAM_CHECK (blink code 5,8)	65
10.1.40	E_ROM_CHECK (blink code 6,1)	65
10.1.41	E_TEST_FAIL (blink code 6,2)	65
10.1.42	E_PKTQ_FULL (blink code 6,3)	65
10.1.43	E_PKTQ_EMPTY (blink code 6,4)	65
10.1.44	E_SERIAL_TX_Q_FULL (blink code 6,5)	65
10.1.45	E_SERIAL_RX_OVERFLOW (blink code 6,6)	66

11 GLOSSARY.....67

List of Figures

Figure 2-1: Full Duplex RDK Installation Wizard.....	4
Figure 3-1: Monitor Main Screen.....	8
Figure 4-1: Isochronous Transmit Setup Dialog.....	14
Figure 4-2: Isochronous Receive Setup Dialog.....	15
Figure 4-3: Quadlet/Block Read Request Dialog.....	20
Figure 4-4: Quadlet/Block Write Request Dialog.....	21
Figure 4-5: Quadlet Lock Request Dialog.....	22
Figure 4-6: Phy Registers Dialog.....	23
Figure 4-7: Link Register Dialog.....	29
Figure 4-8: Camcorder Operations Dialog.....	30
Figure 4-9: Indirect Address Quadlet Dialog.....	31
Figure 4-10: FIFO Size Registers Dialog.....	31
Figure 4-11: Ping Packet.....	32
Figure 4-12: Remote Access Packet.....	33
Figure 4-13: Remote Command Packet.....	34
Figure 4-14: Resume Packet.....	35
Figure 5-1: Software Layer Diagram.....	36
Figure 5-2: API Overview	41
Figure 5-3: Asynchronous Outgoing Request Flow.....	44
Figure 5-4: Asynchronous Incoming Request Flow.....	45

List of Tables

Table 4-1: Transmit Mode Register Values.....	16
Table 4-2: Required Isochronous Transmit Mode Signals	16
Table 4-3: Preset Receive Mode Register Values	17
Table 4-4: Isochronous Receiver Signal Required	17
Table 4-5: Other Output Signals.....	17
Table 4-6: Asynchronous Transmission Parameters	18
Table 4-7: Unsupported Asynchronous Transmission Parameters.....	19
Table 4-8: Common Camcorder Operation Settings	29

1 Introduction

1.1 Philips Semiconductors Full Duplex RDK

The Full Duplex A/V Link Layer Reference Design Kit (RDK) is a versatile platform for rapid prototyping. It is designed to demonstrate the features of the Philips 1394 Link & PHY chipset. Using the RDK, designers can rapidly become familiar with the Philips implementation of the IEEE 1394 Serial Bus standard and product development time can be considerably reduced.

The RDK consists of three parts: the Evaluation Board, the embedded software, and the monitor program.

- The Evaluation Board is populated with all of the switches, headers, connectors, and prototype areas needed to fully understand the Philips chipset. It illustrates the basic connectivity of the PDI1394 family of 1394 devices and demonstrates their appropriate use in a hardware design.
- The embedded software configures the Evaluation Board as a fully functional IEEE 1394 serial bus node. This is extremely useful in prototyping a 1394 product. The software was designed to be extended to your product, with a well designed porting layer, an API to PDI1394 chipset, including a fully featured transaction layer, as well as a sample application. The embedded software also comes with extensive online documentation.
- The monitor program can be run on any PC with a Win32 capable operating system (Windows 9x or NT). The monitor communicates with the embedded software on the Evaluation Board by means of a RS232 serial cable. It allows on-line access to all registers of the PDI1394 chipset and demonstrates many features of the IEEE 1394 serial bus.

The kit features an embedded C51 microcontroller that is simple to use, yet powerful enough for the intended application. You can quickly test programs written during development by downloading them to the Evaluation Board. Interactive debugging is available through standard console I/O functions (printf, scanf), which communicate with a host PC through the Evaluation Board serial port. The result is a flexible development platform for 1394 applications.

1.2 About this Manual

This manual has the following structure:

- Chapter 1, Introduction, is this chapter. It explains the structure of the rest of the document and introduces the RDK.
- Chapter 2, Installation, describes how to install the RDK hardware and software.
- Chapter 3, Monitor Basics, describes the basic workings of the monitor program.

- Chapter 4, Monitor Sections, describes in detail the workings of the monitor program.
- Chapter 5, Embedded Software, describes the software running onboard the Evaluation Board.
- Chapter 6, Serial Interface Documentation, documents the serial interface of the RDK.
- Chapter 7, Software Compliance, documents what is and is not missing from this reference implementation of the IEEE 1394-1995 and 1394A Specification.
- Chapter 8, Customer Support, gives you contact information for RDK support.
- Chapter 9, Troubleshooting, helps resolve some basic problems you may encounter while using the RDK.
- Chapter 10, Error Codes, provides a list of the error codes used by the embedded software
- Chapter 11, Glossary, defines a number of terms used in this manual.

1.3 Writing Conventions

The names of commands, buttons, and menus are displayed in fixed font like the following examples: `monitor`, `Show Events`. The commands may be entered on the keyboard, on-screen buttons, or internal to the software.

1.4 Additional References

The IEEE 1394 Serial Bus standard is well documented in the following publications:

1. *PD11394L21 - 1394 AV Link Layer Controller*, available in PDF form at <http://www.semiconductors.philips.com/1394/>
2. *PD11394P11 - 3-Port Physical Layer Interface*, available in PDF format at <http://www.semiconductors.philips.com/1394/>
3. *80C31/80C51/8751 - CMOS single chip 8-bit microcontrollers*, available in PDF format at <http://www.semiconductors.philips.com/>
4. *IEEE 1394, High Performance Serial Bus*, available from any IEEE publication house, more information can be found at <http://www.1394ta.org/>
5. *AV/C Digital Interface Command Set General Specification*, available from the 1394 Trade Association web site at <http://www.1394ta.org/>
6. *IEC/ISO 61883 Digital Interface for Consumer Electronic AV Equipment Specification*, available from the ISO.

2 Installation

This chapter describes the components of the Full Duplex RDK version 2.0, its minimum system requirements, recommended additional equipment, and the procedures for installing the software and hardware and performing verification tests.

2.1 Unpacking

The RDK includes:

- One 12 volt DC power supply
- One 9 pin male to 9 pin female serial cable
- One IEEE 1394 cable assembly
- One 25 pin female to 9 pin male adapter
- Philips CD-ROM with the RDK software
- Philips s/w license agreement
- Philips Hardware Warranty
- Philips Registration Form
- Philips technical support document
- Philips Data Analyzer Brief
- Philips RDK User's Manual version 2.2.1 (this document)
- Philips RDK H/W Ref Manual version 2.2
- Philips RDK Order Form
- One Philips Full Duplex demo board REV 0

Please verify that the packaging is complete. If it is not, please contact Philips customer support immediately. Contact information is provided in Chapter 8.

2.2 Minimum Requirements

To install and run the monitor program and embedded software development tools, the following are required:

- A PC running Microsoft Windows 9x or NT 4.0
- One free RS-232 port
- A CD-ROM drive
- A minimum of 5 Megabytes of free hard-drive space
- For best results a screen resolution of 1024x768 is recommended

2.3 Recommended Additional Equipment

Development of embedded software for the RDK requires a compiler and a linker for the C51 processor. The Keil C51 integrated development environment is recommended. All RDK embedded software was written using this environment. The Keil IDE is available from Keil Software at <http://www.keil.com/> or 1 (800) 348-8051.

2.4 Software Installation

To install the RDK software:

1. Place the installation CD in the CD-ROM of the target computer.
2. Run the program setup.exe from the installation CD.
3. Use the wizard to select the desired installation options and program locations.

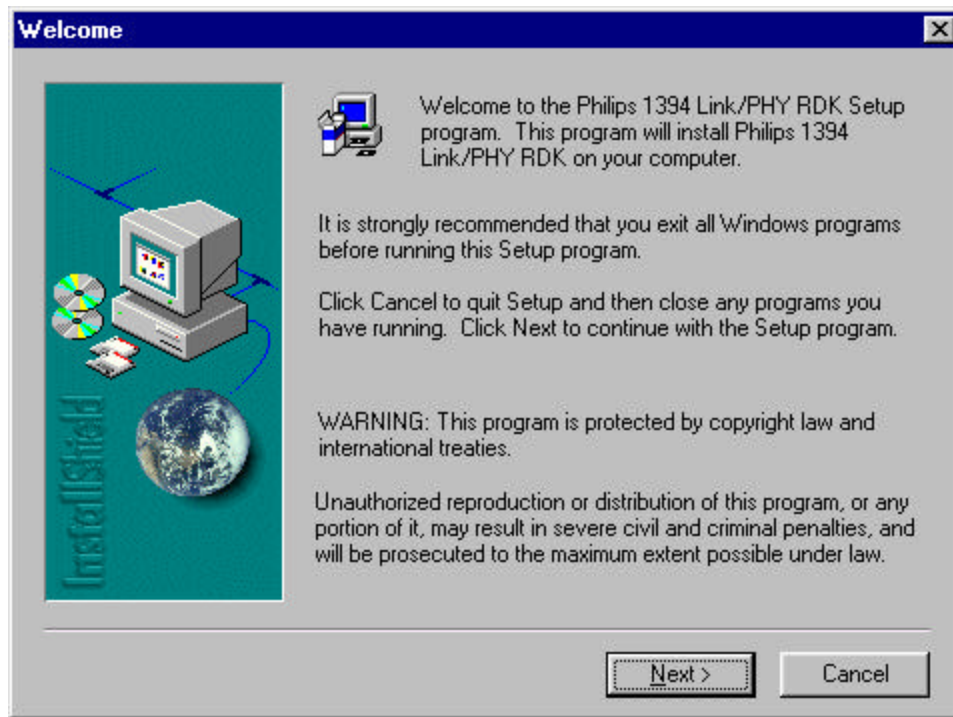


Figure 2-1: Full Duplex RDK Installation Wizard

2.5 Hardware Installation

Note: Some parts of the RDK hardware are very sensitive to static. Please ensure that you are well grounded to prevent damage due to electrostatic discharge.

To install the hardware perform the following steps:

1. Plug the power supply into the Evaluation Board. The power connector is located approximately 1cm from the left edge along the top of the board. The following LEDs should light up:

- a. 5v LED (labelled D7)
- b. 3.3v LED (labelled D6)
- c. PHY Power LED (labelled D8)

If the board has the original EPROM installed the User LED (labeled D2) should begin to flash in a heartbeat style. This means that the program is operating correctly.

2. Press the RESET button (labelled S2) located near the serial port. The User LED should light for a few seconds while the EPROM programs the 8051, and then the heartbeat should start again.
3. Connect the Evaluation Board to the serial port of the host PC using the provided cable. The PC must not share this serial port with other applications while the monitor is running.

2.5.1 Hardware Settings

The Evaluation Board is shipped with the jumpers and switches properly configured for normal use. If further information is required on specific settings please consult the *Hardware Reference Manual*.

2.6 Verifying the Installation

To perform the installation tests you will need to know which port the Evaluation Board is connected to on the PC (COM1 to COM9).

2.6.1 Step One: Verify Embedded Code

A few seconds after power is applied to the board, the embedded software should begin flashing the User LED in a heartbeat pattern at approximately 80 beats per minute to indicate that it is working properly. The LED is located next to the link access LED, by the serial port.

When the embedded processor is heavily loaded the heartbeat pattern will slow, however it should return to ~80 bpm when the load is reduced.

If another 1394 node is connected, the heartbeat may pause briefly while the bus resets, but should resume after a short time.

If the embedded software detects a fatal error, the heartbeat will change to a blink code. Blink codes are sequences of flashes that indicate a problem. For example, a repeating code of four blinks followed by five blinks (blink code 4, 5) corresponds to the error E_CABLE_LOOP, which means that the 1394 nodes are connected in a loop. This is not allowed by the IEEE 1394 specification. For more information on blink codes, see Chapter 10, Error Codes.

Testing the hardware and embedded software setup is a matter of installing the hardware and verifying the presence of a heartbeat.

2.6.2 Step Two: Verify Monitor Program

To test the monitor the hardware and firmware must first be working properly. Ensure that the above steps have been followed to verify the firmware before proceeding with the verification of the monitor.

Start the monitor by clicking on the “Monitor version 2.2.1” menu item from the start menu. By default this icon is placed in the Programs sub-menu of the Start menu under a group entitled “Philips 1394 RDK”.

You can also run the monitor:

1. by selecting Run from the Start menu, browsing to the directory to which you installed the application (by default “C:\Philips_1394_RDK\SW\bin\Monitor”) and choosing FWMonWin.exe; or
2. from the DOS prompt by typing FWMonWin, while in the above directory; and using the `-pCOM#` switch, where # represents 1-9.

If the program is able to attach to the requested port and receives the proper responses, it will display the main screen. The main screen identifies the RDK, and the monitor release number.

If the monitor fails, it will exit with an error message that should help in identifying the problem.

Once the program starts successfully the event viewer window should be open and display that a bus reset was detected. The monitor performs a bus-reset each time it starts to ensure that information such as the board node and bus topography is up to date.

To further verify the monitor is working properly, open the “Register” menu and choose “Link” to see the link-layer register values.

If the board is not connected to any other 1394 nodes, the first register on the left, IDREG, should read 0xFFC0YYZZ. This indicates node 0 on bus 0x3FF, which is the local bus. The link version is YY (00 for PDI1394L11, 01 for PDI1394L21, and 03 for PDI1394L40) and the link revision code is ZZ.

If this is the result you obtain, then congratulations, your RDK is working properly.

3 Monitor Basics

3.1 Introduction


The monitor is a real-time interactive program allowing the developer to examine and control the PDI1394 chipset while the embedded software is running. Access to a full range of 1394 functional layers is made available, from bit-level probing of the Link or Physical device to Bus Layer or Transaction Layer function control.

There are two types of IEEE 1394 transactions: asynchronous and isochronous. These two types are handled differently by the AV Link chip.

3.1.1 Asynchronous

Asynchronous transactions are used for control messages and data transfers where guaranteed delivery is more important than guaranteed timing. There are four parts to an asynchronous transaction, request, request confirmation, response and response confirmation. Confirmations are synchronous; they are placed on the 1394 serial bus immediately after the request or response. Outgoing confirmations are generated directly by the AV Link chip and so are never seen by the monitor. Incoming confirmations are seen asynchronously because of the hardware FIFOs.

There are five types of asynchronous transactions: Block Writes, Block Reads, Quadlet Writes, Quadlet Reads, and Lock transactions.

The monitor displays asynchronous transactions, along with other asynchronous information, such as interrupts, in the Event Window. Click the Event Viewer button  in the toolbar to open the Event Viewer window.

3.1.2 Isochronous

Isochronous transactions are used for data that requires a regular data stream, where a packet arriving late is just as bad as a packet not arriving at all. Audio and video are the two largest applications where guaranteed timing is more important than guaranteed delivery, but other applications, such as video games, can take advantage of this as well.

On the IEEE 1394 bus, isochronous transactions could happen regularly every 125 microseconds. This is guaranteed because the specification allows up to 80% of the bus to be reserved for these transactions. The main advantage of using the Philips AV Link over competing IEEE 1394 Link devices is that it handles isochronous transactions completely in hardware. The transactions are available on the AV Link bus stripped of header information at the time specified in the timestamp. Therefore, these transactions are not visible to the software, which only monitors the host microcomputer bus.

Most devices that use isochronous bandwidth follow the ISO/IEC 61883 specification. This is also supported by the AV LINK in hardware.

3.2 Starting the Monitor

The monitor can be started either through the Start menu, either using the run option or clicking the appropriate icon in the Program submenu, or by typing `fwmonwin` at a DOS prompt while in the appropriate directory. An optional command-line switch allows you to specify the port to use. The syntax of this switch is `-pCOM#` where `#` represents the port (1-9) you elect to use. The software defaults to using COM1.

On startup the software checks to ensure that it is properly connected to an evaluation board running the correct version of the embedded software. If any errors occur during startup, the program displays an error message and quits.

If the monitor program determines that it is properly connected, it will reset the 1394 bus, which can be seen on the `Event Viewer` dialog. This ensures that the program has an up-to-date setting for the node id of the Evaluation Board, the number of boards connected, and the number of self-id packets available to be read.

