



ACE4033CST
Set-Top Box Reference Design Kit
User's Manual

73G7335 000001

Third Edition (September 1997)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS DOCUMENT "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you. IBM does not warrant that the use of the information herein shall be free from third party intellectual property claims.

IBM does not warrant that the contents of this document will meet your requirements or that the document is error-free. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the document. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this document at any time. This document does not imply a commitment by IBM to supply or make generally available the product(s) described herein.

The products described in this document are not intended for use in implantation or other direct life support applications.

All performance data contained in this document was obtained in a specific environment, and is presented as an illustration. The results obtained in other operating environments may vary.

No part of this document may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the written permission of IBM.

Address comments about this document to:

IBM Corporation
Department YM5A
PO Box 12195
Research Triangle Park, NC 27709
ppc400pubs@vnet.ibm.com

IBM may have patents or pending patent applications covering the subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, 500 Columbus Avenue, Thornwood, NY 10594, USA.

©Copyright International Business Machines Corporation 1997. All rights reserved.

Printed in the United States of America.

The following terms are trademarks of IBM Corporation:

IBM
PowerPC
PowerPC 403
PowerPC Architecture
RISCWatch
RISCTrace

Other terms which are trademarks are the property of their respective owners.

Contents

About This Book	xiii
Who Should Use This Book.....	xiii
How This Book is Organized.....	xiii
Numeric Conventions.....	xiv
Highlighting Conventions	xiv
Syntax Diagram Conventions	xv
Contacting the IBM Embedded Systems Solution Center	xvi
Related Publications	xvi
1. Introducing the Set-Top Box (STB) Reference Design Kit	1-1
STB Reference Design Kit Overview	1-1
Advanced STBRP Features.....	1-1
STB Reference Design Kit Hardware Components	1-3
STB Reference Design Kit Software Components	1-3
Cables and Supplies	1-4
User-supplied Components	1-4
2. Host PC Requirements	2-1
Host PC Hardware Requirements.....	2-1
Host PC Software Requirements	2-1
Host PC Data Communications and Networking Requirements.....	2-1
3. Installing Software on the Host PC	3-1
Installation Requirements	3-1
Installing Device Drivers	3-1
Installing a BSP.....	3-1
4. Configuring the Host PC	4-1
Using a Terminal Emulator	4-1
Using HyperTerminal (Windows 95)	4-1
Using Windows Terminal (Windows 3.n and Windows NT).....	4-2
Configuring TCP/IP for STB Development.....	4-2
Configuring TCP/IP for Windows 95	4-3
Configuring TCP/IP for Windows 3.n	4-3
Configuring Trumpet Winsock.....	4-4

Configuring TCP/IP for Windows NT 3.51	4-5
Configuring bootp and tftp to Support ROM Monitor Loads	4-6
Configuring bootp and tftp on the Host PC	4-6
Automatic bootpd and tftpd Startup for Windows 3.n and Windows NT 3.51	4-7
Automatic bootpd and tftpd Startup for Windows 95	4-8
5. Assembling the STB Reference Design Kit Hardware	5-1
STB Reference Design Kit Hardware Components	5-1
Assembling the STB Reference Design Kit Components	5-1
Connecting the DVBE Buffer Card to the DVBE	5-1
Connecting the Ethernet Daughterboard to the STBRP	5-2
Connecting the Transport Stream Cable	5-2
6. Connecting the STB Reference Design Kit Hardware	6-1
Connecting the STBRP and DVBE Serial Ports to the Host PC	6-1
Connecting the STBRP, DVBE, and Host PC to an Ethernet Network	6-1
Connecting the STBRP A/V Outputs to the TV	6-1
Connecting the STBRP and DVBE to Power	6-2
7. Hardware Components	7-1
STBRP Components	7-1
PPC403 Embedded Controller	7-1
Memory	7-4
System Control Logic	7-5
STBP	7-5
Buffering	7-8
Infrared (IR) Transceiver	7-9
DVB Transport Chip	7-10
DVB Descrambler	7-11
MPEG Audio/Video Decoder	7-11
DENC	7-12
Audio DAC	7-12
Ethernet Daughterboard	7-12
Network Interfaces	7-14
Indicator Lamps	7-14
Switch Matrix Keypad	7-15
Headers and Jumpers	7-16

8. STBRP Connectors, Switches, and Jumpers	8-1
Serial Port Connectors.....	8-3
IEEE 1284 Port Connector.....	8-3
SCART Connector	8-4
Composite Video, Left and Right Audio Connector	8-5
S-Video Connector.....	8-6
Expansion Connector.....	8-6
Expansion Connector Signals	8-6
DVB Transport Stream Connector.....	8-10
DVB Transport Stream Connector Signals	8-10
RISCWatch and RISCTrace Connectors	8-13
Auxiliary Power Connector.....	8-15
I ² C Expansion Connector (J8)	8-16
STBRP Test Headers	8-16
Setting the STBRP Jumpers	8-23
Indicator Lamps	8-24
Switches.....	8-24
9. STBRP Board Clock Distribution.....	9-1
10. Using the ROM Monitor	10-1
Accessing the ROM Monitor	10-1
Monitor Selections and Submenus	10-1
ROM Monitor Main Menu	10-1
Selecting Power-On Tests	10-2
Selecting a Boot Device	10-4
Changing IP Addresses	10-5
Using the Ping Test.....	10-7
Disabling and Enabling the Automatic Menu Display.....	10-8
Displaying the Current Configuration	10-8
Saving the Current Configuration.....	10-9
Setting the Baud Rate for Boots over SP1	10-10
Performing a Boot over SP1	10-11
Loading the Demo Programs	10-13
Booting from the pSOS/pROBE Boot Image.....	10-15
Exiting the Main Menu.....	10-16

11. pSOS Platform Support Package.....	11-1
Integrating the pSOS PSP	11-1
Loading the Boot Code	11-1
Modifying the Base pSOS Installation	11-2
Debugging the STBRP Application using SingleStep.....	11-2
The pSOS Device Drivers and System Files.....	11-4
Flash Memory Device Driver	11-4
Transport Demultiplexer Device Driver.....	11-7
MPEG Audio/Video Decoder Device Driver.....	11-10
Digital Video Encoder (DENC) Device Driver	11-12
I ² C Bus Device Driver.....	11-14
EEPROM Device Driver.....	11-17
Ethernet Device Driver.....	11-19
Infrared (IR) Receiver Device Driver.....	11-21
Keypad Device Driver	11-23
Serial Port Device Driver	11-24
IEEE 1284 Port Device Driver	11-25
Smart Card Device Driver.....	11-27
STB Peripheral Chip (STBP) Functions.....	11-29
Server Tasks	11-29
Root Task	11-29
A/V Task	11-30
Debug Task	11-30
Transport Task.....	11-31
Control Task	11-31
Database Task.....	11-32
IR Application Task.....	11-33
User Interface Task	11-33
Conditional Access Support	11-34
Demo Program Source Code	11-34
Diagnostics.....	11-34
Level 1 Diagnostics.....	11-34
Level 2 and Level 3 Diagnostics	11-35
The UI Task	11-36
Additional Diagnostics	11-36
Miscellaneous File and Directory Descriptions.....	11-36

12. OS-9000 Platform Support Package	12-1
Integrating the OS-9000 PSP	12-1
Loading the Boot Code	12-1
Using the OS-9000 PSP Version of FasTrak	12-2
Modifying the Base OS-9000 Installation	12-2
Modifying OS-9000 System Files	12-3
Problems with Base OS-9000 Software	12-4
Using the OS-9000 PSP DAVIDLite Implementation	12-4
The OS-9000 PSP Device Drivers and System Files	12-4
Flash Memory Device Driver	12-4
Transport Demultiplexer Device Driver	12-7
MPFM (MPEG Decoder) Device Driver	12-12
MFM (Graphics) Device Driver	12-15
DENC Device Driver	12-17
I ² C Bus Device Driver	12-18
EEPROM Device Driver	12-20
Ethernet Device Driver	12-22
Infrared (IR) Remote Control Device Driver	12-23
Keypad Device Driver	12-25
Serial Port Device Driver	12-27
IEEE 1284 Port Device Driver	12-29
Smart Card Device Driver	12-30
STBP Chip Functions	12-32
Conditional Access Support	12-33
Demo Program Source Code	12-34
Miscellaneous Files and Directories	12-34
13. Using the Demo Program	13-1
Starting the OS-9000 Demo Program	13-1
Starting the pSOS Demonstration Program	13-2
Using the Demo Program GUI	13-2
The Main Menu	13-2
The OSD Demo Menu	13-3
The MPEG Player Menu	13-4
Using the VT 100 User Interface	13-6
Appendix A. Programmable Logic Equations	A-9
ispGAL22V10C Equations	A-9

ispLSI 1024-90LJ Equations.....	A-10
Appendix B. Bills of Materials	B-1
STBRP Bill of Materials	B-1
Ethernet Daughterboard Bill of Materials.....	B-7
Appendix C. Acronyms and Abbreviations	C-1
Index	X-1

Figures

Figure 1-1. IBM Set-Top-Box Reference Platform Block Diagram	1-2
Figure 7-1. Smart Card Connection Diagram	7-7
Figure 7-2. Data Buffering Diagram	7-8
Figure 7-3. Address Buffering Diagram	7-9
Figure 7-4. Ethernet Daughterboard Layout	7-13
Figure 8-1. STBRP Board Layout	8-2
Figure 8-2. Serial Port Connectors (P2, P3)	8-3
Figure 8-3. IEEE 1284 Port Connector (P1)	8-3
Figure 8-4. SCART Connector (J2)	8-5
Figure 8-5. S-Video Connector (J1)	8-6
Figure 8-6. Expansion Connector (J20)	8-6
Figure 8-7. DVB Transport Stream Front-End Connector (J18)	8-10
Figure 8-8. JTAG Header (J35) (Top View)	8-13
Figure 8-9. RISCTrace Connector (J36) (Top View)	8-14
Figure 8-10. Power Connector (J32)	8-15
Figure 8-11. IIC Expansion Connector (J8)	8-16
Figure 8-12. 20-Pin Test Header	8-16
Figure 9-1. Clocking Diagram	9-1
Figure 13-1. Main Demo Program Menu	13-2
Figure 13-2. OSD Demo Menu	13-3
Figure 13-3. MPEG Player Menu	13-4
Figure 13-4. Program Entry Menu	13-5
Figure 13-5. List/Select Programs Menu	13-6

Tables

Table 7-1. Interrupt Distribution	7-2
Table 7-2. PC403 Bank Registers BR0–BR6	7-3
Table 7-3. PPC403 Bank Register Settings to Control Output Signals CS0–CS6	7-3
Table 7-4. PPC403 BR7 Settings to Control Output Signal CS7	7-4
Table 7-5. STBP GPIO Pins	7-6
Table 7-6. Indicator Lamps D10–D13	7-14
Table 7-7. Switch Matrix Configuration	7-15
Table 7-8. Row-Column Control Bits on the Data Bus	7-15
Table 7-9. Switch Matrix Scan Decoding	7-16
Table 7-10. Switch Matrix Scan	7-16
Table 7-11. J1 Header Signals	7-17
Table 7-12. J6 Test Header Signals	7-17
Table 7-13. J7 Test Header Signals	7-18
Table 7-14. J5 Jumper Setting	7-19
Table 8-1. Serial Port Connector (P2, P3) Pins	8-3
Table 8-2. IEEE 1284 Port Connector (P1) Pins	8-3
Table 8-3. SCART Connector (J2) Pins	8-5
Table 8-4. Expansion Connector (J20) Pin Assignments	8-6
Table 8-5. DVB Transport Stream Connector Signals	8-10
Table 8-6. JTAG Interface (J35) Connections and Resistors	8-13
Table 8-7. RISCTrace Connector Pins(J36)	8-15
Table 8-8. I ² C Expansion Connector (J8) Pins	8-16
Table 8-9. CD21 DRAM Data Bus Test Header (J7)	8-16
Table 8-10. YUV Data Bus Test Header (J9)	8-17
Table 8-11. System Address Bus Test Header (J10)	8-17
Table 8-12. CD21 DRAM Data Bus Test Header (J12)	8-18
Table 8-13. System Data Bus (D0:15) Test Header (J13)	8-18
Table 8-14. CD21 Control Test Header (J14)	8-19
Table 8-15. CD21 DRAM Data Bus Test Header (J15)	8-19
Table 8-16. Transport DRAM Data Bus Test Header (J17)	8-19
Table 8-17. CD21 DRAM Data Bus Test Header (J19)	8-20
Table 8-18. PPC403 Control Signals Test Header (J22)	8-20
Table 8-19. Transport DRAM Signals Test Header (J23)	8-21
Table 8-20. PPC403 Controls/CD21 DRAM Address Bus Test Header (J27)	8-21

Table 8-21. Ready and Interrupt Signals Test Header (J29)	8-22
Table 8-22. System Address Bus (and Others) Test Header (J30)	8-22
Table 8-23. System Data Bus (D16:31) Test Header (J31)	8-23
Table 8-24. Jumper Settings	8-23
Table 8-25. Indicator Lamps	8-24

About This Book

This book describes how to install and use the IBM ACEACE4033CST Set-Top Box (STB) Reference Design Kit. The STB Reference Design Kit provides the hardware and software needed to use the STB Reference Platform (STBRP), an evaluation board that handles digital video broadcast (DVB) transport streams. The STBRP enables designers to evaluate and develop applications for the set-top box market.

Who Should Use This Book

This book is for hardware and software developers who need to evaluate the STBRP to support software development.

Users should understand hardware and software development concepts, tools, and environments. Specifically, users should understand:

- Television set-top box concepts
- The host operating system
- The PowerPC Architecture™ and its implementation in PowerPC 403 (PPC403) controllers
- C and PowerPC assembler language programming

How This Book is Organized

This book contains the following chapters:

- Chapter 1, “Introducing the Set-Top Box (STB) Reference Design Kit,” describes the product and its hardware and software components.
- Chapter 2, “Host PC Requirements,” lists the hardware and software requirements of the host system.
- Chapter 3, “Installing Software on the Host PC,” describes how to install the host software on the host system.
- Chapter 4, “Configuring the Host PC,” describes the steps required to facilitate communications between the host computer and the STB Reference Design Kit.
- Chapter 5, “Assembling the STB Reference Design Kit Hardware,” describes how to assemble the hardware.
- Chapter 6, “Connecting the STB Reference Design Kit Hardware,” describes how to connect the STBRP to the host PC and to a TV.
- Chapter 7, “Hardware Components,” describes the STBRP hardware components and their functions in terms of the overall organization of the STB Reference Design Kit

- Chapter 8, “STBRP Connectors, Switches, and Jumpers,” contains hardware descriptions of the STBRP connectors, switches, and jumpers.
- Chapter 9, “STBRP Board Clock Distribution,” describes transport stream clock distribution on the PPC403.
- Chapter 10, “Using the ROM Monitor,” describes the operations of the ROM Monitor, which runs on the STBRP.
- Chapter 11, “pSOS Platform Support Package,” describes the pSOS Platform Support Package (PSP).
- Chapter 12, “OS-9000 Platform Support Package,” describes the OS-9000 PSP.
- Chapter 13, “Using the Demo Program,” describes how to run and use the demo program provided with the STB Reference Design Kit

This book contains the following appendices:

- Appendix A, “Programmable Logic Equations,” provides the equations for programmable logic devices on the STBRP and Ethernet daughterboard..
- Appendix B, “Bills of Materials,” presents the bill of materials for the STBRP and the Ethernet daughterboard.
- Appendix C, “Acronyms and Abbreviations,” lists acronyms and abbreviations used in this book.

An index follows Appendix C.

Numeric Conventions

In general, numbers are used exactly as shown. Unless noted otherwise, all numbers are in decimal, and, if entered as part of a command, are entered without base prefixes.

Binary numbers are preceded by “0b,” for example: 0b010

Hexadecimal numbers are preceded by “0x,” for example: 0x1A7

In text, the hexadecimal digits A through F appear in uppercase. In commands, these digits are typically entered in lowercase.

Highlighting Conventions

This book uses the following highlighting conventions:





- The names of invariant objects known to software appear in bold type. In some text, however, such as in lists, no special typographic treatment is used. Examples of such objects include:
 - Function and macro names
 - Data types and structures
 - Constants and flags

Names of objects known to the software must be entered exactly as shown.

- Variable names supplied by user programs appear in italic type. In some text, however, such as in lists, no special typographic treatment is used. Examples of these objects include arguments and other parameters.
- No highlighting appears in code examples.

Syntax Diagram Conventions

Throughout this book, diagrams illustrate the syntax for string formats and commands. The following list shows how to read these diagrams:










- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.
- A  symbol begins a diagram.
- A  symbol indicates continuation of a diagram on the next line.
- A  symbol indicates continuation of a diagram from the previous line.
- A  symbol terminates a diagram.
- Keywords are in regular type, and variables are in italics. Keywords must be typed exactly as shown.
- Keywords or variables on the main path of a diagram are required.

 keyword — *variable1* — *variable2* 

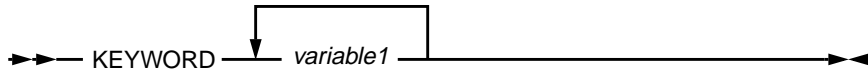
- Keywords or variables shown on branches below the main path are optional.

 keyword —  *variable1*   *variable2*  

- Keywords or variables can appear in a stack, indicating that only one item in a stack can be chosen. If an item in a stack is on the main path, you must choose an item from the stack. If all items in a stack are below the main path, you may choose an item from the stack.
- For example, in the following syntax diagram, you must choose either *variable1* or *variable2*. However, because *variable3* and *variable4* are below the main path, neither is required.

 KEYWORD —  *variable1*   *variable2*   *variable3*  *variable4*  

- A repeat separator is a returning arrow that surrounds a syntax element or group and shows that the element or group can be repeated.



Contacting the IBM Embedded Systems Solution Center

For information about the STBRP Kit and the IBM family of hardware and software products for embedded system developers, call the IBM Embedded Systems Solution Center at (919) 254-1810.

Please send any comments regarding this publication to the following Internet address:

ppc400pubs@vnet.ibm.com

Related Publications

This book refers to the following publications, which are available from your IBM Microelectronics representative:

- *PPPPC403GC Embedded Controller User's Manual* (order number 13H6986)

This book refers to the following standards, which are available from the sponsoring standards organizations.

- *IEEE 1284 - 1994, Standard Signaling Method for a Bidirectional Parallel Interface for Personal Computer (ANSI)* (referred to as IEEE 1284)
- *IEEE 802.3 - 1996, LAN/MAN CSMA/CD Access Method and Physical Layer Specification* (referred to as IEEE 802.3)
- *ISO/IES DIS 7816-3, Information Technology - Identification Cards - Integrated Circuit(s) Cards With Contacts - Part 3 Electronic Signals and Transmission Protocols* (referred to as ISO/IEC 7816)

Chapter 1. Introducing the Set-Top Box (STB) Reference Design Kit

This chapter introduces the IBM ACE4033CST Set-Top Box (STB) Reference Design Kit and its hardware, software, and user-supplied components.

1.1 STB Reference Design Kit Overview

The STB Reference Design Kit accepts, decodes, and displays MPEG2 audio and video from digital video broadcast (DVB) transport streams.

Sample DVB transport streams are input using the DVB Exerciser (DVBE), which connects between the STB Reference Platform (STBRP) board and a host PC, which stores sample DVB transport stream files. Other sources of test DVB transport streams can be connected to the STBRP, such as:

- A generic quadrature phase shift key (QPSK) front-end to provide unscrambled DVB transport streams from satellite systems, such as EchoStar.
- A generic quadrature amplitude modulation (QAM) front-end to provide cable TV feeds

Custom front-end boards can be designed to provide DVB transport streams.

The expansion interface of the STBRP provides a connector that makes the PPC403 bus available to additional logic that may be added to an STB design. Device interfaces can be designed, built, and attached as daughterboards for testing and software development.

An Ethernet daughterboard, supplied with the STB Reference Design Kit and connected to the STBRP peripheral expansion connector, furnishes 10Base2 and 10BaseT Ethernet interfaces. However, because the DVBE requires a 10Base2 Ethernet connection, 10Base2 cabling is typically used.

1.2 Advanced STBRP Features

The STBRP provides ample memory (4MB) for software development.

The STBRP supports internal and external clocking alternatives.

The IR receive mechanism of the STBRP receives and transmits Infrared Data Association/Hewlett-Packard serial infrared (IrDA/HPSIR), 500 KHz amplitude shift key (ASK), and ≈ 38 KHz ASK for TV remote SIR data.

The STBRP provides header connectors for optional test equipment, such as the IBM RISCWatch™ Debugger. This tool allows non-intrusive hardware and software debug through the STBRP JTAG port. For more information about the RISCWatch Debugger, call IBM Embedded PowerPC Technical Support at (919) 543-5701.

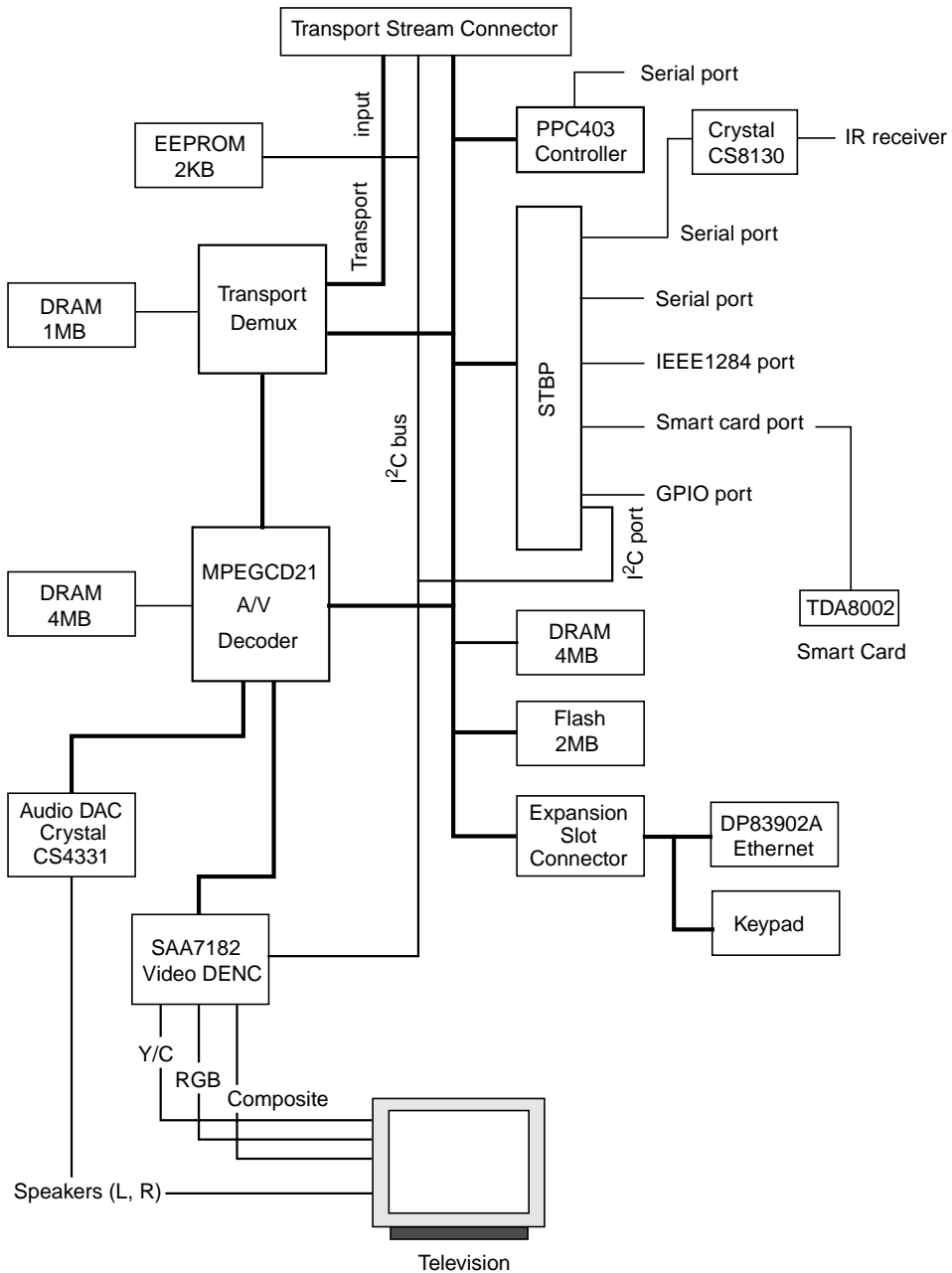


Figure 1-1. IBM Set-Top-Box Reference Platform Block Diagram

1.3 STB Reference Design Kit Hardware Components

The STB Reference Design Kit contains the following hardware components. The STBRP components are described in more detail in Chapter 7, “Hardware Components.”

- **IBM STB Reference Platform (STBRP)**

This is a board with an expansion slot and connectors for a digital video broadcast (DVB) transport stream, composite video, separate video (S-video) and left (L) and right (R) audio, two serial ports, and power. The connectors are described in detail in Chapter 8, “STBRP Connectors, Switches, and Jumpers.”

The STBRP provides the basic functions required for an STB product using a PowerPC 403 (PPC403) controller, an IBM MPEG2 audio/video (A/V) decoder, and an IBM STB Peripheral (STBP) application-specific integrated circuit (ASIC).

- **Ethernet daughterboard**

This board, which provides 10Base2 and 10BaseT Ethernet interfaces, plugs into the STBRP expansion slot.

- **DVB Exerciser (DVBE)**

This pair of boards, comprising a main board and an expansion daughterboard, provides the input DVB transport stream to the STBRP.

The main board uses a PPC403 controller, Ethernet interfaces, and a serial port to communicate with a host PC, where DVB transport stream files are stored.

The daughterboard provides a DVB transport stream connector that connects over a ribbon cable to the STBRP.

- **Cables and Supplies**

Cables are supplied for power, serial communications with a host PC, and separate video (S-video) and audio output.

Two power supplies are provided; these provide power to the STBRP and DVBE.

A universal remote control and smart card are provided.

1.4 STB Reference Design Kit Software Components

The basic STB Reference Design Kit provides a ROM monitor and two real-time operating systems (RTOS) options: pSOS, described in Chapter 11, “pSOS Platform Support Package,” and OS-9000, described in Chapter 12, “OS-9000 Platform Support Package.”

The ROM Monitor, resident in the flash memory module on the STBRP, initializes the PPC403 processor, the serial port, and the Ethernet controllers. The ROM Monitor, described in Chapter 10, “Using the ROM Monitor,” also loads applications from the host PC.

The ROM Monitor is accessed through a terminal attached to serial port 1 on the STBRP. Refer to Section 4.1, “Using a Terminal Emulator,” on p. 4-1, for information about setting up and using a terminal emulator.

1.5 Cables and Supplies

The STB Reference Design Kit includes the following cables:

- Two serial cables that connect an STBRP serial port and the DVBE serial port to a host PC for terminal emulation
- Power line cords for the 5-pin DIN power connectors on the STBRP and DVBE.
- S-video cable
- L/R audio cable
- 2 x 25 ribbon cable to connect the STBRP and DVBE

The STB Reference Design Kit provides power supplies for STBRP and DVBE. Line cords for the power supplies are for use in the USA only.

A universal remote control and smart card are also provided.

1.6 User-supplied Components

The STB Reference Design Kit attaches to a user-supplied host PC. The host PC must be attached to an Ethernet network to communicate with the STBRP and DVBE, which requires a 10Base2 Ethernet connection.

Users must also provide the 10Base2 Ethernet cabling between the STBRP, the DVBE, and the host PC. Each Ethernet connection requires a T-connector. Depending on the network configuration, one or two terminators may be required.

Chapter 2. Host PC Requirements

This chapter describes the hardware, software, and data communications and networking requirements of the host PC connected to the Set-Top Box (STB) Reference Design Kit hardware.

2.1 Host PC Hardware Requirements

A host PC should meet or exceed the following minimum requirements, or provide equivalent capabilities:

- 50/66 MHz Intel 486DX2 processor
- 8MB RAM
- Approximately 70MB of free disk space, if your STB Reference Design Kit is supplied with operating system and software development software
- VGA 640x480 display (minimum); SVGA 1024 x 768 (recommended)
- Serial port
- Ethernet adapter with a 10Base2 connector

2.2 Host PC Software Requirements

Supported host PCs run one of the following operating systems:

- Microsoft Windows 3.1, Windows 3.11, or Windows 3.11 for Workgroups
- Microsoft Windows 95
- Windows NT 3.51

2.3 Host PC Data Communications and Networking Requirements

The host PC communicates with the STBRP over a serial connection and Ethernet.

A serial connection is required for a terminal emulator, running on the host, that communicates with the ROM Monitor running on the STBRP. The Ethernet connection is used for most host PC-to-STBRP communications. If the host PC is not otherwise connected to the Ethernet, a terminator is required.

The host PC communicates with the DVBE using a serial connection for terminal emulation and a Ethernet connection for other host-to-DVBE communications.

The Ethernet connection requires additional hardware in the host PC, including an Ethernet adapter that provides a 10Base2 Ethernet connector (the DVBE requires 10Base2 cabling). The STB Reference Design Kit does not provide any networking hardware or software for a

host PC. If an Ethernet adapter is not already installed in the host PC, consult PC and adapter documentation for requirements and installation instructions.

Chapter 6, "Connecting the STB Reference Design Kit Hardware," describes how to connect the host PC, STBRP, and DVBE to an Ethernet network.

The host PC requires TCP/IP software compatible with the Microsoft Windows Sockets API. Such software is bundled with Windows 95 and Windows NT. Windows 3.*n* users can use Trumpet Winsock, a TCP/IP protocol stack available from **www.trumpet.com** on the World Wide Web. Installation documentation is also found at this site.

Users should refer to the documentation for the terms and conditions of using Trumpet Winsock.

Chapter 4, "Configuring the Host PC," describes how to configure and use the TCP/IP software.

Chapter 3. Installing Software on the Host PC

This chapter describes the procedures for installing the software components of the Set-Top Box (STB) Reference Design Kit on a host PC.

The STB Reference Design Kit software components are:

- Device drivers

The device driver installation diskettes contain device drivers for the STB Reference Platform (STBRP). Two sets of device driver diskettes are provided, one for the pSOS Platform Support Package (PSP) and one for the OS-9000 PSP.

- Board Support Packages (BSPs)

The BSP installation diskettes contain support files for the STBRP. Two sets of BSP installation diskettes are provided, one for the pSOS PSP and one for the OS-9000 PSP.

3.1 Installation Requirements

Before beginning the installation, you must have:

- The device driver and BSP installation diskettes
- A PC running Windows 3.1 or higher, Windows 95, or Windows NT 3.51

3.2 Installing Device Drivers

The following procedure installs the STBRP device drivers from a set of device driver diskettes:

Note: Windows NT users should log on as the Administrator.

1. Insert the diskette labeled "Device Driver Disk 1" into diskette drive A:.
2. Start Windows, if it is not running.
3. Select Run... from the File menu of Program Manager in Windows 3.x, or from the Start menu in Windows 95 and Windows NT.
4. Run the installation program. Enter:
a:setup
in the dialog box and click the OK button.
5. Follow the installation program prompts.

3.3 Installing a BSP

The following procedure installs the STBRP device drivers from a set of device driver diskettes:

1. Insert the diskette labeled "BSP Disk 1" into diskette drive A:.
2. Select Run... from the File menu of Program Manager in Windows 3.*n*, or from the Start menu in Windows 95 and Windows NT.
3. Run the installation program. Enter:
a:setup
in the dialog box and click the OK button.
4. Follow the installation program prompts.

Chapter 4. Configuring the Host PC

This chapter describes the steps taken to configure the host PC to communicate with the STB Reference Platform (STBRP).

4.1 Using a Terminal Emulator

The host PC uses a terminal emulator to access the ROM Monitor. Data is communicated over a connection between a host serial port (COM n on most PCs) and serial port 1 on the STBRP.

4.1.1 Using HyperTerminal (Windows 95)

After data communications and networking connections are made between the host PC and the STBRP and power is supplied to the STBRP, HyperTerminal can serve as a terminal emulator.

The following steps configure and start the HyperTerminal program:

1. Select Start from the Windows 95 task bar.
2. Select Programs.
3. Select Accessories.
4. Select HyperTerminal.
5. Click the No button if you see a window that says "You need to install a modem before you can make a connection. Would you like to do this now?"
6. Select the Hypertrm icon.
7. Enter a name, for example, **stbrp**, and select an icon.
8. Select the following:
 - Connect using Direct to Com 1 (default)
 - Bits per second - 9600
 - Data bits - 8 (default)
 - Parity - None (default)
 - Stop Bits - 1 (default)
 - Flow Control - Hardware (default)
9. Select OK.

After the STBRP is reset, the ROM Monitor menu appears in the Terminal window.

If the ROM Monitor menu does not appear, check the connection between the host PC and the STBRP. If the connection is good but the ROM Monitor menu still does not appear, use a PC setup utility to ensure that the COM port is enabled.

4.1.2 Using Windows Terminal (Windows 3.n and Windows NT)

After all data communications and networking connections are made between the host PC and the STBRP, and power is supplied to the STBRP, the Windows Terminal program can be used as a terminal emulator.

The following steps configure and start the Windows Terminal program:

1. From Windows Program Manager, select Accessories.
2. Select Terminal.
3. Select Settings.
4. Select Communications.
5. Select COM1 (or the appropriate COM port used for S1 serial port set-up).
6. Select Baud Rate 9600, Data Bits 8, Stop Bits 1, Parity None.
7. Select Flow Control Xon/Xoff.
8. Select OK.

After the STBRP is reset, the ROM Monitor menu appears in the Terminal window.

If the ROM Monitor menu does not appear, check the connection between the host PC and the STBRP. If the connection is good but the ROM Monitor menu still does not appear, use the PC setup utility to ensure that the COM port is enabled (refer to your PC publications for information).

Upon exiting the terminal program, you can save your setting in a ***.trm** file (for example, **stbrp.trm**) for future use.

4.2 Configuring TCP/IP for STB Development

An Ethernet network comprising an Ethernet adapter in the host PC (host adapter), the DVBE, and the STBRP Ethernet daughterboard provides appropriate data communications for STB development.

The host PC must provide a TCP/IP package that complies with the Microsoft Windows Socket (Winsock) API. Windows 95, Windows NT, and Windows for Workgroups 3.n provide such a package. Windows 3.n does not; however, a TCP/IP package may already have been installed.

Establishing an Ethernet interface requires IP addresses for the host PC, STBRP, and DVBE. The following IP addresses, which are used in the configuration examples, are suggested:

- 7.1.1.4, host PC
- 7.1.1.5, STBRP
- 7.1.1.6, DVBE

If the host PC is connected to an existing Ethernet network, the IP address of the host PC is already defined. Ask your network administrator for the IP address of the host PC, and how to add the STBRP and DVBE to the existing network.

Record the selected or supplied IP addresses, which will be needed later.

4.2.1 Configuring TCP/IP for Windows 95

A compliant TCP/IP package comes with Windows 95, so no TCP/IP package needs to be installed. If using Ethernet, follow the installation instructions that came with your Ethernet adapter.

The following steps set the Host IP address for the Ethernet connection:

1. Open the My Computer icon on the desktop.
2. Open the Control Panel icon.
3. Open the Network control panel.
4. Add the appropriate Adapter network component for the Ethernet adapter being used (if not already added).
5. Add a Protocol network component of Microsoft - TCP/IP (if not already added).
6. Specify the IP address (**7.1.1.4**, which is used throughout this document, is recommended) and subnet mask (**255.255.240.0**) to be used.