Once the program has successfully started you should see the following screen:

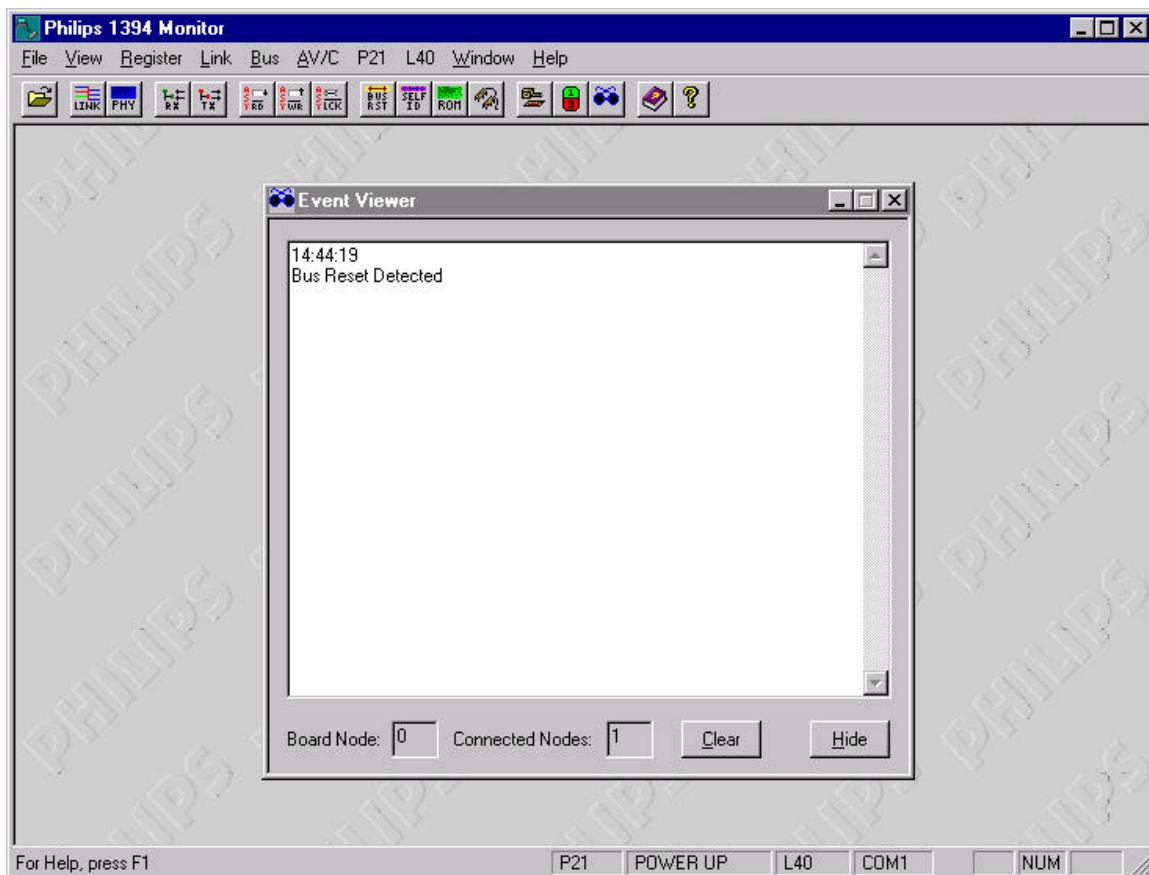


Figure 3-1: Monitor Main Screen

3.3 Program Navigation

Monitor uses the standard Windows GUI. Navigation between various areas of the program is done using menus and toolbar.

The status bar at the bottom of the main window displays a brief description of a toolbar button and, if the cursor rests momentarily on the button, a small pop-up “tooltip” window appears containing the button name.

Commands are given by clicking on the appropriate button. Commands that require additional parameters are entered through a dialog box.

3.3.1 Entering Data

The entering of data into the monitor is accomplished by:

1. typing a value into an edit box; or
2. choosing an option from a drop-down combo box.

Note: Unless otherwise indicated, all input fields expect data to be entered in hexadecimal format without leading characters.

3.4 Communication Settings

By default, the program defaults to the following settings:

Port	Speed	Parity	Data bits	Stop bits
COM1	9600bps	NONE	8	1

The port setting can be changed with the `-pCOM#` command line switch, where # is a number from 1 to 9.

Note: Except for the port number, the supplied embedded program is only designed to work at the default settings so take care in changing them.

3.5 Event Viewer

The Event Viewer receives all events generated by the embedded code. Events fall into two categories, events that inform you of asynchronous bus traffic and those that inform you of interrupts that have occurred.

Certain commands return a response of OK internally, but result in a more detailed event. An example of this is the command to read the configuration ROM. The OK response to this command indicates that the request to read the configuration ROM succeeded; the events following the command actually contain the contents of the configuration ROM CSRs.

An example of a typical Event Window display is:

11:48:37 Each event is tagged with the time at which it occurred.
Bus Reset Detected A bus reset was detected on the local bus.

11:48:37
Received a LNKPHYINT: A Link/PHY interrupt occurred.
Cycle Lost Which interrupt

11:48:54 The next nineteen lines are Self-ID packet information

Self IDs

length: 0x4
CRC: 0x9865
generation_number 0x3
node_count: 0x2
self_id_count: 0x2

Node	Link	GapCnt	Speed	Contend	Power	Init	Reset
0	1	63	1	0	0	1	

Ports:

0:UNCONN
1:PARENT
2:UNCONN

Node	Link	GapCnt	Speed	Contend	Power	Init	Reset
1	1	63	1	1	7	0	

Ports:

0:UNCONN
1:CHILD
2:UNCONN

11:49:15

Transmitted an Asynchronous Request: Performed Quadlet Read Req. of CSR 0x80

spd: 0
(100Mb/s)
tLabel: 0
rt: 0
(retry 1)
tCode: 4
(Quadlet read request)


Dest. ID: 0xFFC1
Dest. Offset High: 0xFFFF

3.6 Loading HEX Files

Recent versions of the L40 RDK have been fitted with flash programmable 89C51 microcontroller units. We recommend using the flash programming method detailed in the Addendum to this User Manual, it's titled Programming L40 RDK Flash Memory Microcontrollers and is included on the CDROM that accompanied this RDK. Please

read the Addendum and load the ISP software from the CDROM in order to use the flash ROM capabilities of the MCU. Benefits of flash programming are (1) non-volatile MCU code and (2) selectable erase/write memory slices that can (3) reduce your programming time.

Hex Download of code to RAM memory space:

The embedded software for the RDK is designed so that a new program can be downloaded and started to take its place. Clicking this button  on the toolbar opens Select Hex File to Download to Board dialog. The following steps detail how to replace the default embedded program with a new one.

Note: It is only possible to successfully download a HEX file when the embedded software is running from the EPROM. To make sure of this, press the reset button on the board before beginning any download.


1. Write the program and compile it;
2. Convert it to a .hex file;
3. If recompiling the supplied embedded program, use the supplied checksum generator, Check, as follows: check 0x12341234 a.hex > b.hex where '0x12341234' is the bottom 32 bits of the globally unique id, found in the 1394 configuration ROM, and 'a' and 'b' can be any file name;
4. Start the monitor;
5. Select Load Hex... from the File menu; and
6. Select the .hex file created in the latter of step 2 (or 3 if it occurred).

At this point, the program should begin downloading. While the hex file is being downloaded, the program displays a dialog informing you that the download is in progress. The monitor is unable to perform any other actions while the download is in progress. A dialog box will appear to inform you that the download has finished.

Even if the new program is perfectly compatible with the shipped version, because you are now running from RAM the download may be followed by a ROM Check error.

3.7 Quitting

The monitor can be shut down in any of the following ways:

1. Clicking  on the main screen;
2. Double-clicking the system menu; or
3. Choosing Exit from the File menu.

Closing the program will not have any effect on the embedded software and will free up the communications port for use by other applications.

4 Monitor Sections

4.1 Isochronous Services

The 1394 architecture provides services for two types of data: asynchronous and isochronous. Isochronous data is guaranteed to have a constant bandwidth, and to arrive within a certain time span. Isochronous data is typically continuous audio or video data that must arrive at a certain rate but not at any specific point in time.

The evaluation board can be used as an isochronous transmitter or receiver by connecting an appropriate device to the byte-wide AV ports on the board and connecting another isochronous receiver/transmitter to one of the 1394 ports. Once the isochronous parameters have been configured for the node, no further intervention is required.

For example, a 1394 capable video camera could be set up to transmit an isochronous video stream to the board node. The board would then transmit that data out either port 1 or port 2 to the receiver.

4.1.1 Transmit Configuration

Configuration of the board node for isochronous transmission involves setting the appropriate bits in the AV Link registers. This dialog allows direct access to register values and breaks out the register values for ease of use.

***Note:** Before changing any field, use the Stop Transmit button in the Isochronous Transmit Setup dialog. This will prevent isochronous transmission error. Once the changes have been made, restart transmission by clicking the Start Transmit button.*

When the board node is used as a transmitter, any channel can be chosen for the transmitter, as long as the receiver on the other end of the 1394 bus knows to expect data on that channel.

***Note:** Many video cameras default to channel 63 as that is the broadcast channel.*

Clicking this button  on the toolbar opens Isochronous Transmit Setup Dialog.

Refer to *Online Help* button  on the toolbar about Transmit Setup for more details.

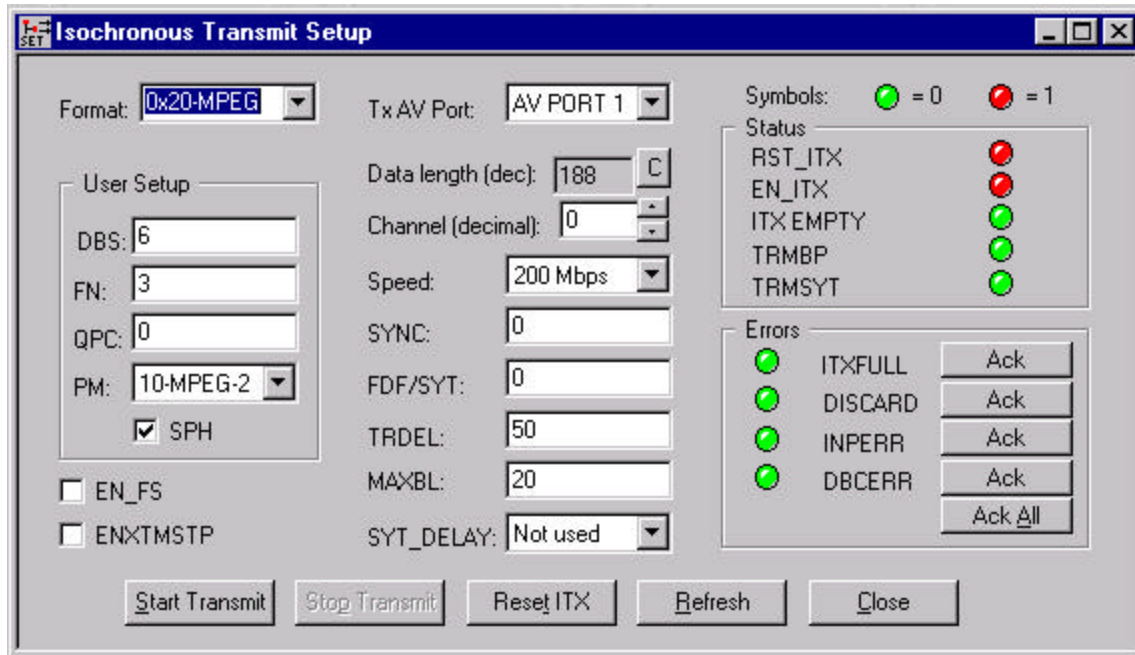


Figure 4-1: Isochronous Transmit Setup Dialog

4.1.2 Receive Configuration

This window allows direct access to the registers required to set up the node as an isochronous receiver.

***Note:** Before changing any field, use the Stop Receive button in the Isochronous Receive Setup dialog. This will prevent isochronous reception error. Once the changes have been made, restart reception by clicking the Start Receive button.*

When the board node is used as a receiver, the channel must correspond to the channel used by the transmitter.

Clicking this button  on the toolbar opens Isochronous Receive Setup Dialog.

Refer to *Online Help* button  on the toolbar about Receive Setup for more details.

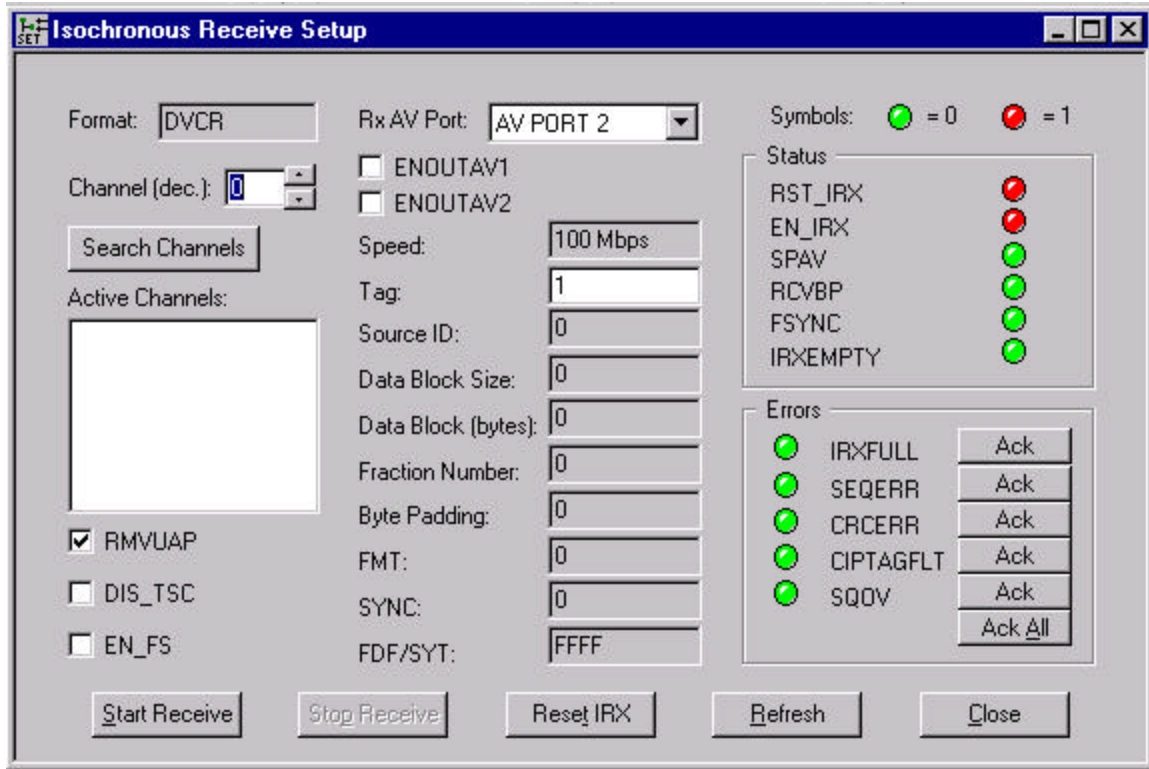


Figure 4-2: Isochronous Receive Setup Dialog

The following table shows the register values for the different preset transmit modes. The embedded software automatically sets these values when entering the mode, so these values are for reference only.

		MPEG-2	DSS	DVC
ITXPKCTL register	(address 0x20)			
TRDEL	(bits 23-16)	0x40-0xA0	0x40-0xA0	0x40-0xA0
MAXBL	(bits 5-8)	16	1	1
EN_ITX	(bit 4)	1	1	1
PM	(bits 3-2)	2	0	1
EN_FS	(bit 1)	0	0	1
RST_ITX	(bit 0)	0	0	0
ITXHQ1 register	(address 0x24)			
DBS	(bits 23-16)	6	35	120

FN	(bits 15-14)	3	0	0
QPC	(bits 13-11)	0	0	0
SPH	(bit 10)	1	1	0
ITXHQ2 register	(address 0x28)			
FMT	(bits 29-24)	32	33	0
FDF/SYT	(bits 23-0)	0	0	0
ITXCTL register	(address 0x34)			
TAG	(bits 15-14)	1	1	1
CHANNEL	(bits 13-8)	0-63	0-63	0-63
SPD	(bits 5-4)	0 / 1	0 / 1	0 / 1
SYNC	(bits 3-0)	0	0	0

Table 4-1: Transmit Mode Register Values

The following table lists the required signals for the preset isochronous transmit modes, where AV_n represents AV Port 1 (J8), or AV Port 2 (J10).

Signal Name	Pin Number(s)	MPEG-2	DSS	DVC (525-60)
Av _n D[0:7]	2, 4, 6, 8, 10, 12, 14, 16	Required	Required	Required
Av _n CLOCK	18	0-25 MHz	3.33 MHz	3.84 MHz
Av _n SYNC	26	Required	Required	Required
Av _n FSYNCIN	32	Don't Care	Don't care	29.97 Hz
Av _n ENDPCK	30	GND	Required	GND
Av _n VALID	24	Required	Required	Required

Table 4-2: Required Isochronous Transmit Mode Signals

The following table shows the register values for the different preset receiver modes. As with the transmit values, they are automatically set by the embedded software when entering the mode.

		MPEG-2	DSS	DVC
IRXPCTL register	(address 0x40)			
EN_IRX	(bit 4)	1	1	1

BPAD	(bits 3-2)	0	2	0
EN_FS	(bit 1)	0	0	1
RST_IRX	(bit 0)	0	0	0
IRXCTL register	(address 0x50)			
TAG	(bits 15-14)	1	1	1
CHANNEL	(bits 13-8)	0-63	0-63	0-63

Table 4-3: Preset Receive Mode Register Values

The only signal that may be required to be supplied for the isochronous receiver is the clock, where AV_n represents AV Port 1 (J8), or AV Port 2 (J10). Alternatively, the RXAP_CLK field may be set to a value other than "00" in the IRXPKTCTL register.

Signal Name	Pin Number	MPEG-2	DSS	DVC (525-60)
AV _n CLOCK	18	0-25 MHz	3.33 MHz	3.84 MHz

Table 4-4: Isochronous Receiver Signal Required

The other signals are outputs, and may or may not appear depending on the mode, where AV_n represents AV Port 1 (J8), or AV Port 2 (J10).

Signal Name	Pin Number(s)	MPEG-2	DSS	DVC (525-60)
AV _n D[0:7]	2, 4 ,6, 8, 10, 12, 14, 16	Yes	Yes	Yes
AV _n SYNC	26	Yes	Yes	Yes
AV _n FSYNCOUT	32	No	No	Yes
AV _n VALID	24	Yes	Yes	Yes
AV _n ERR[0]	22	Yes	Yes	Yes
AV _n ERR[1]	28	Yes	Yes	Yes

Table 4-5: Other Output Signals

4.2 Asynchronous Services

Asynchronous data is the second type of data transmitted on the 1394 bus. It is data that is not guaranteed a fixed bandwidth, and arrives at random times. 20% of the bandwidth is reserved for asynchronous data. Asynchronous data is sent after all isochronous data has been transmitted. Asynchronous data is packaged into packets consisting of one or more quadlets, or 32 bit values.

There are three types of asynchronous transactions defined in the 1394 standard: read, write and lock, and two amounts of data, quadlets, or blocks. Quadlets are simply 32 bit quantities, whereas blocks are a series of sequential 8-bit bytes, with the number of bytes set as part of the transaction. Since only quadlets can be transmitted, blocks are padded with zeros when necessary to the next quadlet.

The various asynchronous transmissions require different parameters; the following table explains the parameter choices:

Parameter	Description and Comments
Speed	The speed of the transaction in bits per second: 100Mb/s 200Mb/s 400Mb/s
Destination Node	The node with which you are exchanging packets
CSR Address	The CSR address you are writing to/reading from. An offset of 0xFFFF F000 0000 (the start of the initial register space) is added to the address you specify.
Data	The quadlet of data you wish to write.
Number of Bytes to Write	The number of bytes you wish to write as part of a <code>Block Write Request</code> . After you close the dialog, you will see a number of dialogs asking for a quadlet to write.
Starting CSR Address	Used in the <code>Block Write Request</code> to determine what CSR address to use as a starting address when writing a block of data.
Extended Transaction Code	Identifies the extended transaction code used in lock requests. This transaction code identifies a mathematical operation to be performed on the data in the CSR register mentioned. mask_swap compare_swap fetch_add little_add bounded_add wrap_add vendor-dependant


Table 4-6: Asynchronous Transmission Parameters


Parameter	Description and Comments
Retry Code	The type of retry protocol to use in a transaction: retry1: dual-phase retry protocol retryX: single-phase retry protocol retryA: dual-phase retry protocol retryB: dual-phase retry protocol <i>Note: retryX is the only protocol currently supported. This is Errata E-2 on the PDI1394L21 errata sheet.</i>

Table 4-7: Unsupported Asynchronous Transmission Parameters

Most of the asynchronous transactions involve sending and receiving asynchronous 1394 events, so the record of the transmitted event as well as any responses or confirmations will appear in the Event Viewer.

4.2.1 Quadlet/Block Read Request

Clicking this button  on the toolbar brings up a dialog that allows you to send a Quadlet or Block Read Request on the 1394 bus, asking that the specified node return the value in the specified CSR register. Enter the CSR Offset to start from and Data Length (in bytes) to read. A Quadlet Read Response packet will return this value.

Refer to *Online Help* button  on the toolbar about Quadlet or Block Read Request for more details.

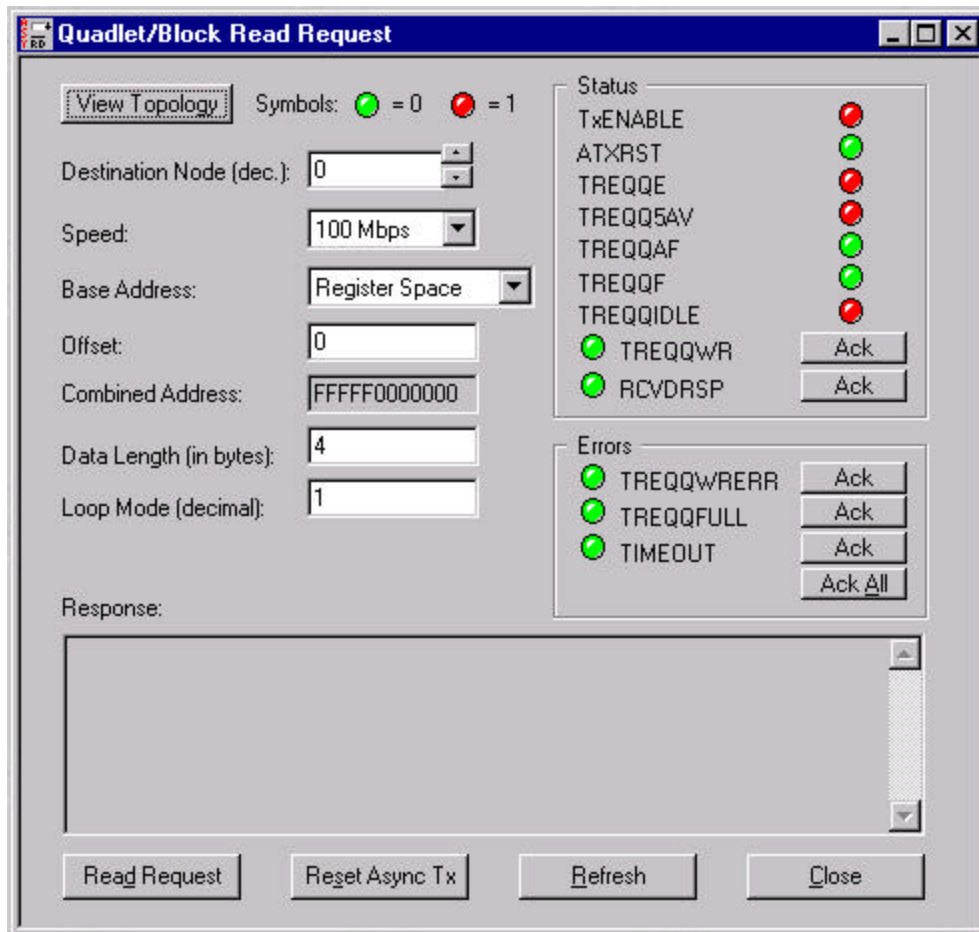




Figure 4-3: Quadlet/Block Read Request Dialog

4.2.2 Quadlet/Block Write Request

Clicking this button  on the toolbar brings up a dialog that allows you to transmit a Quadlet or Block Write Request on the 1394, asking that the specified node change the value of a CSR register to the specified new value. The new value is entered in the edit box right above the Add button.

Refer to *Online Help* button  on the toolbar about Quadlet or Block Write Request for more details.

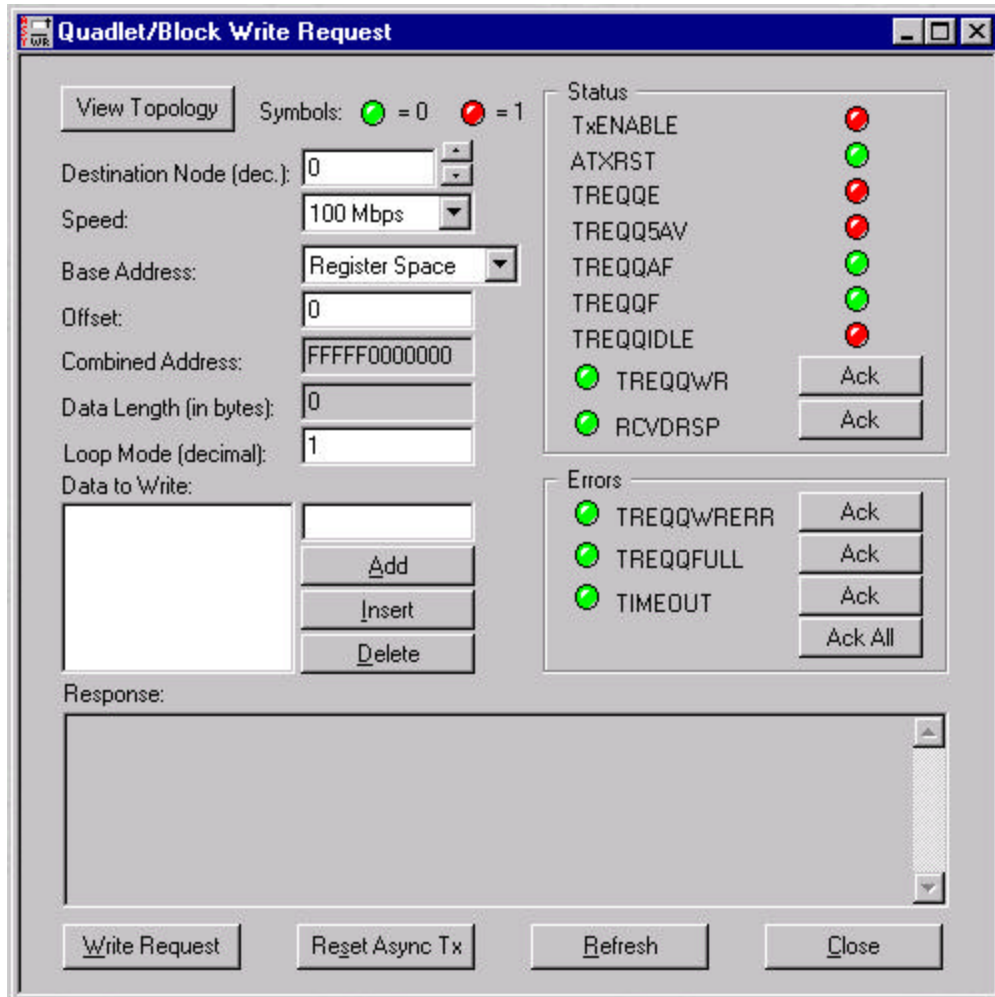



Figure 4-4: Quadlet/Block Write Request Dialog

4.2.3 Quadlet Lock Request

Clicking this button  brings up a dialog that allows you to transmit a Lock Request on the 1394 bus. The 'Lock' transaction combines the read, compare, and conditional write operations. The dialog allows you to choose the type of transaction, the data and test arguments (if applicable), and then transmits the packet to the specified node.

Refer to *Online Help* button  on the toolbar about Quadlet Lock Request for more details.

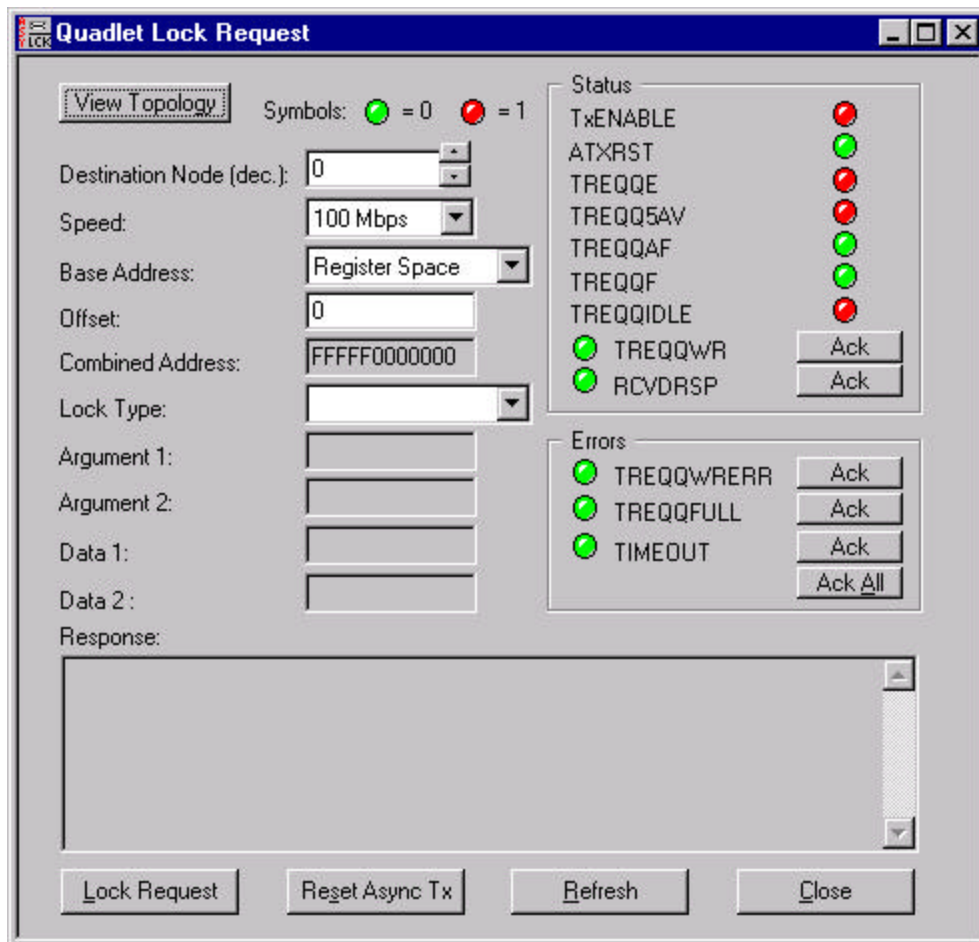


Figure 4-5: Quadlet Lock Request Dialog

The monitor supports the following lock transactions (shown using C/C++ syntax):

Lock Function	Update Function
mask_swap	<code>new_value = data_value + (old_value & ~arg_value);</code>
compare_swap	<code>if (old_value == arg_value) new_value = data_value;</code>
fetch_add	<code>new_value = old_value + data_value;</code>
little_add	<code>(little) new_value = (little) old_value + (little) data_value;</code>
bounded_add	<code>if (old_value != arg_value) new_value = old_value + data_value;</code>
wrap_add	<code>new_value = (old_value != arg_value) ? old_value + data_value data_value;</code>

However, the embedded software only supports compare_swap, as it is the only function required by the 1394 Specification.

4.3 PHY Registers

As the name implies, the PHY Registers dialog allows access to the physical layer registers on the Philips PHY chip. This dialog can be used to diagnose problems with the cable physical layer, or to visualize the operation of the PHY chip.

Clicking this button  on the toolbar opens Phy Registers Dialog.