4.2.2 Configuring TCP/IP for Windows 3.n

To determine whether you need to install a TCP/IP package on Windows 3.n, perform the following steps:

1. Select the Main icon from the Windows Program Manager.
2. Select the File Manager icon.
3. Select File from the menu bar and choose Search.
4. Search for the file **winsock.dll** on all hard disk drives.

If the file **winsock.dll** exists, you probably already have a compliant TCP/IP package. If **winsock.dll** does not exist, you must install a Winsock-compliant TCP/IP package.

A Winsock-compliant TCP/IP package, Trumpet Winsock, can be downloaded from the Trumpet Software International (TSI) World Wide Web (web) site **<http://www.trumpet.com>**.

The following information supplements the installation instructions contained at the TSI web site, and is provided to clarify confusing information.

1. Go to **<http://www.trumpet.com>** and find the installation information for Trumpet Winsock.
2. Download the latest version that provides 16-bit support for Windows 3.*n*. For example, Trumpet Winsock v. 3.0 (in the file **twsk30c.exe**) supports Windows 3.*n* and Windows 95.

Note: The TSI software can be downloaded for an evaluation period. Read the provided information for terms and conditions.

The downloaded version is usually a self-extracting file that has the extension **.exe**.

3. Install the **.exe** file in a new directory (**c:\trumpet**, for example) and change the working directory to the new directory.
4. Run the **.exe** file.

Running the **.exe** file results in the creation of many more files in the new directory.

5. Read **README** files carefully.

Ethernet users should read about packet drivers; you will not be using a modem and you have already determined that there is not a TCP/IP package on your system.

6. Unless directed otherwise by the Trumpet Winsocks documentation, run **install.exe** to start the installation.

You will be prompted for required information.

Note: You may be told that the installer will look for and rename any **winsock.dll** files. If you checked for **winsock.dll** files earlier, such files should not be found.

7. If a setup window appears, defer entering any fields until a later time.
8. When installation is complete, reboot the host PC and start Windows.

4.2.3 Configuring Trumpet Winsock

Trumpet Winsock users can use the following steps as a guide to establishing a local Ethernet interface:

1. TSI provides software for packet drivers. When you first install a host Ethernet adapter, a set of device drivers typically is provided. To use Trumpet Winsock, you must select a packet driver. Some Ethernet adapters provide a packet driver that can be selected. If you buy an Ethernet adapter that does not provide a packet driver, select Help on the Trumpet menu bar to find out how to obtain a packet driver over the Internet. We will assume you have already followed the instructions for installing your Ethernet adapter, have installed Trumpet Winsock, and have chosen a packet driver for use with Trumpet.
2. Read any README files carefully. Pay particular attention to any directions concerning packet drivers.

3. Follow the instructions for “Using the Trumpet Winsock over a packet driver” from the main Trumpet Help window. Follow the instructions for “Installing a packet driver and WINPKT.” At the time of this publication, the WINPKT program is found at the universal resource locator (URL) <ftp://ftp.trumpet.com/winsock/winpkt.com>. The **ndis3pkt** package, referred in Help as a replacement for winpkt, does not work unless you have Windows for WorkGroups, or another Windows version that supports network driver interface specification (NDIS).
4. Use Trumpet Help as a guide to add two lines to the file **autoexec.bat** that are required to establish Ethernet networking.

The first line starts the packet driver that was installed with the host Ethernet adapter. The syntax for this line should be identified in the installation guide for the your Ethernet adapter, or in a file that came with the packet driver.

The second line to add is:

winpkt 0x60

Vector 0x60 is usually the default vector.

5. After updating the file **autexec.bat**, reboot the system to execute the changes.
6. From Windows, double-click on the Trumpet Winsock icon in the Trumpet Winsock Files program group to start Trumpet Winsock.
7. If setup was bypassed during installation, the connection will fail. A Trumpet Winsock window comes up indicating your connection status.
8. Select Setup from the File menu to open the Setup dialog.
9. Specify the IP address (**7.1.1.4**, which is used throughout this document, is recommended).
10. Select Packet driver, set Vector to **60**, Netmask to **255.255.240.0**, and Gateway to **0.0.0.0**.
11. Select OK.
12. Specify, in the **hosts** file in the installed Trumpet directory, the IP addresses to be used, for example:

7.1.1.4	local_enet
7.1.1.5	stbrpb_enet
7.1.1.6	dvbe_enet

After entering all the information, restart Trumpet Winsock for the network setup to take effect.

Before exiting Windows, quit the Trumpet Winsock application. Otherwise, subsequent Trumpet starts may fail. If this occurs, reboot your system.

4.2.4 Configuring TCP/IP for Windows NT 3.51

A Winsock-compliant TCP/IP package comes with Windows NT, so no TCP/IP package needs to be installed.

To configure TCP/IP for Ethernet, double-click on the control panel icon followed by the network icon. Windows NT prompts you through adding an Ethernet adapter and TCP/IP, if this has not already been done.

Specify the IP address (**7.1.1.4**, which is used throughout this document, is recommended) and subnet mask (**255.255.240.0**) to be used.

4.3 Configuring bootp and tftp to Support ROM Monitor Loads

To use the ROM Monitor to download applications to the STBRP, the host PC must be configured to support the **bootp** protocol and **tftp** servers (daemons). This requires two main steps: The host **bootptab** file must be configured, and the **bootp** and **tftp** servers must be made available.

4.3.1 Configuring bootp and tftp on the Host PC

Because not all TCP/IP packages provide the **bootpd** and **tftpd** servers required for ROM Monitor downloads, the servers are provided in the **lib** directory of the OS-9000 Platform Support Package (PSP) and the pSOS PSPs. The provided servers can be installed and used with Winsock-compliant TCP/IP packages.

Because TCP/IP packages vary, this section provides *guidelines* for configuring **bootp** and **tftp**. Consult the documentation for your TCP/IP package for specific details.

To use the **bootpd** and **tftpd** servers provided with the PSPs, modify the file **autoexec.bat** on the host PC to specify the location of the PSP files **bootptab** and **services**, adding a line that sets the **ETC** constant to the directory containing the PSP files **bootptab** and **services**:

- **SET ETC=C:\TRUMPET** (Windows 3.11 Trumpet Winsock)
- **ETC=C:\WINDOWS** (Windows 95)
- **ETC=C:\WINNT35\system32\drivers\etc** (Windows NT 3.51)

The PSPs provide a sample bootptab file, **bootptab.sam**, in the **lib** directory. Copy this file to the **ETC** directory; the file copy must be named **bootptab** with no file extension. Add entries describing the evaluation board to the host PC to the **bootptab** file.

When creating or modifying a **bootptab** file, the following rules apply:

- Blank lines and lines beginning with **#** are ignored.
- Each entry must be on one line.
- Each entry must start with a host name followed by the legends (see the sample **bootptab** file for legend descriptions).
- **A** : separates legends; leave no space between legends.
- A user-supplied **ip** legend specifies the IP address of the host PC.
- If **hd** (home directory) and **bf** (boot file) legends are not provided for a particular entry, the first defined **hd** and **bf** legends in **bootptab** are used by default.

The following entry provides an example:

```
enetc:ht=ethernet:hd=\lib:bf=sampboot.img:bs=ip=7.1.1.5:sm=255.255.255.255:
ha=xxxxxxxxxxxx
```

The value of the Ethernet hardware address field in the **enetc** entry **ha=xxxxxxxxxxxx** should match the 12-character hardware address listed for the Ethernet Boot Source on the ROM Monitor menu.

The file **\lib\sampboot.img** is the source for the application image to be downloaded to the STBRP.

Ensure that the **ht=ethernet** legend is used for the Ethernet connection entry and that the IP address in the **enetc** legend is that of the STBRP. If the suggested IP address was used, 7.1.1.5 is the IP address of the STBRP.

The file **services** (no file extension), which must also exist in the **ETC** directory, must be updated with the port and protocol information for the **bootpd** and **tftpd** servers. To use the servers provided with the PSPs, the following entries must be included in the **services** file:

```
bootps 67/UDP
bootpc 68/UDP
tftp 69/UDP
```

For the update to take effect, TCP/IP must be restarted; this may require restarting the TCP/IP package or rebooting the system.

4.3.2 Automatic bootpd and tftpd Startup for Windows 3.n and Windows NT 3.51

The **bootpd** and **tftpd** servers can start automatically when Windows 3.n or Windows NT 3.51 starts.

Alternatively, the servers can be started using the Program Manager File and Run menus. TCP/IP must be running before the servers can started.

To configure Windows 3.n to start the **bootpd** and **tftpd** servers automatically, add them to the Windows Startup window:

1. Select the Program Manager and open the Startup window.
2. Pull down the Program Manager File menu and select New to display a New Program Object window.
3. In the New Program Object window, select Program Item and click the OK button to open the Program Item Properties window.

The Program Item Properties window requires Description, Command Line and Working Directory values. The following example shows a possible configuration:

```
Description: BOOTPD
Command Line: BOOTPD -C D -H 7.1.1.4
Working Directory: D:\PSPBIN
```

Command Line specifies how to invoke the **bootpd** server.

The **-C** parameter specifies the drive letter used with **bootptab** file entries. Because the colon is a delimiter in **bootptab** file entries, the **bootpd** server uses the **-C** parameter to concatenate a drive letter to the beginning of the **hd:** legend. If no **-C** parameter is specified, the current drive is the default.

The **-H** parameter specifies the IP address of the host PC (set during TCP/IP configuration) to the **bootpd** server.

Working Directory specifies the location of the **bootpd** server program **bootpd.exe**. *PSP* specifies the directory for the appropriate PSP.

Use a similar procedure to configure automatic startup of the **tftpd** server. The Program Item Properties window entries will describe information for the **tftpd** server. The following example shows a possible configuration:

Description: TFTP
Command Line: TFTP
Working Directory: D:\PSP\BIN

Command Line specifies how to invoke the **tftpd** server.

Working Directory specifies the location of the **bootpd** server program **bootpd.exe**. *PSP* specifies the directory for the appropriate PSP.

4.3.3 Automatic bootpd and tftpd Startup for Windows 95

The **bootpd** and **tftpd** servers can start automatically when Windows 95 starts.

To run the **bootpd** and **tftpd** servers automatically when Windows 95 starts, perform the following steps:

1. Select Start from the Windows 95 task bar.
2. Select Setting
3. Select Taskbar
4. Select Start Menu Programs
5. Select Add...
6. In the Command Line field, enter,

```
BOOTPD -c C -h 7.1.1.4
```

where *C* specifies the drive letter containing the boot image and 7.1.1.4 is the host IP address

7. To start the tftp server automatically, follow the preceding steps, but enter

```
TFTP
```

in the command line field.

Chapter 5. Assembling the STB Reference Design Kit Hardware

This chapter describes the components of the STB Reference Design Kit and how those components are assembled into a development platform for STB applications.

The main components of the STB Reference Design Kit hardware are the STB Reference Platform (STBRP) and the Digital Video Broadcast (DVB) Exerciser (DVBE). Each of the main components comprise components that must be assembled after you receive your STB Reference Design Kit.

5.1 STB Reference Design Kit Hardware Components

The following list describes the hardware components of the STB Reference Design Kit are:

- STBRP, which consists of the following hardware components:
 - STB main board (Part No. 42H3082)
 - Ethernet daughterboard (Part No. 42H3083)
- DVBE (Part No. 42H3069 00001)
- DVBE buffer card (Part No. 42H2868)

5.2 Assembling the STB Reference Design Kit Components

The main steps of assembling the STB Reference Design Kit hardware are:

1. Connecting the DVBE to the DVBE expansion connector.
2. Connecting the Ethernet daughterboard to the STBRP expansion connector.
3. Connecting the 2 x 25 data stream cable between the transport stream connectors on the STBRP and the DVBE.

5.3 Connecting the DVBE Buffer Card to the DVBE

The DVBE buffer card connects to the expansion connector on the DVBE.

To connect the DVBE buffer card, insert its connector into the mating expansion connector on the DVBE.

5.4 Connecting the Ethernet Daughterboard to the STBRP

The Ethernet daughterboard connects to the STBRP expansion connector.

To connect the Ethernet daughterboard, insert the daughterboard's Eurocard connector into the mating expansion connector on the STBRP.

5.5 Connecting the Transport Stream Cable

The transport stream cable connects the DVBE to the STBRP. This cable, a 2 x 25 ribbon cable with female connectors at each end, plugs into the mating transport connectors on the DVBE and STBRP.

Chapter 6. Connecting the STB Reference Design Kit Hardware

This chapter describes how to connect the STB Reference Platform (STBRP) and Digital Video Broadcast (DVB) Exerciser (DVBE) serial ports to the host PC, the STBRP Ethernet daughterboard, DVBE, and host PC to an Ethernet network, and the STBRP audio and video (A/V) outputs to a TV set or monitor, and the STBRP and DVBE to their power supplies.

6.1 Connecting the STBRP and DVBE Serial Ports to the Host PC

To connect the STBRP serial port to the host PC, run a serial cable between the STBRP serial port connector P3 and a serial port on the host PC.

To connect the DVBE serial port to the host PC, run a serial cable between the DVBE serial port connector S1 and a serial port on the host PC.

6.2 Connecting the STBRP, DVBE, and Host PC to an Ethernet Network

To connect the STBRP Ethernet daughterboard, DVBE, and host PC to an Ethernet network:

Connect 10Base2 T-connectors to the 10Base2 connectors on the STBRP Ethernet daughterboard (J3), the DVBE (P5), and host PC Ethernet adapter.

Run a 10Base2 cable from the T-connector connected to the STBRP to the T-connector connected to the DVBE. Run another 10Base2 cable from the T-connector connected to the DVBE to the T-connector connected to the host PC Ethernet adapter.

Depending on other Ethernet network connections, you might need to attach terminators to the T-connectors connected to the STBRP Ethernet daughterboard or host PC Ethernet adapter, or both. Ensure that all T-connector sides are connected to the Ethernet network or to a terminator.

6.3 Connecting the STBRP A/V Outputs to the TV

The STBRP supports composite video, SCART, separate video (S-video), and left (L) and right (R) audio outputs.

The following connection example uses the S-video and L and R audio outputs.

To connect the A/V outputs to the TV, run an S-video cable from the S-video output on the STBRP (J1) to the S-video input on the TV. Run audio cables from the L and R audio outputs on the STBRP (J3 and J4) to the L and R audio inputs on the TV.

6.4 Connecting the STBRP and DVBE to Power

To connect the DVBE to power, connect the Ault, Inc. SW 201 power supply to the large DIN-style connector on the STBRP. Alternatively, power can be supplied through a PC-style power connector.

To connect the STBRP to power, connect the ELPAC Power Systems Model WR1 4232 power supply to the large DIN-style connector on the DVBE.

When connecting the STBRP and DVBE to power, connect the power supplies to the boards before connecting the power supplies to an electrical outlet. Otherwise, damage to the boards can result.

Chapter 7. Hardware Components

This chapter describes the major hardware components of the STB Reference Platform (STBRP) and its Ethernet daughterboard.

7.1 STBRP Components

The major components of the STBRP are:

- PowerPC 403 (PPC403) embedded controller
- Memory
- System control logic
- Infrared (IR) transceiver
- STB peripheral (STBP) chip
- Digital broadcast video (DVB) transport and descrambler chips
- MPEG audio/visual (A/V) decoder
- Digital video encoder (DENC)
- Audio digital-to-analog converter (DAC)

7.1.1 PPC403 Embedded Controller

A PPC403 embedded controller serves as the main STB processor. This 32-bit microcontroller offers high performance and functional integration, along with low power consumption. The PPC403 provides on-chip caches, integrated DRAM/SRAM control, and peripherals that reduce system chip count and design complexity.

The PPC403 features:

- PowerPC RISC CPU
- Glueless interfaces to DRAM, SRAM, ROM, and peripherals
- 2KB instruction cache and 1KB write-back data cache, two-way set associative
- Memory management unit (MMU)
- Four DMA channels
- External interrupt controller
- Serial port
- Flexible interface to external bus masters

The RISC CPU features:

- Thirty-two 32-bit general purpose registers
- Branch prediction and folding
- Single-cycle execution for most instructions
- Hardware multiplier and divider for faster integer arithmetic
- Enhanced string and multiple word handling
- Minimal interrupt latency

The PPC403 32-bit data bus supports addressing for 512MB of external memory and memory-mapped input/output (MMIO) using a wide range of memory timing parameters. Byte, half-word, and full-word devices can be directly connected to the bus interface unit.

The PPC403 boots from a byte-wide flash ROM, then executes code from DRAM configured as a 32-bit wide bank controlled by the on-chip DRAM controller. The STBRP and daughterboard use all eight chip selects, but two are still considered to be available because they are connected to the daughterboard connectors.

The on-chip serial port can serve as a software debug port or general-purpose RS-232 port.

DMA channels 2 and 3 connect to the STBP. DMA channel 0 is unused. DMA channel 1 connects to the expansion connector.

The five interrupt inputs (INT0–INT4) connect to various devices. INT2 is reserved for the daughterboard. INT4 is reserved for the front-end expansion module. The critical interrupt input, connected to a “debounced” switch S2 on the main board, is available to aid software development.

7.1.1.1 PPC403 Interrupt Distribution

Table 7-1 shows the PPC403 interrupt distribution.

Table 7-1. Interrupt Distribution

Interrupt	Device
INT0	MPEG
INT1	Transport
INT2	Expansion Module
INT3	STBP
INT4	Front-end Module

7.1.1.2 PPC403 Chip Selects and Address Mapping

Table 7-2 shows how the PPC403 bank registers are associated with the major STBRP devices.

Table 7-2. PPC403 Bank Registers BR0–BR6

Bank Register	Device	Bank Size
BR0	Flash	2MB
BR1	STBP	1MB
BR2	Unused	—
BR3	Expansion	16MB
BR4	Unused	—
BR5	MPEG	8MB
BR6	Transport	16MB

Table 7-3 shows the settings for BR0–BR6.

Table 7-3. PPC403 Bank Register Settings to Control Output Signals $\overline{CS0}$ – $\overline{CS6}$

Field (bits)	Bank Register						
	BR0	BR1	BR2	BR3	BR4	BR5	BR6
BAS (0:7)	11111110	00000001		00010000		00001000	00100000
BS (8:10)	001	000		100		011	100
BU (11:12)	11	11		11		11	11
SLF (13)	0	0		0		0	0
BME (14)	0	0		0		0	0
BW (15:16)	00	00		00		00	01
RE (17)	0	1		1		1	1
TWT (18:23)	000100	000011		000101		000010	000100
CSN (24)	0	1		1		1	1
OEN (25)	0	0		0		1	1
WBN (26)	0	1		1		1	1
WBF (27)	0	0		0		0	0
TH (28:30)	010	010		010		010	010
SD	1	1		1		1	1

Table 7-4 shows the settings of PPC403 Bank Register 7–DRAM Configuration (BR7), which control the system DRAM connected to PPC403 output signal $\overline{CS7}$, if a 4MB DRAM SIMM is installed. If a 16MB SIMM is installed, BR7[BS] = 100.

Table 7-4. PPC403 BR7 Settings to Control Output Signal $\overline{CS7}$

BR7			BR7		
Field	Bits	Setting	Field	Bits	Setting
BAS	0:7	00000000	ARM	19	0
BS	8:10	010	PM	20	1
BU	11:12	11	FAC	21:22	01
SLF	13	1	BAC	23:24	01
ERM	14	0	PCC	25	0
BW	15:16	10	RAR	26	1
IEM	17	0	RR	27:30	0111
RCT	18	0	SD	31	0

7.1.2 Memory

System memory comprises DRAM, a flash boot ROM, and a serial EEPROM

7.1.2.1 System DRAM

System DRAM is a single in-line memory module (SIMM). The installed part is a 5V, 70 nanosecond (ns) 4MB SIMM with fast-page mode access. A single-bank 8MB or a 16MB SIMM can replace the 4MB SIMM. The STBRP allocates only one chip-select for DRAM; hence the single-bank limitation. The four memory-type pins on the SIMM connect to GPIOs on the STBP so that software can verify the type of DRAM. The expected operating mode is 3-2-2-2 wait states per bus cycle.

All DRAM control inputs, which connect directly to the PPC403, are under software control. Chip-select output $\overline{CS7}$ connects to the RAS input pin. The PPC403 outputs $\overline{CAS0}$ and $\overline{CAS1}$ connect to the D0:15 data lines. $\overline{CAS0}$ controls write accesses for the byte on D0:7; $\overline{CAS1}$ controls write accesses for the byte on D8:D15. The processor outputs $\overline{CAS2}$ and $\overline{CAS3}$ connect to the CAS inputs on the D16:31 data lines. $\overline{CAS2}$ controls write accesses for the data byte on D16:23; $\overline{CAS3}$ controls write accesses for the data byte on D24:D31.

System performance can be tested using 16-bit wide memory instead of 32-bit wide memory. Moving jumper shunts from pin1 to pin 2 on jumpers J21, J24, J25, and J28 enables 32-bit operation. Moving these jumper shunts from pin 2 to pin 3 disables half of the 32-bit wide SIMM to provide 16-bit wide memory operation. Note that only half of the memory can be used when using 16-bit wide mode.

7.1.2.2 System Flash Boot ROM

An AMD AM29F016 is the system boot ROM. This 16 megabit (Mb), 5V-only, 90ns, sector-erase flash-memory device is organized as 2MB divided into thirty-two 64KB sectors for flexible erasing. The PPC403 chip-select output $\overline{CS2}$ connects to the ROM chip-select input. The chip-select access type is programmed for four wait states and a write strobe that occurs one clock before the chip-select becomes inactive. No additional control logic is required.

7.1.2.3 Serial EEPROM

This 2KB serial EEPROM, with an I²C interface, connects to the I²C interface on the STBP.

7.1.3 System Control Logic

The STBRP contains an electrically-programmable logic device (EPLD) that provides miscellaneous glue logic. This flash-based part from Lattice Semiconductor provides in-circuit programming capability. The EPLD has 12 input pins that connect to various signals on the STBRP, and 10 I/O pins. Three I/O pins are used; the extra pins and internal gates could be used for additional control logic.

7.1.4 STBP

The STBP is attached to the PPC403 chip as a memory-mapped I/O peripheral controller that provides interfaces to various peripheral devices commonly required in set-top box designs.

The STBP connects to PPC403 chip select $\overline{CS1}$ and to PPC403 DMA channels 2 and 3.

7.1.4.1 Serial Port

The STBP UART 2 port connects to an external serial port.

A driver/receiver/voltage converter converts the serial port signals to standard RS-232 voltages. The converter connects to a 9-pin connector (P2).

7.1.4.2 IEEE 1284 Parallel Port

The STBP provides an IEEE 1284 parallel port. The board provides a standard 25-pin DIN connector (P1) for this port.

The data pins are buffered with a 3.3V transceiver that tolerates 5V input. All IEEE 1284 inputs have 3.3V pullups. This buffer provides sufficient current drive. The 3.3V transceiver simplifies the plane cutouts required by the mix of 5V and 3.3V parts on the board.

7.1.4.3 Smart Card and Conditional Access

To be compatible with the smart card electrical interface requires proper handling of short-circuit and power-supply issues. The TDA8002 handles the smart card interface requirements in a single component.

When answering a reset, the smart card operates at 9600 baud. After the reset is answered, a higher baud rate can be used. This design supports 19 200 baud when a 14.318 MHz oscillator input is connected to pin 1 of the smart card interface. Baud rates are generated inside the STBP using the 27 MHz input clock.

Changing the smart card internal divisor enables asynchronous operation at baud rates greater than 19 200. A socket is available for an external oscillator input into the STB peripheral chip. This socket can accept an oscillator having the exact frequency needed to support the higher baud rate, if this baud rate cannot be derived with the desired accuracy using the 27 MHz system clock.

The smart card interface supports only asynchronous operation.

7.1.4.4 Smart Card/GPIO Connection

The STBP provides the interface to the smart card, 24 GPIO pins and the I²C channel. The GPIO pins are shown in Table 7-5.

Table 7-5. STBP GPIO Pins

GPIO Pin	Function
0	DRAM SIMM pin 67
1	DRAM SIMM pin 68
2	Ring indicator for serial port # 2
3	DCD RS-232 connection for serial port # 2
4:7	General purpose LEDs D4, D5, D6, D7
8	Transistor-to-transistor logic (TTL) low noise block (LNB)-PWR switching signal for transport connector
9	Uncorrectable packet indication from transport connector
10	DRAM SIMM pin 69
11	DRAM SIMM pin 70
12	Smart Card interface AUX1UC
13	Smart Card interface AUX2UC
14	Smart Card interface CLKSEL
15	Smart Card interface CLKDIV1
16	Smart Card interface CLKDIV2
17	Smart Card interface $\overline{\text{CMDVCC}}$
18	Smart Card interface RSTIN

Table 7-5. STBP GPIO Pins (cont.)

GPIO Pin	Function
19	Smart Card interface $\overline{\text{OFF}}$
20	Smart Card interface MODE
21:22	Spare
23	SCART connector BLANKING pin

7.1.4.5 GPIO Connection

Nine of the GPIO pins on the STBP control the TDA8002 smart card interface, as shown in Figure 7-1.

Note: The TDA8002 does not drive the program voltage pin on the smart card connector. Programming of smart cards, without additional circuitry to drive pin 6, is not supported.

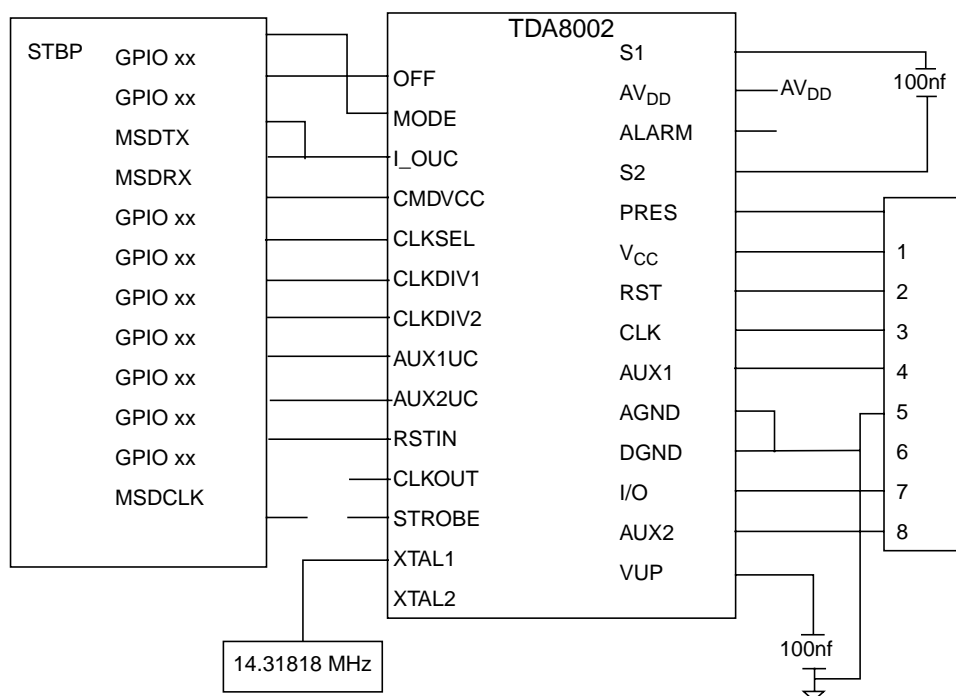


Figure 7-1. Smart Card Connection Diagram

7.1.5 Buffering

Figure 7-2 and Figure 7-3 show the STBRP data and address buffering schemes, respectively.

Additional buffering isolates the daughterboard connectors from the STBRP logic. This is to protect the STBRP against unknown logic rather than reducing the overall bus loading. The load capacitance on the data bus is 78pF, which exceeds the 50pF load specified for the PPC403.

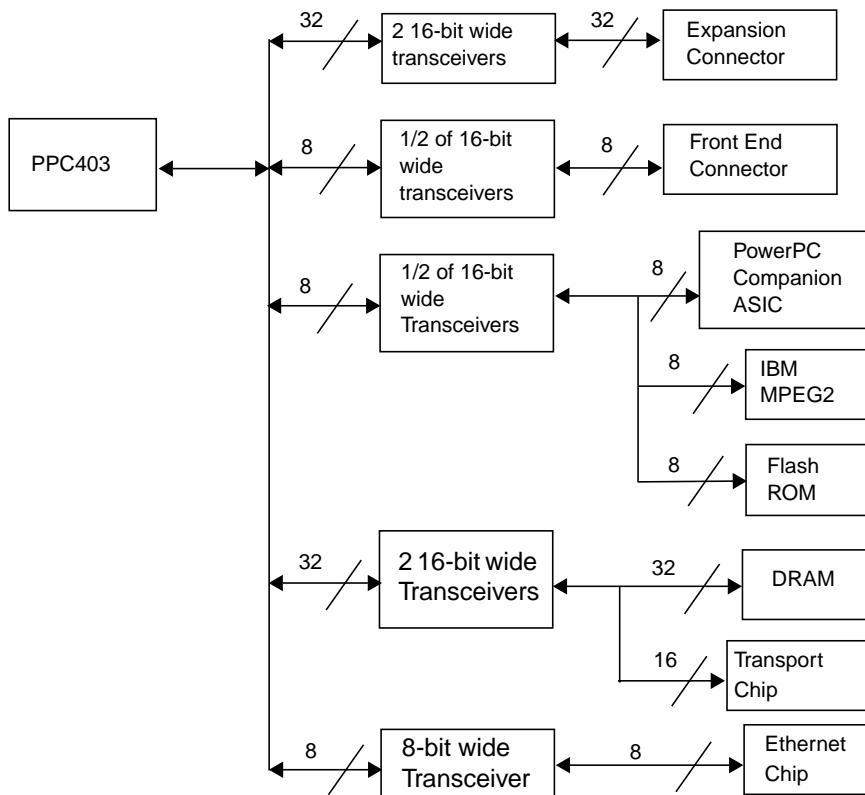


Figure 7-2. Data Buffering Diagram

The DRAM SIMM module attaches to the address buffer used for the daughterboard modules.

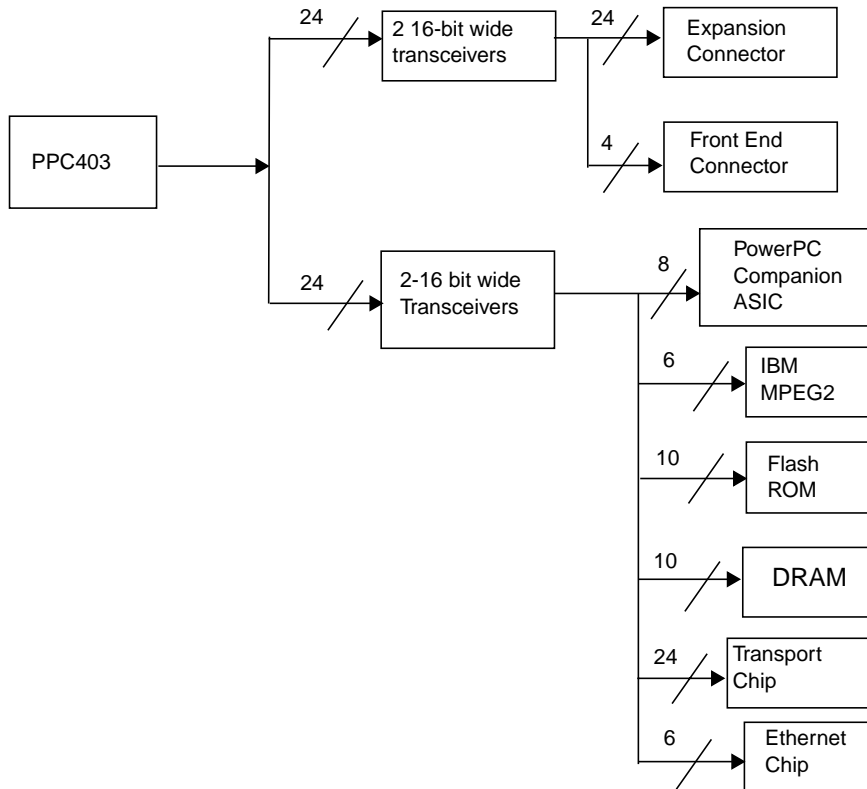


Figure 7-3. Address Buffering Diagram

7.1.6 Infrared (IR) Transceiver

A Crystal Semiconductor CS8130 IR transceiver IC provides an Infrared Data Association (IrDA) interface that decodes the IR received from a standard universal-remote controller. Output from this device connects to the STBP UART port (pins 65–70).

The receive channel includes an on-chip, high-gain PIN-diode amplifier, IrDA/HPSIR, amplitude shift key (ASK), and TV remote-compatible decoder and data-pulse stretcher. The transmit channel includes IrDA/HPSIR, ASK, TV remote-compatible encoder, and LED driver. The required computer data port is UART TXD- and RXD-compatible, with flow control, operating between 1200 baud and 115200 baud. An external PIN diode, transmit LED, and 3.6864 MHz oscillator provide IR transmit/receive capabilities.

To receive TV remote IR, software must write the modulation frequency to the modulator divisor registers. The tolerance on the expected frequency must be written to the receive ASK-timing sensitivity register.

There are two TV remote receive data modes, “oversampled” and “programmed T period.” If the T period mode of the TV remote is not known, it is possible to use the oversampled mode to measure the T period and then to switch to the programmed T period mode to reduce CPU overhead. In programmed T period mode, the part detects and decodes the incoming IR bit stream, assembles these bits into characters with a start and stop bit, and then transmits the character via the UART port.

7.1.7 DVB Transport Chip

The DVB transport chip is a VLSI VES2020 MPEG2 transport demultiplexer. This chip handles MPEG2 transport streams and packetized elementary streams (PES) of up to 60 Mbps, parses the streams to frame the MPEG2 packets, and then routes the extracted packets to either the MPEG2 video/audio decoder, or host processor. The VES2020 also performs program clock reference (PCR) recovery to maintain audio and video synchronization.

The DVB transport chip controls its own DRAM, and can also control DRAM for the host and MPEG audio decoder. However, in the STBRP, the chip controls only its own DRAM.

The DVB transport chip contains a high-performance RISC engine. Its microcode, stored in the STBRP boot ROM, is downloaded on system power-up. VLSI provides the microcode in various versions designed to support different service providers. The STBRP microcode, a version that provides basic DVB functionality, receives and decodes unscrambled DVB data.

An external descrambler interface consisting of eight data outputs, eight data inputs, and control signals, is provided. The DVB transport chip sends an 8-byte key and scrambled data for each received scrambled packet. The descrambled packet is expected to return over the input parallel, byte-wide data path.

The chip select for the DVB transport chip connects to the PPC403 $\overline{CS6}$ pin. The 16-bit host data bus on the transport chip connects to the PPC403 data signals D0:15.

7.1.7.1 Transport DRAM

Two Texas Instruments 5V 256Kb x 16 DRAMs provide 1MB of DVB transport chip DRAM. The DVB transport chip DRAM interface is 16 bits wide, and supports up to 1MB. The STBRP uses the maximum DRAM capacity. The reference designator U30 is for the first bank; U35 is for the second bank.

A typical set-top box product contains less memory.

7.1.7.2 DVB Transport Chip I/O

The DVB transport chip supports a high-speed serial output port. The RISC engine can output MPEG2 transport packets at 40.5 MHz. The three output pins on the high-speed serial output port are connected to the 3-pin header J16. Video and audio connections to

the MPEG decoder use the serial output connections. The transport chip can also support byte-wide, parallel, video output. The parallel video input path into the MPEG device is not used.

The transport input stream comes from the transport connector into the CHDATA(0:7) pins on the DVB transport chip. The port operates in either parallel- or serial-input mode. When a front-end module operates in serial mode, any data bits can be used as the serial input, with the remainder of the data bits tied to ground or left as no-connects.