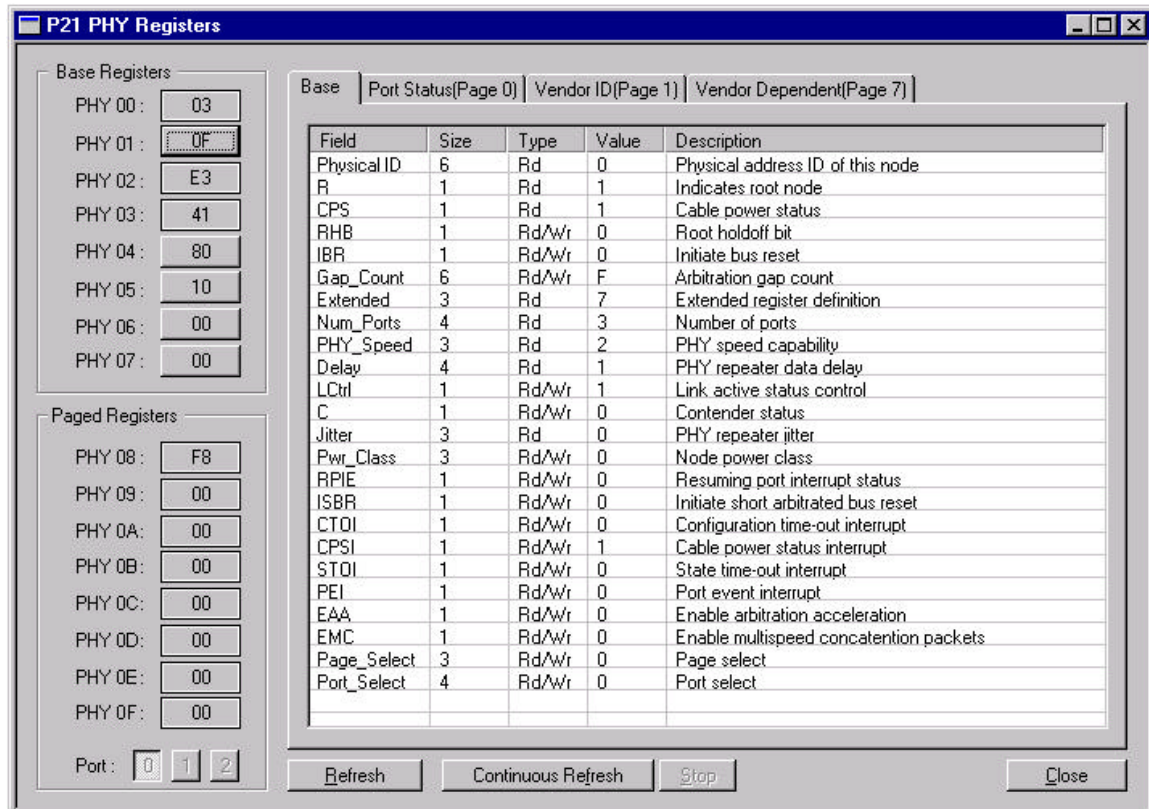


Figure 4-6: Phy Registers Dialog


P21 registers are divided into two groups: Base and Paged, and are displayed as push buttons on the left side of the dialog (PHY 00 through PHY 0F). Register buttons are displayed as “flat” push buttons when the register is read only (R) or “raised” when the register can be modified (R/W). To change a PHY register’s value, just click on the appropriate button and type in a new value. Register field information is displayed within the grid list located to the right of the register buttons. Selecting the desired tab located across the top of the list (Base, Port Status, Vendor ID, or Vendor Dependent) controls the information displayed. When Port Status tab is selected you may use the port 1, 2, or 3 push buttons to view a different port. Use the Refresh or Continuous Refresh buttons to update the dialog’s information.

4.4 Bus Management Menu


This menu allows you to access the functionality of the AV LINK chip relating to the Serial Bus Management layer. It allows the initiation of a bus reset, display of self-id packets or bus topology, and other features described below.

4.4.1 Low-Level Bus Functions


4.4.1.1 Bus Reset

A bus reset is automatically generated when a node is added to or removed from the 1394 bus. This button  allows you to manually generate a bus reset, which ensures that the Event Viewer window contains the current value for the local node ID and the number of nodes connected.

4.4.1.2 Display Self-IDs

Clicking this button  initiates a read of the topology CSR registers in the AV LINK chip and interpretation of the Self-ID packets to provide information about the nodes. The results are returned in the form of an event, which will be visible in the Event Viewer.

4.4.1.3 Show Bus Topology

This option is similar to that of the `Display Self-IDs` button in that the topology registers are read, but the data is returned in a graphical (tree) format. Clicking this button  on the toolbar opens Bus Topology Dialog.

4.4.1.4 Send PHY Config Packet

A PHY config packet serves two purposes: to designate a specific node as root, and to modify the bus gap count. The changes requested by a PHY config packet are sent to all nodes on the bus and the PHY registers are updated accordingly. However, the node with the root hold-off flag will not become root until the next bus reset.

For example, to send a PHY config packet in a 1394 network with two Full Duplex RDK nodes:

1. Use the monitor connected to the node currently set as root.
2. Select `Send PHY Config Packet` from Bus menu.
3. Enter the following as parameters:
 - a. Check the Force Root checkbox
 - b. Enter 0 for the physical ID
 - c. Check the Gap Count checkbox
 - d. Enter 2E for the gap count
 - e. Send the packet by choosing `Send`

4. Examine the PHY Registers menu on the node and verify the following settings:
 - a. Physical ID: 0
 - b. Root Flag: 0
 - c. Root Holdoff: 1
 - d. Gap Count: 0x2E
5. Generate a bus reset and verify that the configuration has changed by reading the registers
 - a. Physical ID: 1
 - b. Root Flag: 1
 - c. Root Holdoff: 1
 - d. Gap Count 0x2E

4.4.1.5 Send Link-On Packet

A link-on packet is used to 'wake-up' the link layer of nodes that are in a power-saving standby mode. This command has no effect on Full Duplex boards as the AV LINK Links are always on.

4.4.2 Serial Bus Management

4.4.2.1 CSR Quick Summary

The CSR package handles all CSR requests except for RESET_START, which is handled immediately by the Link chip.

The CSR package is designed as an implementation guide, it does not provide a full set of registers that is compliant with IEEE 1394-1995 or IEC/ISO 61883.

All required core CSR registers and Serial Bus dependent registers have been fully implemented and should be compliant with IEEE 1394-1995 and IEEE 1212-1994. Some of the optional fields and registers have also been implemented.

Not all registers have full backing code. For instance, although the plug registers are implemented, setting a plug register does not start a connection.

The optional registers MAINT_UTILITY, MESSAGE_REQUEST, and MESSAGE_RESPONSE have been implemented but do not have any effect. They are useful as a scratch test pad area. The monitor also uses this area for board-to-board communications.

The BUSY_TIMEOUT register and the SPLIT_TIMEOUT register have a default that is larger than that specified. These are the recommended values for slow 8051 implementations such as ours. We do not adjust value on other nodes of the network, but that will have to be done to avoid split timeout errors.

Offset	CSR	Quadlet Read	Quadlet Write	Lock	Block Read	Block Write	Description	Reference
000 ₁₆ ,004 ₁₆	STATE_CLEAR STATE_SET	•	•				State and control information	1394-1995 8.3.2.2.1
008 ₁₆	NODE_IDS	•	•				Specifies 16-bit node ID value	1394-1995 8.3.2.2.3
00C ₁₆	RESET_START		•				Resets the PDI1394L21. Implemented in hardware	1394-1995 8.3.2.2.4
018 ₁₆ ,01C ₁₆	SPLIT_TIME_OUT_HI SPLIT_TIME_OUT_LO	•	•				Split-request time-out	1394-1995 8.3.2.2.6
080 ₁₆ -0FC ₁₆	MESSAGE_REQUEST MESSAGE_RESPONSE	•	•	•	•	•	Optional message passing registers	1394-1995 8.3.2.2.11
200 ₁₆	CYCLE_TIME	•	•				For nodes providing isochronous services.	1394-1995 8.3.2.3.1
204 ₁₆	BUS_TIME	•	•				For nodes that require synchronized bus time.	1394-1995 8.3.2.3.2
210 ₁₆	BUSY_TIMEOUT	•	•				For transaction capable nodes.	1394-1995 8.3.2.3.5
21C ₁₆	BUS_MANAGER_ID	•		•			For selecting or locating the bus manager.	1394-1995 8.3.2.3.6
220 ₁₆	BANDWIDTH_AVAILABLE	•		•			Bandwidth allocation	1394-1995 8.3.2.3.7
224 ₁₆ ,228 ₁₆	CHANNELS_AVAILABLE	•		•			Channel allocation	1394-1995 8.3.2.3.8
230 ₁₆	MAINT_UTILITY	•	•				For introducing diagnostic error conditions.	1394-1995 8.3.2.3.10
400 ₁₆ -7FC ₁₆	CONFIG_ROM	•			•		Configuration ROM	1394-1995 8.3.2.5
900 ₁₆	OMPR	•		•			Output Master Plug Register	61883 7.5

904 ₁₆	OPCR	•		•			Output Plug Control Register 0	61883 7.7
980 ₁₆	IMPR	•		•			Input Master Plug Register	61883 7.6
984 ₁₆	IPCR	•		•			Input Plug Control Register 0	61883 7.8
1000 ₁₆ -13FC ₁₆	TOPOLOGY_MAP	•			•		Topology Map	1394-1995 8.3.2.4.1
2000 ₁₆ -2FFC ₁₆	SPEED_MAP	•			•		Speed Map	1394-1995 8.3.2.4.2

4.4.2.2 Read CSR Register

This operation is the same as the read operation available from the Asynchronous Services from Link menu. It sends a `Quadlet Read Request` on the 1394 bus to the requested node asking for the contents of a CSR register to be sent back as a `Quadlet Read Response`.


4.4.2.3 Write to CSR Register

This operation is the same as the `Quadlet Write Request` available from the Asynchronous Services from Link menu. It attempts to have a node on the 1394 bus change its contents to the new value supplied.

4.4.2.4 Lock CSR Register

This operation is identical to the `Quadlet Lock Request` command available through the Asynchronous Services from Link menu.

4.4.2.5 Configuration ROM

Clicking this button  initiates a scan of the configuration of the ROM CSR registers, displaying the formatted result in the Event Viewer.

4.5 Link Registers

The Link Registers dialog displays all but two of the link registers. The registers, `RREQ` and `RRSP`, are used internally and cannot be displayed. The remaining registers are all visible and, where applicable, changeable.

Clicking this button  on the toolbar opens Link Registers Dialog.

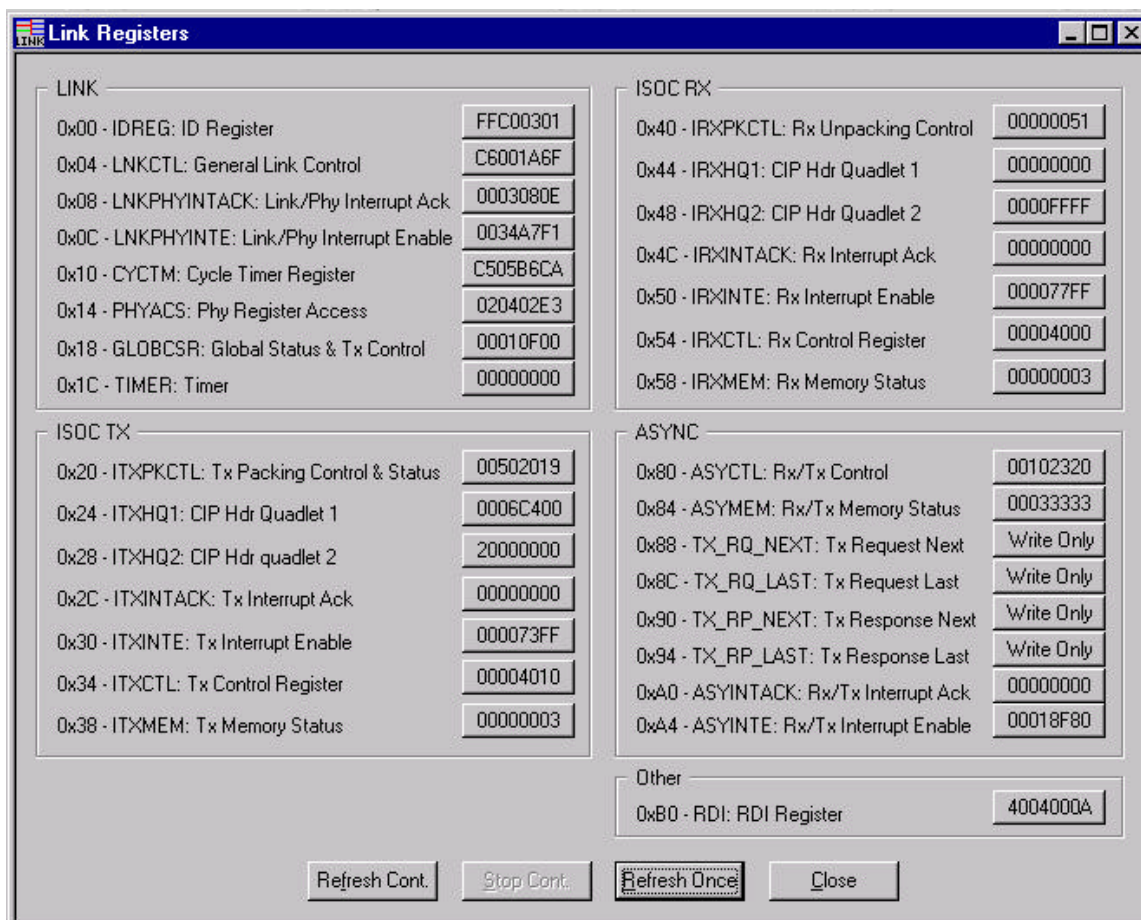


Figure 4-7: Link Register Dialog

To change a register's value, just click on the appropriate button and type in a new value.

4.6 Camcorder Operations

This dialog provides an easy way to send the most common camcorder commands through the 1394 interface. These commands are documented in the *AV/C Digital Interface Command Set General Specification*.

While the AV/C spec provides for a rich variety of operations, the camcorder dialog uses only a commonly implemented subset. All of the operations have the following settings in common:

Setting	Value	Label
Command type (ctype):	0	CONTROL
Subunit Type	4	Video Cassette Recorder (VCR)
Subunit ID	0	Instance Number 0


Table 4-8: Common Camcorder Operation Settings

The commands are a combination of a one-byte opcode followed by a one-byte operand. For example, the Wind opcode is 0xC4, and has four possible opcode values:

1. **High Speed Rewind:** 0x45
2. **Stop:** 0x60
3. **Rewind:** 0x65
4. **Fast Forward:** 0x75

Note: The camcorder commands use a subunit type corresponding to a VCR. This is due to the fact that this type has been shown to be the most compatible with the widest variety of 1394-capable video cameras.

Like the Quadlet/Block Write Request and similar dialogs, the Camcorder Dialog allows a choice of a destination node and speed, then provides buttons which display a graphic showing the camcorder function they provide.

Clicking this button  on the toolbar opens Camcorder Operations Dialog.

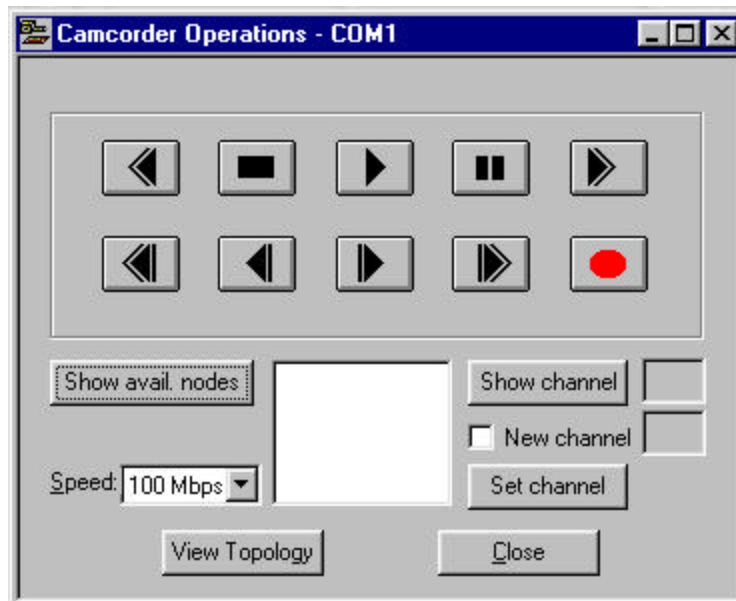


Figure 4-8: Camcorder Operations Dialog

Pressing a button on this dialog generates a Quadlet Write Request to the specified node writing the created AV/C frame to the FCP_COMMAND CSR register at address 0xFFFF F000 0B00.

4.7 L40 Menu

This menu allows you to access new features of the PDI1394L40 chip. There are two menu items under L40 and they are enabled only for L40 chip. Each menu item is used to bring up an appropriate window such as: Indirect Address Quadlet and FIFO Size Registers Block.

4.7.1 Indirect Address Quadlet

This dialog allows access to new L40 individual registers that are addressed via indirect addresses.

Refer to *Online Help* button  on the toolbar about Indirect Address Quadlet for more details.

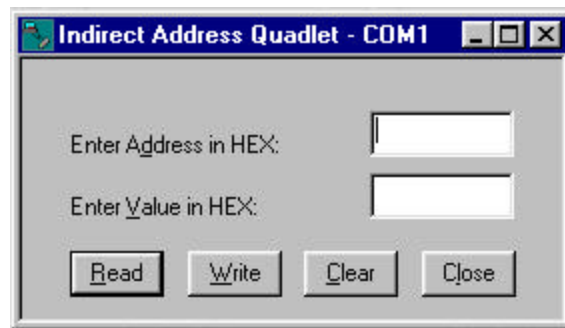



Figure 4-9: Indirect Address Quadlet Dialog

4.7.2 FIFO Size Registers Block

The FIFO Size Registers Block dialog allows you to access the following FIFO registers such as: RRSPSIZE (0x100), RREQSIZE (0x104), TRSPSIZE (0x110), TREQSIZE (0x114), IRXSIZE (0x120), and ITXSIZE (0x130). Consult data sheets for more information.

Refer to *Online Help* button  on the toolbar about FIFO Size Registers Block for more details.

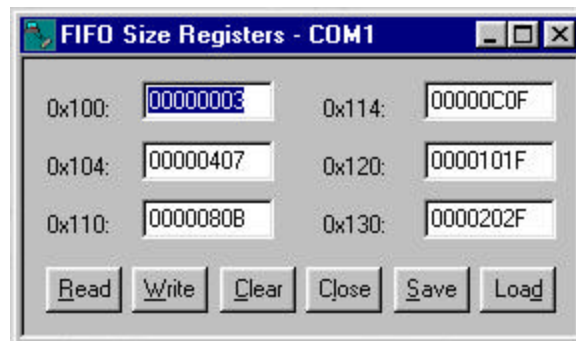


Figure 4-10: FIFO Size Registers Dialog

4.8 Extended PHY Packets

The 1394a supplement identifies extended PHY packet types that include:

- Ping packet
- Remote Access packet
- Remote Reply packet
- Remote Command packet
- Remote Confirmation packet
- Resume packet

Each of these extended packet types can be accessed through **P21** menu by clicking the **Extended PHY Packets...** menu item.

4.8.1 Ping Packet

The ping packet is used to cause the target node to transmit self-ID packet(s) that reflect the current configuration and status of the PHY.

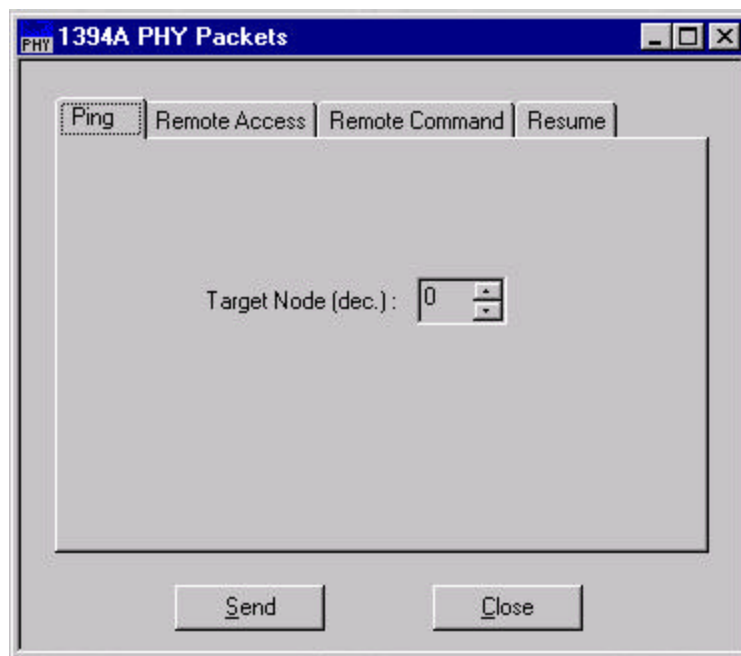



Figure 4-11: Ping Packet

The `Target Node (dec.)` specifies the physical node identifier of the destination of this packet. In response to a ping packet, the target PHY broadcasts its self-packet(s). The activity can be viewed in the **Event Viewer** dialog by clicking Event Viewer  button in the toolbar if it is not opened.

4.8.2 Remote Access and Remote Reply Packets

The remote access packet offers a means of accessing a PHY register within the target node.

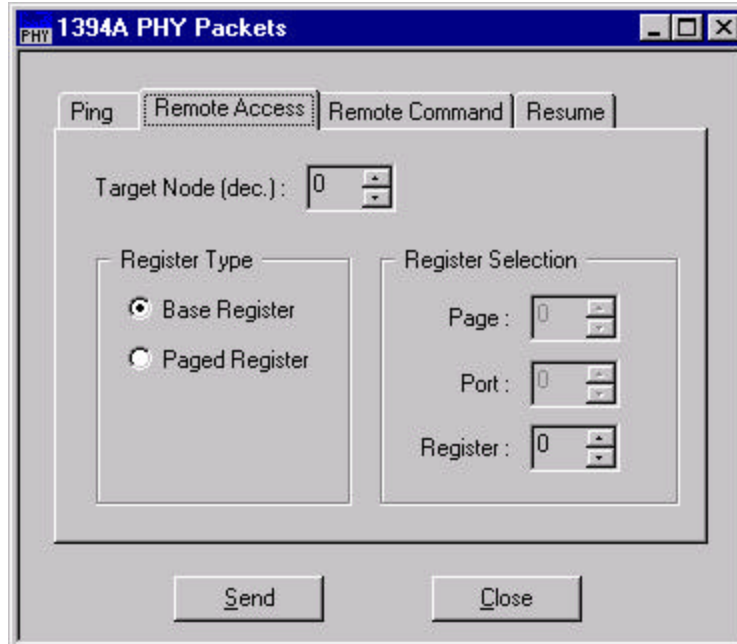


Figure 4-12: Remote Access Packet

The `Target Node (dec.)` specifies the physical node identifier of the destination of this packet. There are two types of registers that are supported for a remote access packet:

- **Base Register:** One of the eight base PHY registers is to be read. The `Register` edit box specifies the target register.
- **Paged Register:** One of the eight paged registers is to be read. The `Page` edit box selects one of the eight groups of paged registers to be read. The `Register` edit box specifies the target register within the selected page. The `Port` edit box identifies the target port if the page is 0 (PHY port registers).

The remote reply packet returns the read data specified by the remote access packet. The data is the current value of the PHY register addressed by the remote access packet. If the register is reserved or unimplemented, the data will be zero.

The activity can be viewed in the **Event Viewer** dialog by clicking Event Viewer  button in the toolbar if it is not opened.

4.8.3 Remote Command and Remote Confirmation Packets

The remote command provides a mechanism to issue a variety of commands to the selected port within the target PHY.

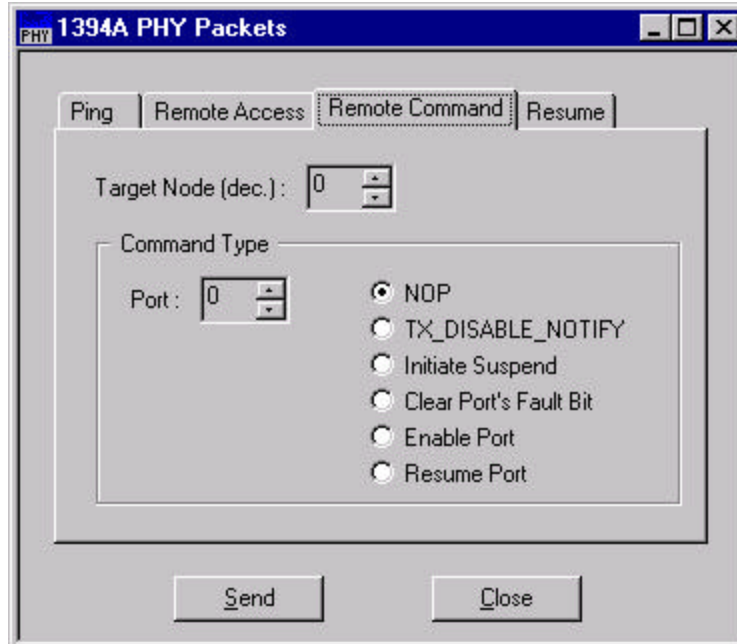


Figure 4-13: Remote Command Packet

The `Target Node (dec.)` specifies the physical node identifier of the destination of this packet. The `Port` edit box selects one of the PHY's ports as the target of the command.

The remote confirmation packet is returned by the node that was targeted by a remote command packet to verify that the command has been successfully processed, or not. The confirmation packet also returns current status for the selected port.

The activity can be viewed in the **Event Viewer** dialog by clicking Event Viewer  button in the toolbar if it is not opened.

4.8.4 Resume Packet

The resume packet is broadcast to all PHYs to command that all connected and suspended ports must resume normal operation.

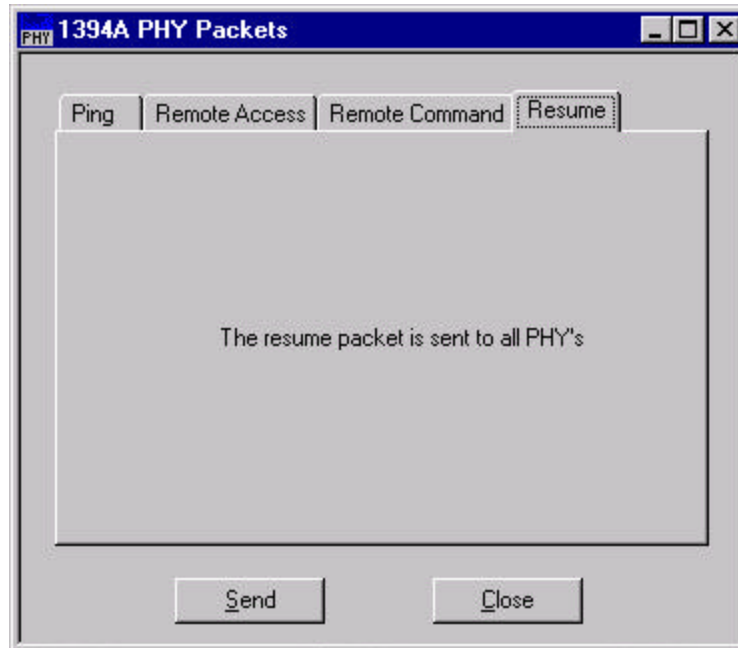



Figure 4-14: Resume Packet

The resume packet is broadcast; therefore there is no reply. The activity can be viewed in the **Event Viewer** dialog by clicking Event Viewer  button in the toolbar if it is not opened.

5 Embedded Software

5.1 Introduction

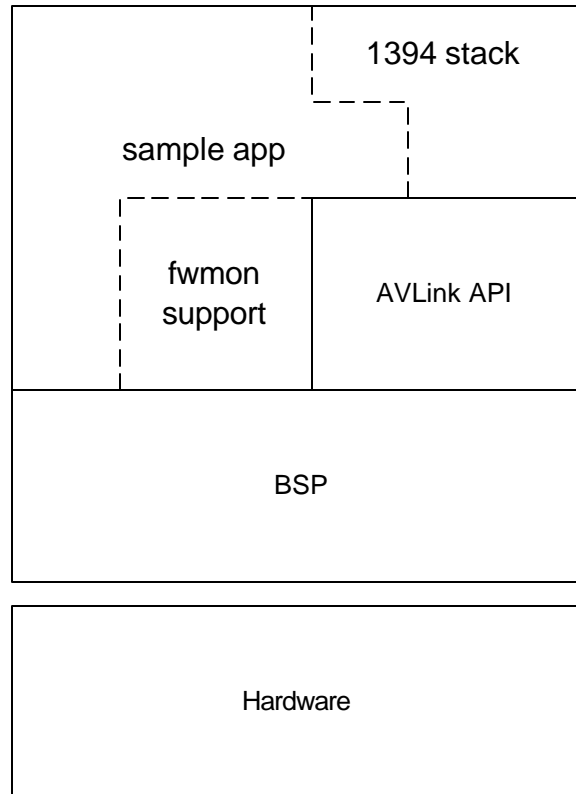


Figure 5-1: Software Layer Diagram

There are three parts to the embedded software delivered with the Philips 1394 RDK. These are the Board Support Package (BSP), the Application Programming Interface (API), and the application.

The application performs two roles:

1. It is a sample that may be used as a guideline in programming user applications and completing the 1394 stack; and
2. It supports the monitor program.

The API provides an interface to the Philips 1394 Link Layer controller chips for user applications and the 1394 protocol stack.

The BSP is a layer that abstracts the hardware and the operating system functionality to increase portability, modularity, and functionality. It is written in a combination of assembly language, Keil C Compiler specific C and ANSI C so that the API and the application may be written in highly portable ANSI C.

There is no operating system included with the Philips 1394 RDK. However, services traditionally provided by an operating system that would be useful to those using the RDK software, programming with the API or porting the system are included in the BSP.

The first service that is provided by the BSP is hardware abstraction. The module in the BSP that is primarily responsible for this duty is the HW module. The Startup, SelfTest, Serial and Timer modules are the only other modules in the system and the BSP that directly access hardware, as these modules also provide hardware abstraction.

The other major service that is provided by the BSP is operating system abstraction. Event, PktQ, Watermark, and Error are the modules that primarily perform this task. Because there is no operating system, these modules implement the required features as well as abstracting them. These modules have been written to be as portable as possible, and could be used as is with an operating system, but efficiency would be gained by using operating system features. The only exception is the Watermark module, which was written in assembly language.

5.2 BSP Architecture Notes

This section talks about the BSP, its implementation and some of the issues involved in using the BSP and porting it to another platform. It is not just useful to those porting the BSP; it explains some of the structure of it and some of the tradeoffs made of in its implementation. Therefore, it is useful for those just using the BSP as well. This section does not explain details of the BSP. The documentation that describes each BSP module contains these details. As always, the code is the final reference.

5.2.1 The Current Platform

The Evaluation Board uses an 8051 with 64K of RAM and 64K of ROM as its microprocessor system. The software is mainly written in C using the Keil C compiler.

The 8051 was not designed to run high level languages. The largest limitation is the 128-byte stack limitation. On the 8051, the stack is held in internal memory, which is limited to 256 bytes.

If the Keil 8051 C compiler stored all function parameters and automatic variables on the stack, you would very quickly run out of stack space. The Keil compiler stores function parameters and automatic variables in fixed memory locations. This reduces the stack usage for a function call to two bytes for the return address, but also has the effect of making functions non-reentrant. Functions may be explicitly declared reentrant, however. The only function that does so is the BRRecurseDepth function.

The RDK is written using the large memory model. This means that all variables are stored in external memory. Pointers are generally declared using the `_XDATA` declarator. This makes them smaller and more efficient. `_XDATA` may simply be `#define'd` away on other systems.

5.2.2 Threads

As there is no operating system underlying the current implementation, threads may seem like a misstatement. In the current system, this refers only to the main loop thread and to the interrupt service routine thread, emphasizing the fact that the code was written with the possibility of porting to a true multitasking operating system in mind.

In some places in the documentation, the interrupt service routine is modeled as a thread that runs completely within a critical section. In other places functions with embedded critical sections have a functionally identical counterpart with the suffix ISR. This indicates a function without the critical section markers, designed to be called from interrupt service routines. In some cases, functions with the suffix ISR have been created that are identical to the non-ISR functions. These are created because of the non-reentrant nature of the 8051 - we cannot call from the ISR functions that are also called by the main loop thread. These identical functions may be removed when porting to a different platform.

5.2.3 Semaphores

Throughout the code, there are variables that are used in a manner similar to semaphores. These have been noted in the comments and documentation. Their usage as semaphores imposes limits on how they may be used. For example, one thread may only be allowed to increment the variable while the other thread may only decrement it. Variable access may also have to be protected by a critical section.

When porting the code it may be desirable to replace these pseudo-variables with a true semaphore.

One particular semaphore function in the HW module is called LocalLock. This function works similarly to the 1394 compare and swap operation. A search for this function as well as a search for critical sections (denoted by ENTER_CRITICAL_SECTION and DISABLE_LINK_INTERRUPTS), should quickly locate most pseudo-semaphores and other thread-aware code structures.

5.2.4 Events

One key component of the BSP and the RDK software is the message passing mechanism implemented inside the Event package. You may refer to the documentation on the Event package in Section 5.3 for further details.

Events have two primary purposes, they are used to pass messages between threads, and they are used to pass messages up the protocol stack. Function calls may be used to pass messages down the protocol stack.

Events also provide the scheduling for this system. The only duties of the main loop are dispatching events and flashing the heartbeat.

All of the events used by the program are listed in the Event List.

5.2.5 Memory Management

There is no memory management in the system, which has two major consequences:

1. In cases where it may be desirable to have memory allocation, the function usually calls for a pointer to scratch memory. This shifts the responsibility for the memory allocation to the application, which has a better grasp on memory usage and can reuse memory easier than can the BSP.
2. Many BSP packages are hardcoded in ways that would not be necessary in an environment with a malloc. The Packet Queue (PktQ) module is perhaps the best example of this - all queues use #define's to represent the queue size, length and handle rather than using the dynamic allocation that would be more convenient.