The serial video input to the transport chip allows its parallel port to serve as the PPC403 interface. This precludes sourcing of MPEG data directly from the PPC403 into the MPEG decoder because the strobe lines for parallel MPEG input are not driven to clock in the data. To input MPEG data from the PPC403, the audio and/or video queue in the transport memory must be loaded with the required data. The PPC403 interface can be told to clock it through the serial interface into the compressed data FIFO in the transport chip.

An external 27 MHz voltage-controlled crystal oscillator (VCXO) is connected to the transport chip.

7.1.8 DVB Descrambler

The VLSI VES0010 DVB descrambler implements a descrambling algorithm defined by the European Project for Digital Video Broadcasting in *European Project for Digital Video Broadcasting - Common Scrambling Specifications Dec. 1994*. The DVB descrambler chip supports descrambling at the transport stream level, and at the packetized elementary stream level.

The DVB descrambler chip, which connects directly to the DVB transport chip, requires no additional control logic.

Unencrypted DVB transport streams do not require descrambling; the default code configuration does not require the DVB descrambler chip to be present.

Note: STB Reference Design Kits do not have the DVB descrambler chip installed on the STBRP due to export restrictions.

7.1.9 MPEG Audio/Video Decoder

The IBM MPEGCD21 provides a single-chip solution for decoding MPEG-2 main profile at main level video-packetized elementary stream and MPEG Layers I & II audio packetized elementary stream. The MPEGCD21 receives serial video and audio streams from the VLSI transport stream device.

The processor interface connects to the PPC403 D0:7 signals. The processor chip select signal $\overline{CS5}$ controls accesses to this device. The interface to the MPEG DRAM is 64 bits wide. The video-out connection to the video digital encoder (DENC) is through a 16-bit YUV data port. The connection to the audio DAC is serial.

The 27 MHz clock source for the MPEGCD21 is jumper-selectable between the voltage-controlled oscillator (VCXO) clock controlled by the transport device, or an alternative clock oscillator.

Eight Texas Instruments 5V 256Kb x 16 DRAMs provide the 4MB of DRAM attached to the MPEG device. Because the MPEG data interface is 64 bits wide, two banks of four devices are required.

The maximum amount of supported MPEG DRAM is provided for maximum functionality. Many current National Television Standards Committee (NTSC) set-top box designs provide 2 MB of MPEG DRAM.

7.1.10 DENC

A Phillips Semiconductor SAA7182 (a phase alternate line (PAL)/NTSC/something essentially contrary to the American method (SECAM) encoder) is the DENC. This part supports closed-caption television and Teletext. The pixel frequency is 13.5 MHz.

The DENC simultaneously outputs combined video blanking and sync (CVBS), luma/chroma (Y/C), and red-green-blue (RGB). The video outputs connect to the S-video, composite video, and SCART connectors using the standard recommendation for isolation and termination.

The board uses an analog V_{cc} (power) and ground cutout for the video and audio drivers and connectors, for signal-to-noise protection.

7.1.11 Audio DAC

The Crystal audio DAC CS4331-KS connects a serial data stream to the MPEG device between +5V and ground. The left (L) and right (R) outputs connect to the composite video output connector and the SCART connector.

7.2 Ethernet Daughterboard

The Ethernet daughterboard provides the STBRP with 10Base2 and 10BaseT connections. This low part count daughterboard connects to the STBRP through the expansion connector. The dual DMA (local and remote) capabilities of the ST-NIC and with 8 KB of buffer SRAM, which can be extended to 32KB, allow the Ethernet daughterboard to appear to the system as a standard I/O port.

The Ethernet daughterboard supports two physical layer options, 10BaseT (using an RJ45 connector) and 10Base2 (using a BNC connector). The local DMA channel of the ST-NIC buffers packets between local memory (8 KB of buffer SRAM) and the network; the remote DMA channel passes data between local memory and the system using an I/O Port. The I/O port architecture isolates the PPC403 from the network traffic and provides the simplest system interface. The dual DMA control logic is implemented in an iSP1024 FPGA chip.

The Ethernet daughterboard also provides a matrix of 9 switches that can be used as control buttons for the STBRP. The switch matrix, which interfaces to the system using the FPGA control logic, also appears to the system as a standard I/O port

Figure 7-4 illustrates the layout of the Ethernet daughterboard.

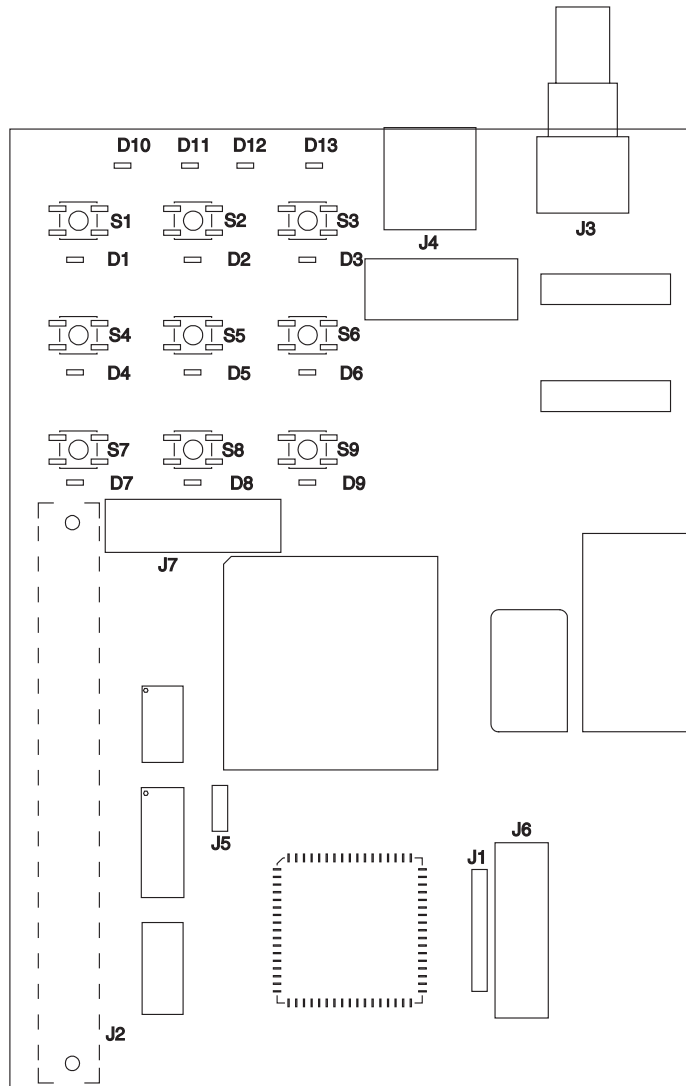


Figure 7-4. Ethernet Daughterboard Layout

7.2.1 Network Interfaces

The Ethernet daughterboard supports two physical-layer options: 10BaseT and 10Base2. Because the DVBE provides only a 10Base2 interface, using the 10Base2 interface is highly recommended.

When using 10Base2, an on-board transceiver (the CTI) enables the daughterboard to directly connect to the network. To use the 10Base2 interface, leave J7.19 and J7.20 unjumped.

The 10BaseT interface enables direct connection to the network using the RJ-45 phone jack. When using the 10BaseT, J7.19 and J7.20 must be tied together by a jumper.

7.2.2 Indicator Lamps

Table 7-6 lists the indicator lamps on the Ethernet daughterboard.i

Table 7-6. Indicator Lamps D10–D13

Lamp	Name	Function
D1/		Prevent ghost switch
D2/		Prevent ghost switch
D3/		Prevent ghost switch
D4/ r		Prevent ghost switch
D5/		Prevent ghost switch
D6/		Prevent ghost switch
D7/		Prevent ghost switch
D8/		Prevent ghost switch
D9/		Prevent ghost switch
D10/Clear	Rx	When lit indicates Ethernet receiving data
D11/Clear	Tx	When lit indicates Ethernet sending data
D12/Clear	COL	When lit indicates Ethernet traffic collision
D13/Clear	PWR	When lit indicates power on Ethernet board

7.2.3 Switch Matrix Keypad

The switch matrix consists of nine switches arranged in a 3 x 3 matrix; the three rows and three columns connect to the FPGA.

Table 7-7 shows the configuration of the switch matrix.

Table 7-7. Switch Matrix Configuration

	Column 0	Column 1	Column 2
Row 0	S1	S2	S3
Row 1	S4	S5	S6
Row 2	S7	S8	S9

The PPC403 issues an I/O port write to the FPGA to control the column line voltages. The FPGA latches data from D4:6 into three D flip-flops and output them to the column lines. To read the status of the row lines, the PPC403 issues a I/O port read to the FPGA. The FPGA decodes the address and puts the row data on D4:6 by controlling the output enable on the bidirectional I/O pins

Table 7-8 shows the row-column control bits on the data bus.

Table 7-8. Row-Column Control Bits on the Data Bus

	Column 0	Column 1	Column 2
Row 0	D6		
Row 1		D5	
Row 2			D4

The PPC403 must scan the switch matrix to know which key was pressed. The scan starts from column 0, when the PPC403 sends 0b011 to the FPGA on D4:6. The FPGA latches the data into flip-flops and outputs the data to the switch matrix columns. Then, the PPC403 immediately reads the rows back (also from D4:6). The PPC403 can directly read the row lines, which are connected to the FPGA.

The FPGA decodes the address for the switch and the $\overline{R/W}$ signal to controls the bidirectional I/O.

Schottky diodes implemented on each key eliminate ghost keys.

Table 7-9 describes the switch matrix scan decoding.

Table 7-9. Switch Matrix Scan Decoding

D4:6	Switch Pressed
001	S1 and S4
011	S1
101	S4
110	S7

When the PPC403 finishes scanning column 0, it outputs 101 to D4:6 to the FPGA and scans column 1, then column 2. Software can specify the function of, and control, any key.

Table 7-10 describes the switch matrix scan.

Table 7-10. Switch Matrix Scan

PPC403 sends 011		PPC403 sends 101		PPC403 sends 110	
Data Read	Switch Pressed	Data Read	Switch Pressed	Data Read	Switch Pressed
000	S1, S4, S7	000	S2, S5, S8	000	S3, S6, S9
001	S1, S4	001	S2, S5	001	S3, S6
010	S1, S7	010	S2, S8	010	S3, S9
011	S1	011	S2	011	S3
100	S4, S7	100	S5, S8	100	S6, S9
101	S4	101	S5	101	S6
110	S7	110	S8	110	S9
111	None	111	None	111	None

7.2.4 Headers and Jumpers

Two 20-pin test headers (J6, J7), an 8-pin interface header (J1), and a 3-pin jumper (J5) are on the Ethernet daughterboard.

7.2.4.1 J1 Header Signals

J1 is not a test header. It connects to the ispLSI1024 FPGA for on-chip programming. Table 7-11 describes the J1 header signals.

Table 7-11. J1 Header Signals

Pin	Signal	Description
1	+5VD	Power
2	SDO	Serial data out
3	SDI	Serial data in
4	ispEN	Serial program enable
5	NC	
6	MODE	Mode
7	GND	Ground
8	SCLK	Serial clock

7.2.4.2 J6 Test Header Signals

Table 7-12 describes the J6 test header signals.

Table 7-12. J6 Test Header Signals

Pin	Signal	Description
1	27MHz	System clock
2	NC	
3	NC	
4	BACK	Bus acknowledge
5	BREQ	ST-NIC bus request
6	PRQ	Port request for remote DMA
7	CS	Chip select for daughterboard devices
8	E_ACK	Ethernet ACK signal
9	R/W	Read/write#
10	WBE0	Byte enable for D0:7
11	SWR	ST-NIC register write control signal
12	SRD	ST-NIC register read control signal

Table 7-12. J6 Test Header Signals

Pin	Signal	Description
13	SBA	Transceiver buffer latch control signal (B to A)
14	SAB	Transceiver buffer latch control signal (A to B)
15	\overline{G}	Transceiver buffer output enable
16	E_DIR	Transceiver buffer data flow direction control
17	\overline{RACK}	Read from transceiver buffer complete, from system to ST-NIC
18	\overline{WACK}	Write to transceiver buffer complete, from system to ST-NIC
19	$\overline{ETHERNETCS}$	ST-NIC chip select for register access
20	GND	Ground

On test header J7, pins 19 and 20 select AUI/TPI mode. For AUI mode, leave pins 19 and 20 open; for TPI mode, short pin 19 and 20.

7.2.4.3 J7 Test Header Signals

Table 7-13 describe the J7 test header signals.

Table 7-13. J7 Test Header Signals

Pin	Signal	Description
1	20MHZ	ST-NIC clock
2	NC	
3	NC	
4	\overline{PRD}	Transceiver buffer read from ST-NIC
5	\overline{PWR}	Transceiver buffer write from ST-NIC
6	EXADDR26	System address bus
7	EXADDR27	System address bus
8	EXADDR28	System address bus
9	EXADDR29	System address bus
10	EXADDR30	System address bus
11	EXADDR31	System address bus
12	EAD7	ST-NIC local address and data bus
13	EAD6	ST-NIC local address and data bus
14	EAD5	ST-NIC local address and data bus

Table 7-13. J7 Test Header Signals

Pin	Signal	Description
15	EAD4	ST-NIC local address and data bus
16	EAD3	ST-NIC local address and data bus
17	EAD2	ST-NIC local address and data bus
18	EAD1	ST-NIC local address and data bus
19	EAD0	ST-NIC local address and data bus
20	GND	Ground

7.2.4.4 J5 Jumper Settings

The 3-pin jumper J5 extends packet memory. Pins 1 and 2 are shorted to support the 8KB of installed packet memory. If packet memory is expanded to 16KB or 32KB, pins 2 and 3 should be shorted.

Table 7-14 shows the J5 pin settings.

Table 7-14. J5 Jumper Setting

Memory Size	8Kb x 8	16Kb x 8 or 32Kb x 8
Jumper J5	1 and 2	2 and 3

7.2.4.5 J8 Jumper Settings

The 2-pin jumper J8 selects the Ethernet physical layer interface.

If J8 is open, the 10Base2 Ethernet connector is active.

If J8 is shorted, the 10BaseT Ethernet connector is active.

Chapter 8. STBRP Connectors, Switches, and Jumpers

The STB Reference Platform (STBRP) provides a variety of connectors, switches, and jumpers.

This chapter describes the connectors and their associated signals, and provides functional descriptions of the switches and jumpers.

Figure 8-1 shows the location of the connectors, switches, and jumpers.

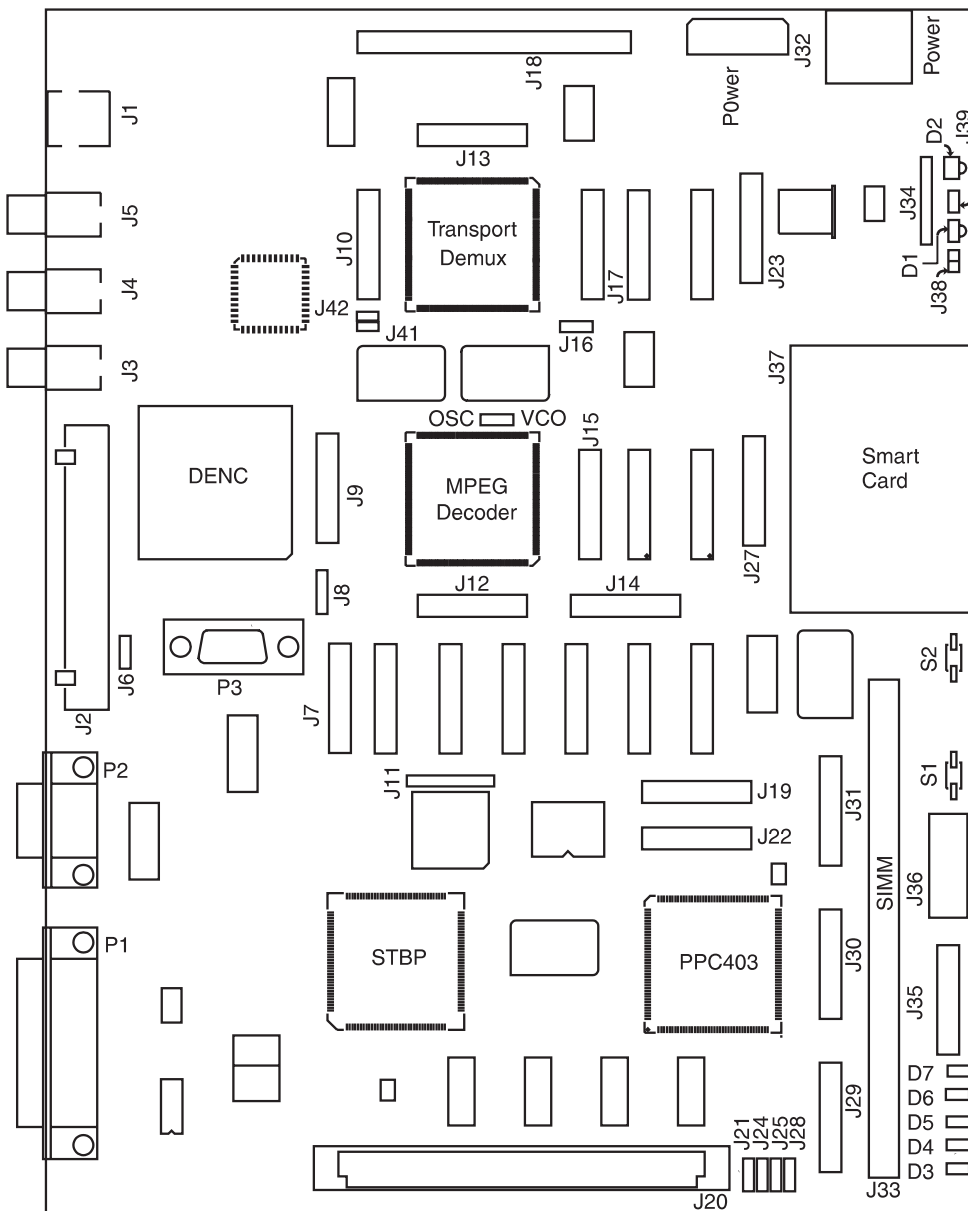


Figure 8-1. STBRP Board Layout

8.1 Serial Port Connectors

Lines from the PPC403 and STBP are buffered to RS-232 levels and connected to 9-pin D-connectors (P2, P3), as shown in Figure 8-2.

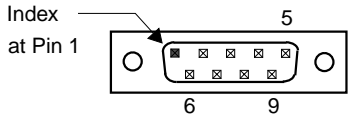


Figure 8-2. Serial Port Connectors (P2, P3)

The PPC403 serial port connects to P2; the STBP serial port connects to P3.

Table 8-1 describes the signal-to-pin assignments for the serial port connectors.

Table 8-1. Serial Port Connector (P2, P3) Pins

Pin	Signal	Description	Pin	Signal	Description
1	CD	Carrier detect	6	DSR	Data set ready
2	RXD	Transmit data	7	RTS	Request to send
3	TXD	Receive data	8	CTS	Clear to send
4	DTR	Data terminal ready	9	—	No connect
5	GND	Ground			

8.2 IEEE 1284 Port Connector

The IEEE 1284 parallel port (P1) uses a standard 25-pin D-connector (P1), as shown in Figure 8-3.

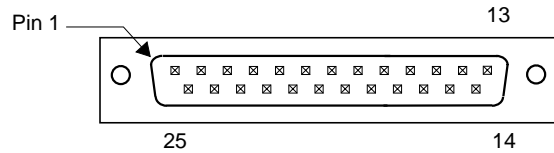


Figure 8-3. IEEE 1284 Port Connector (P1)

Table 8-2 shows the signal-to-pin assignments for the IEEE 1284 connector.

Table 8-2. IEEE 1284 Port Connector (P1) Pins

Pin	Signal	Pin	Signal
1	STROBE	14	AUTO-FD
2	PD7	15	ERROR
3	PD6	16	INIT

Table 8-2. IEEE 1284 Port Connector (P1) Pins

Pin	Signal	Pin	Signal
4	PD5	17	Selectin
5	PD4	18	GND
6	PD3	19	GND
7	PD2	20	GND
8	PD1	21	GND
9	PD0	22	GND
10	$\overline{\text{ACK}}$	23	GND
11	BUSY	24	GND
12	PE	25	GND
13	SELECT		

8.3 SCART Connector

The STBRP provides a composite video broadcast signal (SCART, also called Peritel) connector (J2), as shown in Figure 8-4. The SCART connector provides the following TV SCART pinout:

- Red, green, blue (RGB)
- Composite video broadcast signal (CVBS)
- Luma/chroma (Y/C)
- Blanking or fast switch
- Audio L and R
- Slow switch/4:3 and 16:9

The outputs from the digital video encoder (DENC) drive the RGB, CVBS and Y/C pins. The audio DAC drives the audio left (L) and right (R) pins.

Pin 8 of the SCART connector is defined in the EN50049-1 European SCART standard to be a function switch or slow switch. Level 0 (0V–2V) is named television-broadcast reproduction. Level 1A (4.5V–7V) indicates reproduction of an external source with an aspect ratio of 16:9. Level 1B (9.5V–12V) indicates peritelevision reproduction.

Pin 8 is connected by jumper shunts to +5V, +12V, or ground on STBRP header J6. Tie pin 1 to pin 2 for 16:9 aspect ratio operation and tie pin 2 to pin 3 for peritelevision; no jumpers are used for 4:3 aspect ratio operation. If peritelevision is not required in the STB design, a GPIO pin can drive pin 8. In such a design, a buffer must boost the 3V output high level of the STBP to the 4.5V minimum level for 16:9 aspect ratio operation.

Pin 16 is the blanking control. A 0V–0.4V range is a logical 0; +1V–3V is a logical 1. Pin 16 is connected to a GPIO pin on the STBP ASIC.

Pin 20, a video input pin, is left unconnected because this design provides no video input circuitry. Pin 18, the video input return, is also left unconnected. Pins 10 and 12 are intercommunication data lines; no connections are permitted. Pin 14, the blanking return, is connected to digital ground.

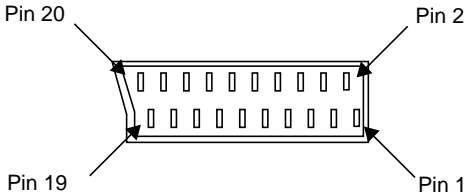


Figure 8-4. SCART Connector (J2)

Table 8-3 shows the signal-to-pin assignments for the SCART connector.

Table 8-3. SCART Connector (J2) Pins

Pin	Signal	Pin	Signal
1	Audio out R	11	GREEN
2	NC	12	NC
3	Audio out L	13	RED ground
4	Audio ground	14	Digital ground
5	BLUE ground	15	RED
6	NC	16	GPIO
7	BLUE	17	Video ground
8	+5v, +12V or GND	18	NC
9	GREEN ground	19	Composite Video out
10	NC	20	NC

8.4 Composite Video, Left and Right Audio Connector

Three standard RCA jack connectors output composite video and L and R audio. The source of the signals are the video DENC and Audio DAC parts. J4 is audio left, J3 is audio right, and J5 is composite video.

8.5 S-Video Connector

The STBRP contains a standard S-video connector (J1) that connects to the Y/C outputs from the DENC.

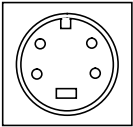


Figure 8-5. S-Video Connector (J1)

8.6 Expansion Connector

The expansion connector (J20), which enables the connection of a daughterboard, provides buffered processor address and data buses, clock, chip select, interrupt, read/write-direction control, data acknowledge, data write byte enables, IIC bus, and reset signals. The connector is a 96-pin Eurocard connector, as shown in Figure 8-6.

Timing and control of the expansion connector bus is identical to the PPC403, except for small timing differences caused by the use of buffered signals. The buffers protect logic on the main board against errors or possible misconnection of user-designed daughterboard logic.

The STBRP comes with an Ethernet daughterboard that also provides an array of nine user-input switches that can simulate an STB front panel.

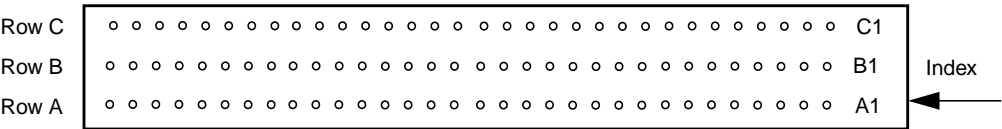


Figure 8-6. Expansion Connector (J20)

8.6.1 Expansion Connector Signals

Table 8-4 shows the signal-to-pin assignments for the expansion connector.

Table 8-4. Expansion Connector (J20) Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
A1	EXDATA0	B1	GND	C1	EXADDR8
A2	EXDATA1	B2	+12V	C2	EXADDR9
A3	EXDATA2	B3	GND	C3	EXADDR10
A4	EXDATA3	B4	GND	C4	EXINT2

Table 8-4. Expansion Connector (J20) Pin Assignments (cont.)

Pin	Signal	Pin	Signal	Pin	Signal
A5	EXDATA4	B5	$\overline{\text{EXDMAR1}}$	C5	EXADDR11
A6	EXDATA5	B6	GND	C6	EXADDR12
A7	EXDATA6	B7	GND	C7	EXADDR13
A8	EXDATA7	B8	$\overline{\text{EXWBE0}}$	C8	$\overline{\text{EXCS3}}$
A9	EXDATA8	B9	$\overline{\text{EXWBE1}}$	C9	EXADDR14
A10	EXDATA9	B10	GND	C10	EXADDR15
A11	EXDATA10	B11	GND	C11	EXADDR16
A12	EXDATA11	B12	GND	C12	$\overline{\text{EXRW}}$
A13	EXDATA12	B13	GND	C13	EXADDR17
A14	EXDATA13	B14	$\overline{\text{EXDMAA1}}$	C14	EXADDR18
A15	EXDATA14	B15	GND	C15	EXADDR19
A16	EXDATA15	B16	GND	C16	$\overline{\text{EXRESET}}$
A17	EXDATA16	B17	GND	C17	EXADDR20
A18	EXDATA17	B18	$\overline{\text{EXINT2}}$	C18	EXADDR21
A19	EXDATA18	B19	GND	C19	EXADDR22
A20	EXDATA19	B20	GND	C20	EXACK
A21	EXDATA20	B21	GND	C21	EXADDR23
A22	EXDATA21	B22	GND	C22	EXADDR24
A23	EXDATA22	B23	GND	C23	EXADDR25
A24	EXDATA23	B24	GND	C24	IIC_CLK
A25	EXDATA24	B25	GND	C25	EXADDR26
A26	EXDATA25	B26	GND	C26	EXADDR27
A27	EXDATA26	B27	GND	C27	EXADDR28
A28	EXDATA27	B28	GND	C28	EXADDR29
A29	EXDATA28	B29	GND	C29	EXADDR30
A30	EXDATA29	B30	+5VD	C30	EXADDR31
A31	EXDATA30	B31	+5VD	C31	IIC_DAT
A32	EXDATA31	B32	+5VD	C32	EX_27_MHz

8.6.1.1 EX-27_MHZ (5V input)

The 27 MHz PPC403 input clock is buffered and passed through the expansion connector as the signal EX-27_MHZ, which provides clocking for daughterboard logic that has less than 1.2 nanosecond (ns) delay from the processor input clock.

8.6.1.2 EXCS3 (5V input)

The PPC403 output $\overline{\text{CS3}}$ (pin 152) is buffered and passed through the expansion connector as the signal EXCS3, which provides a chip select and controls accesses to daughterboard logic.

Software sets $\overline{\text{EXCS3}}$ to define the base address and size allocated to the daughterboard.

8.6.1.3 EXRD/WR (5V input)

The PPC403 output $\overline{\text{R/W}}$ (pin 127) is buffered and passed through the expansion connector as the signal EXRD/WR, which controls the direction of data transfers.

$\overline{\text{EXRD/WR}}$, when high, indicates a processor read; when low, a processor write.

8.6.1.4 EXACK (5V output)

Daughterboard logic drives $\overline{\text{EXACK}}$ low to indicate either that read data is ready on the expansion data bus, or that data presented by the processor during a write cycle was latched or accepted by the expansion card logic.

STBRP control logic (an electronically programmable logic device (EPLD)) combines EXACK with other acknowledge signals and then drives the processor's PPC403 input READY (pin 13).

The control logic delay is less than 10 ns.

8.6.1.5 EXDMAR1 and EXDMAA1 (5V input)

A PPC403 DMA channel is connected to the expansion connector: $\overline{\text{DMAR1}}$ and $\overline{\text{DMAA1}}$ are buffered to become EXDMAR1 and EXDMAA1.

EXDMAA1 should only be used as an input to expansion card logic. $\overline{\text{EXDMAR1}}$, the request input to the PPC403, should be driven by daughterboard logic if a DMA operation is requested. The bidirectional PPC403 DMA-control pin associated with this DMA channel is buffered and connected to the EXTC1 pin on the expansion connector.

8.6.1.6 EXDATA(0:31) (5V input/output)

The PPC403 data bus is buffered to become EXDATA(0:31); EXDATA0 corresponds to D0 (pin 42) of the PPC403. These lines are presented as 3.3V logic levels when driven by the processor during writes, but are buffered with a 5V part to 5V logic levels.

The transceiver, a 5V part, can be driven by either 5V or 3.3V outputs from the daughterboard logic during processor reads.

Note that the processor data bus should only be driven by daughterboard logic only when $\overline{\text{EXCS3}}$ is active low and EXRD/WR is high, indicating a processor read from the daughterboard logic.

8.6.1.7 EXADDR(8:31) (5V input)

The PPC403 address bus is buffered to become the signals $\text{EXADDR}(8:31)$. $\text{EXADDR} 8$ corresponds to A8 (pin 94) of the PPC403. The address pins, buffered to 5V, are connected to the PPC403 DRAM SIMM and the expansion connector. $\text{EXADDR}(28:31)$ also connect to the transport stream connector.

$\text{EXADDR}(8:31)$, which are always driven by the PPC403, should be used only as inputs to daughterboard logic.

$\text{EXADDR}(30:31)$ are dual-function, for address or byte enable.

8.6.1.8 $\overline{\text{EXRESET}}$ (5V input)

The PPC403 signal $\overline{\text{RESET}}$ is buffered and passed through the expansion connector as $\overline{\text{EXRESET}}$.

$\overline{\text{EXRESET}}$, which is active at power-on or when using the reset switch, is driven at 5V.

$\overline{\text{EXRESET}}$ should be used only as an input to expansion card logic.

8.6.1.9 $\overline{\text{EXINT2}}$ (5V output)

The expansion connector output $\overline{\text{EXINT2}}$ is buffered and connected to $\overline{\text{INT2}}$ (pin 33) on the PPC403. $\overline{\text{EXINT2}}$ provides an interrupt mechanism for daughterboard logic.

This PPC403 interrupt, reserved for use by daughterboard logic, is not shared with any devices on the STBRP.

A 10K ohm pullup resistor is connected to this line on the STBRP.

8.6.1.10 $\overline{\text{EXWBE0}}$, $\overline{\text{EXWBE1}}$ (5V inputs)

The PPC403 byte enables $\overline{\text{WBE0}}$ (pin 122) and $\overline{\text{WBE1}}$ (pin 122) are buffered using an IDT162244 and passed through the expansion connector as $\overline{\text{EXWBE0}}$ and $\overline{\text{EXWBE1}}$. These pins, driven at 5V, should be used only as inputs to daughterboard logic.

8.6.1.11 IIC_DAT, IIC_CLK (Expansion input/output)

The I²C bus on the STBRP connects to the expansion connector as pins IIC_DAT and IIC_CLK. Note that these pins are not buffered.

I²C bus voltage levels are used.

8.7 DVB Transport Stream Connector

The DVB transport stream connector (J18) inputs sample DVB transport streams from the DVBE into the STBRP using the supplied 2 x 25 ribbon cable. The sample DVB transport streams enable users to decode and display samples of MPEG2 audio and video in NTSC or PAL format.

The DVB transport stream connector supports the connection of many types of transport stream front-ends using a cabled connection to a separate board or a direct connection to a daughterboard. Most newer transport chips are controlled using I²C, but some transport chips have 8-bit, control-data ports. For this reason, signals are connected so that both I²C control and parallel control can be used. The parallel control path is a buffered connection to the PPC403 data bus bits D0:7. Four buffered address lines and direction control, dedicated chip select, data ready, and reset signals are also provided.

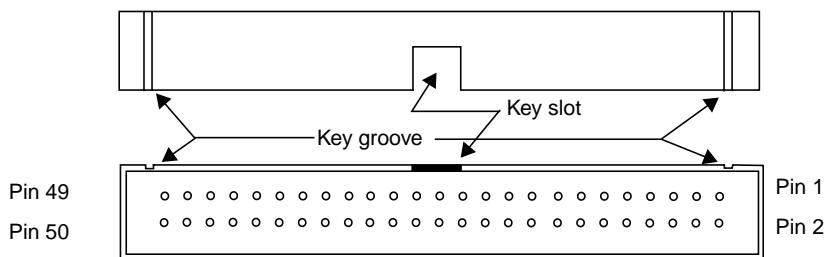


Figure 8-7. DVB Transport Stream Front-End Connector (J18)

8.7.1 DVB Transport Stream Connector Signals

Table 8-5 shows the signal-to-pin assignments for the transport stream front-end connector.

Table 8-5. DVB Transport Stream Connector Signals

Pin	Signal	Pin	Signal
1	NBPWN0	26	IIC_CLKT
2	+5VD	27	GND
3	EXADDR31	28	IIC_DATT
4	EXADDR30	29	GND
5	EXADDR29	30	
6	EXADDR28	31	EXINT4
7	NBLNBNF_PWR	32	GND
8	GND	33	
9	GND	34	GND
10	NBCHDATA0	35	NBACK/DS

Table 8-5. DVB Transport Stream Connector Signals

Pin	Signal	Pin	Signal
11	NBCHDATA1	36	GND
12	NBCHDATA2	37	$\overline{\text{EXRESET}}$
13	NBCHDATA3	38	GND
14	NBCHDATA4	39	$\overline{\text{EXR/W}}$
15	NBCHDATA5	40	$\overline{\text{BCS2}}$
16	NBCHDATA6	41	+12V
17	NBCHDATA7	42	GND
18	GND	43	TXDATA0
19	NBCHCLK	44	TXDATA1
20	NBCHDEN	45	TXDATA2
21	NBFE_ERROR	46	TXDATA3
22	GND	47	TXDATA4
23	GND	48	TXDATA5
24	+5VD	49	TXDATA6
25	+5VD	50	TXDATA7

8.7.1.1 NBCHDATA(0:7)

The NBCHDATA(0:7) data bus pins connect from the transport stream connector buffer to the transport stream input pins on the transport chip. The buffer adds a maximum delay of 1.2 nanoseconds. The signals NBCHDEN (for channel data enable), and NBCHCLK (channel clock) are buffered using an FET switch before connection to the transport chip.

8.7.1.2 NBLNBNF_PWR

NBLNBNF_PWR is buffered through an FET switch and connected to a general purpose I/O (GPIO) on the STPB (pin 52) as signal LBNBNF_PWR. NBLNBNF_PWR can be an input or an output; its most common use is as an input to the front-end logic that controls transponder selection.

8.7.1.3 NBFE_ERROR

NBFE_ERROR is buffered through a 3.3V SN74LVT16244ADGG buffer and connected to a GPIO on the STBP (pin 51) as signal FE_ERROR. NBFE_ERROR, an output from the front-end, can be driven to 5V levels because the receiving buffer is 5V-tolerant. NBFE_ERROR is expected to be used as the error input from the front-end, but the signal can also serve as a general-purpose input.