5.2.6 Interrupts

It should be noted that there are two interrupt modules that handle link interrupts. (The Serial module and the Timer module also include ISR's). The first one is INTBSP, which includes the platform specific error code. It is a simple module whose primary purpose is to call into the INT module when an interrupt occurs.

5.2.7 Errors

The various error return codes are listed in Chapter 10, Error Codes.

Many functions return a value of type RETURN_CODE. This is an integer type of some size (an unsigned char in the current implementation). All of the possible values that the function may return are listed in the function documentation.

RETURN_CODEs greater than or equal to E_ERROR in value are error conditions. Those below E_ERROR in value are not error conditions.

It is possible to ignore the return code if you do not care about the distinction between them. Before returning with an error code, the function always calls the macro ERROR(). In the current implementation, this macro sends a signal to the monitor. In production code, it is expected that you will choose to define away this macro. In preproduction code however, this macro could prove useful.

Some errors occur in places where it is impossible to return an error code. In this case, the code will call the FATAL_ERROR() macro.

See the error module documentation for more information on the ERROR() and FATAL_ERROR() macros.

5.2.8 Watermarks

The BSP also includes the Watermark module, which supports a stack watermark. Watermarks are documented in the documentation of that module.

5.2.9 Porting to a new Platform

When porting the code to a new platform, only the BSP section of the code needs to be modified. However, depending on the type of port done, only some sections of the BSP may need to be ported.

If the new port is another 8051 platform with no operating system, only the HW, Startup, and Error modules have to be modified.

If the new port is a completely different platform but still does not contain an operating system, the IntBSP, Serial, SelfTest, Watermark, and memcpy51 modules must be modified.

See the documentation for the Int module. This describes some optional changes.

5.2.10 Porting to an Operating System

When porting to a system that includes an operating system, there are several things that can be done that may make the code more efficient. Of course, the new platform changes will probably have to be done as well.

The PktQ and Event modules implement functionality that is normally present in an operating system. Rewriting these two packages so that they use operating system services will provide greater functionality. For example, the PktQ package maps almost directly to the message queue library in VxWorks.

5.2.11 Porting to Another 1394 Link

The code and the API is fairly Philips specific. For example, packets are formatted in the Philips specific format rather than the 1394 format. Therefore, porting to a non-Philips link chip is not recommended. If the link chip uses the same packet format, the only packages that need to be changed are the Int, AsyHW, IsoHW, and LinkHW packages.

Porting to a new Philips link chip should not be too difficult. Unless major modifications have been made, only the LinkHW module may need to be changed.

5.3 API Overview

This section is designed to give an overview on how the modules work together.

The diagram below gives a picture of how three of the most significant API modules in the embedded program fit together. The AsyHW and the IsoHW constitute what is often termed the Hardware Abstraction Layer. The asynchronous module sits on top of the AsyHW module and provides an interface for assembling asynchronous packets, which is called the transaction layer in the 1394 specification. There is no corresponding module for the Isoch communications - the L21 hardware and the hardware attached to the AV port is responsible for assembling the packets.

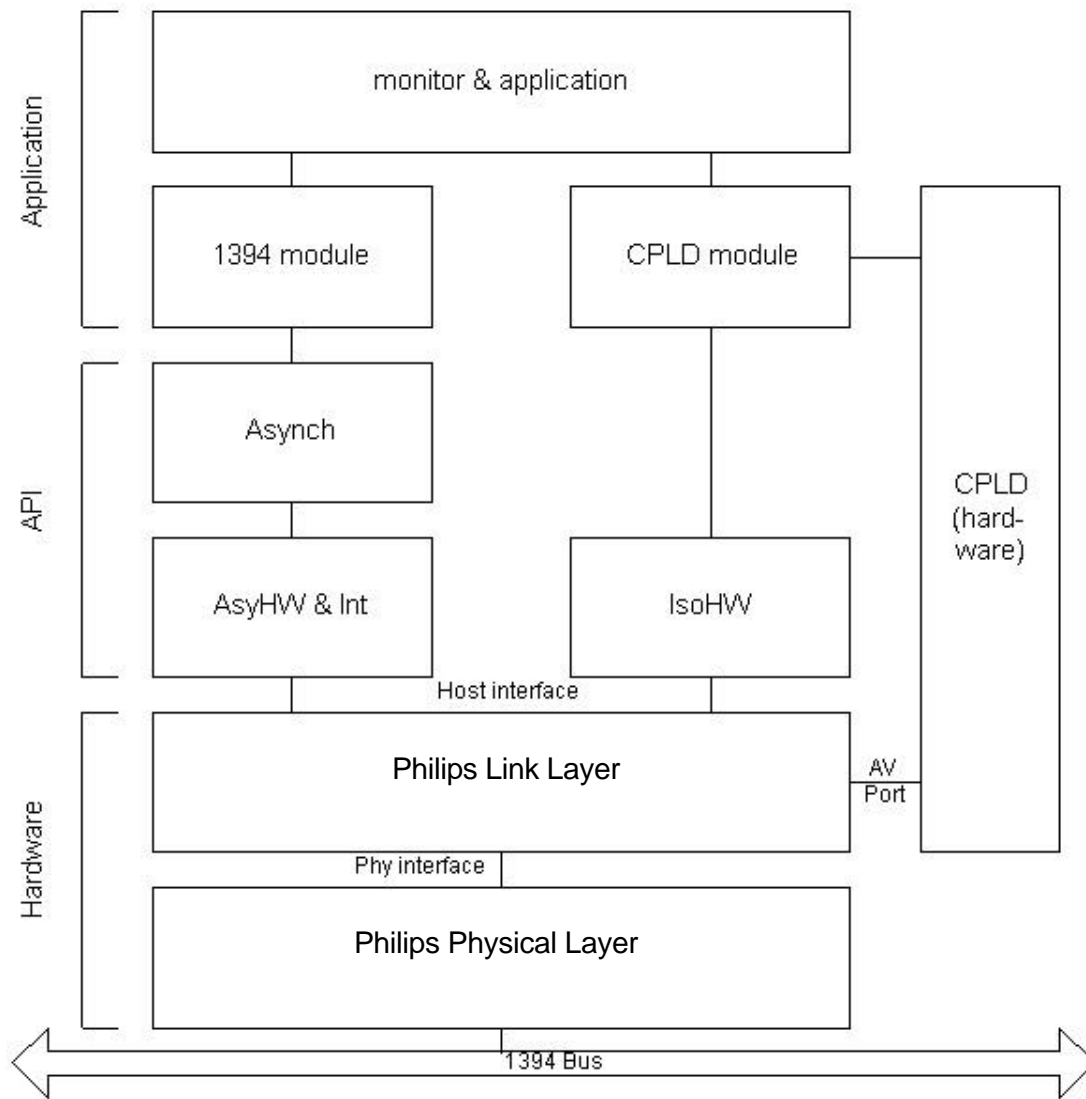


Figure 5-2: API Overview

The hardware abstraction modules do not talk directly to hardware (although they are the only modules that truly understand the structure of the module). They both use the GS module to read and write registers. The constants that they use are defined in the LinkHW module. The link's memory mapped I/O location is specified in the HW module in the BSP, along with macros that access these locations.

The Int package is also considered a hardware abstraction module. Its responsibilities are for those hardware parts that have to be serviced inside of an interrupt subroutine.

The only other API module is the Bus Reset module. This module handles the Bus Reset interrupt, collecting and verifying the self-ids. It also builds the topology map and speed map CSR's.

5.3.1 Asynch Module

By far the most important module in the API is the Asynch module, the IEEE 1394 transaction layer, which provides a non-blocking interface. There are two ways to use the non-blocking interface. The best method is to use the completionEvent parameter to define an event. The Asynch module will then throw the event when the transaction is complete.

The other way is to poll the state parameter in the request structure. When it reaches REQUEST_COMPLETE or ABORTED_ERROR, the transaction is complete.

Note: This module operates in a non-blocking manner. This means that most functions return to the calling function immediately. The request is not complete until a message is sent indicating that it is complete.

More information can be found in the API Programmer's Reference Manual.

5.3.1.1 Overview

This module is responsible for handling asynchronous transactions. It is designed to be fairly device independent, depending on the hardware abstraction of the AsyHW module. This module corresponds to the transaction layer in the IEEE-1394 specification.

This software matches requests, responses, and confirmations, provides timeouts, and handles other abnormal conditions such as bus resets. It is also capable of answering internal requests.

This layer does not determine which module is responsible for answering a request. This task is handled by the 1394 module.

5.3.1.2 AsynchRequestParams structure

Most of the functions in this module take a pointer to an AsynchRequestParams structure as a parameter. This structure forms the backbone of this module, and has the following members.

IN USER_LABEL userLabel

The user label. This may be used to identify the structure any way that is desired. This label is never used by the embedded software.

IN QUAD_XDATA *requestPacket

A pointer to the first four quadlets of the request (or to three quadlets if the request is only three quadlets long). This consists of the entire packet for all packet types except block & lock requests.

IN QUAD_XDATA *requestPacketBody

A pointer to the data block for block and lock requests. If the request is not a block or lock request, this pointer is not used.

IN QUAD_XDATA *confPacket

A pointer to a QUAD where the confirmation packet will be stored.

IN QUAD_XDATA *responsePacket

A pointer to a buffer which will hold 4 quadlets where the response packet will be stored, except in the case of a block read or response, where this pointer is only used to store the header.

IN QUAD_XDATA *responsePacketBody

A pointer to a buffer, which will hold the block, read response body data. If a block read response is not received, this pointer is not used.

IN QUAD_XDATA *packetTail

A pointer to a QUAD, which holds the Philips specific tail appended to received packets.

IN EVENT_ID completionEvent

This event will be sent when the service is complete.

IN SMALLINT generationCount

This is the generation count that corresponds with the request. If the generation count is different from the current generation count, this means that a bus reset has occurred. This means that the request is invalid, asynchronous traffic does not survive a bus reset because the node id's may have changed between generations. To read the current generation count, use `AsynchCurrentGeneration()`. `E_BUS_RESET` is returned if the generation count does not match `AsynchCurrentGeneration()` at any point during the progress of the request.

OUT enum AsynchRequestState state

Holds the progress of the service

OUT RETURN_CODE errorCode

If state == `ABORTED_ERROR`, this holds the error.

struct AsynchRequestScratch scratch

A block of memory used by the service to hold status variables and other private information.

5.3.1.3 Implementation Details

The following diagrams show the program flow for incoming and outgoing asynchronous requests. There is descriptive text describing the boxes in the API Programmer's Reference Manual.

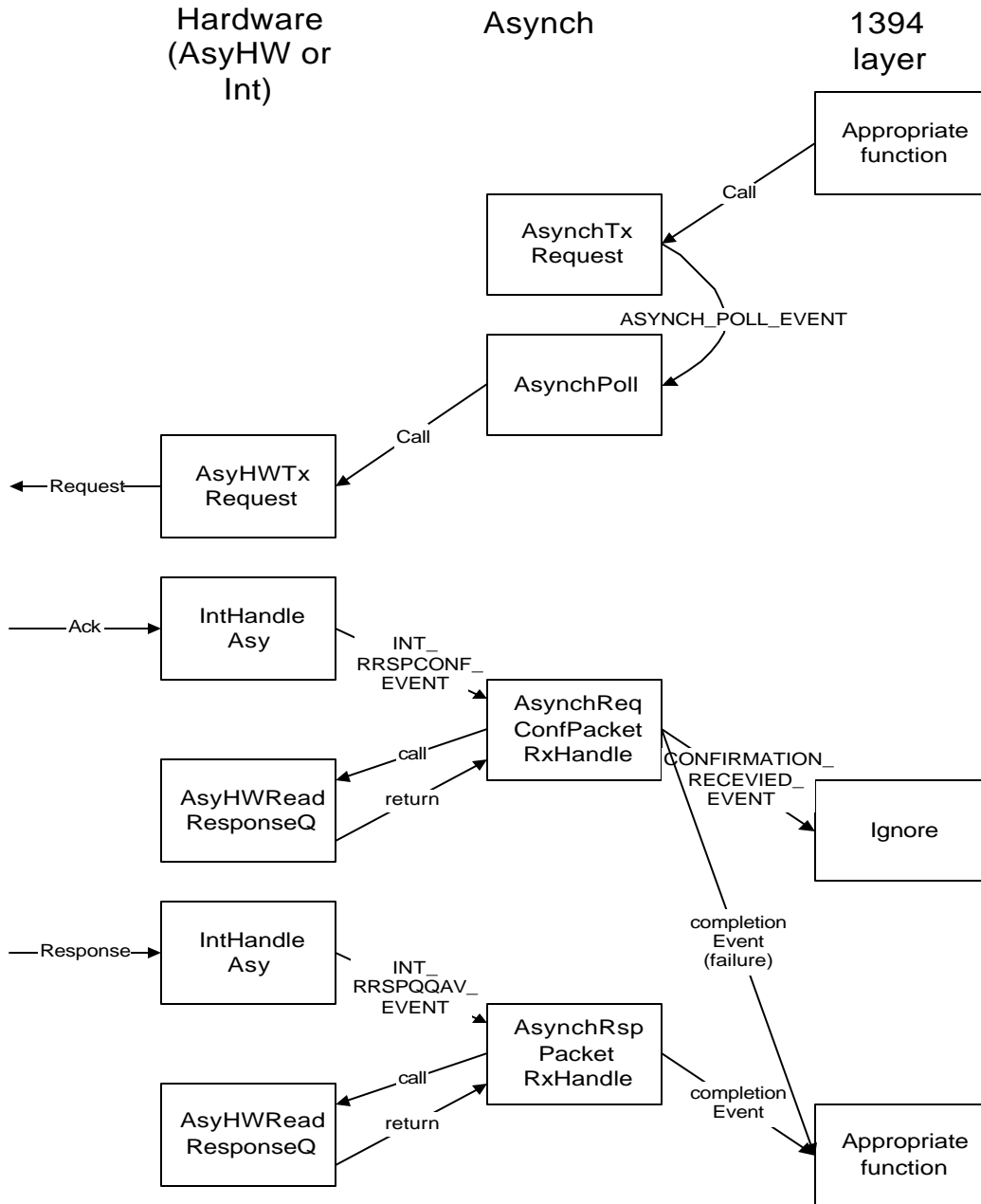


Figure 5-3: Asynchronous Outgoing Request Flow

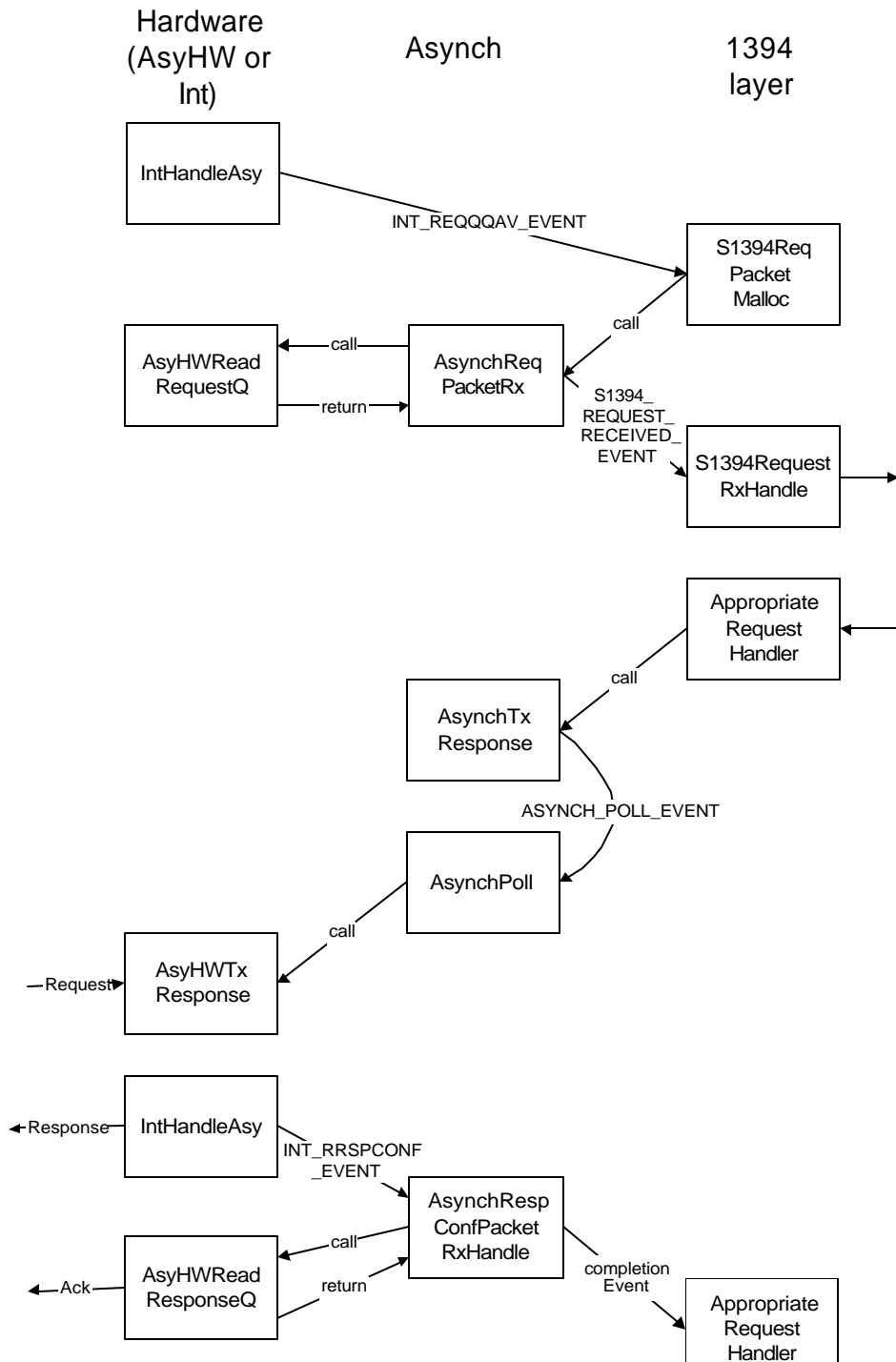


Figure 5-4: Asynchronous Incoming Request Flow

5.3.1.4 Porting to a Preemptive Multitasking System

If this transaction layer is ported to a system that contains a preemptive multitasking system, it will work as long as only one thread sends asynchronous requests. However, the software may be much more useful if more than one thread can send requests.