8.7.1.4 IIC_CLKT, IIC_DATT

The IIC_CLKT and IIC_DATT pins connect to the I²C bus pins on the STBRP. IIC_CLKT and IIC_DATT can connect to the I²C bus master on the transport chip, or to the I²C bus master on the STBP. Jumpers J41 and J42 control this selection.

If both jumpers are open, the connection is to the I²C bus master on the transport chip. If both jumpers are closed, the connection is to the I²C bus master on the STBP.

Note that the IIC connections to the expansion connectors are not buffered.

8.7.1.5 TXDATA(0:7)

The pins TXDATA(0:7) are bidirectional data pins that are connected to the PPC403 data bus bits D0:D7, through a 5V 74FCT162245 transceiver. The PPC403 drives these pins during write cycles (when $\overline{\text{EXR/W}}$ is low), and the expansion module should drive these lines during PPC403 reads from the expansion module indicated by $\overline{\text{EXR/W}}$ high and $\overline{\text{EXCS0}}$ active low. $\overline{\text{EXR/W}}$ is the buffered version (through a 5V 74FCT16244) of the PPC403 signal R/W. This pin is driven at 5V and should only be used as an input to expansion card logic.

8.7.1.6 $\overline{\text{BCS2}}$

$\overline{\text{BCS2}}$ is the buffered version of the PPC403 chip-select signal $\overline{\text{CS2}}$. $\overline{\text{BCS2}}$ is driven at 5V logic levels and should only be used as an input to expansion card logic.

8.7.1.7 EXADDR(28:31)

EXADDR(28:31) are buffered PPC403 address bits. These pins, driven at 5V, should be used only as inputs to daughterboard logic for address decode purposes.

8.7.1.8 $\overline{\text{EXINT4}}$

The input signal $\overline{\text{EXINT4}}$ from the front-end connector is buffered with a 5V IDT162244 and then connected to the $\overline{\text{INT4}}$ (pin35) on the PPC403. This signal provides an interrupt mechanism for expansion board logic. This interrupt is reserved for the front-end module and is not shared with any other main board devices. A 10k-ohm pullup resistor is connected to this line on the main board.

8.7.1.9 $\overline{\text{NBACK/DS}}$

$\overline{\text{NBACK/DS}}$ is buffered through an FET switch and then into the control-logic EPLD as signal ACK/DS on pin 3. This is a 3.3V or 5V output from the front-end module that can be used to acknowledge and end processor accesses to the front-end module. The EPLD combines acknowledge signals from the expansion module, front-end module, and STBP to drive the PPC403 ready input.

8.7.1.10 NBPWN0

NBPWN0 is buffered through an FET switch and then connected to the pulse-width modulator pin on the STBP.

8.7.1.11 EXRESET

The PPC403 $\overline{\text{RESET}}$ signal is buffered through an 74FCT16244 and passed through the expansion connector as $\overline{\text{EXRESET}}$. This signal is active on power up or via the on-board push-button.

$\overline{\text{EXRESET}}$, driven at 5V, and should be used only as an input to daughterboard logic.

8.8 RISCWatch and RISCTrace Connectors

The 16-pin RISCWatch JTAG connector (J35) and 20-pin RISCTrace connector (J36) attach to a JTAG debugger to support hardware debug and software development.

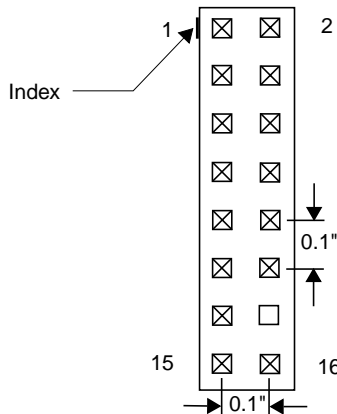


Figure 8-8. JTAG Header (J35) (Top View)

Table 8-6 lists the signal names and positions on the JTAG header.

Table 8-6. JTAG Interface (J35) Connections and Resistors

Header Pin	Signal	Description	PPC403 Pin	Board Resistors ¹
1	TDO	JTAG test data out	16	
2	NC	To be left unconnected		
3	TDI	JTAG test data in	8	10K Ω PU
4	NC	To be left unconnected		
5	NC	To be left unconnected		
6	+POWER ²	Power (status signal, not processor V_{DD})		1K Ω SR ³
7	TCK	JTAG test clock	6	10K Ω PU
8	NC	To be left unconnected		
9	TMS	JTAG test mode select	7	10K Ω PU

Table 8-6. JTAG Interface (J35) Connections and Resistors

Header Pin	Signal	Description	PPC403 Pin	Board Resistors ¹
10	NC	To be left unconnected		
11	HALT	Sreset_RISCWATCH (not Processor)	9	10K Ω PU
12	NC	To be left unconnected		
13	NC	To be left unconnected		
14	KEY	Pin in this position should be removed.		
15	NC	To be left unconnected		
16	GND	Ground		

¹PU = pullup resistor, SR = series resistor

²The +POWER signal is sourced from the STBRP and is used as a reference signal. It should not be the power signal supplied to the PPC403 (either+ 3.3V or +5V). This signal does not supply power to the RISCWatch hardware.

³This 1K ohm series resistor provides short current-limiting protection only. If the resistor is present, it should be 1K ohm or less.

Figure 8-9 illustrates the RISCTrace connector.

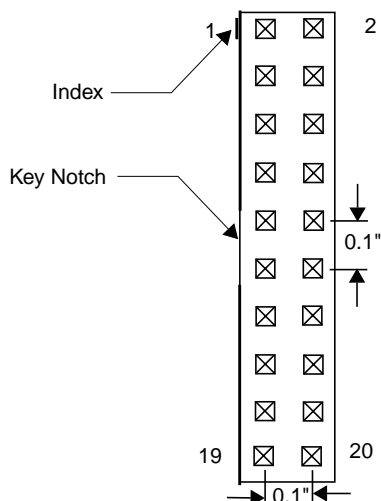


Figure 8-9. RISCTrace Connector (J36) (Top View)

Table 8-7 lists the RISCTrace connector pins.

Table 8-7. RISCTrace Connector Pins(J36)

Pin	Signal	Pin	Signal
1	NC	11	NC
2	NC	12	CN
3	SysClk3	13	TS0
4	NC	14	ST1
5	CN	15	TS2
6	NC	15	TS3
7	NC	17	TS4
8	NC	18	TC5
9	NC	19	TS6
10	NC	20	GND

8.9 Auxiliary Power Connector

Figure 8-10 shows the PC-type four-position, large power-connector (J32) that provides power to the STBRP.

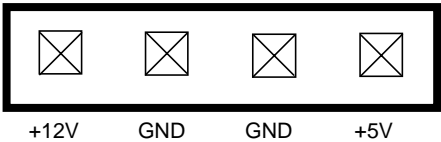


Figure 8-10. Power Connector (J32)

The STBRP can be powered using a stand-alone power supply that provides PC-type output connectors. The pins consist of 2 grounds, +5V, and +12V. Typically, the provided power supply is used, connected using the provided line cord to the J40 power connector.

8.10 I²C Expansion Connector (J8)

The STBRP provides a 4-pin I²C expansion connector (header J8, shown in Figure 8-11) that can connect an additional IIC device to the on-board I²C main bus.

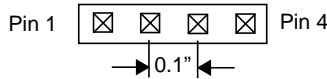


Figure 8-11. IIC Expansion Connector (J8)

Table 8-8 shows the signal-to-pin assignments for the I²C expansion connector.

Table 8-8. I²C Expansion Connector (J8) Pins

Pin	Signal	Pin	Signal
1	+5V	3	IIC_DAT
2	IIC_CLK	4	Ground

8.11 STBRP Test Headers

The board contains many 20-pin test headers that can be used during debug for hookups to logic analyzers and other external test equipment. These connectors (J7, J9, J10, J12–J15, J17, J19, J22, J23, J27, J29, J30 and J31) connect to most of the buses, clocks, and control signals on the STBRP. In addition, the four GPIOs from the STBP ASIC connect to on-board LEDs that are also connected to one of these test headers. Each header also contains a ground pin. The pinouts for these headers are shown in Tables 8-9 through 8-23 and in the schematics.

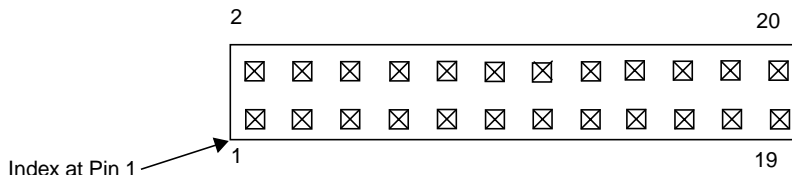


Figure 8-12. 20-Pin Test Header

Table 8-9 shows the signal-to-pin assignments for the test header J7.

Table 8-9. CD21 DRAM Data Bus Test Header (J7)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	CD2_MD56	CD21 DRAM data
2	NC		12	CD2_MD55	CD21 DRAM data
3	NC		13	CD2_MD54	CD21 DRAM data
4	CD2_MD63	CD21 DRAM data	14	CD2_MD53	CD21 DRAM data

Table 8-9. CD21 DRAM Data Bus Test Header (J7)

Pin	Signal	Description	Pin	Signal	Description
4	CD2_MD62	CD21 DRAM data	15	CD2_MD52	CD21 DRAM data
6	CD2_MD61	CD21 DRAM data	16	CD2_MD51	CD21 DRAM data
7	CD2_MD60	CD21 DRAM data	17	CD2_MD50	CD21 DRAM data
8	CD2_MD59	CD21 DRAM data	18	CD2_MD49	CD21 DRAM data
9	CD2_MD58	CD21 DRAM data	19	CD2_MD48	CD21 DRAM data
10	CD2_MD57	CD21 DRAM data	20	GND	Ground

Table 8-10. YUV Data Bus Test Header (J9)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	MP_YUV8	YUV data
2	NC		12	MP_YUV7	YUV data
3	NC		13	MP_YUV6	YUV data
4	MP_YUV15	YUV data	14	MP_YUV5	YUV data
5	MP_YUV14	YUV data	15	MP_YUV4	YUV data
6	MP_YUV13	YUV data	16	MP_YUV3	YUV data
7	MP_YUV12	YUV data	17	MP_YUV2	YUV data
8	MP_YUV11	YUV data	18	MP_YUV1	YUV data
9	MP_YUV10	YUV data	19	MP_YUV0	YUV data
10	MP_YUV9	YUV data	20	GND	Ground

Table 8-11. System Address Bus Test Header (J10)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	A23	System Address
2	NC		12	A24	System Address
3	NC		13	A25	System Address
4	A16		14	A26	System Address
5	A17	System Address	15	A27	System Address
6	A18	System Address	16	A28	System Address
7	A19	System Address	17	A29	System Address

Table 8-11. System Address Bus Test Header (J10)

Pin	Signal	Description	Pin	Signal	Description
8	A20	System Address	18	A30	System Address
9	A21	System Address	19	A31	System Address
10	A22	System Address	20	GND	Ground

Table 8-12. CD21 DRAM Data Bus Test Header (J12)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	CD2_MD40	CD21 DRAM data
2	NC		12	CD2_MD39	CD21 DRAM data
3	NC		13	CD2_MD38	CD21 DRAM data
4	CD2_MD47	CD21 DRAM data	14	CD2_MD37	CD21 DRAM data
5	CD2_MD46	CD21 DRAM data	15	CD2_MD36	CD21 DRAM data
6	CD2_MD45	CD21 DRAM data	16	CD2_MD35	CD21 DRAM data
7	CD2_MD44	CD21 DRAM data	17	CD2_MD34	CD21 DRAM data
8	CD2_MD43	CD21 DRAM data	18	CD2_MD33	CD21 DRAM data
9	CD2_MD42	CD21 DRAM data	19	CD2_MD32	CD21 DRAM data
10	CD2_MD41	CD21 DRAM data	20	GND	Ground

Table 8-13. System Data Bus (D0:15) Test Header (J13)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	D7	System data bus
2	NC		12	D8	System data bus
3	NC		13	D9	System data bus
4	D0	System data bus	14	D10	System data bus
5	D1	System data bus	15	D11	System data bus
6	D2	System data bus	16	D12	System data bus
7	D3	System data bus	17	D13	System data bus
8	D4	System data bus	18	D14	System data bus
9	D5	System data bus	19	D15	System data bus
10	D6	System data bus	20	GND	Ground

Table 8-14. CD21 Control Test Header (J14)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	HSYNC	HSYNC from DENC
2	NC		12	$\overline{\text{CD2_WT0}}$	CD21 DRAM write enable 0
3	NC		13	$\overline{\text{CD2_WT1}}$	CD21 DRAM write enable 1
4	VDREQ#	Video data request	14	$\overline{\text{CD2_WT2}}$	CD21 DRAM write enable 2
5	ADREQ#	Audio data request	15	$\overline{\text{CD2_WT3}}$	CD21 DRAM write enable 3
6	SERDAT	Serial data in	16	$\overline{\text{CD2_CAS1}}$	CD21 DRAM CAS1
7	SERCLK	Serial clock in	17	$\overline{\text{CD2_CAS0}}$	CD21 DRAM CAS0
8	ACLK_CD2	Audio clock in	18	$\overline{\text{CD2_MOE}}$	CD21 DRAM output enable
9	ADAT_CD2	Audio data in	19	$\overline{\text{CD2_RAS}}$	CD21 DRAM RAS
10	$\overline{\text{VSYNC}}$	VSYNC from DENC	20	GND	Ground

Table 8-15. CD21 DRAM Data Bus Test Header (J15)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	CD2_MD8	CD21 DRAM data
2	NC		12	CD2_MD7	CD21 DRAM data
3	NC		13	CD2_MD6	CD21 DRAM data
4	CD2_MD15	CD21 DRAM data	14	CD2_MD5	CD21 DRAM data
5	CD2_MD14	CD21 DRAM data	15	CD2_MD4	CD21 DRAM data
6	CD2_MD13	CD21 DRAM data	16	CD2_MD3	CD21 DRAM data
7	CD2_MD12	CD21 DRAM data	17	CD2_MD2	CD21 DRAM data
8	CD2_MD11	CD21 DRAM data	18	CD2_MD1	CD21 DRAM data
9	CD2_MD10	CD21 DRAM data	19	CD2_MD0	CD21 DRAM data
10	CD2_MD9	CD21 DRAM data	20	GND	Ground

Table 8-16. Transport DRAM Data Bus Test Header (J17)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	TMD8	Transport DRAM data
2	NC		12	TMD7	Transport DRAM data

Table 8-16. Transport DRAM Data Bus Test Header (J17)

Pin	Signal	Description	Pin	Signal	Description
3	NC		13	TMD6	Transport DRAM data
4	TMD15	Transport DRAM data	14	TMD5	Transport DRAM data
5	TMD14	Transport DRAM data	15	TMD4	Transport DRAM data
6	TMD13	Transport DRAM data	16	TMD3	Transport DRAM data
7	TMD12	Transport DRAM data	17	TMD2	Transport DRAM data
8	TMD11	Transport DRAM data	18	TMD1	Transport DRAM data
9	TMD10	Transport DRAM data	19	TMD0	Transport DRAM data
10	TMD9	Transport DRAM data	20	GND	Ground

Table 8-17. CD21 DRAM Data Bus Test Header (J19)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	CD2_MD24	CD21 DRAM data
2	NC		12	CD2_MD23	CD21 DRAM data
3	NC		13	CD2_MD22	CD21 DRAM data
4	CD2_MD31	CD21 DRAM data	14	CD2_MD21	CD21 DRAM data
5	CD2_MD30	CD21 DRAM data	15	CD2_MD20	CD21 DRAM data
6	CD2_MD29	CD21 DRAM data	16	CD2_MD19	CD21 DRAM data
7	CD2_MD28	CD21 DRAM data	17	CD2_MD18	CD21 DRAM data
8	CD2_MD27	CD21 DRAM data	18	CD2_MD17	CD21 DRAM data
9	CD2_MD26	CD21 DRAM data	19	CD2_MD16	CD21 DRAM data
10	CD2_MD25	CD21 DRAM data	20	GND	Ground

Table 8-18. PPC403 Control Signals Test Header (J22)

Pin	Signal	Description	Pin	Signal	Description
1	ASIC_27MHZ	27 MHz clock output from STBP	11	$\overline{\text{CS}}3$	Chip select 3
2			12	$\overline{\text{CS}}4$	Chip select 4
3			13	$\overline{\text{CS}}5$	Chip select 5
4	$\overline{\text{MWE}}$	System DRAM write enable	14	$\overline{\text{CS}}6$	Chip select 6
5	$\overline{\text{MOE}}$	System DRAM output enable	15	$\overline{\text{MRAS}}0$	System DRAM RAS0

Table 8-18. PPC403 Control Signals Test Header (J22)

Pin	Signal	Description	Pin	Signal	Description
6	\overline{OE}	SRAM output enable	16	$\overline{MCAS0}$	System DRAM CAS0
7	$\overline{R/W}$	SRAM read/write	17	$\overline{MCAS1}$	System DRAM CAS1
8	$\overline{CS0}$	Chip select 0	18	$\overline{MCAS2}$	System DRAM CAS2
9	$\overline{CS1}$	Chip select 1	19	$\overline{MCAS3}$	System DRAM CAS3
10	$\overline{CS2}$	Chip select 2	20	GND	Ground

Table 8-19. Transport DRAM Signals Test Header (J23)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	TMA1	
2	NC		12	TMA0	Transport DRAM address
3	NC		13	NC	
4	TMA8	Transport DRAM address	14	NC	
5	TMA7	Transport DRAM address	15	$\overline{T_MWE}$	Transport DRAM write enable
6	TMA6	Transport DRAM address	16	$\overline{T_MRAS1}$	Transport DRAM RAS1
7	TMA5	Transport DRAM address	17	$\overline{T_MRAS0}$	Transport DRAM RAS0
8	TMA4	Transport DRAM address	18	$\overline{T_MCASH}$	Transport DRAM CASH
9	TMA3	Transport DRAM address	19	$\overline{T_MCASL}$	Transport DRAM CASL
10	TMA2	Transport DRAM address	29	GND	Ground

Table 8-20. PPC403 Controls/CD21 DRAM Address Bus Test Header (J27)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	CD2_MA8	CD 21 DRAM address
2	NC		12	CD2_MA7	CD 21 DRAM address
3	NC		13	CD2_MA6	CD 21 DRAM address
4	NC		14	CD2_MA5	CD 21 DRAM address
5	\overline{RESET}	PPC403 system reset signal	15	CD2_MA4	CD 21 DRAM address
6	$\overline{WBE0}$	PPC403 byte enable 0	16	CD2_MA3	CD 21 DRAM address
7	$\overline{WBE1}$	PPC403 byte enable 1	17	CD2_MA2	CD 21 DRAM address
8	$\overline{WBE2}$	PPC403 byte enable 2	18	CD2_MA1	CD 21 DRAM address

Table 8-20. PPC403 Controls/CD21 DRAM Address Bus Test Header (J27)

Pin	Signal	Description	Pin	Signal	Description
9	WBE3	PPC403 byte enable 3	19	CD2_MA0	CD 21 DRAM address
10	CD2_MA8	CD 21 DRAM address	20	Ground	

Table 8-21. Ready and Interrupt Signals Test Header (J29)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	INT4	Interrupt 4
2	NC		12	INT3	Interrupt 3
3	NC		13	INT2	Interrupt 2
4	NC		14	INT1	Interrupt 1
5	X_READY	PPC403 ready	15	INT0	Interrupt 0
6	ACK_EXP	Expansion card ready	16	LED8	LED 8
7	NC		17	LED7	LED 7
8	VES_ACK	Transport IC ready	18	LED6	LED 6
9	ASIC_READY	ATB_ASIC ready	19	LED5	LED 5
10	NC		20	GND	Ground

Table 8-22. System Address Bus (and Others) Test Header (J30)

Pin	Signal	Description	Pin	Signal	Description
1	27MHZ	27 MHz system clock	11	A7	System address
2	NC		12	A8	System address
3	NC		13	A9	System address
4	NC		14	A10	System address
5	NC		15	A11	System address
6	NC		16	A12	System address
7	NC		17	A13	System address
8	TTXDAT	Teletext data	18	A14	System address
9	TTXREQ	Teletext request	19	A15	System address
10	A6	System address	20	GND	Ground

Table 8-23. System Data Bus (D16:31) Test Header (J31)

Pin	Signal	Description	Pin	Signal	Description
1	NC		11	D23	System data
2	NC		12	D24	System data
3	NC		13	D25	System data
4	D16	System data	14	D26	System data
5	D17	System data	15	D27	System data
6	D18	System data	16	D28	System data
7	D19	System data	17	D29	System data
8	D20	System data	18	D30	System data
9	D21	System data	19	D31	System data
10	D22	System data	20	GND	Ground

8.12 Setting the STBRP Jumpers

Eight jumpers are on the STBRP; Table 8-24 describes their functions and settings.

Table 8-24. Jumper Settings

Jumper	Function	Settings
J6	Function switch for SCART connector	Jumper 1 to 2: BIPIO Jumper 2 to 3: peritelevision open for 4:3
J21	Memory bus width select	Jumper 1 to 2: 32-bit memory bus Jumper 2 to 3: 16-bit memory bus
J24	Memory bus width select	Jumper 1 to 2: 32-bit memory bus Jumper 2 to 3: 16-bit memory bus
J25	Memory bus width select	Jumper 1 to 2: 32-bit memory bus Jumper 2 to 3: 16-bit memory bus
J28	Memory bus width select	Jumper 1 to 2: 32-bit memory bus Jumper 2 to 3: 16-bit memory bus
J38	IrDA on-board LEDs	Short: select on-board LED
J39	IrDA on-board LEDs	Short: select on-board LED
OSC/VCO	Clock select	See Chapter 9, "STBRP Board Clock Distribution"
J41	I ² C bus master select	See Section 8.7.1.4
J41	I ² C bus master select	See Section 8.7.1.4

8.13 Indicator Lamps

Table 8-25 describes the STBRP indicator lamps.

Table 8-25. Indicator Lamps

Lamp, Color	Name	Function
D1, clear	IR, photo diode	RX
D2, red	IR, LED	TX
D3, clear	Power	When on, indicates power is supplied to STBRP.
D4, clear		Programmable
D5, clear		Programmable
D6, clear		Programmable
D7, clear		Programmable

8.14 Switches

The STBRP has two push-button switches:

- S1, which resets the STBRP
- S2, which sends a critical interrupt to the PPC403 interrupt handler

Chapter 9. STBRP Board Clock Distribution

The transport chip causes the companion 27 MHz voltage control crystal oscillator (VCXO) to lock to clocking information in the transport stream. The locked clock is passed to the MPEG decoder chip, and to the STBP when the jumper (OSC/VCO) is set between pin 1 and pin 2, as shown in Figure 9-1.

An alternate approach for board clocking is to isolate the rest of the board logic from the phase-locked clock. If the alternate approach is desired, the 27 MHz free-running clock oscillator serve as the source clock. The default jumper setting, from pin 1 to pin 2, is changed to jumper from pin 2 to pin 3.

Figure 9-1 illustrates the STBRP clock distribution.

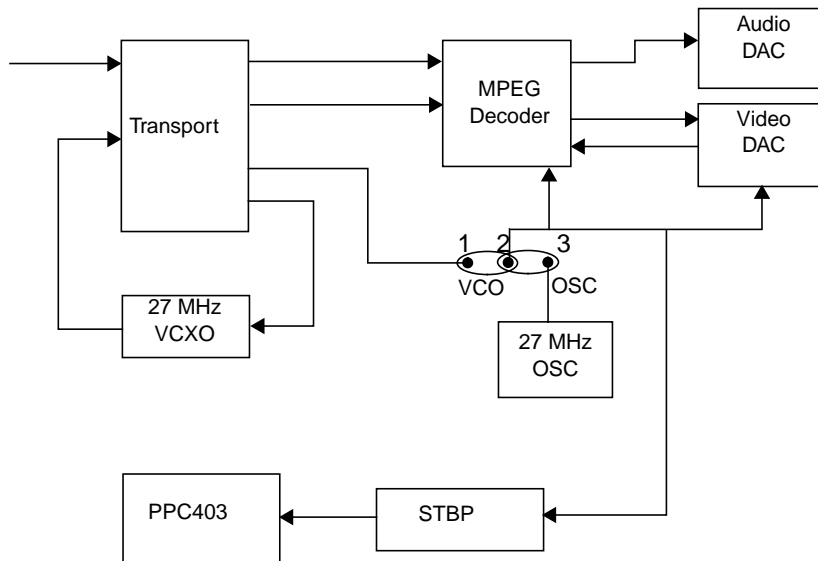


Figure 9-1. Clocking Diagram

Chapter 10. Using the ROM Monitor

This chapter describes the STB Reference Platform (STBRP) ROM Monitor program. This ROM-resident program, which performs board- and chip-level initialization, provides a user interface menu that supports board diagnostics, program downloads, and access to ROM-resident demos.

10.1 Accessing the ROM Monitor

The ROM Monitor runs as part of the boot code in the on-board flash memory. The ROM Monitor expects an emulated VT100 type ASCII display, attached to serial port 1 (P3) on the STBRP, to be running on the host PC. The line protocol parameters are 9600 baud, eight bits per character, no parity, and one stop bit.

After the terminal emulator is configured properly, a user can access the ROM Monitor menus, use the ping test, and load applications onto the STBRP, or run preloaded ROM-resident demos.

The STBRP downloads applications using Ethernet. The STBRP also supports the downloading of programs using P3, if the terminal emulator supports Kermit binary file transfers.

10.2 Monitor Selections and Submenus

The main menu, shown in Section 10.2.1, is displayed after the STBRP is reset and the ROM Monitor initializes. Note that some values in the sample menus, in particular the ROM Monitor version, IP addresses, and the hardware address of the Ethernet daughterboard, can differ from those shown. Note that in the menus, the local IP address is the IP address assigned to the STBRP; the remote IP is address assigned to the host PC.

Each menu option is described in subsequent sections.

Option 8 saves configuration changes and enables changed values to persist through subsequent power-ons and resets. The ROM Monitor stores configuration data in flash memory.

10.2.1 ROM Monitor Main Menu

The following menu is displayed after the board has been reset:

```
403GC 1.4 ROM Monitor (3/27/97)
----- System Info -----
Processor speed = 27 MHz
Bus speed      = 27 MHz
Amount of DRAM  = 16 MBytes
-----
--- Device Configuration ---
```

```

Power-On Test Devices:
 000 Enabled System Memory [RAM]
 001 Disabled Ethernet [ENET]
-----
Boot Sources:
 001 Disabled Ethernet [ENET]
                        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
 002 Enabled OS9K STB Image [OS9K]
 003 Enabled PSOS STB Image [PSOS]
 004 Enabled PSOS/PROBE BOOT [PROBE]
 005 Disabled Serial Port 1 [S1]
                        Baud = 9600
-----
Debugger : Disabled
-----
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Not used
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->

```

10.2.2 Selecting Power-On Tests

Option 1 in the main menu selects power-on tests. These tests run when the menu exits and before the ROM loader begins **bootp** processing.

```

 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Not Used
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->1

```

Selecting option 1 displays the following submenu:

```
--- ENABLE AND DISABLE POWER-ON TESTS ---
Power-On Test Devices:
    000 Enabled System Memory [RAM]
    001 Disabled Ethernet [ENET]
-----
select device to change ->
```

Selecting a test toggles its status. For example, because the System Memory test is enabled in the preceding menu, selecting 0 at the prompt disables it.

```
select device to change ->0                [Selects system memory]
```

After the selection is made, the new setting is displayed, followed by the main menu.

```
select device to change ->0
- [RAM] test is disabled                [Message describing change]
```

```
Power-On Test Devices:
    000 Disabled System Memory [RAM]
    001 Disabled Ethernet [ENET]
-----
Boot Sources:
    001 Enabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
    002 Enabled OS9K STB Image [OS9K]
    003 Enabled PSOS STB Image [PSOS]
    004 Enabled PSOS/PROBE BOOT [PROBE]
    005 Disabled Serial Port 1 [S1]
        Baud = 9600
-----
Debugger : Disabled
-----
    1 - Enable/disable tests
    2 - Enable/disable boot devices
    3 - Change IP addresses
    4 - Ping test
    5 - Not used
    6 - Toggle automatic menu
    7 - Display configuration
    8 - Save changes to configuration
    9 - Set baud rate for s1 boot
    0 - Exit menu and continue
->
```

Use Option 8 to save configuration changes before exiting the menu, if desired. Otherwise, changes are lost when you exit the menu or when the STBRP is reset.

10.2.3 Selecting a Boot Device

Select Option 2 in the main menu to enable or disable a boot device.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->2
```

Selecting option 2 displays the following submenu:

```
--- ENABLE AND DISABLE BOOT DEVICES ---
Boot Sources:
  001 Enabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
  002 Enabled OS9K STB Image [OS9K]
  003 Enabled PSOS STB Image [PSOS]
  004 Enabled PSOS/PROBE BOOT [PROBE]
  005 Enabled Serial Port 1 [S1]
        Baud = 9600
-----
Debugger : Disabled
-----
select device to change ->
```

Selecting a device toggles its boot status. Selecting 5, for example, disables Serial Port 1 as a boot device.

```
select device to change ->5          [Selects serial port]
```

After the selection has been made, the new setting is displayed, followed by the main menu.

```
select device to change ->5
  [S1] boot is disabled          [Message describing change]
```

```
Power-On Test Devices:
  000 Disabled System Memory [RAM]
  001 Disabled Ethernet [ENET]
-----
Boot Sources:
  001 Disabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
  002 Enabled OS9K STB Image [OS9K]
  003 Enabled PSOS STB Image [PSOS]
  004 Enabled PSOS/PROBE BOOT [PROBE]
```

```

005 Disabled Serial Port 1 [S1]
      Baud = 9600
-----
Debugger : Disabled
-----
    1 - Enable/disable tests
    2 - Enable/disable boot devices
    3 - Change IP addresses
    4 - Ping test
    5 - Not used
    6 - Toggle automatic menu
    7 - Display configuration
    8 - Save changes to configuration
    9 - Set baud rate for s1 boot
    0 - Exit menu and continue
->

```

When the user selects option 0 and exits from the monitor menu, the ROM Monitor tries to boot an application image using the enabled boot sources in their listed order. In the preceding example, the ROM Monitor would try to boot over Ethernet, which is the first enabled boot source. If more than one boot source is enabled, and an attempt to boot over the first enabled device fails, the ROM Monitor tries to boot using the next enabled device.

10.2.4 Changing IP Addresses

Option 3 in the main menu enables users to change the IP addresses of the STBRP and the host PC. These addresses are used for **bootp** processing and the host connectivity ping test.

Note: The “local” IP address is that of the STBRP; the remote IP address is that of the host PC. The IP addresses must match those set during host PC configuration.

```

    1 - Enable/disable tests
    2 - Enable/disable boot devices
    3 - Change IP addresses
    4 - Ping test
    5 - Not used
    6 - Toggle automatic menu
    7 - Display configuration
    8 - Save changes to configuration
    9 - Set baud rate for s1 boot
    0 - Exit menu and continue
->3

```

When option 3 is selected, the following submenu is displayed:

```

--- CHANGE IP ADDRESS ---
Device List:
    001 Enabled Ethernet [ENET]
           local=7.1.1.5 remote=7.1.1.4 hwaddr=1000abcdef55
-----

```

select device to change ->

Select the appropriate device:

select device to change ->1 [Selects Ethernet]

When a valid device is selected, the following submenu is displayed:

- 1 - Change local address
- 2 - Change remote address
- 0 - Return to main menu

->

Make the appropriate selection. To change the IP address of the STBRP, would select option 1, Change local address:

->1 [Selects the local address]
Current IP address = (7.1.1.5 [Displays the current value]
Enter new IP address -> Enter IP address in dot notation (e. g., 8.1.1.2)

Now enter the new IP address in dotted decimal notation:

7.1.1.5

After the selection is entered, the new configuration is displayed, followed by the main menu:

Power-On Test Devices:

- 000 Disabled System Memory [RAM]
- 001 Disabled Ethernet [ENET]

Boot Sources:

- 001 Disabled Ethernet [ENET]
local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
- 002 Enabled OS9K STB Image [OS9K]
- 003 Enabled PSOS STB Image [PSOS]
- 004 Enabled PSOS/PROBE BOOT [PROBE]
- 005 Disabled Serial Port 1 [S1]
Baud = 9600

Debugger : Disabled

- 1 - Enable/disable tests
- 2 - Enable/disable boot devices
- 3 - Change IP addresses
- 4 - Ping test
- 5 - Not used
- 6 - Toggle automatic menu
- 7 - Display configuration
- 8 - Save changes to configuration
- 9 - Set baud rate for s1 boot
- 0 - Exit menu and continue

->

Repeat option 3 as required to set all IP addresses to appropriate values.

Use option 8 to save the configuration changes.

10.2.5 Using the Ping Test

Option 4 in the main menu selects the ping test. The ping test should be performed after setting IP addresses to perform a basic test of host PC-to-STBRP connectivity. If the ping test fails, application cannot be downloaded to the STBRP. Local and remote IP addresses are used as the source and destination of the ICMP ping packets.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->4
```

When option 4 is selected, the current configuration is displayed, followed by another command prompt:

```
--- PING TEST ---
Device List:
    001 Enabled Ethernet [ENET]
           local=7.1.1.83 remote=7.1.1.38 hwaddr=081122334459
-----
select device to ping ->
```

Select the appropriate device to ping (in this case only Ethernet is enabled):

```
select device to ping ->1 [selects the Ethernet port]
```

If the board can ping the host, a message similar to the following should appear:

```
Using [ENET] to ping. press any key to stop.
PING 7.1.1.4 56 data bytes
78 bytes from 7.1.1.4: icmp_seq=0 ttl=255 time=2 ms
78 bytes from 7.1.1.4: icmp_seq=2 ttl=255 time=1 ms
```

Hitting any key terminates the ping test and displays the ping statistics report and the main menu.

```
--- 7.1.1.4 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
```

```
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->
```

If the ping test fails:

- Verify that the cables are connected properly.
- Verify TCP/IP is running on the host.
- Verify that the local and remote IP addresses are set correctly. The local IP address should be that of the STBRP and the remote IP address should be that of the host PC.

Note: The ROM Monitor does not respond to an inbound ping test from the host PC unless the ROM Monitor ping test is active on the STBRP at the same time.

10.2.6 Disabling and Enabling the Automatic Menu Display

Option 6 in the main menu toggles the automatic ROM Monitor menu display when the STBRP boots.

After option 6 is selected and the configuration is saved (using Option 8), menu display continues to function until the user exits from the main menu. After the next power-on or reset, however, the ROM Monitor menu is not displayed. This downloads the application automatically; no menu input is required. This feature also enables a user to download an application when no cable is connected to SP1 (that is, without a terminal emulator).

If automatic menu display is disabled, and a terminal emulator is attached, pressing any key during the first five seconds of the STBRP boot displays the ROM Monitor main menu. Otherwise, application download processing starts without displaying the main menu.

10.2.7 Displaying the Current Configuration

Option 7 displays the current configuration, followed by the main menu.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->7
Power-On Test Devices:
000 Disabled System Memory [RAM]
001 Disabled Ethernet [ENET]
-----
```

```

Boot Sources:
 001 Disabled Ethernet [ENET]
           local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
 002 Enabled OS9K STB Image [OS9K]
 003 Enabled PSOS STB Image [PSOS]
 004 Enabled PSOS/PROBE BOOT [PROBE]
 005 Disabled Serial Port 1 [S1]
           Baud = 9600
-----
Debugger : Disabled
-----
 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Not used
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->

```

When a selected menu operation alters configuration settings, the current configuration is automatically redisplayed.

10.2.8 Saving the Current Configuration

Option 8 saves the current configuration for subsequent power-ons and resets, displays a confirmation message, and displays the main menu:

```

 1 - Enable/disable tests
 2 - Enable/disable boot devices
 3 - Change IP addresses
 4 - Ping test
 5 - Not used
 6 - Toggle automatic menu
 7 - Display configuration
 8 - Save changes to configuration
 9 - Set baud rate for s1 boot
 0 - Exit menu and continue
->8

```

Selecting option 8 saves the current configuration

The configuration is saved in flash memory on the STBRP and retained until a new configuration is saved.

10.2.9 Setting the Baud Rate for Boots over SP1

Option 9 sets the baud rate used by SP1 when it is used as a device to download applications. Using SP1 to download applications requires a VT100 terminal emulator that supports Kermit binary file transfers. The Windows Terminal emulator performs Kermit binary file transfers, but the baud rate is limited to 19200. To use serial port 1 as a device to download programs at speeds up to 115200 baud, you can download a Kermit terminal emulator available as shareware from the <http://www.columbia.edu/kermit> Internet site.

```
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->9
Select a baud rate for S1 boot
1 -          9600
2 -         19200
3 -         28800
4 -         38400
5 -         57600
6 -        115200
=>4
```

After the new baud rate is selected, the ROM Monitor displays the new configuration and the main menu.

```
--- Device Configuration ---
Power-On Test Devices:
000 Enabled System Memory [RAM]
001 Disabled Ethernet [ENET]
-----
Boot Sources:
001 Disabled Ethernet [ENET]
      local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
002 Disabled OS9K STB Image [OS9K]
003 Disabled PSOS STB Image [PSOS]
004 Disabled PSOS/PROBE BOOT [PROBE]
005 Enabled Serial Port 1 [S1]
      Baud = 38400
-----
Debugger : Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
```

```

3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->

```

Use Option 8 to save the selected speed after reset and power-on.

10.2.10 Performing a Boot over SP1

To perform an S1 boot, the terminal emulator must support Kermit file transfer. The file must be a valid boot image and must be sent in binary mode. If the selected SP1 baud rate is not 9600, the terminal emulator must set to run at the selected SP1 baud rate before it downloads the file, and reset to 9600 baud after the download finishes.

The following example shows downloading of the file **usr_samp.img** file:

```

--- Device Configuration ---
Power-On Test Devices:
  000 Enabled System Memory [RAM]
  001 Disabled Ethernet [ENET]
-----
Boot Sources:
  001 Disabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
  002 Disabled OS9K STB Image [OS9K]
  003 Disabled PSOS STB Image [PSOS]
  004 Disabled PSOS/PROBE BOOT [PROBE]
  005 Enabled Serial Port 1 [S1]
        Baud = 38400
-----
Debugger : Disabled
-----
1 - Enable/disable tests
2 - Enable/disable boot devices
3 - Change IP addresses
4 - Ping test
5 - Not used
6 - Toggle automatic menu
7 - Display configuration
8 - Save changes to configuration
9 - Set baud rate for s1 boot
0 - Exit menu and continue
->0
Booting from [S1] Serial Port 1...

PLEASE NOTE: You must now...

```

- a. Exit from terminal emulation mode
- b. Modify the baud rate of your host session
- c. Transmit a file to the target in binary mode
- d. Reset the host baud rate to 9600
- e. Reenter terminal emulation mode
- f. Hit enter to execute the downloaded program

Get into terminal emulator command mode, change the line speed to match that selected using option 9, and set the file format to binary, as shown in the following sample session.

```
^\\c
(Back at waterdeep)
C-Kermit>set speed 38400
/dev/tty0, 38400 bps
C-Kermit>set file type bin
```

You can now load the file.

```
C-Kermit>send usr_samp.img
SF
Type escape character (^\\) followed by:
X to cancel file, CR to resend current packet
Z to cancel group, A for status report
E to send Error packet, Ctrl-C to quit immediately:

Sending: usr_samp.img => USR_SAMP.IMG
Size: 164864, Type: binary
.....
.....
.....
.... [OK]
ZB
```

When loading is completed, you must change the baud rate back to 9600 bps before continuing.

```
C-Kermit>set speed 9600
/dev/tty0, 9600 bps
```

After setting the baud rate to 9600 bps, reconnect to the terminal emulator and press Enter to complete the download.

```
C-Kermit>con
Connecting to /dev/tty0, speed 9600.
The escape character is Ctrl-\\ (ASCII 28, FS)
Type the escape character followed by C to get back,
or followed by ? to see other options

Loaded successfully ...
Entry point at 0x22f20 ...

Hello user!
```

Your ROM Monitor version is : 2.1

Your Evaluation Board has 33554432 bytes of DRAM installed.

Your Ethernet controller's network address is : 123456777777

usr_samp done!

10.2.11 Loading the Demo Programs

The ROM Monitor contains pSOS and OS-9000 demo programs that can be executed from the ROM Monitor main menu using the same mechanism used to boot from a device.

To load a demo program, specify the desired demo program version as the first enabled boot device and exit the ROM Monitor. The pSOS PSP and OS-9000 PSP demo programs are described in loaded as described in Chapter 13, "Using the Demo Program," describes the demo program.

After the demo program finishes, reset the STBRP to access the ROM Monitor main menu.

10.2.11.1 Running the pSOS Demo Program

To run the pSOS demo program, enable the pSOS boot image as the boot source and exit the ROM Monitor.

```
--- Device Configuration ---
Power-On Test Devices:
  000 Enabled System Memory [RAM]
  001 Disabled Ethernet [ENET]
-----
Boot Sources:
  001 Disabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
  002 Disabled OS9K STB Image [OS9K]
  003 Enabled PSOS STB Image [PSOS]
  004 Disabled PSOS/PROBE BOOT [PROBE]
  005 Disabled Serial Port 1 [S1]
        Baud = 38400
-----
Debugger : Disabled
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Not used
  6 - Toggle automatic menu
  7 - Display configuration
  8 - Save changes to configuration
  9 - Set baud rate for s1 boot
```

```
    0 - Exit menu and continue
->0
Booting from [PSOS] PSOS STB Image...
```

```
Standard output device initialized...
System clock initialized...
Diagnostics menu:
ch      - Control Command Help
dh      - DENC diagnostic menu
eh      - Serial EEPROM diagnostics menu
h       - Display this menu
ih      - Infrared diagnostics menu
lh      - Debug log command help
mh      - MPEG diagnostics menu
ph      - Parallel port diagnostics menu
sh      - Smart card diagnostics menu
th      - Transport demux command help
uh      - Modem diagnostics menu
kt      - Key Pad test
ap      - Demo Application
```

Enter one of the above choices or 'h' to see this menu again
>

10.2.11.2 Running the OS-9000 Demo Program

To run the OS-9000 demo program, enable the OS-9000 boot image as the boot source and exit the ROM Monitor.

```
--- Device Configuration ---
Power-On Test Devices:
  000 Enabled System Memory [RAM]
  001 Disabled Ethernet [ENET]
-----
Boot Sources:
  001 Disabled Ethernet [ENET]
        local=7.1.1.5 remote=7.1.1.4 hwaddr=081122334459
  002 Enabled OS9K STB Image [OS9K]
  003 Disabled PSOS STB Image [PSOS]
  004 Disabled PSOS/PROBE BOOT [PROBE]
  005 Disabled Serial Port 1 [S1]
        Baud = 38400
-----
Debugger : Disabled
-----
  1 - Enable/disable tests
  2 - Enable/disable boot devices
  3 - Change IP addresses
  4 - Ping test
  5 - Not used
```



```

    6 - Toggle automatic menu
    7 - Display configuration
    8 - Save changes to configuration
    9 - Set baud rate for s1 boot
    0 - Exit menu and continue
->0
Booting from [OS9K] OS9K STB Image...

OS-9000 Bootstrap for the PowerPC(tm)

Now trying to Boot loaded system in-place.
Now searching memory ($00140000 - $002fffff) for an OS-9000 Kernel...

An OS-9000 kernel was found at $00140000
A valid OS-9000 bootfile was found.
[1]$

```

10.2.12 Booting from the pSOS/pROBE Boot Image

The ROM Monitor contains a pSOS/pROBE ROM image that enables attachment of the STBRP to the SingleStep debugger from SDS. This debugger provides task-aware source level debugging for pSOS.

To boot from this image, enable the ROM Monitor PSOS/PROBE BOOT device, disable all other ROM Monitor boot devices, and exit the ROM Monitor.

```

--- Device Configuration ---
Power-On Test Devices:
    000 Disabled System Memory [RAM]
    001 Disabled Ethernet [ENET]
-----
Boot Sources:
    001 Disabled Ethernet [ENET]
           local=7.1.1.4 remote=7.1.1.5 hwaddr=001020304055
    002 Disabled OS9K STB Image [OS9K]
    003 Disabled PSOS STB Image [PSOS]
    004 Enabled PSOS/PROBE BOOT [PROBE]
    005 Disabled Serial Port 1 [S1]
-----
Debugger: Disabled
-----
    1 - Enable/disable tests
    2 - Enable/disable boot devices
    3 - Change IP addresses
    4 - Ping test
    5 - Not Used
    6 - Toggle automatic menu
    7 - Display configuration
    8 - Save changes to configuration
    9 - Set baud rate for s1 boot

```

```

    0 - Exit menu and continue
->0
Booting from [PROBE] PSOS/PROBE BOOT...

pSOSystem V2.1.3
Copyright (c) 1991 - 1996, Integrated Systems, Inc.
-----
--
STARTUP MODE:
    Boot into pROBE+ standalone mode
NETWORK INTERFACE PARAMETERS:
    LAN interface is disabled
HARDWARE PARAMETERS:
    Serial channels will use a baud rate of 9600
    This board's Ethernet hardware address is 0:1:CC:DD:EE:F0
    After board is reset, startup code will wait 60 seconds
-----
To change any of this, press any key within 60 seconds

```

Section 11.1.3, “Debugging the STBRP Application using SingleStep,” on p. 11-2, provides more instructions for attaching the SingleStep debugger to the STBRP.

10.2.13 Exiting the Main Menu

Option 0 exits the ROM Monitor main menu, leaving the ROM Monitor active.

If a debugger is active before option 0 is selected, the ROM Monitor waits for the debugger to start on the host PC.

In all other cases, option 0 causes the ROM Monitor to try to load an application from the host PC to the STBRP over the enabled boot devices. When loaded successfully, the application is executed.

Chapter 11. pSOS Platform Support Package

The Set-Top Box (STB) Reference Design Kit provides software, based on the pSOS™ real-time operating system (RTOS) and OptIC™ (pSOS drawing package for embedded applications), that runs on the STB Reference Platform (STBRP). This software, called the pSOS Platform Support Package (PSP), is provided as a set of source code and a set of executable files.

This chapter describes how to modify pSOS for PowerPC v. 2.1.3 for STB application development on a host PC, and how to use the pSOS PSP.

Note: To use the pSOS PSP, pSOS v 2.1.3, SDS™ v. 7.1, and Diab Data v. 4.0 must be installed on the host PC. Only limited-time evaluation copies of pSOS, Diab Data, and SDS are included in the STB Reference Design Kit. The pSOS PSP provides only limited OptIC functionality

pSOS and OptIC can be purchased from Integrated Systems Corporation. Contact Integrated Systems Corporation for more information

The pSOS PSP contains all of the device drivers needed to develop applications for the STBRP. A supplied demo program serves as a starting point for development and evaluation.

The reader needs to be familiar with pSOS and OptIC.

11.1 Integrating the pSOS PSP

The pSOS PSP must be carefully integrated with a base install of pSOS to run on the STBRP. After the integration is complete, the resulting pSOS executable image runs on the STBRP.

Alternatively, the sample image provided with the pSOS PSP can be executed on the STBRP. The sample compiled application file, named **ram.elf**, can be found in the **lib** subdirectory.

11.1.1 Loading the Boot Code

After a hardware reset, the PPC403 fetches a jump instruction at address 0xFFFFF0C in the STBRP flash boot block. This instruction branches to the first instruction of boot code. An intermediate program, the ROM Monitor, resides in flash memory and supports the loading of pSOS executable images. The ROM Monitor receives control out of reset. Chapter 10, "Using the ROM Monitor," describes the ROM Monitor.

The process for loading images is the same as that described Chapter 10.

11.1.2 Modifying the Base pSOS Installation

The following software packages must be installed from the diskettes and CD-ROMs provided with the STBRP.

- STBRP device drivers
- STBRP board support package
- Diab Data compiler and linker
- pSOS system
- SingleStep debugger

The Diab Data, pSOS, and SingleStep packages require license keys to install and operate. Some keys may have been included in the STBRP. The other necessary keys can be obtained from Integrated Systems, Inc. (ISI).

Note: Refer to the pSOS PSP README files installed in the pSOS PSP directories for information about keys included in the pSOS PSP and obtaining keys from ISI. The README files also include detailed installation and usage instructions for each software package provided with the STB Reference Design Kit.

After the pSOS PSP packages are installed, they must be configured to work correctly with each other and the STBRP. This is accomplished by running the **stbsetup.bat** program installed in the pSOS PSP root directory. Refer to the README files in the pSOS PSP root directory for more information about using the **stbsetup.bat** program.

After all of the software packages have been configured, the sample STBRP application can be built by changing to the **app** subdirectory and typing the command **make**. This creates the following files:

- **ram.elf**, the ELF executable application image used by the debugger.
- **ram.hex**, the application in S-record format, suitable for download using the pSOS/pROBE **tftp** loader.
- **ram.img**, the application in boot image format, suitable for download using the ROM Monitor.
- **ram.map**, an ASCII file containing the addresses of the global objects in the application.

11.1.3 Debugging the STBRP Application using SingleStep

The sample STBRP application is compiled with optimization turned on. This results in faster execution of the sample code but makes the application more difficult to debug. To turn compiler optimization off, edit the file **Makefile** in the **app** directory to remove the **-O** flag from the **COMP_OPTS** line. Execute **make clean** and **make** commands for the removal of the option to take effect.

11.1.3.1 Configuring pROBE on the STBRP

To debug the STBRP application using SingleStep, boot the pSOS/pROBE image in the pSOS PSP. Section 10.2.12, “Bootting from the pSOS/pROBE Boot Image,” on p. 10-15, describes how to boot using the pSOS/pROBE image. The pROBE connection information must be configured before connecting to the debugger.

To configure the connection information, press a key on the terminal emulator within 60 seconds after the pROBE boot information is displayed. Then:

1. Type **m** and press Enter to modify the information.
2. Type **3** and press Enter to select debugging over Ethernet.
3. Type **y** and press Enter to configure a LAN interface.
4. Enter the IP address of the STBRP (for example, 7.1.1.5) and press Enter.

Press Enter after each of the subsequent prompts (except for the STBRP Ethernet hardware address prompt) to accept the default configuration information. You may also want to reduce the start up time to 5 seconds instead of 60 seconds.

Type **y** and at the prompt and press Enter to configure the Ethernet hardware address. The hardware address is a 12-digit hexadecimal string printed on the sticker attached to the smart card reader on the STBRP. This address must be entered one byte (2 hexadecimal digits) at a time, starting at byte 0, which is the left most byte.

After the configuration information is correct, type **c** and press Enter to continue when prompted. The configuration information is saved in the STBRP flash chip (this takes about five seconds). A message indicates that the STBRP is ready to connect to the debugger.

11.1.3.2 Configuring a SingleStep Debug Session

A SingleStep debug session must be configured by entering information in all of the tabs in the Debug window, as illustrated in the following procedure:

1. Start the SingleStep to Probe application and open the Debug window.
2. Select the application to debug. Click the File tab and type the name of the file to debug, for example **c:\psosstb\app\ram.elf**.
3. Configure the network connection. Click the Connection tab, select the Network Host button, enter the IP address for the board (for example, **7.1.1.5**), and select the UDP button.
4. Select the processor. Click the Processor tab and select **PPC403** from the processor selection list.
5. Configure the other options. Click the Options tab and ensure that all Loading options are selected, and that no Executing options are selected.
6. Configure the logging options. Click the Logging tab and select the Log to Screen (always) button.
7. Press Enter to start the session.

The SingleStep debugger creates the temporary files needed to debug the selected application, loads the application in the STBRP DRAM and initializes the application, stopping at the application entry point in **root.c**. This process can take a while, depending on the complexity and size of the image and the speed of the host PC. After the application is been initialized, it can be started and debugged using the Run (green light) and other debugger buttons. Refer to the SingleStep and pSOS documentation CD-ROM for more information about debugging pSOS applications.

11.1.3.3 Configuring the Sample Application

The sample application is configured by modifying the file **sys_conf.h** in the **include** subdirectory.

Initially, the application is configured to be loaded and run using the SingleStep debugger. To build a standalone image that can be loaded and run without using the debugger, modify the file **sys_conf.h** as follows:

1. Change the **SC_SD_PARAMETERS** parameter to **SYS_CONF**.
2. Change the **SE_DEBUG_MODE** parameter to **DBG_AP**.
3. Change the **SC_PROBE**, **SC_PROBE_DISASM**, **SC_PROBE_CIE**, **SC_PROBE_QUERY**, **SC_PROBE_DEBUG**, and **SC_PNA** parameters to **NO**.

For more information about the **sys_conf.h** file parameters, refer to the pSOS documentation CD-ROM.

11.2 The pSOS Device Drivers and System Files

The device drivers provided in the pSOS PSP are standard pSOS drivers that return to the caller only after the called function finishes.

The pSOS driver interface provides the following entry points: **de_init()**, **de_open()**, **de_close()**, **de_read()**, **de_write()**, and **de_cntrl()**. When a driver does not require one of these entry points, a stub function that returns an error is included in its place.

11.2.1 Flash Memory Device Driver

The flash memory driver provides write access to the AMD Am29F016 2MB flash memory device, along with routines to reset the flash, erase a flash sector, or erase the entire flash memory. Other functions obtain the manufacturer and device codes. The application task formats and manages the flash memory.

Note: Use caution when exercising the flash device driver to avoid overwriting the STBRP software in the flash memory.

The **nvr_task()** task provides an interface to the flash memory device driver.

11.2.1.1 Testing the Flash Device Driver

To test the flash memory, run the **FLASH_DIAGNOSTICS** I/O control code against the flash memory device.

11.2.1.2 Flash Device Driver Functions

The following table describes flash device driver functions.

Device Driver Function	Description
flash_init()	Creates a flash memory device driver semaphore used during read, write, and control operations to ensure exclusive access to the flash memory. If a semaphore cannot be created, an error is returned.
flash_open()	Ensures exclusive access before writing to the flash or performing control functions. If exclusive access is denied, an error is returned.
flash_close()	Releases exclusive access after writing to the flash or performing control functions. If exclusive access is not released, an error is returned.
flash_read()	Reads a flash address location.
flash_write()	Writes <i>n</i> consecutive bytes to the flash starting at a specified address. flash_write() requires the following parameters: number of bytes to be written, destination address, and a pointer to the data.
flash_cntrl()	Performs status and control functions, such as returning the manufacturer or device codes, resetting the flash, erasing individual flash sectors, or erasing the entire flash. When flash_cntrl() is called, an opcode, passed as an argument, determines which control function is executed.

11.2.1.2.1 flash_read()

A single-byte read operation is performed by writing the following command sequence and then reading data from the requested address:

```
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0xF0
read data from flash source address
```

Multibyte read operations can be performed directly against the flash device without using the device driver **flash_read()** function.

11.2.1.2.2 **flash_write()**

A write operation (byte program) is performed by writing the following command sequence followed by writing the source byte to the requested address:

```
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0xA0
write source data to flash destination address
```

When the DQ7 bit is equal to the most significant bit just written, the write is complete. During a write operation, attempting to read the device results in the complement of the data most recently written to DQ7.

11.2.1.2.3 **FLASH_ERASE_ALL I/O Control Code**

The **FLASH_ERASE_ALL** control code erases all flash sectors that are not hardware-protected. This is accomplished by writing the following command sequence:

```
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0x80
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0x10
```

Polling DQ7 indicates when a chip erase operation is complete. During a chip erase operation, an attempt to read the flash produces 0 at the DQ7 output. After a chip erase operation finishes, 0xFF is returned for all data read from flash.

11.2.1.2.4 **FLASH_ERASE_SECTOR I/O Control Code**

The **FLASH_ERASE_SECTOR** control code erases one of the 32 flash sectors. This is accomplished by writing the following command sequence:

```
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0x80
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash sector address data 0x30
```

Polling DQ7 indicates when a sector erase operation is complete. During a sector erase operation, an attempt to read the device produces a 0 at the DQ7 output. After a sector erase operation finishes, attempting to read the flash produces a 1 at the DQ7 output.

11.2.1.2.5 GET_MFG_CODE I/O Control Code

The **GET_MFG_CODE** control code returns the manufacturer code for AMD, which is 0x01. The control code is obtained by writing the following command sequence:

```
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0x90
read data from flash address 0x0000 to get back 0x01
```

11.2.1.2.6 GET_DEVICE_CODE I/O Control Code

The **GET_DEVICE_CODE** control code returns the device code for the Am29F016 flash, which is 0xAD. The device code is obtained by writing the following command sequence:

```
write to flash address 0x5555 data 0xAA
write to flash address 0x2AAA data 0x55
write to flash address 0x5555 data 0x90
read data from flash address 0x0001 to get back 0xAD
```

11.2.1.2.7 RESET_FLASH I/O Control Code

The **RESET_FLASH** control code resets the flash by writing the following command sequence:

```
write to flash address 0x5555 data 0xF0
```

11.2.1.3 Flash Device Driver Source Code Files

The following table lists the directories and file names for the flash device driver source code.

Directory and File Name	Description
app\flashdvr.c	Flash device driver and diagnostic program.
include\flashdvr.h	Flash device driver global definitions.
app\nvr_task.c	Sample flash device driver interface task.

11.2.2 Transport Demultiplexer Device Driver

The transport demultiplexer device driver provides the interface between the application program and the VLSI VES2020 MPEG-2 transport demultiplexer. The transport demultiplexer device driver supports high-level control of the demultiplexing of an incoming transport stream, manages queues for receiving MPEG-2 PES data (audio and video streams) and PSI (DVB SI sections), and delivers the sections to the application program for special processing.

xpr_task() provides the interface to the transport demultiplexer device driver.

The application software uses device driver calls to initiate, setup, and control the transport demultiplexer, including requests for stream data from the device.

When PSI data is available at the transport device, the transport interrupt service routine (ISR) performs initial processing of the data and forwards the information to the transport task for intermediate processing.

The transport task collects data from the transport circular queues and forwards the data to the application software using pSOS messages.

The application processes the transport data and, if necessary, modifies the setup of the transport device.

The file **xprdrv.h** defines the device driver interface and the format of the messages that return data to the application.

The transport demultiplexer supports data collection on 15 circular queues, excluding the internal RISC engine message queue and the high-speed data port pseudo-queue. Four queues are reserved for video, audio, teletext, and subtitles, leaving 11 queues for application-defined data.

The application must allocate transport DRAM for each of the transport circular queues, except for the messaging queue, which is allocated at system initialization. Each queue can be initialized only once. The base software reserves 512KB of the 1MB transport DRAM for the queues.

Each transport queue can receive data from one or more PIDs. The transport demultiplexer driver and the transport demultiplexer support up to 32 PID definitions. A PID definition consists of:

- The PID to be demultiplexed
- The transport queue in which to place the data
- A set of buffers, allocated by the application, for delivering transport data to the application
- A set of up to four section prefilter definitions
- The pSOS message queue ID that receives the transport data

Up to 32 delivery buffers can be defined. The multi-buffer scheme allows for temporary delays in the processing of the data by the application. Each block of data delivered to the application uses an available buffer. The application releases the buffer back to the driver after processing.

The following steps are performed during the internal initialization of the transport demultiplexer driver and the transport device:

1. Create a pSOS semaphore to guarantee exclusive device driver access.
2. Reset the transport device and configure transport memory.
3. Set up the ISR for the transport demultiplexer, and enable the transport interrupt.

4. Run a memory test on the transport DRAM.
5. Initialize video and audio operation.
6. Download the RISC microcode into the transport DRAM and initialize the microcode data tables.
7. Set up a transport queue to receive internal messages from the RISC microcode.
8. Start the RISC engine.

11.2.2.1 Testing the Transport Demultiplexer

Executing the demo program tests the transport demultiplexer device.

11.2.2.2 Transport Demultiplexer Device Driver Functions

The following table describes transport demultiplexer device driver functions.

Device Driver Function	Description
xpr_init()	Initializes the transport driver and device. This should only be called by the base software during start-up.
xpr_open()	Gains exclusive access to the transport device.
xpr_close()	Releases exclusive access to the transport device.
xpr_read()	Not implemented.
xpr_write()	Not implemented.
xpr_cntrl()	Manages transport queues, sets up PIDs to be demultiplexed, and changes channels (audio and video PIDs).

The following table lists the input/output (I/O) control codes for the transport demultiplexer device driver.

I/O Control Code	Description
XPR_CMD_DOWNLOAD_UCODE	Stops RISC, downloads, starts RISC.
XPR_CMD_DIAG	Main diagnostic/test command.
XPR_CMD_INIT_QUEUE	Initializes queue size.
XPR_CMD_SETUP_PID	Sets up a PID.
XPR_CMD_STOP_PID	Turns off a PID.
XPR_CMD_START_PID	Turns on a PID.
XPR_CMD_STOP_QUEUE	Turns off all PIDs on a queue.

I/O Control Code	Description
XPR_CMD_START_QUEUE	Turns on all PIDs on a queue.
XPR_CMD_STOP_CHANNEL	Turns off current channel A/V.
XPR_CMD_START_CHANNEL	Sets up new channel and starts it.
XPR_CMD_RELEASE_BUF	Signals that the application is done with section buffer.
XPR_CMD_FORWARD_MSG	Signals that the application wants Messaging Queue messages.

11.2.2.3 Transport Demultiplexer Device Driver Source Code Files

The pSOS PSP does not provide source code, which is proprietary, for the transport demultiplexer device driver. Instead, the pSOS PSP provides the required binary code.

The following table lists the directory and file names for the transport demultiplexer device driver and transport demultiplexer task include files.

Directory and File Name	Description
app\xpdrv.h	Upper layer transport demultiplexer device driver.
include\xpdrv.h	Transport demultiplexer device driver global definitions.

11.2.2.4 Transport Demultiplexer Device Driver Module

The file **liblv2020.o** is the transport demultiplexer device driver module.

11.2.3 MPEG Audio/Video Decoder Device Driver

This driver provides an interface to the IBM MPEGCD21 audio/video (A/V) decoder.

The MPEGCD21 registers are memory-mapped using the PPC403 Bank Register BR5. The define **MPEG_BASE_ADDR** in the definition file provides the address of the first register. The DRAM associated with the chip can only be accessed via the MPEG audio/video decoder registers. User read/write access to both registers and attached DRAM is provided via the **de_cntrl()** interface. **dcc_task()** task processes close caption events associated with audio/video device driver.

The driver initializes the MPEG hardware and performs setup and control operations for MPEG video and audio data streams. The MPEG chip synchronizes audio and video presentation internally; the driver need only specify the method used and provide a method of loading the system time clock with an accurate value. System clock recovery is provided externally by the transport driver.

An associated ISR handles interrupt requests by determining the cause of the interrupt and uses the pSOS message mechanism to dispatch appropriate code. Except for event

logging, the only message traffic generated by the ISR is the indication of “top or even” vertical sync. This message is sent to the A/V task to control closed-caption processing.

The OpTIC driver also accesses registers and memory on the MPEGCD21. Exclusion is handled as necessary by routines in the driver implementation files.

11.2.3.1 Testing the MPEG A/V Decoder

Executing the demo program tests the MPEG A/V decoder.

11.2.3.2 MPEG A/V Decoder Device Driver Functions

The following table describes the MPEG A/V decoder device driver functions.

Device Driver Function	Description
dcc_init()	Verifies communication with the chip. Specifies the initial configuration (NTSC, 4 x 3, default OSD size, 90 KHz clock), resets the chip, loads microcode for video and audio sections, and installs the interrupt handler.
dcc_open()	Not implemented.
dcc_close()	Not implemented.
dcc_read()	Not implemented.
dcc_write()	Not implemented.
dcc_cntrl()	Performs various control functions. When called, an opcode passed as an argument, determines which control function is run.

The following table lists the I/O control codes for the MPEG audio/video decoder device driver.

I/O Control Code	Description
DCC_SET_CFG	Set decoder configuration.
DCC_GET_CFG	Retrieves decoder configuration.
DCC_RESET	Resets the chip.
DCC_AUDIO_MUTE	Stops audio output but continues decode.
DCC_VIDEO_BLANK	Stops video output but continues decode.
DCC_VIDEO_FREEZE	Displays the current frame.
DCC_DECODE_STOP	Stops decode of audio, video, or both.
DCC_DECODE_START	Stops decode of audio, video, or both.

I/O Control Code	Description
DCC_DECODE_RESET	Resets decode of audio, video, or both.
DCC_CHAN_STOP	Stops decode of audio and video.
DCC_CHAN_SWITCH	Resets buffers and prepares for decode of audio and video.
DCC_AVSYNC	Enables/disables/specifies synchronization.

11.2.3.3 MPEG A/V Decoder Device Driver Source Code Files

The following table lists the directories and file names for the MPEG A/V decoder device driver source code.

Directory and File Name	Description
app\mpegdrv.c	MPEG A/V decoder upper layer device driver.
include\mpegdrv.h	MPEG A/V decoder device driver global definitions.
app\cd2.c	MPEG A/V decoder low-level functions.
include\cd2.h	MPEG A/V decoder low level function prototypes. Definitions of the interface between the pSOS device driver and low level functions.
app\cd21osd.c	MPEG A/V decoder OSD test functions.
apps\cd21ucod.c	MPEG A/V decoder microcode.
include\cd21regs.h	Audio/video decoder register definitions.
app\optcdvr.c	OpTIC device driver.

11.2.4 Digital Video Encoder (DENC) Device Driver

The DENC device driver handles the initialization and continued operation of the Philips SAA7182 DENC. The DENC device driver configures the DENC control registers by requesting access to the STBP I²C bus using the I²C driver. The DENC device driver communicates with the DENC using the I²C bus.

The DENC device driver controls the output format (PAL/NTSC/SECAM) and insertion of closed-caption and teletext data into the encoded output stream.

The DENC device driver performs the following specific video operations:

- Read video encoder status byte
- Return the state of the closed-caption mode (ON or OFF)
- Set the video mode (PAL, NTSC, or SECAM)
- Set closed caption to ENABLED or DISABLED

- Set color bars ON or OFF
- Sets video output (turns the output DAC on and off)
- Writes to any DENC control register
- Writes closed-caption data and extended data

11.2.4.1 Testing the DENC

Running the demo program tests the DENC.

11.2.4.2 DENC Device Driver Functions

The following table describes the DENC device driver functions.

Device Driver Function	Description
denc_init()	Creates a DENC device driver semaphore to be used when reading or writing to the DENC. An error is returned if a semaphore cannot be created. Any built-in test routines are called from this routine.
denc_open()	Ensures exclusive access to the DENC and initializes the I ² C controller. An error is returned if exclusive access to the I ² C is denied. I ² C device initialization takes place as a result of this function call. After this initialization, the function NTSCPHILInit() is called to initialize the actual video encoder.
denc_close()	Releases exclusive access to the DENC and I ² C bus. An error is returned if exclusive access is not released.
denc_read()	Reads the DENC registers. A pointer to a data structure is passed to this function. The data structure contains the register to be read and the number of bytes to be read (the maximum number of bytes is 1). The value read is returned in the data structure element variable called DrvInp->u.wrdata.val. An error is returned if the read from the DENC was unsuccessful.
denc_write()	Writes to the DENC registers. An error is returned if the write to the DENC is unsuccessful. A pointer to a data structure is passed to this function. The data structure contains the register to be written to, the value to be written, and the number of bytes to be written (the maximum number of bytes is 1).
denc_cntrl()	Performs specific DENC operations. A pointer to a data structure is passed to this function. The data structure contains an opcode parameter that tells the DENC device driver which operation to execute.

The following table lists the DENC device driver I/O control codes.

I/O Control Code	Description
DENC_GET_CHIP_STATUS	Retrieves DENC status.
DENC_GET_CC_STATE	Retrieves close caption state.
DENC_SET_VIDEO_MODE	Changes the video mode between NTSC and PAL.
DENC_SET_CLOSE_CAP	Enables/disables close caption.
DENC_COLOR_BAR	Enables/disables color bar display.
DENC_WRITE_VIDEO_REG	Writes DENC register.
DENC_WRITE_CC_DATA	Writes close caption data.
DENC_DIAGNOSTICS	Executes DENC diagnostics functions.

11.2.4.3 DENC Device Driver Source Code Files

The following table lists the directories and file names for the DENC device driver source code.

Directory and File Name	Description
include\denclmsgs.h	DENC device driver message definitions.
include\denclivr.h	DENC device driver defines.
app\denclivr.c	DENC device driver.

11.2.5 I²C Bus Device Driver

The I²C bus device driver handles uses the STBP I²C controller to handle the read and write requests for the I²C bus. The I²C bus device driver is not called directly by any application, but is instead called by other drivers. The DENC and EEPROM device drivers request access to the I²C bus device driver to perform read and write operations.

No explicit ISRs are associated with the I²C bus.

The I²C device driver performs the following steps to set up status and control registers to initialize the STBP I²C bus:

1. Clear the status register.
2. Write 0x8F into the extended status register.
3. Write 0x02 into the clock divide register.
4. Write 0x39 into the interrupt mask register.
5. Clear the transfer count register.

6. Clear the pending status register.
7. Flush the master data buffer and set the I²C to transfer data at a rate of 100 KHz. This is equivalent to writing 0x43 into the mode control register.
8. Clear the control register.

11.2.5.1 Reading Data from I²C Bus

Packets received over the I²C bus are buffered in the STBP master data buffer. The master data buffer is essentially a 4-byte deep FIFO. The master buffer must be flushed using **iic_read()** before receiving a packet. Setting the flush master data buffer bit in the mode control register flushes the master buffer. Next, the low master address register must be loaded with the slave address of the target I²C device. The routine then verifies that the number of bytes to be received is greater than 0 and less than the maximum number defined in the **i2cmsgs.h** header file. If neither condition is met, an error is returned.

Before **iic_read()** starts to receiving the number of bytes requested, it must configure the control register. The configuration depends on which byte in the I²C packet is being received. If a received byte is not the last byte in the I²C packet, **iic_read()** must set the control register to chain each byte transferred. After each byte is received, the pending transfer status bit is cleared, but a stop condition is not sent because more bytes are expected.

After the last byte is received, the pending transfer status bit is cleared and a stop condition is sent on the I²C bus to indicate the end of an I²C transfer. If each byte is not received within a specific amount of time, **iic_read()** times out and returns an error to the calling function.

Finally, **iic_read()** checks the status register for any errors that may have occurred during the transfer. An error code is returned to the calling function through the `p->err` variable.

11.2.5.2 Writing Data to I²C Bus

iic_write() must flush the STBP master data buffer before sending a packet. This is done by setting the flush master data buffer bit in the mode control register. Packets to be sent over the I²C bus are then loaded into the master data buffer.

Next, the low master address register must be loaded with the slave address of the target I²C device.

If the I²C packet to be sent is 1- to 4-bytes long, **iic_write()** loads all bytes into the master data buffer and configures the control register to send the packet. If the packet is not sent within a specific amount of time, the routine times out and returns an error to the calling function.

If the I²C packet to be sent contains more than four bytes, but less than the maximum number defined in the **i2cmsgs.h** header file, **iic_write()** proceeds as follows: first, it configures the control register, depending on whether the last byte in the I²C packet is being sent. If the byte being sent is not the last in the packet, **iic_write()** must set up the control register to chain each byte transferred. After each byte is sent, the pending transfer status bit is cleared, but a stop condition is not sent because more bytes are expected.

After the last byte is sent, the pending transfer status bit is cleared and a stop condition is sent on the I²C bus to indicate the end of an I²C transfer. If each byte is not sent within a specific amount of time, the routine times out and returns an error to the calling function.

Finally, the routine checks the status register for any errors that may have occurred during the transfer. An error code is returned to the calling function using the p->err variable.

11.2.5.3 Testing the I²C Bus Controller

Running the demo program tests the I²C controller on the STBP device.

11.2.5.4 I²C Device Driver Functions

The following table describes I²C device driver functions.

Device Driver Function	Description
iic_init()	Creates a I ² C driver semaphore to be used when reading or writing to some I ² C device. An error is returned if a semaphore cannot be created.
iic_open()	Ensures exclusive access to the I ² C bus by acquiring a semaphore. An error is returned if exclusive access is denied. Actual I ² C device initialization takes place in this function.
iic_close()	Releases exclusive access to the I ² C bus by releasing the semaphore. An error is returned if exclusive access is not released.
iic_read()	Receives an I ² C packet up to 64 bytes long (one slave address byte and 63 data bytes). This function is passed a pointer to a structure that contains the I ² C device slave address, the number of bytes to read, and a buffer to store the data received.
iic_write()	Transfers an I ² C packet up to 64 bytes long (one slave address byte and 63 data bytes). This function is passed a pointer to a structure that contains the I ² C device slave address, the number of bytes to send, and a buffer containing the data to be sent.
iic_cntrl()	Not implemented.

11.2.5.5 I²C Device Driver Source Code Files

The following table lists the directories and file names for the source code containing the I²C device driver.

Directory and File Name	Description
include\i2cmsgs.h	I ² C device driver message definitions.
include\i2cdvr.h	I ² C device driver definitions.

Directory and File Name	Description
app\i2cdvr.c	I ² C device driver.
app\i2c_task.c	Task performing actual I ² C device initialization.

11.2.6 EEPROM Device Driver

The EEPROM device driver provides read and write access to the Xicor X24165 Advanced 2-Wire Serial EEPROM device connected to the STBP I²C interface. The EEPROM driver incorporates specific knowledge about the characteristics of the serial memory device used, such as device address, maximum capacity, addressing characteristics, maximum write sequence length, and write timing. The format and management of non-volatile data is left up to the application.

During initialization, a semaphore is created for use during reads and writes to ensure exclusive access to this device, and EEPROM data is copied to shadow RAM for use by the application software.

The **de_open()** or **de_close()** interface (which could be used to ensure/enforce permanent exclusive access) is not implemented. The implementation ensures exclusive access only during one read or write entry into the EEPROM driver.

11.2.6.1 Reading Data from the EEPROM Device

During a read operation, a semaphore ensures exclusive access to the EEPROM. **de_open()** is performed on the I²C to ensure exclusive access to the I²C device. After the read operation, exclusive access to both devices is released.

The **EEPROM_read()** function uses the following parameters to define the read operation:

- Length in bytes (1 through the number of bytes in the EEPROM)
- The EEPROM offset (0 through the last byte)
- Destination address

To set up an EEPROM read operation, a write must first select the read address. No data is actually written, but the EEPROM internal address pointer is set to point to the selected address.

After the read is initiated, the EEPROM starts at the location of the current internal address pointer. The maximum number of bytes that can be read from the EEPROM is limited by the I²C interface. Note that this maximum is different from the maximum number of bytes that can be written to the EEPROM, because writing involves an additional limit specific to the EEPROM.

The EEPROM driver considers the I²C limit when reading from the device. If more byte reads are required than the I²C limit allows, the driver issues multiple reads to the EEPROM until all requested bytes are read. Unlike a write operation, a read operation has no hardware limitation of the EEPROM device on timing of sequential reads.

11.2.6.2 Writing Data to the EEPROM Device

Write protection is disabled by writing to the EEPROM write protect register before a write operation. Write protection is enabled after a write operation is completed. The EEPROM's write protect register, the only EEPROM control register, can protect all of the EEPROM or specific EEPROM blocks. This implementation of the EEPROM device driver enables and disables protection of the entire EEPROM; block protection is not used.

During a write operation, a semaphore ensures exclusive access to the EEPROM.

de_open() is performed on the I²C to ensure exclusive access to the I²C device. After the write operation finishes, exclusive access to both devices is released.

The **EEPROM_write()** operation uses the following specified parameters to define and bound the write operation:

- Length in bytes (1 through the maximum bytes in EEPROM)
- EEPROM destination offset (0 through the last byte)
- source address of data to write

To set up the write operation for this device, the EEPROM device driver disables the EEPROM write protection, then determines the maximum number of bytes per page that can be written sequentially (not to exceed the number of bytes requested for writing). The EEPROM driver has two limitations on the number of bytes that can be written, so the driver uses the lesser of the two in one operation.

One limit is imposed by the I²C implementation; the other is imposed by the EEPROM device. The EEPROM device is limited by the number of address bits that the device automatically updates in the auto-increment address mode. If more byte writes are required than the limit allows, this driver issues multiple writes to the EEPROM until the requested bytes are written.

The EEPROM driver ensures that all multiple-byte writes start on a proper "page" boundary. If a write is requested that does not start on a page boundary, the EEPROM driver divides the write operation into units that do start on a proper boundary. The byte and page limitations are transparent to the caller of the EEPROM driver.

Note that, unlike a read operation, a write operation has a hardware limitation set by the EEPROM device on the timing of sequential writes. After a page write (of any number of bytes up to the maximum page limit), a fixed delay is required before more write operations are attempted.

11.2.6.3 Running the EEPROM Diagnostics

To test the EEPROM, run the **EPC_DIAGNOSTICS** I/O control code against the EEPROM.

11.2.6.4 EEPROM Device Driver Functions

The following table describes EEPROM device driver functions.

Device Driver Function	Description
EEPROM_init()	Copies EEPROM content into shadow RAM.
EEPROM_open()	Not implemented.
EEPROM_close()	Not implemented.
EEPROM_read()	Reads the specified number of bytes from the specified offset in the EEPROM and copies them to the specified destination address.
EEPROM_write()	Writes the specified number of bytes from the specified source address to the specified offset in EEPROM.
EEPROM_cntrl()	Enables/disables the EEPROM write protect feature (EPC_SET_WRITE_PROTECT). Sets the debug flag (EPC_DBG_FLAGS). Executes EEPROM diagnostics (EPC_DIAGNOSTICS).

11.2.6.5 EEPROM Device Driver Source Code Files

The following table lists the directories and file names for the source code containing the EEPROM device driver.

Directory and File Name	Description
include\leepdvr.h	EEPROM device driver definitions.
app\leepdvr.c	EEPROM device driver.

11.2.7 Ethernet Device Driver

The Ethernet device driver performs four basic functions. It determines whether the Ethernet daughterboard is present, provides low-level access and control for the National Semiconductor DP83902A Ethernet controller chip on the daughterboard, handles the transmission and reception of Ethernet messages across the LAN, and provides a high-speed communication link to pNA, the pSOS network component.

The Ethernet device driver is accessed when pNA calls the Ethernet driver routine **NiLan()**. PNA passes **NiLan()** a function code to execute and a pointer to a parameter block set up by pNA.

On power-up, pNA passes the initialization function code passed to **NiLan()**.

The Ethernet device driver writes to multicast address register 0 inside the ST-NIC chip to determine whether the Ethernet daughterboard is present. Then, the driver reads back the

contents of multicast register 0. If the value read back is identical to the value written, the Ethernet daughterboard is present. Note that normal Ethernet accesses rely on a ready line to end the Ethernet access. During this test, external ready is disabled to prevent the processor from hanging if the Ethernet daughterboard is not present. An internal ready is instead generated after 64 clocks to allow the maximum time for the Ethernet register access to take place.

Because of hardware limitations on the Ethernet interface, there is no guarantee that any Ethernet register access can complete in less than 128 clocks if the Ethernet chip is enabled and receiving or transmitting packets. Therefore, the PTD bit in the PPC403GC IOCR register that disables bus error faults after 128 wait states is set.

After the ST-NIC chip is initialized, software control is returned to pSOS. pNA polls the Ethernet device driver to determine whether messages have been received or transmitted, passing the polling function code **NI_POLL** to **NiLan()**.

The transfer of each byte between the Ethernet device driver and the Ethernet controller is performed by assembly code in file the **asm.s** (the **TxCpyData()** function also performs data transfers). The data transfer code polls the PRQ pin memory-mapped to address 0x71000030 coming from the PAL on the Ethernet daughterboard. The PRQ pin tells the processor when data can be read or written to the port address memory-mapped at address 0x71000010.

11.2.7.1 Testing Ethernet

Performing a ping test against the STBRP is a sufficient Ethernet diagnostic. pNA must be initialized and set up before this test to be performed.

11.2.7.2 Ethernet Device Driver Functions

The Ethernet device driver is accessed using the pNA network stack. The only function visible to the pNA network stack is **NiLan()**.

11.2.7.3 Ethernet Device Driver Source Code Files

The following table lists the directories and file names for the Ethernet device driver source code.

Directory and File Name	Description
bsp\lan.h	Ethernet device driver definitions.
bsp\lan.c	Ethernet device driver.
bsp\asm.s	Low level data transfer functions.

11.2.8 Infrared (IR) Receiver Device Driver

This infrared (IR) transceiver device driver provides the interface between the application and the Crystal CS8130 Infrared Transceiver, which is accessed using the UART 1 port on the STBP.

This IR transceiver device driver only supports receiving signals from an infrared remote control.

The universal remote control provided in the STB Reference Design Kit must be programmed with Zenith VCR code 034 and set to operate in VCR mode.

The default baud rate of 9600 between the transceiver and UART is used. When this baud rate is changed, the application program must ensure that the IR transceiver baud rate registers are updated before the UART baud rate registers are updated.

When an interrupt is detected, the ISR disables the associated interrupt and sends an event to a higher-level IR transceiver server task for processing. This task is responsible for accepting raw infrared transceiver button events from the driver ISR and generating key messages for the registered application task. This task has direct access to all functions of the IR transceiver driver. The server task reads all buffered data until a predetermined number of sequential 0xFF values are detected. This sequence indicates the end of a key press packet. When the end condition is detected, the IR transceiver is reset, and the ISR is enabled to allow the next key press to be detected.

To reset the IR transceiver, its mode is changed to any mode other than its current mode and then is changed back to the current TV remote mode. This stops the IR transceiver from sending trailing 0xFF bytes to the serial port. After the IR transceiver is reset, the serial port receiver FIFO is cleared to remove all unread bytes from the FIFO.

After the first three bytes are read from the serial port the packet is passed through a key decoder. The key decoder performs a table lookup to determine which key is being pressed. When a known key is decoded, its value is sent to the application task that previously registered to receive this data.

An application task can register to be the final destination of the key press indicator by making the appropriate call to the IR transceiver device driver **cntrl()** function and passing the ID of the message queue to which the key indicator is to be sent. Note that the IR server task is responsible for enabling this interrupt. This design decision was made because once data starts arriving from the IR transceiver, it does not stop until forced to do so by the set-top application program. There is no more need for an interrupt, and a task can begin polling the serial port for data until the end of a key press byte stream has been detected.

11.2.8.1 Testing the IR Transceiver

To test the IR transceiver, run the **IRC_DIAGNOSTICS** I/O control code against the IR transceiver device. Running the demo program further tests the IR transceiver device.

11.2.8.2 IR Transceiver Device Driver Functions

The following table describes the IR transceiver device driver functions.

Device Driver Function	Description
Ir_init()	Initializes device and driver.
Ir_open()	Not implemented.
Ir_close()	Not implemented.
Ir_read()	Returns bytes held in UART FIFO that were detected by the IR transceiver.
Ir_write()	Not implemented.
Ir_cntrl()	Performs specific infrared receiver operations

The following table lists the I/O control codes for the IR transceiver device driver.

I/O Control Code	Description
IRC_GET_STATUS	Returns value of the infrared receiver status register and value of the infrared receiver revision register.
IRC_SET_UART_BAUD	Changes UART baud rate.
IRC_SET_IR_BAUD	Changes infrared receiver baud rate.
IRC_WRITE_REG	Writes infrared receiver device register.
IRC_RESET	Resets the infrared receiver.
IRC_REGISTER_RX_QID	Registers an application task to receive key press information.
IRC_DIAGNOSTICS	Performs infrared receiver diagnostics.

11.2.8.3 IR Receiver Device Driver Source Code Files

The following table lists the directories and file names for the IR transceiver device driver source code.

Directory and File Name	Description
app\ir_task.c	Infrared receiver key processing task.
include\ir_task.h	Definitions for the interface between infrared receiver device driver and key processing task.
include\irdb1.h	Remote control key table prototypes.
app\irdvr.c	Infrared receiver device driver.

Directory and File Name	Description
app\irdvr.h	Infrared receiver device driver definitions.
include\irregs.h	Infrared receiver register definitions.

11.2.9 Keypad Device Driver

The keypad device driver polls the keypad to detect a key press at intervals controlled by the application program. By default, the polling interval is two clock ticks; the keypad device driver checks the keypad matrix every two clock ticks. The application program also controls the key repeat intervals. When a key is pressed and held down, the keypad device driver records the second key press after the key is held down position during the initial repeat count polls. The application program also controls the subsequent repeat count.

To use the keypad, the Ethernet daughterboard must be connected to the STBRP.

11.2.9.1 Running the Keypad Diagnostics

To run the keypad diagnostics, run the **key_test()** function or the demo program.

11.2.9.2 Keypad Device Driver Functions

The following table describes keypad device driver functions.

Device Driver Function	Description
KeypadInit()	Initializes device driver.
KeypadOpen()	Not implemented.
KeypadClose()	Not implemented.
KeypadRead()	Reads bytes retrieved from the keypad matrix.
KeypadWrite()	Not implemented.
keypadCntrl()	Performs specific keypad device operations

The following table lists the I/O control codes for the keypad device driver.

I/O Control Code	Description
KEYPAD_SET_REPEAT	Sets key repeat time.
KEYPAD_SET_READ_TIMEOUT	Sets read timeout.

11.2.9.3 Keypad Device Driver Source Code Files

The following table lists the directories and file names for the keypad device driver source code.

Directory and File Name	Description
include\keypad.h	Device driver interface definitions for keypad driver.
include\keycode.h	Prototypes and functions in common keypad driver code.
app\keydrivr.c	Keypad device driver.
app\keycode.c	Keypad device driver functions.

11.2.10 Serial Port Device Driver

The serial port driver provides generic initialization, read, write, and control access to an RS-232 port implemented in the STBRP.

11.2.10.1 Testing the Serial Port

To test the serial port, run the **MC_DIAGNOSTICS** I/O control code against serial the port device.

11.2.10.2 Serial Port Device Driver Functions

The following table describes serial port device driver functions.

Device Driver Function	Description
Modem_init()	Initializes the modem port.
Modem_open()	Not implemented.
Modem_close()	Not implemented.
Modem_read()	Reads data from the modem port.
Modem_write()	Writes data to the modem port.
Modem_cntrl()	Performs specific serial port operations.

The following table lists the I/O control codes for the serial port device driver.

I/O Control Code	Description
MC_CONFIG	Set serial port configuration (data bits, stop bits, parity, carrier detect).
MC_GET_RING_STATUS	Check status of the ring indicator GPIO line.

I/O Control Code	Description
MC_GET_CARRIER_STATUS	Check status of the carrier detect GPIO line.
MC_DEBUG_FLAGS	Toggle debug status.
MC_GET_GEN_STATUS	Retrieve serial port device status.
MC_RESET	Reset the serial port.
MC_SET_UART_BAUD	Set the serial port baud rate.
MC_LOOPBACK_TEST	Perform serial port loopback test.
MC_DIAGNOSTICS	Performs serial port diagnostics.

11.2.10.3 Serial Port Device Driver Source Code Files

The following table lists the directories and file names for the serial port device driver source code.

Directory and File Name	Description
app\modemdvr.c	Serial port device driver.
include\modemdvr.h	Serial port device driver definitions.

11.2.11 IEEE 1284 Port Device Driver

The IEEE 1284 device driver provides slave functionality for the IEEE 1284 port on the STBRP, enabling the STBRP to receive data from a standard parallel port. The IEEE 1284 device driver supports temporary or permanent exclusive control of the IEEE 1284 device.

During initialization, exclusive ownership is disabled and the IEEE 1284 control registers are reset and initialized. The port is initialized to operate in standard compatible mode, with the hardware automatically handling all data acknowledgments. The semaphores used to enforce exclusivity are created at this time, and the data destination pointer is set to null, indicating received data is not expected.

The **PAR_open()** driver call locks the device for exclusive access by a task. The lock can be temporary or permanent. A temporary lock is released when the **PAR_close()** function is called. A permanent lock is established if **PAR_close()** is never called; no other task will have access to the device. If another device attempts to gain access using **PAR_open()**, the task will wait forever on a semaphore.

A call to **PAR_open()** is not required before reading or writing to the device. **PAR_open()** can be used if a task wishes to ensure that no other task will be allowed to access the device at the same time. (Note that there is a second level of simultaneous access protection implemented directly in the **PAR_read()** and **PAR_write()** driver calls for instances when **PAR_open()** is not invoked.)

After the IEEE 1284 port is opened, the IEEE 1284 port device driver returns a unique access key to the calling task. This key must be used in all subsequent entries to the IEEE 1284 port device driver. If the device is opened and the returned access key is not used in subsequent driver calls, an exclusive device error will be generated. If the device is not opened first, the key is not required.

To read from the IEEE 1284 port, a task invokes **de_read()** against this driver and specifies the number of bytes to read, a destination pointer for the bytes read, and a timeout value. This is a blocking call due to the use of a semaphore in this function. The function will block until the specified number of bytes is read or until the timeout period expires. The timeout period does not begin until after the first byte is received, and the timeout period is reset after each successive byte is received. A timeout occurs if a new byte is not received within the timeout period after the previous byte.

Any task that needs to read from the IEEE 1284 port must expect to receive data and must invoke the IEEE 1284 port device driver to read from the port before the transmitter begins sending. If this does not occur, the transmitter gets a busy signal from the STBP IEEE 1284 port and cannot continue.

A reading task should know how many bytes of data to expect so that it can request a read of that many bytes. The reading task must have preallocated memory to contain the data read from the IEEE 1284 port, and must pass a pointer to the start of this memory location.

Actual copying of data from the IEEE 1284 port to the designated memory area does not occur in **PAR_read()** function. The copy is implemented by the IEEE 1284 ISR. The **PAR_read()** function simply monitors the progress of the ISR copy to know when to return.

When all desired bytes have been copied, the ISR will release the semaphore that is blocking **PAR_read()** function. The ISR disables itself, awaiting the next **read()**. Each **read()** enables the ISR to begin the copy process.

11.2.11.1 Testing the IEEE 1284 Port

To test the IEEE 1284 port, run the **PC_DIAGNOSTICS** I/O control code against the IEEE 1284 port device.

11.2.11.2 IEEE 1284 Port Device Driver Functions

The following table describes IEEE 1284 port device driver functions.

Device Driver Function	Description
PAR_init()	Initializes driver and device state.
PAR_open()	Ensures exclusive access.
PAR_close()	Releases exclusive access.
PAR_read()	Reads data from device.
PAR_write()	Not implemented.

Device Driver Function	Description
PAR_cntrl()	Performs IEEE 1284 port diagnostics.

11.2.11.3 IEEE 1284 Port Device Driver Source Code Files

The following table lists the directories and file names for the source code containing the parallel port device driver.

Directory and File Name	Description
app\pardvr.c	IEEE 1284 port device driver.
include\pardvr.h	IEEE 1284 port device driver definitions.

11.2.12 Smart Card Device Driver

The smart card device driver provides access to, and control of, a smart card using the TDA 8002 smart card interface (SCI) device. Communication between the application program and the TDA 8002 is handled using a STBRP UART interface. The TDA 8002 is an interface device for asynchronous and synchronous cards. It is ISO 7816 compatible and handles automatic ISO 7816 activation and deactivation sequences of these cards. This driver only handles processing common to all smart cards and is not designed to work with any particular card. Specific smart card driver implementation can not occur until a specific smart card is selected for inclusion in a given hardware design.

When an interrupt related to a smart card occurs, the ISR first determines the cause of the interrupt. Either a card IN/OUT transition took place or smart card data is available in the UART.

If the interrupt was been caused by the insertion of a card, that fact is stored for later access, the interrupt polarity is changed as appropriate to detect the next card transition, and the interrupt is enabled. If a card was removed, that fact will be stored for later access, the SCI is deactivated (if it was active), the interrupt polarity is changed as appropriate to detect the next card transition, and the interrupt is enabled. During this processing, invalid card state transitions are monitored and handled.

If the interrupt was caused by the presence of data in the UART FIFO, the ISR will begin polling the UART data ready status register, extracting data from the FIFO and storing it in a database for later access. After the ISR exits, a task can command the smart card device driver to access the data stored by the ISR. This is how the answer to reset string is stored and accessed.

11.2.12.1 Testing the SCI

To test the SCI, run the **SCC_DIAGNOSTICS** I/O control code against the serial port device.

11.2.12.2 Smart Card Device Driver Functions

The following table describes smart card device driver functions.

Device Driver Function	Description
SC_init()	Initializes the device driver and smart card interface.
SC_open()	Ensures exclusive access.
SC_close()	Releases exclusive access.
SC_read()	Not implemented.
SC_write()	Not implemented.
SC_cntrl()	Performs specific smart card operations.

The following table lists the I/O control codes for the smart card device driver.

I/O Control Code	Description
SCC_DISGNOSTICS	Performs smart card diagnostics.
SCC_DEUMP_CARD_STATE	Returns status of the interface in terms of physical card presence or absence and success of any reset sequence applied as well as data which was read from the card.
SCC_ACTIVATE	Invokes the activation sequence in the interface card to transition the interface from an idle mode to the active mode. The driver ensures that this activation will only be attempted if the card is currently inserted.
SCC_RESET	Executes a physical card reset sequence.
SCC_DEACTIVATE	Invokes the deactivation sequence in the interface card to transition the interface from an active mode to the idle mode.
SCC_DUMP_INPUT	Prints data received from the smart card.
SCC_LOOPBACK_TEST	Performs loopback test on the serial port attached to the smart card interface.

11.2.12.3 Smart Card Device Driver Source Code Files

The following table lists the directories and file names for the smart card device driver source code.

Directory and File Name	Description
app\scard.c	Smart card device driver.

Directory and File Name	Description
include\scard.h	Smart card device driver definitions.

11.2.13 STB Peripheral Chip (STBP) Functions

Although there is no driver for the STBP, a set of interface functions simplifies the access to the device by multiple drivers. These routines provide generic initialization, allow for enabling and disabling interrupts and establishing interrupt handlers, and allow for the reading and writing of general-purpose I/O (GPIO) pins. The main STBP ISR is also included.

The initialization functions set the direction for the GPIO pins, set the type and sense of interrupts, install the interrupt handler, and load a default interrupt mask.

The ISR reads the interrupt status register and searches a table of interrupt handlers to find the routine that will handle the interrupt. Drivers that want to perform interrupt processing register a handler routine. After the entire table is processed, any interrupts that have not been handled are assumed to be unwanted and are masked off.

11.2.13.1 STBP Source Code Files

The following table lists the directories and file names for the source code containing the smart card device driver.

Directory and File Name	Description
app\asic.c	STBP functions.
include\stbp.h	STBP definitions

11.3 Server Tasks

A task is a loop of code with a defined scheduling priority and dedicated memory for stack and register values. It is the entity controlled by the pSOS scheduler and may be pre-empted by an interrupt or a higher-priority task. As such, a task serves as a useful method for dividing and prioritizing the work load of the system. Tasks are also useful for providing a higher-level view of underlying functionality.

All system tasks send and receive messages through wrapper calls above the pSOS message facilities. This enables message traffic to be logged as an aid in the debugging process. The logging facilities are under compile time control and can easily be removed if desired.

11.3.1 Root Task

The root task is created by the pSOS operating system and entered when pSOS initialization is complete. In the base system, the task is responsible for sequencing the

creation and start-up of system tasks. It is also responsible for installing any needed drivers that are not installed by the tasks themselves.

When system task creation is complete, the root task calls the application entry point to initialize the application tasks.

11.3.2 A/V Task

The A/V task provides higher-level processing for the MPEG driver and low-level control for the application task. The intent is to provide a “mid-level” place for those tasks that must be coordinated with the A/V output processing.

Only a test closed-caption facility is implemented, which serves as a model of how extended data service and teletext processing might be done. For historical reasons, a few DENC diagnostic functions are also implemented in this task.

The following table lists the messages accepted by the A/V task.

I/O Control Code	Description
DCC_MSG_EVEN_SYNC	Message from the MPEG decoder driver indicating a “top or even” field is being output. Used to indicate write of closed-caption data to the DENC.
DCC_MSG_ODD_SYNC	Message from the MPEG decoder driver indicating a “bottom or odd” field is being output. Used to indicate write of closed-caption data to the DENC.
DCC_MSG_CC_DATA_IN	Adds close caption data.
DCC_MSG_DIAG	String containing one of the following diagnostic commands: ah (help message), ac (closed-caption test), adb (DENC blank test), adc (DENC color bar test).
DCC_MSG_CC_CONTROL	Enable/disable closed-caption output commands. CC_ENABLE, enables closed caption output. CC_DISABLE, disables closed-caption output.

11.3.3 Debug Task

The debug task aids in debugging the STBRP hardware and software. Because of huge data transfers and tight timing, it is impractical to debug the system using print statements. The debug task resolves this problem by providing a history buffer that can store information about system events for later analysis.

The following table lists the messages accepted by the debug task.

I/O Control Code	Description
DBG_MSG_LOG	Makes a new log entry. Each entry is time stamped and added to the history buffer.
DBG_MSG_LOG_CNTR	Controls log buffer operation and display. The exact operation is encoded in an opcode as follows: DBG_RESET_LOG (clears history buffer), DBG_DUMP_LOG (formats and displays contents of history buffer), DBG_LOG_START (enables the accumulation of history entries), DBG_LOG_STOP (disables accumulation of history entries).
DBG_MSG_LED_BLINK	Toggles the board led.

11.3.4 Transport Task

The transport task is responsible for the initial processing of the data in the transport circular queues. When the transport device generates a queue interrupt, the transport ISR determines which queues have data in them and forwards this information to the transport server task using a pSOS message.

The transport server task processes the queue data on a scheduled basis (i.e., processing occurs outside of the ISR). The task then forwards the transport section data to the application task that originally set up the PID.

11.3.5 Control Task

The control task controls the state of the system. It sends commands to and receives messages from all other tasks in the system. It provides a higher-level decision-making structure for the software. This frees the remaining tasks from having to know and control the state of other tasks in the system. When operations require synchronization among several tasks, the control task determines and controls the sequence of events.

The control task is really part of the application and is spawned as an application task. The following is an example of its intended use:

The user interface task sends a message to change to another channel. The control task orchestrates this action by sending the appropriate sequence of messages to the front end, transport, conditional access, MPEG decode, and output display tasks

In the base software, the control task is included only to control those sequencing functions required for debugging and testing the associated drivers.

The following table lists the messages accepted by the control task.

I/O Control Code	Description
MCP_MSG_AV_NEW_PIDS	Change channel. <ol style="list-style-type: none"> 1 Stop current MPEG decoder channels. 2 Stop demux of current audio/video and PCR logic. 3 Start PCR process logic. 4 Execute MPEG decoder channel change. 5 Start demux of indicated audio/video. 6 Start decode and play of audio/video.
MCP_MSG_AV_RESET	Change channel and reset MPEGCD21. <ol style="list-style-type: none"> 1 Stop play and decode of audio/video. 2 Stop demux of current audio/video and PCR logic. 3 Reset MPEG decoder chip. 4 Start demux of indicated audio/video. 5 Start PCR process logic. 6 Start decode and play of audio/video.
MCP_MSG_V_RESET	Reset video pipeline: <ol style="list-style-type: none"> 1 Stop play and decode of current video stream. 2 Stop demux of current video stream. 3 Reset video decode logic. 4 Start demux of indicated video stream. 5 Start decode and play of video stream.
MCP_MSG_A_RESET	Reset audio pipeline: <ol style="list-style-type: none"> 1 Stop play and decode of current audio stream. 2 Stop demux of current audio stream. 3 Reset audio decode logic. 4 Start demux of indicated audio stream. 5 Start decode and play of audio stream.

11.3.6 Database Task

At the application level, code must request and process sections (MPEG and DVB tables) from the incoming multiplexed stream, and allow a viewer to select a program to view. The database task provided with the base software fills portions of this role.

The database task parses and validates MPEG-2 program-specific information (PSI) and DVB service information (SI) obtained as a result of PIDs that are set up during the initialization of this task.

To get MPEG PSI and/or DVB SI sections into the application-level database task, the following steps are performed:

1. Create a pSOS message queue (done only once).
2. Initialize a transport queue (done only once).
3. Create one or more buffers large enough to hold the expected section data (done only once).
4. Set up a PID to use that queue. Include filter information, destination buffer information, and response message queue information (the PID is automatically started).
5. Wait for data to arrive at the message queue. When it arrives, process section data.
6. Release the buffer for reuse by the transport driver.

After processing the received section data, additional PIDs may need to be set up. For example, after the PAT is received, the PID for the selected PMT is located and set up.

11.3.7 IR Application Task

This task is implemented solely to illustrate how a high-level application task registers with the infrared receiver driver and receive keystroke messages when an remote control key is pressed.

It will be left to the final application task to determine what to do with these keys once they are received.

11.3.8 User Interface Task

This task accepts user input and requests actions from the rest of the system. These requests to the system take the form of device driver calls and messages sent to tasks.

This task is used most often to initiate the processing of diagnostic testing based on user inputs.

11.3.8.1 Server Task Source Code Files

The following table lists the directories and file names for the source code containing the server tasks.

Directory/File Name	Description
app\root.c	Initial task created by the pSOS system.
app\dcc_task.c	Audio/video decoder high level functions.
app\dbg_task.c	Message logging task.
app\xpr_task.c	Transport demultiplexer task.
app\mcpc_task.c	Audio/video control task.
app\dbt_task.c	DVB SI processing task.
app\dbtsi.c	DVB SI parsing code.

Directory\File Name	Description
include\dbtsi.h	Definitions for the DVB SI parsing code.
appl\ira_task.c	Sample application infrared receiver task.
appl\ui_task.c	User interface task.

11.4 Conditional Access Support

Because of the diverse and proprietary nature of conditional access solutions, the implementation of a completely generic system is impossible. Instead, the STBRP provides the basic building blocks of a conditional access system rather than an actual implementation.

The STBRP provides software interfaces to a smart card, a DVB descrambler, non-volatile storage, and an auxiliary serial port that can communicate with a modem for back-channel support.

11.5 Demo Program Source Code

The following table shows the directories and file names for the source code used to build the demo program.

Directory\File Name	Description
app\democode.c	General purpose functions used by demonstration program
app\demomain.c	Demonstration program man.
include\demo.h	Function prototypes and common defines for demonstration program

11.6 Diagnostics

Three levels of diagnostics are implemented. Level 1 diagnostics occur during initialization and perform relatively simple device checks. Level 2 diagnostics can occur after initialization, depending on the user's request, and can be performed based on the status of the Level 1 tests. Level 3 diagnostics can also occur after initialization. This level of testing involves coordination between multiple drivers.

Level 2 and Level 3 tests, which are more detailed than Level 1 diagnostics, may require additional user input.

11.6.1 Level 1 Diagnostics

These diagnostics are run automatically at start-up when each driver is initialized via the **de_init()** call. The tests are invoked from the root task before application tasks are started.

The driver will return one of the following status values to the root task:

- OK
- Diagnostics test failure(`xxx_DIAGNOSTIC_FAILURE`)
- A non-diagnostic error value (these may be broken out into multiple, more specific error values as desired by the driver)

Drivers may print error and warning information to the screen while performing these tests. Drivers should not print general information to the screen unless it is very pertinent, nor should drivers prompt the user for input while performing these tests.

These tests will test as much as possible about the associated device to confirm that the device is operating properly without user input.

Examples of Level 1 diagnostics include:

- Confirming that a device exists
- Performing a loopback test to check I/O port or memory, or to check register settings

If a driver fails Level 1 diagnostic testing, the root task continues to invoke all other driver initialization and starts up all nominal tasks. This informs the user of how many drivers are in error. The UI task is started to allow the user to perform Level 2 testing. (Note that it is recognized that the first failure may lead to subsequent failures, but the user must determine that.)

11.6.2 Level 2 and Level 3 Diagnostics

These diagnostics are invoked after initialization, if a user requests a more detailed evaluation. Level 2 tests a single driver; Level 3 tests require coordination between drivers. Level 3 diagnostics are grouped to be processed by a message sent to a task from the UI task. Note that this is not a driver call.

Entry into the drivers to perform these tests is through a **`de_cntrl()`** call with an opcode of `xxx_DIAGNOSTIC` (where `xxx` ensures a unique name for all drivers). A second **`de_cntrl()`** parameter is a pointer to the command line string.

When the driver is invoked for a Level 2 or Level 3 test, the driver takes over the user interface and communicates directly with the user to request further data, if required. It then scans the input line to obtain user responses as required.

When the test is complete, the driver returns and possibly receives a new diagnostic test command. Any other tasks or drivers that are processing nominal data simultaneously during Level 2 or Level 3 testing may output text to the screen.

Level 2 and Level 3 diagnostics can be as intricate as a developer wishes to implement. For example, these tests can allow a user to read and write to any or all memory and register locations, or perform more elaborate loopback tests.

For each diagnostic, the driver returns one of the following status values to the calling task:

- OK
- Could not execute test (`xxx_DIAGNOSTIC_ABORTED`)

The OK status value is returned even if the test detected an error. This is because the user, at this level of testing, is directly informed of any errors so that he or she can take appropriate action. The OK status is also returned by any driver that has no diagnostic test to execute.

The abort status is returned if the driver was never able to begin testing, never informed the user of any error, and aborted the test. The abort condition can always be handled by a driver, that no driver will ever actually return this abort value.

11.6.3 The UI Task

The UI task prints a high-level menu to list the available Level 2 and Level 3 diagnostics. If a user wants more details about a test and its associated options, the user types the appropriate help command. The UI task makes a **de_cntrl()** call to the appropriate driver and returns the command string entered by the user. (The same occurs if a task is actually executed.) The invoked driver then processes the help command string and displays the associated menu.

After a device driver executes a user request (using **de_cntrl()**), the device driver returns to the UI task to await the next request. The device driver does not take control of the user interface until the user finishes testing a specific device driver.

Note that all device driver-specific menus are printed by the device driver, not by the UI task. Items include a list of the tests the driver can perform and the menu to describe the tests.

11.6.4 Additional Diagnostics

Tests to evaluate inter-driver operability and full system functionality are considered Level 4 diagnostics. These are the responsibility of the application. Level 1, Level 2, and Level 3 diagnostics do not perform any of this type of high-level testing.

11.7 Miscellaneous File and Directory Descriptions

The following table shows additional files included in the STPRP kit.

Directory\File Name	Description
lib\tftpd.exe	TFTP daemon executable file.
lib\tftp.ico	TFTP daemon icon.
lib\tftp.res	TFTP daemon resource file.
lib\bootpd.exe	BOOTP daemon executable file.
lib\bootpd.ico	BOOTP daemon icon.
lib\bootpd.res	BOOTP daemon resource file.

Directory\File Name	Description
readme	Readme file describing steps required to install and integrate all the software.
readme.drv	Readme file associated with the Device Driver package.
readme.bsp	Readme file associated with the BSP package.
stbsetup.bat	DOS BAT file used in software integration.
include\v2020.h	Transport demultiplexer definitions.
lib\v2020.o	Transport demultiplexer device driver.
lib\liboptic.a	OptIC library.
lib\ram.elf	Compiled sample application in ELF file format.
lib\prbxxxxx.img	pSOS pROBE+ image file. This is a self burning file that can be downloaded using ROM Monitor.
lib\stbxxxxx.img	ROM Monitor image file. This is a self burning file that can be downloaded using ROM Monitor.
lib\appxxxxx.img	Sample application image file. This is a self burning file that can be downloaded using ROM Monitor.
lib\stoimg.exe	Program that converts S-record format file to an image file that can be loaded using ROM Monitor.
lib\bootptab.sam	Sample bootptab file.
app\makefile	Top level makefile.

Chapter 12. OS-9000 Platform Support Package

The Set-Top Box (STB) Reference Design Kit provides software, based on the OS-9000™ real-time operating system (RTOS) and DAVIDLite™, that runs on the STB Reference Platform (STBRP). This software, called the OS-9000 Platform Support Package (PSP), is provided as a set of source code and a set of executable files.

This chapter describes how to install and modify OS-9000 for PowerPC v. 2.1 for STB application development on a host PC, and how to use the OS-9000 PSP.

Note: To use the OS-9000 PSP, FasTrak v. 2.1.1, supplied with the STB Reference Design Kit, must be installed on the host PC. Because of the limited DAVIDLite functionality provided in the OS-9000 PSP, some device driver descriptor files cannot be recompiled until the full DAVIDLite product is installed.

Full versions of OS-9000, DAVIDLite, and FasTrak must be purchased from Microware Systems Corporation before developing complete STB applications. Contact Microware Systems Corporation for more information.

The OS-9000 PSP contains all of the device drivers needed to develop applications for the STBRP. A supplied demo program serves as a starting point for development and evaluation.

The reader needs to be familiar with OS-9000 and DAVIDLite.

12.1 Integrating the OS-9000 PSP

The OS-9000 PSP must be carefully integrated with a base installation of OS-9000 to run on the STBRP. After the integration is complete, the resulting OS-9000 executable image runs on the STBRP.

Alternatively, the sample image provided with the OS-9000 PSP can be executed on the STBRP. The sample image file, named **evbboot**, is in the **lib** subdirectory.

12.1.1 Loading the Boot Code

After a hardware reset, the processor fetches a jump instruction at address 0xFFFFF0C in the STBRP flash boot block. This instruction branches to the first instruction of boot code. An intermediate program, the ROM Monitor, resides in flash memory and supports the loading of OS-9000 executable images. The ROM Monitor receives control out of reset. Chapter 10, "Using the ROM Monitor," describes the ROM Monitor.

The process for loading images is the same as that described in Chapter 10.

12.1.2 Using the OS-9000 PSP Version of FasTrak

The evaluation version of FasTrak v. 2.1.1 supplied with the OS-9000 PSP permits recompilation of the supplied device drivers and the ROM-resident demo program.

The downloadable OS-9000 image (found in the **evbboot** file) cannot be rebuilt until OS-9000 for PowerPC v. 2.1 is installed. The recompiled modules can be loaded into the target system using the FasTrak target tool.

Refer to the README files on the OS-9000 PSP installation diskettes for detailed instructions for using FasTrak and downloading modules to the STBRP.

12.1.3 Modifying the Base OS-9000 Installation

The following procedure modifies the base installation of OS-9000 v. 2.1 on the host PC and integrates the OS-9000 PSP. Section 12.1.4 describes the changes to system files.

The procedure assumes that OS-9000 is installed under the **C:\MWOS** directory, and that the OS-9000 PSP is installed under the **C:\STB** directory. If different directories are used, directory paths in the following procedure must be changed accordingly.

1. Verify that the base installation builds successfully. Change to the directory **C:\MWOS\OS9000\403\PORTS\403GAEVB** and recompile the system using the **os9make** command.
2. Duplicate the directory **403GAEVB**. Create a directory called **403GCSTB** in the same base path and copy the files in the directory **403GAEVB** to the directory **403GCSTB**.
3. Verify the duplicated system build. Change to the directory **..\403GCSTB** and recompile the system using the **os9make** command. Then, run **os9make clean** as a safeguard, before incorporating the OS-9000 PSP modifications.
4. Replace the **systype.h** file in the directory **403GCSTB** with the **systype.h** file in the directory **C:\STB\SYS**.
5. Change to the directory **..\ROM\ROMCORE** and replace the file **sysinit.c** with the file **sysinit.c** in the directory **C:\STB\SYS**.
6. Change to the directory **..\BOOTLIST** and replace the **rom_mods.ml** file with the file **rom_mods.ml** in **C:\STB\SYS**.
7. Replace the **rom.bl** file in the directory **BOOTLIST** with the file **rom.bl** in **C:\STB\SYS**.
8. Copy the files **init.c**, **misc.c**, and **ns8390.h** in **C:\MWOS\SRC\IO\INET\DRV\NS8390** under different names and replace **init.c**, **misc.c**, and **ns8390.h** with files of the same name found in **C:\STB\INET**. Replace the file **systype.des** in the directory **403GCSTB\ISP** with the file **systype.des** in **C:\STB\INET**. These files support Ethernet.
9. Change to the directory **C:\MWOS\OS9000\403\PORTS\403GCSTB\ROM\EVBBOOT** and recompile the system using the **os9make** command. This generates an executable image that integrates the base OS-9000 system and the OS-9000 PSP. The image, named **evbboot**, is placed in the directory **..\CMDS\BOOTOBJ\S\ROM**.

12.1.4 Modifying OS-9000 System Files

The following OS-9000 system files must be modified before OS-9000 can boot on the STBRP:

- C:\MWOS\OS9000\403\PORTS\403GAEVB\systype.h
- C:\MWOS\OS9000\403\PORTS\403GAEVB\ROM\ROMCORE\sysinit.c
- C:\MWOS\OS9000\403\PORTS\403GAEVB\BOOTLIST\rom.bl
- C:\MWOS\OS9000\403\PORTS\403GAEVB\BOOTLIST\rom_mods.ml

If the base OS-9000 system is not installed in the **C:\MWOS** directory, change the directory path accordingly.

The OS-9000 PSP provides sample **systype.h**, **sysinit.c**, **rom.bl**, and **rom_mods.ml** files that have already been modified appropriately. The modified sample files are installed in the directory **sys**. Copy the 403GAEVB directory to the 403GCSTB directory to preserve the original files. After the directory is copied, the listed files can replace the provided files.

The modified system files in OS-9000 PSP work properly only with OS-9000 v. 2.1. If another OS-9000 version is used, the listed system files must be changed manually, as described in Section 12.1.4.1 through Section 12.1.4.4.

12.1.4.1 Modifying the File systype.h

Modify the **systype.h** file as follows:

- Change the **SERCLK** define value from **7372800** to **27000000** (in two places).
- Change the **PICLK** define value from **10000000** to **27027000**.
- Change the **EXTENSIONS** define value from **OS9P2 fpu cache403** to **OS9P2 fpu**.
- Change the **PREIOS** define value from **"** to **cache**.

12.1.4.2 Modifying the File sysinit.c

Modify the **sysinit.c** file as follows:

- Change the **IOCR_DEFAULT** define value **0x04002880**.
- Change the **EXIER_DEFAULT** define value to **0x0c000000**.
- Add the following defines:

```
#define BR0 0xFE380405
__asm__("BR0_UPPER set %0", __obj_constant(BR0>>16));
__asm__("BR0_LOWER set %0", __obj_constant(BR0&0xFFFF));
#define BR1 0x011843A4
#define BR2 0xFF00BFFF
#define BR3 0x109845A5
#define BR4 0xFF00BFFF
```

```
#define BR5 0x087842E5
#define BR6 0x2098C4A5
#define BR7 0x005D0AAE
#define ICACHE_ON
```

- Write zeros to the PPC403 special purpose registers Storage Guarded Register (SGR) and Data Cache Write-thru Register (DCWR).
- Initialize the PPC403 device control register (DCR) Bank Register 0 (BR0).
- In **sysinit1()**, initialize the PPC403 DCRs BR1 through BR7.
- In **sysinit2()**, delete the code that disables 16550 UART interrupts.

12.1.4.3 Modifying the File rom.bl

Add all supplied modules in the OS-9000 PSP.

12.1.4.4 Modifying the File rom_mods.ml

Remove the communication protocol modules and the 16550 device driver.

12.1.5 Problems with Base OS-9000 Software

The following list describes problems found in the base OS-9000 software.

- The **fpu** module on the distribution media is back level and does not work.
- The **sysid** module causes an alignment exception.
- The **nppd** daemon must be started after the **i2cdrvr**, **irdrvr**, **keydrv** device drivers are started. Otherwise, a machine check exception occurs.

12.1.6 Using the OS-9000 PSP DAVIDLite Implementation

The OS-9000 PSP includes portion of DAVIDLite. The complete DAVIDLite package is required to recompile some OS-9000 PSP device descriptors. DAVIDLite is available from Microware Systems Corporation.

12.2 The OS-9000 PSP Device Drivers and System Files

The OS-9000 PSP provides the device drivers and system files described in this section.

12.2.1 Flash Memory Device Driver

The flash memory driver provides write access to the AMD Am29F016 2MB flash memory device, along with routines to reset the flash, erase a flash sector, or erase the entire flash memory. Other functions obtain the manufacturer and device codes. The application task formats and manages the flash memory.

12.2.1.1 Running the Flash Diagnostics

Run the **flashsamp** sample program to test the flash. If **flashsamp** completes without printing error messages, the flash device is functioning properly.

12.2.1.2 Flash Device Driver Functions

The **flashlib.l** file provides the following functions.

Function	Description
init_flash()	Initializes the flash device driver.
write_flash()	Writes data to the flash.

The following table shows the flash device driver implementation-specific **_os_getstat()** and **_os_setstat()** codes and their functions.

Function	Description
SS_SETADDR	Sets the address to write to. This is the offset from the start of the flash (FLASH_START, in flash.h).
SS_SETLEN	Sets the length, in bytes, of data to be written.
SS_SETFLAGS	When set to FLASH_FLAG_PROGRESS, specifies the progress indicator (spinning wheel) is to be shown while the flash is being programmed.

12.2.1.2.1 init_flash()

Synopsis

```
#include <flash.h>
int init_flash(void);
```

Description

init_flash() initializes the flash device driver and checks the flash manufacturer device code. If successful, **init_flash()** returns 0; otherwise, a non-zero value is returned.

Errors

FLASH_DEVICE_ERROR is returned when an invalid flash manufacturer device code is detected

12.2.1.2.2 write_flash()

Synopsis

```
#include <flash.h>
int write_flash(char * offset, int length, char* data, int flags);
```

Description

write_flash() writes data to the flash device. The flash is read like normal RAM, so no special function is needed for reading. If successful, **write_flash()** returns 0; otherwise, a non-zero value is returned.

offset specifies the offset in bytes from the start of the flash address space (constant **FLASH_START** in **flash.h**).

length specifies the length of the data pointed to by data.

char* data points to the data to write to the flash. The flash is divided into 64KB sectors; **write_flash()** calculates the sectors to write to, based on **offset** and **length**. Sectors that are written are erased before any data is written. Written data is read back to verify that it was written correctly.

int flags takes any of the following values, which can be ORed together:

FLASH_FLAG_PROGRESS displays a progress indicator as the flash is being written.

FLASH_FLAG_WARNING displays a warning before the flash is written, along with messages for errors found during verification.

FLASH_FLAG_PROMPT prompts the user to confirm that the flash will be overwritten before proceeding.

Errors

FLASH_OPEN_ERROR is returned if open of the **/f1** descriptor failed

FLASH_PROMPT_ERROR is returned if **FLASH_FLAG_PROMPT** was specified and the user cancelled the operation

FLASH_ERROR is returned if the write failed

FLASH_VERIFY_ERROR is returned if errors are found during verification.

12.2.1.3 Flash Device Driver Source Code Files

The following table lists the directory and file names for the flash device driver, device driver descriptor file, and diagnostic program source code.

Directory and File Name	Description
flash\makefile	Makefile for the driver, descriptor and library.
flash\flshsamp.c	Diagnostics application.

Directory and File Name	Description
flash\sampmake.mak	Makefile for the diagnostics application.
flash\desc\makefile	Descriptor makefile.
flash\desc\systype.h	Descriptor definition file.
flash\desc\fl1.c	Dummy file for os9make.
flash\desc\systype.des	Flash device specific defines.
flash\desc\am29f016.des	Descriptor device specific storage definition.
flash\drv\flash.c	Device driver high level functions.
flash\drv\flashdrv.c	Low level device driver.
flash\drv\debug.c	Functions supporting device driver debug.
flash\drv\makefile	Device driver makefile.
flash\drv\drvmain.c	Device driver main function.
flash\drv\defsfile	Device driver main include file.
flash\drv\am29f016.h	Definitions specific to flash device driver.
include\flash.h	Flash include file.

12.2.1.4 Flash Device Driver Modules

The following table lists the flash device driver modules.

Module	Description
flashsamp	Flash diagnostics program.
am29f016	Flash device driver.
flashlib.1	Library containing user callable functions.
fl1	Flash device driver descriptor.

12.2.2 Transport Demultiplexer Device Driver

The transport demultiplexer device driver provides the interface between the application software and the VLSI VES2020 transport demultiplexer. It allows high-level control of the demultiplexing of an incoming transport stream. The device driver manages queues for receiving MPEG-2 PES data (audio and video streams) and PSI (DVB SI) sections and delivers these sections to the application software for specialized processing and channel selection.

The transport demultiplexer is controlled using the DAVIDLite DUXMAN interface. An application uses **_os_setstat()** calls to the MPFM device driver to indirectly control this interface. The source code for the demonstration software described in Section 12.4 provides an example of using this interface.

To use the DUXMAN interface, the modules **dmxucode** (microcode for the transport demultiplexer), **dux** (the device driver descriptor) and **dxm_drvr** (the device driver) must be incorporated into the OS-9000 executable image.

The transport demultiplexer device driver has the following limitations:

- The processing of program specific information (PSI) data is limited to program association table (PAT) and program map table (PMT) section information.
- PAT processing assumes that the PAT is fully contained in section 0. Section 0 can be split across multiple transport packets.
- PMT processing assumes that all PMT information for a given packet identification (PID) is contained in a transport packet. Additionally, multiple PMTs can be mapped to the same PMT PID. If multiple PMTs are mapped to the same PMT PID, the associated PMTs can be returned in the same buffer using a single call, unless a PMT is split across multiple transport packets.
- When program clock reference (PCR) PID filtering is enabled, DUXMAN detects PCR events and discontinuity. An application must explicitly handle these events by inserting the **EV_PCR** or **EV_PCR_DISCONTINUITY** events (or both) and providing a callback function to DUXMAN. The demo program included in the OS-9000 PSP includes an example of how PCR processing could be done using one of these events.

12.2.2.1 Testing the Transport Demultiplexer Device Driver

Successful execution of the demo program tests the transport device driver.

12.2.2.2 Transport Demultiplexer Device Driver Functions

The following table lists the supported DUXMAN functions. The DAVIDLite documentation provides detailed function descriptions.

Function	Description
_os_dvm_link()	Establishes proper linkage with the DUXMAN driver module.
_os_dvm_unlink()	Terminates linkage with the DUXMAN driver module.
_os_dxm_flush()	Stops the filtering of all active PIDs in the system and flushes the PAT.
_os_dxm_getstat()	Gets transport demultiplexer hardware registers.
_os_dxm_setstat()	Sets transport demultiplexer hardware registers.
_os_dxm_pid_addbuf()	Not implemented, returns -1.
_os_dxm_pid_delete()	Deletes PIDs associated with video or audio elementary streams.

Function	Description
<code>_os_dxm_pid_event()</code>	Inserts or removes the events EV_PCR and EV_PCR_DISCONTINUITY.
<code>_os_dxm_pid_getpsi()</code>	Retrieves PAT or PMT information.
<code>_os_dxm_pid_insert()</code>	Inserts PIDs associated with audio or video elementary streams.
<code>_os_dxm_pid_status()</code>	Not implemented, returns -1.

12.2.2.2.1 `_os_dvm_link()`

`_os_dvm_link()` must be used to establish proper linkage with the DUXMAN driver module. Calling `_os_dvm_link()` also loads and initializes the transport demultiplexer. When `_os_open()` is called to open an MPFM audio/video path, MPFM transparently calls `_os_dvm_link()` to establish proper linkage to the DUXMAN driver.

`_os_dvm_link()` initializes DUXMAN and returns a device manager handle to MPFM to be used in subsequent calls. During DUXMAN initialization, the transport demultiplexer is initialized. Initialization includes setting initial register values, testing on-chip DRAM, setting up data queues, loading microcode, installing the interrupt handler, and initiating the execution of the microcode and setup for the initial filtering of PAT information.

DUXMAN loads microcode into the transport demultiplexer by linking to a microcode data module, **dmxucode**, and copying the microcode into on-chip DRAM.

`_os_dvm_link()` creates an OS-9000 event, **duxman**, that is used as a timeout mechanism during the retrieval of PMT information. This event is resident until DUXMAN is terminated using `_os_dvm_unlink()`.

The interrupt handler is installed using a vector number 5 and a priority 5 (refer to `_os_irq()` in the OS-9000 for PowerPC documentation). The interrupt handler remains installed until DUXMAN is terminated using `_os_dvm_unlink()`.

`_os_dvm_link()` sets up five data queues: a 256KB video queue for direct transfer to the video decoder; an 8KB audio queue for direct transfer to the audio decoder; an 8KB message queue for the capture of private and PTS data (currently disabled); a 4KB queue for capturing PAT sections and a 4KB queue for capturing PMT sections.

`_os_dvm_link()` enables the bit in the PPC403 External Interrupt Enable Register (EXIER) used to detect transport demultiplexer interrupts.

12.2.2.2.2 `_os_dvm_unlink()`

`_os_dvm_unlink()` must be called as part of the termination process when exiting an application. All transport functions are terminated, queues are flushed, memory is deallocated and execution of the transport demultiplexer microcode stops.

The **duxman** event created in `_os_dvm_link()` is deleted, the interrupt handler is deactivated, and the PPC403 EXIER bit used to detect transport demultiplexer interrupts is disabled.

When **_os_close()** is called to close an MPFM audio/video path, MPFM transparently calls **_os_dvm_unlink()** as part of termination.

12.2.2.2.3 **_os_dxm_flush()**

Calling **_os_dxm_flush()** causes the filtering of all active PIDs in the system to cease and the PAT table to be flushed. Since DUXMAN caches the PAT, PID 0 is automatically reinserted to force a PAT refresh.

This function can be accessed though MPFM using **_os_setstat()** calls with a **code** parameter value of **DECOD_S_DXM_FLUSH**.

12.2.2.2.4 **_os_dxm_pid_delete()**

_os_dxm_pid_delete() deletes PIDs associated with video or audio elementary streams. Using this function to handle PIDs associated with private data is not supported. The function can also delete a program clock reference (PCR) PID, because it can be transmitted separately from the current video and audio PIDs.

This function can be accessed though MPFM using **_os_setstat()** calls with a **code** parameter value of **DECOD_S_DXM_PID_DEL**.

12.2.2.2.5 **_os_dxm_pid_event()**

_os_dxm_pid_event() currently supports the insertion and removal of two events, **EV_PCR** and **EV_PCR_DISCONTINUITY**. Attempts to insert or remove any other events return **-1**.

_os_dxm_pid_event() can be accessed though MPFM using **_os_setstat()** calls with a **code** parameter value of **DECOD_S_DXM_PID_EVENT**.

12.2.2.2.6 **_os_dxm_pid_getpsi()**

_os_dxm_pid_getpsi() retrieves of PAT or PMT information. Retrieval of CAT information is not supported. Retrieved PAT information is returned directly from a cached copy. PAT information is refreshed immediately following a call to **_os_dxm_flush()**, or when a new version of the PAT arrives in the transport stream.

PMT information is retrieved from the incoming transport stream on demand. An OS-9000 alarm triggers a timeout by the **duxman** event if the requested PMT does not arrive within 200 clock ticks. The timeout duration can be altered by calls to **_os_dxm_setstat()**. The **duxman** event (see the preceding **_os_dvm_link()** description) is triggered from either the interrupt handler, when a valid PMT has arrived, or from a timeout function signaled when the alarm timeout expires.

Automatic detection of a new PAT version relies on the convention that the PAT version number increases by one. If the PAT version changes in any other manner, such as changing input streams, the new PAT is not detected until the cached PAT is flushed.

_os_dxm_pid_getpsi() can be accessed though MPFM using **_os_setstat()** calls with a **code** parameter value of **DECOD_G_DXM_GET_PSI**.

12.2.2.2.7 `_os_dxm_pid_insert()`

`_os_dxm_pid_insert()` allows the insertion of PIDs associated with audio or video elementary streams. Setting the PCR bit, as described in the DAVIDLite documentation, triggers the detection of incoming PCR events, but the occurrence of these events does not report to the application unless the **EV_PCR** or **EV_PCR_DISCONTINUITY** events are enabled.

`_os_dxm_pid_insert()` also supports the insertion of a PCR PID separately from the audio and video stream PIDs if PCR is to be received in this manner.

This function can be accessed through MPFM using `_os_setstat()` calls with a **code** parameter value of **DECOD_S_DXM_PID_INS**.

12.2.2.2.8 `_os_dxm_setstat()`

`_os_dxm_setstat()` sets the transport demultiplexer hardware registers and can toggle the CHCKLP bit in the packet framer control register. Refer to the *VLSI VES2020 MPEG 2 Transport Demultiplexer Device Functional Specification* for detailed register information.

`_os_dxm_setstat()` can also alter the timeout associated with calls to `_os_dxm_pid_getpsi()` when that function is used to capture the PMT. The default timeout value is 200 clock ticks.

`_os_dxm_setstat()` can be accessed through MPFM using `_os_setstat()` calls with a **code** parameter value of **DECOD_S_DXM_WRITE_REG**, **DECOD_S_DXM_FLIPCLK** or **DECOD_S_DXM_PMT_TIMEOUT**.

Set the parameter **parm_blk** according to the **stat_decod_t** type (refer to the file **decod.h**). Register access requires a **reg_number** parameter to specify an offset to the register set base address. The register value is returned in the value field. When writing the PMT timeout value, the specify the timeout value in the value field.

12.2.2.2.9 `_os_dxm_getstat()`

`_os_dxm_getstat()` gets the transport demultiplexer hardware registers. Refer to the *VLSI VES2020 MPEG 2 Transport Demultiplexer Device Functional Specification* for detailed register information.

`_os_dxm_getstat()` can be accessed through MPFM using `_os_getstat()` calls with a **code** parameter value of **DECOD_G_DXM_READ_REG**, and **DECOD_G_DXM_PMT_TIMEOUT**.

12.2.2.3 Transport Demultiplexer Device Driver Source Code Files

The OS-9000 PSP does not provide source code, which is proprietary, for the transport demultiplexer device driver. Instead, the OS-9000 PSP provides the required binary modules.

12.2.2.4 Transport Demultiplexer Device Driver Modules

The following table lists the transport demultiplexer device driver modules.

Module	Description
dux	Transport demultiplexer descriptor module.
dxm_drvr	Transport demultiplexer device driver.
dmxucode	Transport demultiplexer microcode module.

12.2.3 MPFM (MPEG Decoder) Device Driver

The MPFM device driver controls operation of the CD21 MPEG decoder. High-level MPFM function calls result in calls to the device driver. The MPFM device driver are accessed by opening a video (**mv**) or audio (**ma**) descriptor. These descriptor files are included in the OS-9000 PSP.

Not all code required to build the descriptor files is included. DAVIDLite software is required to build the descriptor files.

The MPFM device driver installs an interrupt handler to retrieve information on the currently-playing video stream. The MPFM device driver also establishes a connection with the demultiplexer device driver using the **_os_dvm_link()** system call. To initialize the MPFM device driver correctly, the demultiplexer (DUXMAN) driver must be in the boot image.

The module **mp_cd21** contains the MPFM device driver. The library that contains the low-level MPEG decoder functions is called **decodlib.l**. The DAVIDLite software provides source code for the upper layer of the MPFM driver. To recompile the MPFM device driver, the upper layer of the driver is compiled and is supplied in binary format in file **mpcomm.l**.

The MPFM device driver does not support Microvision and close caption functions.

12.2.3.1 Testing the MPFM Device Driver

Successful execution of the demo program tests the MPFM device driver.

12.2.3.2 MPFM Device Driver Functions

Most of the functions in the MPFM device driver are accessed by issuing OS-9000 system calls. Consult the OS-9000 documentation for the specific functions used to control playback of the MPEG data.

The following table shows the implementation specific **_os_getstat()** and **_os_setstat()** codes and their function. The **_os_setstat()** and **_os_getstat()** calls can be issued against the **ma** or **mv** descriptor.

When calling **_os_setstat()** or **_os_getstat()**, the third function parameter should point to the **stat_decod** or **stat_dxm** structure. The **stat_decod** structure pointer should be used when executing MPEG decoder-specific calls. The **stat_dxm** structure should be used

when executing calls dealing with the transport demultiplexer. The table also indicates which structure pointer and structure members should be used with specific **_os_getstat()** and **_os_setstat()** calls.

Function	Description
DECOD_G_READ_REG	_os_getstat() call (stat_decod.reg_number). Returns value for the specified MPEG decoder register.
DECOD_S_WRITE_REG	_os_setstat() call (stat_decod.reg_number and stat_decod.value). Writes a value to the specified MPEG decoder register.
DECOD_G_AUDIO_STATUS	_os_getstat() call (stat_decod.value). Returns current audio status. A status value of 0x8000 indicates that audio is playing. A value of 0x4000 indicates that the audio processor is waiting for data. A value of 0x0000 indicates that the audio processor was reset.
DECOD_S_CONF	_os_setstat() call (stat_decod.conf_values). Reconfigures MPEG decoder according to specified values.
DECOD_S_EXE_CMD	_os_setstat() call (stat_decod.value). Executes specified control command. List of commands can be found in the MPEG decoder manual.
DECOD_G_STC	_os_getstat() call (stat_decod.stc). Retrieves the STC value from MPEG decoder registers 2, 3, and 4.
DECOD_S_STC	_os_setstat() call (stat_decod.stc). Assigns the STC values to the MPEG decoder registers 2, 3 and 4.
DECOD_S_DXM_FLUSH	_os_setstat() call. Calls _os_dxm_flush().
DECOD_S_DXM_PID_ADD_BUF	_os_setstat() call (dxm_stat.buf, dxm_stat.buf, dxm_stat.bufsize). Calls _os_dxm_pid_addbuf(). Not implemented in transport demultiplexer device driver.
DECOD_S_DXM_PID_DEL	_os_setstat() call (dxm_stat.pid, sxm_stat.buflen). Calls _os_dxm_pid_delete().
DECOD_S_DXM_PID_EVENT	_os_setstat() call (dxm_stat.pid, dxm_stat.ev_id, dxm_stat.ev_flag, dxm_stat.ev_parmblk, dxm_stat.ev_handler, dxm_stat.hdlstat). Calls _os_dxm_pid_event().
DECOD_G_DXM_GET_PSI	_os_getstat() call (dxm_stat.pid, dxm_stat.table_id, dxm_stat.table_rev, dxm_stat.buf, dxm_stat.bufsize). Calls _os_dxm_pid_getpsi().
DECOD_S_DXM_PID_INS	_os_setstat() call (dxm_stat.pid, dxm_stat.istr_type, dxm_stat.ostr_type, dxm_stat.buf, dxm_stat.bufsize). Calls _os_dxm_pid_insert().

Function	Description
DECOD_G_DXM_PID_STAT	_os_getstat() call (dxm_stat.pid, dxm_stat.blk). Calls _os_dxm_pid_status(). Not implemented in the transport demultiplexer device driver.
DECOD_G_DXM_READ_REG	_os_getstat() call. Calls _os_gestat() against demultiplexer device using DXM_G_READ_REG code.
DECOD_S_DXM_WRITE_REG	_os_setstat() call. Calls _os_setstat() against demultiplexer device using DXM_S_WRITE_REG code.
DECOD_S_DXM_FLIP_CLK	_os_setstat() call. Calls _os_setstat() against the demultiplexer device using the DXM_S_FLIP_FCR_CHKCLP code.
DECOD_G_DXM_PMT_TIMEOUT	_os_setstat() call. Calls _os_setstat() against the demultiplexer device using DXM_G_PMT_TIMEOUT.
DECOD_S_DXM_PMT_TIMEOUT	_os_setstat() call. Calls _os_setstat() against the demultiplexer device using DXM_S_PMT_TIMEOUT.

12.2.3.3 MPFM Device Driver Source Code Files

The following table lists the directories and file names for the source code containing MPEG decoder functions.

Directory\File Name	Description
decod\decod.c	MPEG decoder common functions.
decod\cd21acod	Audio microcode file.
decod\cd21vcod	Video microcode file.
decod\makefile	Makefile for the MPEG decoder functions and microcode modules.
desc\makefile	MPFM device driver descriptor makefile. This makefile creates mv and ma descriptors.
desc\mpfm_des.h	MPFM device driver descriptor include file.
drv\makefile	Device driver makefile.
drv\drv.c	Low level device driver code.
include\decod.h	MPEG decoder device include file.

12.2.3.4 MPFM Device Driver Modules

The following table lists the MPFM device driver modules.

Module	Description
ma	Audio MPFM descriptor module.
mv	Video MPFM descriptor module.
cd21acod	Audio microcode data module.
cd21vcod	Video microcode data module.
decodlib.l	Low level MPEG decoder functions library.
mp_cd21	MPFM audio/video device driver.
mpcomm.l	Upper portion of the MPFM device driver.

12.2.4 MFM (Graphics) Device Driver

The MFM device driver handles the MPEG decoder on-screen display (OSD) functionality and enables graphics to be displayed on the attached TV. The MFM device driver also handles MAUI graphics calls.

For the MFM device driver to function correctly, the digital video encoder (DENC) and MPFM drivers must be initialized. Issuing an **_os_open()** call against the MPFM and DENC descriptor files initializes both devices, using default settings.

Not all code required to build the descriptor files is included. DAVIDLite software is required to build the descriptor files.

The MFM device driver imposes the following restrictions, due to hardware limitations:

- Each view port is limited to 16 colors.
- Bit blit operations must be performed on an 8-byte boundary.
- Pixel size is fixed at 4 bits.
- View port regions can not overlap.
- Two view ports cannot occupy the same horizontal line.

The MAUI **gfx_set_vport_intensity()** system call alters the shading/blending level of displayed graphics. When RGB colors are assigned to a given view port, the upper 8 bits of the color value indicate whether the given color participates in blending/shading. If ones are placed in the upper 8 bits, changing view port intensity affects the given color. For example, an RGB color value of 0x0000FF00 (dark green) is not affected by changes in view port intensity. Changes in view port intensity, however, affect an RGB color value of 0x0100FF00 (dark green). View port intensity changes effects the blending of non-transparent colors and the shading of a transparent color (represented by an RGB value of 0x00000000).

12.2.4.1 Testing the MFM Device Driver

Successful execution of the demo program tests the MFM device driver.

12.2.4.2 MFM Device Driver Functions

No MFM driver functions can be directly called by user programs. The driver functions are called as a result of executing a call to an upper-level MAUI graphics subsystem.

12.2.4.3 MFM Device Driver Source Code

The following table lists the directories and file names for the MFM device driver source code containing the MFM device driver.

Directory\File Name	Description
mfm\osddrvr.c	All the function required by the MFM device driver.
mfm\makefile	Makefile for the descriptor, device driver and configuration description block.
mfm\config.h	Include file describing MPEG decoder low level MFM driver function names and some graphic device capabilities
mfm\global.h	Include file describing resolution and other capabilities.
mfm\hardware.h	Hardware specific definitions.
mfm\mfm_desc.h	Descriptor include file.
mfm\static.h	Device driver static storage definitions.
mfm\cdbl.a	Configuration description block source code.

12.2.4.4 MFM Device Driver Modules

The following table lists the MFM device driver modules.

Module	Description
mfm_desc	MFM driver descriptor
mfmupp.r	Upper layer for the MFM driver
mp_cd21o	MFM device driver
cdbl	Configuration description block

12.2.5 DENC Device Driver

The DENC device driver is a standard SCF device driver accessed using the OS-9000 device driver functions, such as `_os_open()`, `_os_setstat()`, `_os_read()` and `_os_close()`.

The DENC is accessed using the I²C bus. The I²C device driver must be installed before using the DENC device driver.

If the DENC device driver descriptor is open, the DENC initializes in the default NTSC mode.

12.2.5.1 Running the DENC Diagnostics

Run the **dencsamp** sample program to test the DENC. If **dencsamp** completes without printing error messages, the DENC is functioning properly.

12.2.5.2 DENC Device Driver Functions

Several function codes are defined for use with the `_os_setstat()` function.

Function	Description
SS_I2C_SETREG	Sets the register number that will be written to in a subsequent <code>_os_write()</code> call. If more than one byte of data is written in the <code>_os_write()</code> , it will be written into successive registers, starting with the one specified in this <code>_os_setstat()</code> call.
SS_DENC_COLOR_BAR_ON	Turns the color bar on.
SS_DENC_COLOUR_BAR_ON	Turns the color bar on.
SS_DENC_COLOR_BAR_OFF	Turns the color bar off.
SS_DENC_COLOUR_BAR_OFF	Turns the color bar off.
SS_DENC_SET_VIDEO_MODE	Sets the video mode to NTSC, PAL or SECAM, using one of these constants: <code>DENC_NTSC_MODE</code> , <code>DENC_PAL_MODE</code> , <code>DENC_SECAM_MODE</code> .

12.2.5.3 DENC Device Driver Source Code Files

The following table lists the directories and file names for the source code containing the DENC device driver.

Directory\File Name	Description
denc\makefile	Makefile for building the device driver and the descriptor.
denc\dencsamp.c	DENC sample program.
denc\sampmake.mak	Makefile for building the DENC sample program.
denc\desc\systype.h	Descriptor definition file.

Directory\File Name	Description
denc\desc\makefile	DENC descriptor makefile.
denc\desc\saa7182.des	DENC device driver specific definitions.
denc\desc\systype.des	DENC device specific defines.
denc\desc\denc1.c	Dummy file for the makefile.
denc\drv\dencdrv.c	DENC device driver low level functions.
denc\drv\defsfile	Device driver include file.
denc\drv\debug.c	DENC device driver debug functions.
denc\drv\dvrmain.c	DENC device driver main function.
denc\drv\makefile	Device driver makefile.
include\denc.h	DENC device driver application program include file.

12.2.5.4 DENC Device Driver Modules

The following table lists the DENC device driver modules.

Module	Description
denc1	DENC descriptor file.
dencsamp	DENC diagnostic application program module.
saa7182	DENC device driver module.

12.2.6 I²C Bus Device Driver

The I²C bus is used to communicate with several devices on the STBRP. The DENC and EEPROM devices are accessed only through the I²C bus. Additionally, a front-end device can connect to the I²C bus. The I²C bus controller on the STBRP is used to communicate with other I²C devices.

The I²C driver module creates two events, **i2c_s_event** and **i2c_w_event**. **i2c_s_event** serializes access to the I²C device driver. **i2c_w_event** indicates that an application program is requesting an action from the device driver. The device driver also creates two named pipes, **i2c_req** and **i2c_res**. The named pipes send messages between the application program and the device driver.

The I²C port is programmed to operate in 100KHz mode; the EEPROM device does not operate in 400KHz fast mode. The I²C port on the STBRP operating as a master for read and write operations.

If a sub address field is not required, up to four bytes of data can be transferred during a read or write operation. If a sub address is required, up to three bytes can be transferred. If more than four bytes must be transferred, multiple reads or writes must be performed. The I²C bus device driver polls the status register until the data is transferred.

No interrupt handler is installed to receive I²C bus interrupts.

The I²C device driver is contained in the **i2cdrvr** module. This driver must be started before calling any function using to the I²C bus.

To start the I²C device driver, type the **i2cdrvr&** command in the OS-9000 shell, or include **i2cdrvr** in the **SYS_PARAMS** define in the file **systype.h**.)

12.2.6.1 Testing the I²C Bus Controller

No specific diagnostic program test the I²C bus. If the DENC and EEPROM test procedure run successfully, the I²C bus is operational.

12.2.6.2 I²C Bus Device Driver Functions

The following table lists the I²C bus functions that can be called from an application program. The device driver must be running in order for the following functions to work correctly.

Function	Description
i2c_setup()	Initiates communication between the application program and the I ² C bus driver. This function must be called before any other I ² C bus functions can be used.
i2c_term()	Terminates communication between the application program and the I ² C bus driver. This function must be called after the application program is finished using I ² C device.
i2c_reinitialize()	Reinitializes the I ² C bus registers.
i2c_read()	Issues a read command on the I ² C bus.
i2c_write()	Issues a write command on the I ² C bus.

12.2.6.3 I²C Bus Device Driver Source Code Files

The following list shows the directories and file names for the I²C bus device driver source code.

Directory\File Name	Description
i2c\i2c.c	Functions used by the application programs.
i2c\i2cdrvr.c	Device driver.
i2c\makefile	Makefile for the I ² C bus, driver and library.
include\i2c.h	I ² C bus include file.

12.2.6.4 I²C Bus Device Driver Modules

The following table lists the I²C bus device driver modules.

Module	Description
i2cdrvr	I ² C bus device driver module.
i2clib.l	I ² C bus device driver application program function library.

12.2.7 EEPROM Device Driver

The EEPROM device driver is an SCF-based device driver accessed using the OS-9000 device driver functions **_os_open()** and **_os_close()**. The path identifier returned from **_os_open()** is used in the user functions described below.

The EEPROM device is accessed using the I²C bus, so the I²C device driver must be installed before using the EEPROM device driver.

12.2.7.1 Running the EEPROM Diagnostics

Run the **eeprsamp** sample program to test the EEPROM. If **eeprsamp** completes without printing error messages, the EEPROM device is functioning properly.

12.2.7.2 EEPROM Device Driver Functions

The following functions are located in the **eeplib.l** file.

Function	Description
eeeprom_write()	Writes data to the EEPROM.
eeeprom_read()	Reads data to the EEPROM.

The following table shows the EEPROM device driver implementation specific **_os_getstat()** and **_os_setstat()** codes and their function.

Function	Description
SS_EEPROM_SET_ADDR	Sets the address to write to. This is the offset from the start of the EEPROM.
SS_EEPROM_SET_READLEN	Sets the length, in bytes, of data to be read.

12.2.7.2.1 eeeprom_write()

Synopsis