Access to the hardware must always be serialized, as it has no concept of threads. Therefore, this module should be run in its own, separate thread. So that more than one thread can send and receive packets, one design would be to place pointers to outgoing `AsynchRequestParams` structures in a thread safe queue. Functions like `AsynchTxRequest` may read the structure off of the queue one at a time.

A preemptive multitasking system has access to proper semaphores. This suggests that an alternative mechanism for determining when a request is complete may be used. A semaphore can be added to the `AsynchRequestParams` structure that is claimed by the `Asynch` package when it receives the request, and then released when the structure reaches the `REQUEST_COMPLETE` or `ABORTED_ERROR` state. Adding the semaphore is not required but may make using this transaction layer easier.

5.3.1.5 `AsynchTxRequest`

This function queues an asynchronous request for transmission. It is an entry point for the asynchronous request service. This service transmits requests and collects the confirmations and the responses. It matches the confirmations and responses with the request and sends messages with these values.

This function supports quadlet and block read and write requests, as well as lock requests. Broadcast writes are also supported.

This function expects a fully formed packet in the format specified in the Philips 1394 AV Link specification. The only exception is the `TLabel`. A `TLabel` is generated and inserted into the packet. If `TLabel` type functionality is desired, the `userLabel` parameter may be used for this.

This service accepts packets addressed to this node. If this is the case, the packet is handled internally. Specifically, a `REQUEST_RECIEVED_EVENT` event is thrown. This functionality may be disabled for a small memory savings with the `RESPOND_TO_OWN_REQUESTS` macro.

PHY packets may also be transmitted with this function. If this is the case, `requestPacket` should point to the Philips-specific header quadlet and `requestPacketBody` should point to the PHY quadlet. There should be a free quadlet after the PHY quadlet as the check quadlet is inserted there. PHY packets are not echoed to the internal node.

An `AsynchRequestParams` structure holds the parameters for this service. This structure is used for the duration of the service. It must not be modified until the `completionEvent` message is sent.

When the confirmation is received, a `CONFIRMATION_RECEIVED_EVENT` message is sent with a pointer to the request structure as its parameter. This message is not sent for requests addressed to this node. The event may be safely ignored.

When the service has completed, the `completionEvent` message is sent. The parameter is a pointer to the original request structure. This message is always sent, even if an error has occurred and the message was never sent. When this message is sent, the state will always be `REQUEST_COMPLETE` or `ABORTED_ERROR`.

The state parameter is a monotonically increasing value. There may be internal state values that are not externally published, but the user application may poll the state register to determine if major milestones have been passed. For example, if (state >= CONFIRMATION_RECEIVED), the application may determine that the confirmation has been received and processed without receiving the CONFIRMATION_RECEIVED_EVENT message.

5.4 Application Overview

Besides the API and the BSP, the supplied code has been lumped together into the application. However, the modules in the application section perform several distinct tasks.

5.4.1 Main

Main is responsible for initializing all of the packages and then executing the main loop. The main loop is responsible for two things: blinking the LED in a heartbeat pattern, and dispatching events.

5.4.2 Monitor Support

The Monitor package provides support for the host monitor. It implements the interface to the host monitor described in the interface documentation. The only exception is that the Error module sends the error messages to the monitor.

5.4.3 1394 Reference Implementation

The majority of the application section implements a reference 1394 application. The two modules that do this are the 1394 and the CSR packages.

The reference implementation is far from complete, yet may implement as much as is needed, depending on the application. The reference implementation is not in an API format - to change the nature of this section, the code itself has to be changed. These two packages have also been kept deliberately simple, so changing them should not be difficult.

Chapter 7, Software Compliance documents where the Full Duplex RDK is non-compliant.

5.4.4 Example Code

The Pkt1394 package has two main duties: to implement a remote command architecture, and to provide an example of how the API may be used. This package has not been widely tested.

6 Serial Interface Documentation

6.1 Overview

The serial interface between the monitor and the embedded software is a standard RS-232 port running at 9600 bps, 8N1. All passage of messages between the monitor and embedded software takes place using packets consisting of a line of ASCII text followed by a line feed ('\r' or 0x0D). The embedded software does not echo characters back.

There are four possible types of packets that may be sent over the serial port. Commands are sent from the host to the embedded software, and responses, event and errors are sent from the embedded software to the host.

If a response, event, or error is received from another node on the bus instead of the local node, the string is preceded by a nodeid string. See the Pkt1394 package for more details on this capability.

Note: All packets in commands and events are in the Philips format; please refer to the PD11394L21 data sheet for more information

6.2 Commands and Responses

Commands always follow the form `command arg0 arg1 ... argN CRC32`. The CRC32 value follows the reference implementation in section 6.4 of IEEE 1394-1995. If there are no arguments, the CRC32 value may be omitted. Only the first four characters of the command are significant.

So that the serial port is useful when a terminal program is used instead of the monitor, the CRC32 value may be replaced by a zero.

Responses are always in the format: `response resp0 resp1 ... respN CRC32`. The response portion is an ASCII string. The most common responses are: OK, CRC_bad, busy, or 'huh?'. The response data is numeric represented as an eight digit hexadecimal string. If there is no response data, the CRC32 value is omitted.

A response may be terminated by a !! glyph. This indicates that the response was prematurely terminated by an error. The error packet will follow.

All numeric responses and parameters are in hexadecimal. The embedded software always prints out eight digits for each number regardless of whether the number is 32 bits or shorter. Command parameters do not have to be eight digits long.

6.2.1 Register access

`rdreg register`

Reads the AVLink register at offset *register* and returns its value. *register* must be a quadlet aligned offset between 0 and FF.

wrreg register value

Writes the quadlet *value* to the AVLink register *register*. *register* must be a quadlet aligned offset between 0 and FF. Returns OK.

rdphy register

Reads the PHY register at offset *register* and returns its value. *register* must be an offset between 0 and F.

wrphy register value

Writes the quadlet *value* to the PHY register *register*. *register* must be an offset between 0 and F. Returns OK.

rdbir start_register number_of_registers

Reads the AV Link L40 registers at offset *start_register* with how many registers to be read *number_of_registers* and returns their values.

wrbir start_register arg0 arg1 ... argN

Writes the arguments *arg0 arg1 ... argN* to AV Link L40 registers at offset *start_register*. Returns OK.

6.2.2 Asynchronous communications

txrq quadlets

Transmits an asynchronous request packet on the 1394 bus. Returns OK. The 1394 asynchronous response is returned inside of events.

txphy quadlets

Transmits a PHY packet packet on the 1394 bus. Returns OK.

asyhold

Temporarily stops asynchronous transmission. All packets for transmission will be queued but not transmitted. Returns OK.

asygo

Releases the asynchronous hold. All packets queued for transmission will be sent. Returns OK.

6.2.3 Isochronous communications

isoinit type MAXBL DBS portdirection

This function initializes the isochronous portion of the chip. *type* selects the format

of the transmission: 1 for MPEG2, 2 for DSS, 3 for DVC, and 4 for cPLD. *MAXBL* and *DBS* select the maximum number of blocks per cycle and the data block size for the transmission. If *portdirection* is 1, *isoinit* sets port 1 to transmit and port 2 to receive. If the *portdirection* is 0, port 1 is set to receive and port 2 to transmit. Returns OK.

txgo

Starts the isochronous transmitter. Returns OK.

rxgo

Starts the isochronous receiver. Returns OK.

txstop

Stops the isochronous transmitter. Returns OK.

rxstop

Stops the isochronous receiver. Returns OK.

txchannel channel speed

Sets the isochronous transmission channel to *channel*, and the *speed* of that channel to 0=100Mbps, 1=200Mbps, 2=400Mbps. Returns OK.

rxchannel channel

Sets the isochronous receiver channel to *channel*. Returns OK.

6.2.4 Misc. 1394 functions

busreset

Starts a bus reset. Returns OK.

crquad offset quadlet

Sets the configuration ROM quadlet *#offset* to *quadlet*. Note that changes do not go into effect until *crset* is requested. As well, the *STATE_SET* CSR should be set to reject asynchronous transactions between the first call to *crquad* and the call to *crset*. Returns OK.

crset num_quadlets

Transfers all the quadlets transmitted via *crquad* into the configuration ROM CSR space. Returns OK.

6.2.5 Debug & Verification commands

hello

Returns the response 'yo'. This command is used to verify communications

ldon

Turns the LED on. Returns OK.

ldoff

Turns the LED off. Returns OK.

rdxdata *address*

Reads and returns a byte from xdata memory *address*.

wrxdata *address value*

Writes the byte *value* to xdata memory *address*. Returns OK.

version

Returns the version string.

wipememory

Erases the code area in XData in preparation for a download. Returns OK.

6.3 Events

Events are differentiated from normal responses by the characters !%, which start the event. Events always follow the format **!%EVENT *quadlets* %!**. The last quadlet of the series is a CRC32 value of the form in Section 6.4 of IEEE 1394-1995. If the ending %! is replaced with !! this indicates that not all of the quadlets of the event were transmitted.

REQRX *quadlets*

quadlets were received as an asynchronous request.

RSPRX *quadlets*

quadlets were received as an asynchronous response.

RSP *quadlets*

quadlets were synthesized as a response to a request to an internal node.

REQCONF *quadlets*

quadlet was received as a confirmation packet after a request was transmitted.

RSPCONF *quadlets*

quadlet was received as a confirmation packet after a response was transmitted.

REQTX *quadlets*

quadlets were transmitted as an asynchronous request.

RSPTX *quadlets*

quadlets were transmitted as an asynchronous response.

BUS_RESET *node# self-ids nodes*

A bus reset was detected. *node#* specifies the new node id. *self-ids* specifies how many self-id packets were received. *nodes* indicates how many nodes there are on the bus. The self-id *quadlets* are not transferred. They may be determined by reading the topology map CSR.

POWER_ON

The next four events all correspond to interrupts that were noticed by the embedded software but that the embedded software was unable to handle. The *quadlet* that is transmitted with the interrupt notification is a bit mapped register corresponding to the Interrupt Acknowledge register in the AVLink chip.

Only interrupts that the embedded software does not know how to handle are returned using these events. Many of these interrupts are acknowledged and disabled inside of the ISR in case these unknown interrupts are capable of flooding the system. Therefore, if you wish to see these interrupts in the future, you will have to re-enable them.

- LNKPHYINT *register*
- ASYINT *register*
- IRXINT *register*
- ITXINT *register*

6.4 Downloading

If a command begins with ":" it is interpreted as a line in a .HEX file for software downloading. The line is interpreted and copied into RAM. If the checksum matches, OK is returned, otherwise error is returned. Lines that return error should be resent. The last line of a .HEX file has an end of file flag. If the monitor notices this flag, the software resets the board to execute the code in RAM.

Ensure that you do not swamp the 8051 with data when downloading the file. The easiest way to do this is to wait for the OK response before sending the next line. If you get an error response, the software expects you to resend the line.

7 Software Compliance

7.1 Overview

This RDK deals with the IEEE 1394-1995 and 1394A Serial Bus standard and various other related standards. This chapter details the compliance of the RDK to these various standards, specifying what is supported and what is not.

There are several ways in which standards are not followed (Please refer to the following sections for more details):

- The software does non-compliant things that are standard practice;
- Some unsupported functionality can be flagged as supported to allow the RDK user the greatest flexibility;
- Certain functionality exists which goes beyond what is required in order to best demonstrate features of the Philips Link and PHY chipset;
- Certain functionality is marked as optional in the spec and has not been implemented; and
- Particularly in the AV/C area, certain things are outside the scope of the RDK.

7.2 Cycle Master

The embedded code should make the software fully compliant and capable of performing Cycle Master duties. The software starts cycle master activity if it is the root, which is slightly non-compliant with the IEEE –1394-1995 and 1394A specification, but follows common practice. See section 8.4.2.6 of the specification for more details

Most of the tasks required for Cycle Master activities are performed by the AV LINK without assistance from software.

7.3 Isochronous Resource Manager (IRM)

The embedded code is not fully IRM capable. The IRM CSR registers, as defined in IEEE 1394-1995, are supplied. However, the state machine, as specified in IEEE 1394-1995 8.5.2, has not been implemented. As well, it does not activate a cycle master as specified in section 8.4.2.6 of the specification.

Even though the code is not fully IRM capable, the RDK hardware can be set so that the node is advertised as IRM capable to a bus reset. This is the contender bit, and can be set by setting S6 switch 4, labeled CMC.

7.4 Bus Manager

The embedded code is not Bus Manager compliant, and does not contend for the position of Bus Manager. The code collects the Self-ID's and supplies them in the form of the

TOPOLOGY_MAP CSR. The SPEED_MAP CSR is also generated. These two CSR's are needed for a Bus Manager compliant implementation, but may be removed to save memory for non-Bus Manager applications.

The only non-obvious change that will have to be made during removal of the TOPOLOGY_MAP is the function AsynchCurrentGeneration(), which uses the TOPOLOGY_MAP generation count variable to keep track of bus resets.

7.5 ISO/IEC 61883 and AV/C

Since the RDK is a reference design kit and not an end-user application, only subsets of the Connection Management Procedures (CMP) and the Function Control Protocol (FCP) were implemented. These subsets are presented as an implementation guide and not a full implementation of the standard.

For example, the CSR Plug Control Registers (PCR) and Master Plug Registers have been supplied, as required by ISO/IEC 61883, but setting these registers does not establish connections. That is because the RDK is designed to be a multi-use reference kit that has no application identity (i.e. a camera) that requires the connection protocol.

For the FCP, a very practical example that allows the RDK to access and control a camera on the network is included. This example shows a part of the command set that can be used with an AV/C compliant device on the network.

The AV LINK, once configured, transmits fully compliant isochronous packets.

8 Customer Support

Philips Semiconductors are committed to giving you, our customer, the best possible technical support. If your system appears to be functioning incorrectly, please attempt a self-diagnosis with the help of the Troubleshooting section, Chapter 9.

If you are still having troubles, please follow these steps before contacting our technical support teams:

1. Be sure to read the relevant sections of the documentation. Many times the answer is right there.
2. Document the problem you are experiencing. Be as specific as you can. It is also useful to know the following:
 - a. The RDK board serial number;
 - b. Your operating system release version; and, if applicable
 - c. Settings used during software compilation

For assistance on installation and operation of the Full Duplex RDK, please contact:

Philips Semiconductors

1394 Applications and Marketing Group

Email: 1394@philips.com

URL: <http://www.semiconductors.philips.com/1394>

For assistance on Philips Semiconductors A/V Link Layer Controller (AV LINK) or the Physical Layer Interface (PDI1394P11), please contact:

your local Philips sales representative

or send email to: 1394@philips.com

For assistance on other Philips Semiconductor parts, please contact:

your local Philips sales representative

9 Troubleshooting

9.1 General Problems

Problem: The User LED is blinking strangely.

Solution: This may be the heartbeat, indicating that embedded software is running smoothly. If the beat is regular and seems to blink, at around 80 beats per minute, this is the heartbeat

If the beat is significantly slower and repeats in a pattern of X flashes followed by Y flashes, this indicates an embedded error. See Chapter 10: Error Codes, for more information.

9.2 Serial communications

Problem: The monitor refuses to start, complaining about being unable to communicate with the embedded code.

Solutions: In order of simplicity:

1. Verify the physical connections, close and re-start the monitor.
2. Make sure the port you are using is the correct one and is not shared with other applications. Some other applications may still have their port driver running even if they are closed correctly. In that case, you may have to reboot your computer.
3. The embedded software may be in an unusual state, ensure an EPROM with the original embedded code is in the socket and press the reset button.

9.3 Link and PHY Register Windows

Problem: The register buttons become blank when the `Read Once` button is clicked.

Solution: You have a communications problem. See section 9.2: Serial Communications and try again after waiting for the read attempt to time out.

9.4 Asynchronous Transactions

This section assumes that any of the problems discussed above are not occurring.

Problem: The monitor asks for the parameters, then nothing happens.

Solution: The problem is probably with your node. Verify that the LNKCTL register begins with C600, the ASYCTL register displays 00100320, and that the ASYINTACK register shows 00003F8C.

Problem: An embedded error `E_CONF_NO_RESPONSE` is seen.

Solution: This result occurs if you are sending your packet to a non-existent node. Make sure that the destination node is between 0 and 3E. If the problem persists, verify that the LNKCTL register of the destination node begins with C600.

Problem: The monitor asks for the parameters, then “Request split time out” appears on the screen.

Solution: Your node received an acknowledgement but no response. First make sure that the LNKCTL register begins with C600. Then verify that the ASYINTACK register is 00003F8C. If they are correct, check the same registers on the destination node and ensure that the ASYCTL register is 00100320.

9.5 Bus Resets

This section assumes that all tests in the serial communications and registers subsections have passed.

Problem: Initiating a bus reset or plugging/unplugging doesn't seem to have any effect.

Solution: Verify that the fifth digit of the LNKPHYINTE register is 2 (LNKPHYINTE is supposed to be 00042000)

9.6 Loading HEX Files

This section assumes that there are no serial communications problems.

Problem: An error message reading, “Error wiping memory in preparation for download” is seen.

Solution: The `wipememory` command failed. This can be the result of a serial communications problem or an embedded error.

Problem: An error message reading, “Too many errors at line *n* when downloading hex file” is seen.

Solution: Either there is an error in that particular line of the hex file, so it should be recompiled, or too many total errors have occurred during the download. This may happen with imperfect communications, and with a very large hex file.

The only solution if the hex file is correct is to try again, or improve the quality of the serial link.

10 Error Codes

10.1 Overview

There are two types of errors that can be brought up by the embedded software: fatal errors and non-fatal errors.

The following section lists the errors and associated blink codes. For example, a blink code of (4,5) would be shown as four blinks, a short pause, five blinks, and then a long pause before the sequence repeats. These blink codes will replace the normal heartbeat pattern on the User LED in the event of a fatal error.

Although non-fatal errors have blink codes associated with them, they are handled by the monitor software and are not displayed on the User LED.

The fatal errors are:

- E_CABLE_LOOP
- E_TOO_MANY_BUS_RESETS
- E_EVENT_Q_FULL
- E_PHY_TIMEOUT
- E_LINKPHY_FATAL_INTERRUPT
- E_ASY_FATAL_INTERRUPT
- IRX_FATAL_INTERRUPT
- ITX_FATAL_INTERRUPT
- E_RAM_CHECK
- E_PKTQ_FULL
- E_PKTQ_EMPTY

***Note:** The DONE return code specifies that the function completed without errors. It also indicates that the service has completed if it is found in `AsynchRequestParams.errorCode`. Compare with the PENDING return code.*

10.1.1 PENDING (blink code 1,2)

This return code indicates that the request has been queued and will be completed at a subsequent time.

Returned by:

- AsynchTxRequest
- AsynchTxResponse

- AsynchRspPacketRxHandle
- AsynchPoll

10.1.2 E_ERROR (blink code 1,3)

This error code is never returned. It is a placeholder indicating the minimum integral value for RETURN_CODE that indicates an error. In other words, all values less than this code are normal return codes; all greater than it are error codes.

10.1.3 E_REQ_MALLOC_TIMEOUT (blink code 1,4)

This error indicates that the embedded software was unable to allocate memory for a request after a large number of retries. This indicates that the previous request did not time out correctly.

10.1.4 E_ASYTXREQ (blink code 1,5)

This error code indicates that a hardware error occurred during the transmission of an asynchronous request packet.

Returned by:

- AsyHWTxRequest
- AsynchPoll

10.1.5 E_ASYTXRESP (blink code 1,6)

This error code indicates that a hardware error occurred during the transmission of an asynchronous response packet.

Returned by:

- AsyHWTxResponse

10.1.6 E_NO_QUADLETS_AVAILABLE (blink code 1,7)

This error code indicates that the number of quadlets requested is not in the queue.

Returned by:

- AsyHWReadReqQ

10.1.7 E_ROUTER_TABLE_FULL (blink code 1,8)

This error code indicates that the asynchronous request transmission service was unable to obtain a router table entry for the TLABEL.

Returned by:

- AsynchTxRequest
- AsynchTxResponse

10.1.8 E_LOOP_DETECTED (blink code 2,1)

This error code indicates that the code detected a loop in an internal linked list. Most likely, a pointer to a parameter structure was passed to a function when the parameter structure was still in use.

Returned by:

- AsynchTxRequest
- AsynchTxResponse

10.1.9 E_BUS_RESET (blink code 2,2)

Not an error per se, but a code that is returned when a request is unable to be completed because a bus reset has interrupted it. Also, note that the request structure may be corrupted.

In the case of a request, this indicates that the request may or may not have been transmitted, and that the node receiving the request may or may not have acted on the request.

In the case of a response, this indicates that the response may or may not have been transmitted.

In the case of an outgoing request, this error must be handled carefully. In some cases it is enough to retransmit the request. In others, it may be desirable to read some state information on the target to determine if the request was or was not processed. In a very few cases this error may be ignored.

Returned by:

- AsyHWTxRequest
- AsyHWTxResponse
- AsyHWReadReqQ
- AsynchTxRequest
- AsynchTxResponse
- AsynchRspPacketRxHandle
- AsynchReqConfPacketRxHandle
- AsynchRespConfPacketRxHandle
- AsynchReqPacketRx
- AsynchPoll

10.1.10 E_RESP_MANGLED (blink code 2,3)

This error code indicates that somehow the asynchronous response packet received was corrupted or an unexpected confirmation arrived.

Returned by:

- AsynchRspPacketRxHandle

10.1.11 E_CONF_MANGLED (blink code 2,4)

This error code indicates that somehow the confirmation packet was corrupted or an unexpected confirmation arrived.

Returned by:

- AsynchReqConfPacketRxHandle
- AsynchRespConfPacketRxHandle

10.1.12 E_CONF_NO_RESPONSE (blink code 2,5)

This error code indicates that the confirmation received did not indicate acknowledge pending. There will not be a response. See confPacket for additional details.

Note: The most common source of this error is that the destination node does not exist.

Returned by:

- AsynchReqConfPacketRxHandle

10.1.13 E_CONF_ERROR (blink code 2,6)

This error code indicates that the confirmation received did not indicate acknowledge complete.

Returned by:

- AsynchRespConfPacketRxHandle

10.1.14 E_CONF_TIMEOUT (blink code 2,7)

This error code indicates that confirmation was not received within the timeout period.

10.1.15 E_REQ_MANGLED (blink code 2,8)

This error code indicates that a corrupted request was received.

Returned by:

- AsynchReqPacketRx

10.1.16 E_PACKET_MANGLED (blink code 3,1)

An error that is returned when an AsynchRequestParams structure is in an unexpected state.

Returned by:

- AsynchTxRequest
- AsynchTxResponse
- AsynchPoll

10.1.17 E_UNSUPPORTED_TCODE (blink code 3,2)

This error code indicates that either you attempted to use the wrong function to look at a packet, or the packet was corrupted.

Returned by:

- AsynchPoll

10.1.18 E_TIMEOUT (blink code 3,3)

This error code indicates that no response was received to an asynchronous request within a BUSY_TIMEOUT interval.

10.1.19 E_BR_NO_IDVALID (blink code 3,4)

This error code indicates that the bus reset software timed out waiting for the IDVALID bit to be set after a bus reset. This error often indicates a hardware problem..

Returned by:

- BREventHandle

10.1.20 E_BR_NO_HEADER (blink code 3,5)

Received a self-id packet missing the 0x000000E0 header quadlet.

Returned by:

- BREventHandle

10.1.21 E_BR_MULTIPLE_HEADERS (blink code 3,6)

Self-id includes multiple header quadlets i.e. multiple 0x000000E0 quadlets.

Returned by:

- BREventHandle

10.1.22 E_BR_SIDQAV_NOT_SEEN (blink code 3,7)

Bit SIDQAV was not set by the Link upon the reception of a self-id packet.

Returned by:

- BREventHandle

10.1.23 E_BR_PACKET_TIMEOUT (blink code 3,8)

Received a truncated self-id packet. This could be the result of a bus reset in the middle of the self-id phase.

Returned by:

- BREventHandle

10.1.24 E_BR_ACK_DATA_ERROR (blink code 4,1)

This error code indicates that the link terminated the self-id sequence with "ACK_DATA_ERROR".

Returned by:

- BREventHandle

10.1.25 E_BR_INVALID_PACKET (blink code 4,2)

This error code indicates that an invalid packet was seen in the self-id sequence.

Returned by:

- BREventHandle

10.1.26 E_INVALID_TOPO_MAP (blink code 4,3)

This error code indicates that BRBusTree tried to build a bus tree from an invalid topology map.

Returned by:

- BRBusTree

10.1.27 E_INVALID_NODE_TREE (blink code 4,4)

This error code indicates that the structure nodeTree (generated by BRBusTree) was incorrectly built.

Returned by:

- BRBusTree

10.1.28 E_CABLE_LOOP (blink code 4,5)

This error code indicates that the bus topology is not a tree. The 1394 cables are connected in a loop.

Returned by:

- BRBusResetHappened

10.1.29 E_BR_CYCLE_START (blink code 4,6)

This error code indicates that a cycle start packet was inserted into the self-id data.

Returned by:

- BREventHandle

10.1.30 E_TOO_MANY_BUS_RESETS (blink code 4,7)

This error code indicates that too many bus resets with errors were noticed.

10.1.31 E_INTERNAL (blink code 4,8)

Internal error.

10.1.32 E_EVENT_Q_FULL (blink code 5,1)

This error code indicates that the Event Q has overflowed. The size may be adjusted inside the PktQ module. However, the most likely cause is either that the system is hung or overburdened and is no longer dispatching events, or that something has started an infinite loop creating events.

10.1.33 E_INVALID_EVENT_ID (blink code 5,2)

This error code indicates that somehow, the event queue was corrupted.

Returned by:

- EventHandle

10.1.34 E_PHY_TIMEOUT (blink code 5,3)

This error code indicates that during a read or write to the PHY, the procedure timed out. If this was not an error, change PHY_TIMEOUT.

Returned by:

- GSReadPhyReg
- GSWritePhyReg

10.1.35 E_LINKPHY_FATAL_INTERRUPT (blink code 5,4)

This error code indicates that a fatal LINKPHY interrupt occurred, or an interrupt that the API uses was mistakenly enabled.

Returned by:

- HandleLinkPhyInt

10.1.36 E_ASY_FATAL_INTERRUPT (blink code 5,5)

A fatal ASY interrupt occurred, or an interrupt that the API uses was mistakenly enabled.

Returned by:

- IntHandleAsy

10.1.37 IRX_FATAL_INTERRUPT (blink code 5,6)

This error code indicates that a fatal IRX interrupt occurred, or an interrupt that the API uses was mistakenly enabled.

10.1.38 ITX_FATAL_INTERRUPT (blink code 5,7)

This error code indicates that a fatal ITX interrupt occurred, or an interrupt that the API uses was mistakenly enabled.

10.1.39 E_RAM_CHECK (blink code 5,8)

This error code indicates that the RAM failed the initial start up diagnostics.

10.1.40 E_ROM_CHECK (blink code 6,1)

This error code indicates that the ROM failed the initial start up diagnostics. If you are running from RAM then ignore this error.

10.1.41 E_TEST_FAIL (blink code 6,2)

This error code indicates that one of the manufacturing tests failed. Check the serial port output for more information.

10.1.42 E_PKTQ_FULL (blink code 6,3)

This error is probably in user code. The RDK code generally calls PktQFull() before writing to the queue so that a more specific error code may be returned.

Returned by:

- PktQWrite

10.1.43 E_PKTQ_EMPTY (blink code 6,4)

This error code indicates that one of the packet queues was empty but somebody tried to get a packet anyway. Call PktQEmpty() before attempting to get a packet.

Returned by:

- PktQRead

10.1.44 E_SERIAL_TX_Q_FULL (blink code 6,5)

This error code indicates that the serial port's transmit queue is full.

Returned by:

- SerialPutChar

10.1.45 E_SERIAL_RX_OVERFLOW (blink code 6,6)

This error code indicates that the serial port's Rx buffer has overflowed. There is no form of flow control in the serial package. However, if you wait for a response each line, you will not have any troubles.

11 Glossary

The various acronyms, abbreviations, and special terms used frequently in this manual are here defined for convenient reference.

1394	A fast external serial bus standard originally developed by Apple under the name FireWire. Other names for this technology include I-link, and Lynx.
80C51	A standard, low cost microprocessor
ActiveX	A set of rules defined by Microsoft for how applications should share information.
API	Application Program Interface. A set of routines, protocols, and tools for building software applications.
ASCII	American Standard Code for Information Interchange
AV	Audio/Video
AV/C	Audio/Video Command set
Bpm	Beats Per Minute
Bps	Bits Per Second
Block	A series of sequential 8-bit bytes, with the number of bytes set as part of the transaction.
Breakout	An expansion of a register into its constituent fields.
BSP	Board Support Package
C51 Processor	See 80C51
CMOS	Complimentary Metal Oxide Semiconductor
Command and Status Registers	Software registers known as Command and Status Registers. A CSR occupies 32 bits, that is, one quadlet.
CPLD	Complex Programmable Logic Device
CSR	Command and Status Register
DSS	Digital Satellite System
DVC	Digital Video Camera
DVD	Digital Versatile Disc
Enables	Bits set to a value in order to enable a function.
EPROM	Erasable Programmable Read-Only Memory
GUI	Graphical User Interface
IEC	International Electrotechnical Committee
IEEE	Institute of Electrical and Electronic Engineers
IRM	Isochronous Resource Manager
ISO	International Standards Organization

Isochronous	The constant-bandwidth capability of 1394 bus architecture to transmit continuous data streams, particularly the transport of digital audio and video signals. A 1394 node can be configured to transmit or receive one isochronous data stream at a time. Once the node has been configured with the various isochronous parameters, no further intervention is required. The process of transmission or reception of the data stream is handled by the PDI1394 chipset.
LED	Light Emitting Diode
MB/s	Megabytes per second
MPEG-2	(MPEG stands for Motion Picture Experts Group). A high compression isochronous format, commonly seen in satellite TV, video cameras, and DVD players.
PDF	Portable Document Format
PDI1394L21	Identifier of the link chip which handles transactions between 1394 boards. The PDI1394L21 Philips Semiconductors 1394 Audio/Video Link Layer Controller is an IEEE 1394-1995 compliant link layer controller featuring an embedded AV layer interface. The AV layer is designed to pack and unpack application data packets for transmission over an IEEE 1394 bus using isochronous data transfers. It runs at 49.978 MHz and uses 3.3 V. For detailed information refer to the <i>Hardware Reference</i> and the <i>PDI1394L21 datasheet</i> .
PHY	Registers of the PDI1394P11 3-port physical layer interface.
Port Direction	The direction setting of the PDI1394L21.
Quadlet	A length of 4 bytes, 32-bit quantities.
RDK	Reference Design Kit
Registers	A high speed data storage area within a CPU.
RO	Read-Only
ROM	Read-Only Memory
RS-232	A standard interface for connecting serial devices.
VCR	Video Cassette Recorder
Win32	The Microsoft Windows API for developing 32-bit applications.
Win9x	The Microsoft Windows 95 or 98 operating system.

For more information please contact:

Philips Semiconductors

Email: 1394@philips.com

URL: <http://www.semiconductors.philips.com/1394>