```
#include <eeeprom.h>
int eeeprom_write(path_id path, int address, int len, unsigned char *data);
```

Description

eeeprom_write() writes data into the EEPROM device.

path is obtained from the **_os_open()** function.

address specifies the address where the data will be written.

len specifies its length of data to be written.

data points to the data to be written.

Returns 0 for success, non-0 otherwise.

12.2.7.2.2 eeeprom_read()

Synopsis

```
#include <eeeprom.h>
```

```
int eeeprom_read(path_id path, int address, int len, unsigned char *data);
```

Description

eeeprom_read() reads data from the EEPROM device.

path is the value returned from the **_os_open()** function.

If successful, **eeeprom_read()** returns 0. Otherwise, **eeeprom_read()** returns -1.

address specifies the address where data is read from the EEPROM.

len specifies length of data to be read.

data points to the space where the data will be held.

12.2.7.3 EEPROM Device Driver Source Code Files

The following table lists the directories and file names for the EEPROM device driver source code.

Directory\File Name	Description
eeeprom\makefile	Makefile for building the device driver and the descriptor.
eeeprom\eeep.rsamp.c	EEPROM sample program.
eeeprom\sampmake.mak	Makefile for building the EEPROM sample program.
eeeprom\desc\systype.h	Descriptor definition file.
eeeprom\desc\makefile	EEPROM descriptor makefile.
eeeprom\desc\x24165.des	EEPROM device driver specific definitions.
eeeprom\desc\systype.des	EEPROM device specific defines.

Directory\File Name	Description
eprom\desc\ee1.c	Dummy file for the makefile.
eprom\drvrx24165dv.c	EEPROM device driver low level functions.
eprom\drv\defsfile	Device driver include file.
eprom\drv\debug.c	EEPROM device driver debug functions.
eprom\drv\dvrmain.c	EEPROM device driver main function.
eprom\drv\eprom.c	EEPROM high level functions.
eprom\drv\makefile	Device driver makefile.
include\eprom.h	EEPROM device driver application program include file.

12.2.7.4 EEPROM Device Driver Modules

The following table lists the EEPROM device driver modules.

Module	Description
eeprsamp	EEPROM sample program.
ee1	EEPROM descriptor file.
eeplib.l	EEPROM application callable functions.
x24165	EEPROM device driver module.

12.2.8 Ethernet Device Driver

The Ethernet device driver provided in the basic OS-9000 package must be modified to run on the STBRP.

Use the following procedure to modify the Ethernet device driver. If OS-9000 was installed in a directory other than **C:\MWOS**, change the directory path accordingly:

1. Replace the files **misc.c**, **init.c**, and **ns8390.h** in the directory **C:\MWOS\SRC\IO\INET\DRV\NS8390** with the files **misc.c**, **init.c**, and **ns8390.h** in the directory **C:\TBD\Inet**.
2. Replace the file **systype.des** in the **C:\MWOS\OS9000\403\PORTS\403GCSTB\ISP** directory with the file **systype.des** in the directory **C:\TBD\enet**.
3. . Run the command **os9make** in the directory **C:\MWOS\OS9000\403\PORTS\403GCSTB\ISP** to build the new device driver.

This procedure creates a device driver that runs on the STBRP. Additional steps must be performed to configure TCP/IP for OS-9000. Refer to OS-9000 documentation for information about configuring TCP/IP.

12.2.9 Infrared (IR) Remote Control Device Driver

The IR remote control function is implemented using a Crystal CS8130 IR transceiver and serial port 1 (SP1) of the STBP. The IR transceiver uses SP1 to communicate with the device driver.

The IR remote control device driver only receives signals from an IR remote control.

To use the universal remote control supplied in the kit, program it with Zenith VCR code 034 and set it to operate in VCR mode.

The IR remote control device driver, in the module **irdrvr**, must be started before calling any functions related to the remote control. The **irdrvr&** command, typed to the OS-9000 shell, starts the IR remote control device driver. Alternatively, **irdrvr&** can be included in the **SYS_PARAMS** define in the file **systype.h**.

The IR remote control driver module creates two events, **ir_s_event** and **ir_w_event**. **ir_s_event** serializes access to the IR remote control device driver. **ir_w_event** indicates that a character was received from the IR transceiver, or that an application program is requesting an action from the IR remote control device driver.

The IR remote control driver creates three named pipes, **ir_req**, **ir_res**, and **ir_keys**. The **ir_req** pipe and **ir_res** pipe communicate between the driver and the application program. The **ir_keys** pipe buffers keys detected by the remote control.

The IR remote control device driver installs an interrupt handler for STBP serial port 1 to receive incoming characters from the IR transceiver. The interrupt handler is installed at priority 5.

When the IR remote control device driver is instructed to monitor for remote control keys, the driver initializes the STBP serial port and the IR transceiver and enables the receiver portion of the IR transceiver. When a remote control key is pressed, the IR transceiver sends multiple characters representing the key. After the characters are sent, the IR transceiver sends multiple 0xFF characters until the receiver portion of the IR transceiver is disabled. The IR remote control device driver disables the receiver after detecting that the characters representing a key were received. The receiver is immediately enabled again to receive the next key.

12.2.9.1 Testing the IR Transceiver

Run the test program **irtest** to test IR transceiver functionality. To start **irtest**, type **irtest** at the OS-9000 shell prompt. The test program echoes all pressed IR remote control keys to the terminal. If the correct keys display, the IR transceiver functions properly.

12.2.9.2 Additional IR Remote Control Key Functions

Several remote control keys perform additional functions. Pressing the Power key exits the diagnostic program. Pressing the Enter key changes the amount of debug information printed to the terminal. The Fast Forward key displays all IR transceiver register values. Pressing the reverse button enables the user to enter (at the terminal) a different UART

baud rate and the IR transceiver. Pressing the Play key allows the user to write a value to any IR transceiver register.

Note: Writing an arbitrary value to some registers can cause the IR remote control device driver to fail.

12.2.9.3 IR Remote Control Device Driver Functions

The following table lists the IR remote control functions that can be called from an application program. The IR remote control device driver must be running for the functions to work correctly.

Function	Description
ir_setup()	Initiates communication between the application program and the IR remote control device driver. This function must be called before any other IR functions are used.
it_terminate()	Terminates communication between the application program and the IR remote control device driver. This function must be called after the application program finishes using the IR remote control device.
ir_key_count()	Returns the number of keys in the receive pipe. When a remote control key is detected, the IR remote control device driver places the code for that key in the receive pipe.
ir_read_key()	Retrieves a key from receive pipe. If there are no keys in the pipe, the calling application program blocks.
ir_start_monitor()	Instructs the IR remote control device driver to start monitoring for IR remote control keys. Internally, the driver enables the IR transceiver receive circuit. No keys can be received before this call.
ir_stop_monitor()	Instructs the IR remote control device driver to stop monitoring for the IR remote control keys and shuts down the IR transceiver receive circuit.
ir_debug_commands()	Causes the IR remote control device driver to print debug messages for commands issued to the IR transceiver.
ir_debug_all()	Causes the IR remote control device driver to print all debug messages and to display all the characters received from the remote control.
ir_debug_off()	This function turns off all the debug messages.
ir_write_reg()	This function allows the application program to write to any IR transceiver register.
ir_set_baud()	Changes UART or IR transceiver baud rates.
ir_get_status()	Returns information about the IR transceiver registers.

12.2.9.4 IR Remote Control Device Driver Source Code Files

The following table lists the directories and file names for the IR remote control device driver source code.

Directory/File Name	Description
ir\ir.c	Functions used by device driver and the application programs.
ir\irdrvr.c	Device driver.
ir\irtest.c	Diagnostic/test program.
ir\makefile	Makefile for the remote control library, driver and test program.
include\ir.h	Remote control include file.

12.2.9.5 IR Remote Control Device Driver Modules

The following table lists the IR remote control device driver modules.

Module	Description
irdrvr	IR remote control device driver.
irlib.l	IR remote control application program functions.
irtest	IR remote control diagnostic program.

12.2.10 Keypad Device Driver

The keypad device driver polls the keypad to detect a key press at intervals controlled by the application program. By default, the polling interval is two clock ticks; the keypad device driver checks the keypad matrix every two ticks. The application program also controls the key repeat intervals. When a key is pressed and held down, the keypad device driver records the second key press after the key is held down position during the initial repeat count polls. The application program also controls the subsequent repeat count.

To use the keypad, the Ethernet daughterboard must be connected to the STBRP.

The keypad device driver, in the **keydrv** module, must be started before calling any function dealing with the keypad. The **keydrv&** command, typed to the OS-9000 shell, starts the keypad device driver. Alternatively, **keydrv&** can be entered in the **SYS_PARAMS** define in the file **systype.h**.

The keypad device driver module creates two events, **key_s_event** and **key_w_event**. **key_s_event** serializes access to the device driver. **key_w_event** indicates that an application program is requesting an action from the device driver.

The keypad device driver creates three named pipes, **key_req**, **key_res**, and **key_keys**. The **key_req** and **key_res** pipes send messages between the application program and the device driver. The **key_keys** pipe transfers the keys between the device driver and the application program.

12.2.10.1 Testing the Keypad

The **keytest** diagnostic program tests the keypad. After the keypad device driver is started, **keytest** can be started. **keytest** displays a menu describing key usage. When a keypad key is pressed, the number corresponding to the pressed key is written to the terminal. If all keypad keys are pressed and echo on the screen, the keypad device functions properly.

12.2.10.2 Keypad Device Driver Functions

The following table lists the keypad functions that can be called from an application program.

Function	Description
key_setup()	Sets up communication between the application and the keypad device driver. This function must be called before any other functions can be called by the application program.
key_terminate()	Terminates communication between the application program and the keypad device driver. This function must be called before the application that used smart card device driver exits.
key_start_monitor()	This function requests that the device driver starts monitoring the keypad for key presses.
key_stop_monitor()	This function requests that the device driver stops monitoring the keypad for key presses. No keys can be received until the device driver starts monitoring for the key presses.
key_key_count()	Returns number of keys present in the receive pipe. When key press is detected the device driver will place the number for that key in the receive pipe.
key_read_key()	Reads the key number detected by the device driver. If no keys have been detected by the device driver this call will block until a key is detected.
key_set_sample_rate()	Changes the rate with which the device driver checks for key presses. The parameter passed to this function is specified in ticks.
key_set_initial_repeat_count()	Sets the initial key repeat count.
key_set_repeat_count()	Sets the key repeat count, after the first repeat action.

12.2.10.3 Keypad Device Driver Source Code Files

The following table lists the directories and file names for the keypad device driver source code.

Directory\File Name	Description
key\key.c	Source file for the keypad application program functions.
key\keytest.c	Source for the keypad diagnostics program.
key\keydrv.c	Source for the keypad device driver program.
key\makefile	Makefile for the device driver, keypad test program and library.
include\keypad.h	Keypad include file.

12.2.10.4 Keypad Device Driver Modules

The following table lists the keypad device driver modules.

Module	Description
keydrv	Keypad device driver
keytest	Keypad diagnostics program
keylib.l	Library containing user callable functions

12.2.11 Serial Port Device Driver

The serial port device driver is a standard SCF device driver accessed using OS-9000 device driver functions, such as **_os_open()**, **_os_gs_ready()**, **_os_read()** and **_os_close()**.

12.2.11.1 Running the Serial Port Diagnostics

Run the **sersamp** sample program to test the serial port. If the sample program completes without printing error messages, the serial port device is functioning properly.

12.2.11.2 Serial Port Device Driver Functions

This device driver, which has been modified to work with the STBP UART, is identical to the 16550 UART device driver provided with the OS-9000 operating system.

12.2.11.3 Serial Port Device Driver Source Code

The following table lists the directories and file names for the serial port device driver source code.

Directory\File Name	Description
uart\makefile	Makefile for building the device driver and the descriptor.
uart\sersamp.c	Serial port sample program.
uart\sampmake.mak	Makefile for building the serial port sample program.
uart\desc\systype.h	Descriptor definition file.
uart\desc\makefile	Serial port descriptor makefile.
uart\desc\stbuart.des	Serial port device driver specific definitions.
uart\desc\systype.des	Serial port device specific defines.
uart\desc\ua1.c	Dummy file for the makefile.
uart\drv\irq.c	Serial port interrupt service function.
uart\drv\init.c	Serial port device driver initialization function.
uart\drv\stbuart.c	Low level serial port functions.
uart\drv\write.c	Serial port write function.
uart\drv\term.c	Serial port device driver termination function.
uart\drv\status.c	Device driver _os_getstat() and _os_setstat() functions.
uart\drv\defsfile	Device driver include file.
uart\drv\drvmain.c	Serial port device driver main function.
uart\drv\makefile	Device driver makefile.
uart\drv\stbuart.h	Serial port device driver application program include file.

12.2.11.4 Serial Port Device Driver Modules

The following table lists the serial port device driver modules.

Module	Description
ua1	Serial port descriptor file.
stbuart	Serial port device driver.
sersamp	Serial port sample program.

12.2.12 IEEE 1284 Port Device Driver

The IEEE 1284 port device driver is a standard SCF device driver accessed using OS-9000 device driver functions, such as `_os_open()`, `_os_gs_ready()`, `_os_read()` and `_os_close()`.

12.2.12.1 Running the IEEE 1284 Port Diagnostics

Run the **parsamp** program to test the IEEE 1284 port. If the sample program completes without printing error messages, the IEEE 1284 port device is functioning properly.

12.2.12.2 IEEE 1284 Port Device Driver Functions

The IEEE 1284 port device driver does not have any `_os_getstat()` or `_os_setstat()` functions.

12.2.12.3 IEEE 1284 Port Device Driver Source Code Files

The following table lists the directories and file names for the IEEE 1284 device driver source code.

Directory\File Name	Description
parallel\makefile	Makefile for building the device driver and the descriptor.
parallel\parsamp.c	IEEE 1284 port sample program.
parallel\sampmake.mak	Makefile for building the IEEE 1284 port sample program.
parallel\desc\systype.h	Descriptor definition file.
parallel\desc\makefile	IEEE 1284 port descriptor makefile.
parallel\desc\stbpar.des	IEEE 1284 port device driver specific definitions.
parallel\desc\systype.des	IEEE 1284 port device specific defines.
parallel\desc\pa1.c	Dummy file for the makefile.
parallel\drvrlirq.c	IEEE 1284 port interrupt service function.
parallel\drvrlinit.c	IEEE 1284 port device driver initialization function.
parallel\drvrlstbpar.c	Low level IEEE 1284 port functions.
parallel\drvrlwrite.c	IEEE 1284 port write function.
parallel\drvrlterm.c	IEEE 1284 port device driver termination function.
parallel\drvrlstatus.c	Device driver <code>_os_getstat()</code> and <code>_os_setstat()</code> functions.
parallel\drvrldefsfile	Device driver include file.
parallel\drvrldebug.c	IEEE 1284 port device driver debug functions.

Directory\File Name	Description
parallel\drv\drvmain.c	IEEE 1284 port device driver main function.
parallel\drv\makefile	Device driver makefile.
parallel\drv\stbpar.h	IEEE 1284 port device driver application program include file.

12.2.12.4 IEEE 1284 Port Device Driver Modules

The following table lists the IEEE 1284 port device driver modules.

Module	Description
pa1	IEEE 1284 port descriptor file.
stbpar	IEEE 1284 port device driver.
parsamp	IEEE 1284 port sample program.

12.2.13 Smart Card Device Driver

The smart card device driver uses the smart card interface in the STBP and the Philips 8002 smart card analog interface chip. Additionally, seven GPIO on the STBP control the 8002 chip. A smart card communicates using the UART in the STBP.

The STBRP supports only microprocessor type asynchronous smart cards. Because each smart card uses a different command set, the smart card device driver supports only generic read and write operations. These read and write operations are performed using T0 protocols, as specified in the ISO/IEC 7816-3 standard.

The smart card device driver installs an interrupt handler that is activated when a smart card is inserted or removed or when a character is received by the STBP.

The smart card device driver, contained in the **scdrvr** module, must be started before calling any function dealing with the smart card interface (SCI). Typing the **scdrvr&** command to the OS-9000 shell starts the smart card device driver. Alternatively, include **scdrvr** in the **SYS_PARAMS** define in the file **systype.h**.

The smart card device driver creates two events, **sc_s_event** and **sc_w_event**. **sc_s_event** serializes access to the smart card device driver. **sc_w_event** indicates that an application program is requesting an action from the smart card device driver.

The smart card device driver creates two named pipes: **sc_req** and **sc_res**, which send messages between the application program and the device driver.

12.2.13.1 Testing the SCI

The test program **sctest** tests the SCI. The provided smart card must be inserted into the smart card reader, component side down, and the smart card device driver must be started.

After **sctest** starts, it reads input from the terminal and performs the action specified by the single-letter smart card commands.

The following commands must be tested; steps 1 and 2 must be done first.

1. **i**; activates smart card monitoring by the smart card device driver
2. **a**; activates the smart card. Verify that the following answer to reset string is received from the card: 3b, f8, 11, 20, 03, 40, ff, ff, ff, ff, ff, 12, 10, 90, 00.
3. **z**; creates password 1 in the smart card. This action can be performed only once on each smart card. Verify that the ack code is set to 40, sw1 is set to 90, and that sw2 is set to 00. Because smart cards use different command sets, this option only works with the provided smart card supplied in the kit.
4. **y**; presents the password 1 command. This action can be performed any number of times after password 1 is created. Verify that the ack code is set to 42, sw1 is set to 90, and that sw2 is set to 00. Because smart cards use different command sets, this option only works with the provided smart card supplied in the kit.

Typing **h** while **sctest** is running displays the test program help menu.

12.2.13.2 Smart Card Device Driver Functions

The following table lists the smart card device driver functions that can be called from the application program.

Function	Description
sc_setup()	Sets up communication between the application and the smart card device driver. This function must be called before any other functions can be called by the application program.
sc_terminate	Terminates communication between the application program and the smart card device driver. This function must be called before the application using the smart card device driver exits.
sc_start_monitor()	Requests the device driver to start monitoring the SCI. This function must be called before the smart card is activated or any other operation relating to the smart card is performed.
sc_stop_monitor()	Instructs the device driver to stop monitoring the smart card interface.
sc_get_status()	Retrieves current smart card status.
sc_reset_card()	Performs the reset operation for the smart card.
sc_sctivate_card()	Performs activation sequence for the smart card.
sc_deactivate_card()	Deactivates the card.
sc_t0_write()	Performs T0 protocol write operations
sc_t0_read()	Performs T0 protocol read operations.

Function	Description
sc_loopback()	Performs a loopback operation on the serial port attached to the smart card.

12.2.13.3 Smart Card Device Driver Source Code Files

The following table lists the directories and file names for smart card device driver source code.

Directory/File Name	Description
sc\sc.c	Functions used by the application program and device driver.
sc\sctest.c	Test program for the smart card device.
sc\scdrvr.c	Device driver.
sc\makefile	Makefile for the smart card, driver, test program and library.
include\sc.h	Smart card device include file.

12.2.13.4 Smart Card Device Driver Modules

The following table lists the smart card device driver modules.

Module	Description
scdrv	Smart card device driver.
sctest	Smart card diagnostics program.
sclib.l	Library containing user callable functions.

12.2.14 STBP Chip Functions

Code in the **gpiolib.l** library provides functions that handle general purpose input/output (GPIO) and STBP interrupts.

12.2.14.1 Testing the STBP Chip Functions

To test the STBP, all other diagnostics programs, along with the test program **ledtest**, must be run. In addition the program must also be executed. **ledtest** turns the LEDs on the STBRP off and on. If the STBP functions properly, the output of the LEDs is visible.

12.2.14.2 STBP Functions

The following table lists the STBP functions that can be called from the application program. The interrupt functions deal only with the STBP interrupts.

Function	Description
gpio_write()	Writes given data to the specified GPIO line.
gpio_read()	Reads data from the given GPIO line.
gpio_set_direction()	Initializes direction (in/out) for the specified GPIO line.
asic_int_set_level()	Initializes the interrupt level for a given interrupt.
asic_int_set_polarity()	Sets polarity for given interrupt level.
asic_int_on()	Enables specified interrupt
asic_int_off()	Disables specified interrupt
asic_int_get_status()	Reads interrupt status for a given interrupt number.

12.2.14.3 STBP Source Code

The following table lists the directories and file names for the STBP source code.

Directory\File Name	Description
gpio/gpio.c	STB Peripheral Chip functions
gpio/ledtest.c	Source code for the test program
gpio/makefile	Makefile for the test program and the STBP library
include/asic.h	Include file for the STBP functions

12.2.14.4 STBP Modules

The following lists the STBP modules.

Module	Description
ledtest	STBP GPIO test program
gpilib.l	Library containing the STBP functions

12.3 Conditional Access Support

Because of the diverse and proprietary nature of conditional access solutions, the implementation of a completely generic system is impossible. Accordingly, the STBRP attempts to provide the basic building blocks of a conditional access system rather than an actual implementation.

The STBRP provides software interfaces to a smart card, a DVB descrambler, non-volatile storage, and an auxiliary serial port that could communicate with a back-channel modem.

12.4 Demo Program Source Code

The STB Reference Design Kit provides a demo program that runs on the STBRP. The following table shows the directories and file names for the demo program.

Directory\File Name	Description
demo\demo.c	General purpose functions used by demonstration program
demo\makefile	Makefile for the demolib.l library
include\dmeo.h	Function prototypes and common defines for demonstration program
stb\main.c	Main source code for the stb module
stb\ev_h.c	PCR event handler code. This code must be in a separate file so that it can be compiled without stack checking option
stb\makefile	Makefile for the stb module

12.5 Miscellaneous Files and Directories

The following table shows additional files included in the STPRP kit.

Directory\File Name	Description
include\demux.h	Transport demultiplexer device driver include file.
lib\tftpd.exe	TFTP daemon executable file.
lib\tftp.ico	TFTP daemon icon.
lib\tftp.res	TFTP daemon resource file.
lib\bootpd.exe	BOOTP daemon executable file.
lib\bootpd.ico	BOOTP daemon icon.
lib\bootpd.res	BOOTP daemon resource file.
lib\make\common.mak	Include file for the makefile.
lib\bootptab.sam	Sample bootptab file.
lib\makefile	Top level makefile.
lib\evbboot	Sample application image file.

Directory\File Name	Description
lib\stbxxxxx.img	ROM Monitor image file. This is a self burning file that can be downloaded using ROM Monitor.
lib\appxxxxx.img	Sample application image file. This is a self burning file that can be downloaded using ROM Monitor.

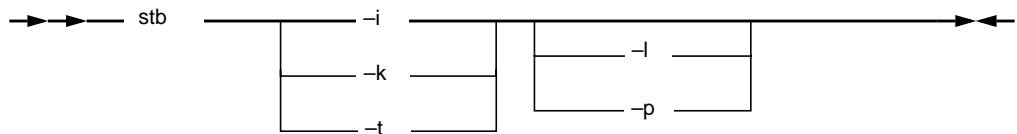
Chapter 13. Using the Demo Program

The Set-Top Box (STB) Reference Design Kit provides a demo program that runs on the STB Reference Platform (STBRP). The demo program can run under either the OS-9000™ real-time operating system (RTOS) or the pSOS™ RTOS.

This chapter describes how to use the demo program.

13.1 Starting the OS-9000 Demo Program

The demo program, contained in the module **stb**, can start in several modes. The following syntax diagram describes the different options. One of the first three possible input options must be specified in order for the demonstration program to start properly.



- i** Enables infrared (IR) remote input.
- k** Enables keypad input. The Ethernet daughterboard must be present.
- t** Enables terminal input. I/O is over a VT100 terminal emulator session using the OS-9000 shell.
- l** Initializes the MPEG decoder in low-memory mode; only 2MB of MPEG memory is used.
- p** Configures the MPEG decoder and digital video encoder (DENC) to run in PAL output mode instead of the default output NTSC mode.

Before running **stb**, the I²C device driver must be started. To start the I²C device driver, enter the **i2cdrv&** command at a OS-9000 shell prompt.

If IR remote control is used, the IR transceiver device driver must be started. To start the IR transceiver device driver, enter the **irdrvr** command at a OS-9000 shell prompt. The keypad device driver, if required, is started by running the **keydrv** command from the OS-9000 shell.

The **stb** program contains two user interfaces, a graphical user interface (GUI) or VT100 terminal emulation. If the selected input is IR remote control or the keypad, the GUI is used. Section 13.3 describes the menus and the features of the graphical user interface. Section 13.4 describes the ASCII interface of the VT100 interface.

13.2 Starting the pSOS Demonstration Program

To start the demo program, enter the **ap** command in the terminal emulator. Subsequent input comes from the IR remote control or from the keypad on the Ethernet daughterboard.

13.3 Using the Demo Program GUI

The GUI menus, displayed on the screen of the connected TV, can be navigated using the IR remote control or the keypad on the Ethernet daughterboard. Pressing the numeric key that corresponds to the number on an on-screen push button selects a push button. When a push button is selected, it changes color. After a push button is selected, pressing the Enter key causes the demo program to perform the action associated with the push button.

If the IR remote control is used, the Channel Up and Channel Down keys navigate the push buttons.

If the keypad is used, key 9 is the Enter key.

13.3.1 The Main Menu

Figure 13-1 illustrates the main menu.

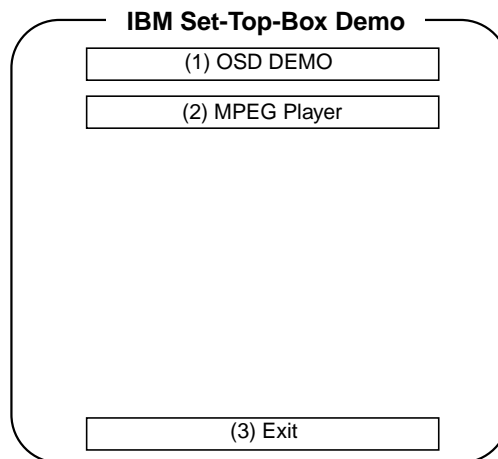


Figure 13-1. Main Demo Program Menu

Selecting the OSD Demo push button displays the OSD menu. See Section 13.3.2.

Selecting the MPEG player push button displays the MPEG Player menu. See Section 13.3.3.

Selecting the Exit push button causes the demonstration application to exit.

13.3.2 The OSD Demo Menu

The OSD demo menu, illustrate in Figure 13-2, displays several OSD demo choices.

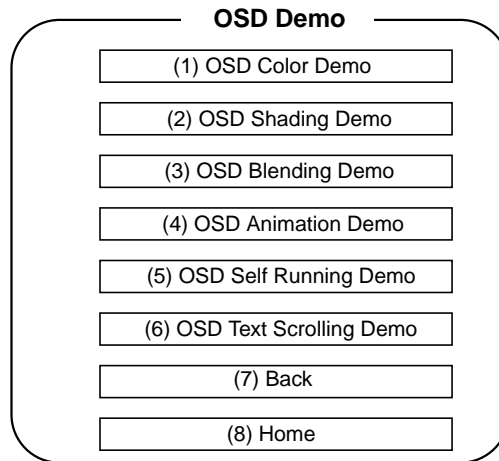


Figure 13-2. OSD Demo Menu

Selecting the OSD Color Demo push button displays each of the 16 colors that can be simultaneously supported in the MPEG decoder OSD region.

Selecting the OSD Shading Demo push button shows the shading function. Shading allows for darkening the video, thus showing through the transparent color.

Selecting the OSD Blending Demo push button shows how video can be mixed with the OSD graphics.

Selecting the OSD animation demo push button starts a simple application that frequently updates the OSD graphics.

Selecting the OSD Self Running Demo push button starts a demo that can run unattended. The self-running demo continuously displays OSD demo screens 1, 2, 3, 4, and 6. Pressing any key on the IR remote control or keypad stops the self-running demo.

Selecting the OSD Text Scrolling Demo push button horizontally scrolls text across the screen.

Selecting the Back or Home push button returns to the main menu.

13.3.3 The MPEG Player Menu

Figure 13-3 illustrates the MPEG Player menu, which displays the MPEG player shell options.

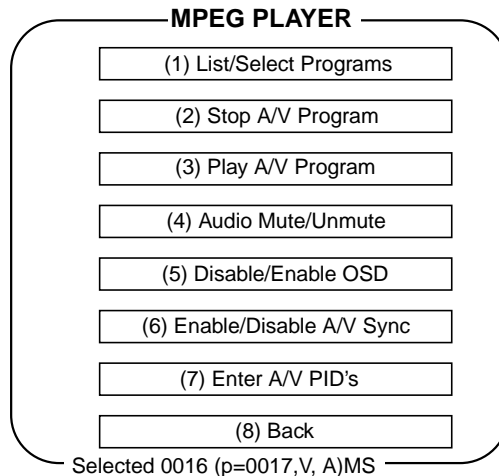


Figure 13-3. MPEG Player Menu

Selecting the List/Select Programs push button displays the List/Select Programs menu, described in Section 13.3.3.2. When the List/Select Programs menu is selected, the demo program tries to acquire current program information from the current stream. Program information is retrieved from the MPEG PAT and PMT.

Selecting the Stop A/V Program push button stops playback of the current program and flushes the PAT table, which must be flushed before new program information can be retrieved.

Selecting the Play A/V Program push button starts playback of the selected program. Program information for the selected program is displayed on the status line:

- The number inside the parentheses is the program number of the current playing.
- V indicates that the playing program contains video data.
- A indicates that currently playing program contains audio data.
- M, outside the parentheses, indicates that audio is muted
- S indicates that audio/video (A/V) synchronization is enabled and that all PCR events will be processed (This option is available only in the OS-9000 version of the demo program).
- s indicates that the PCR event is processed only if discontinuity is detected (A capital S is used in the pSOS version of the demo program).

Selecting the Audio Mute/Unmute push button toggles muting of the audio output.

Selecting the Disable/Enable OSD push button toggles OSD. If OSD is disabled, pressing any key enables OSD.

Selecting the Enable/Disable A/V Sync push button toggles A/V synchronization and toggles the type of A/V synchronization. See Section 13.3.3.1 for more information.

Selecting the Enter A/V PID's push button selects the Enter Program Information menu. If the PAT or PMT cannot be retrieved from the stream, the Enter Program Information menu enables manual entry of program information.

13.3.3.1 The Program Menu

Figure 13-4 illustrates the program entry menu.

Enter Program Information

Input data followed by Enter

Program (0-99):

Audio PID (0-8191):

Video PID (0-8191):

PCR PID (0-8191):

(7) Back

(8) Home

Figure 13-4. Program Entry Menu

If the Enter Program Information menu is selected, the demo program no longer attempts to filter the PAT and PMT tables to automatically acquire program information

The Enter program Information menu cannot be exited until program information is entered. If the Enter program Information menu is entered accidentally, put a 0 in each menu field:

- Program ID
- Program audio PID
- Program video PID
- Program PCR PID

After filling in each field, press the Enter key to continue to the next field. After all fields are entered, button 7 is highlighted to indicate that the input was processed.

If the IR remote control is used for input, pressing Rewind erases the most recently entered number. If the keypad is used for input, numbers must be entered in octal. Key 8 enters the number 0.

13.3.3.2 The List/Select Programs Menu

Figure 13-5 the List/Select Programs menu, which selects the program to be played.

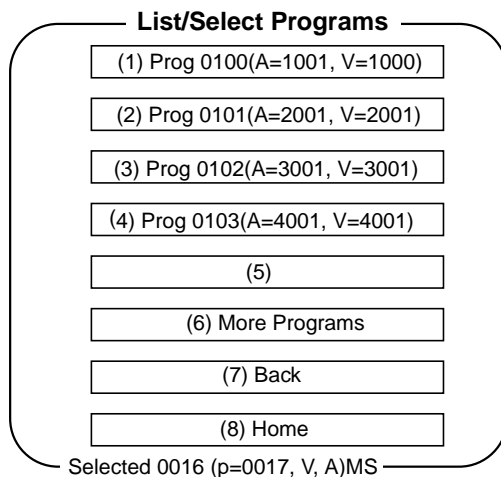


Figure 13-5. List/Select Programs Menu

The demo program attempts to acquire program information from PAT and PMT tables. If program information is entered manually in the Enter Program Information menu, the demo program does not attempt to acquire program information from the stream. Instead, it displays currently available program information. The process of acquiring program information from the stream can be lengthy.

To select a program, pressing the associated number key followed by the Enter key.

The status line at the bottom of the screen is not displayed while the demonstration program acquires program information. Available programs can be selected when the status line is displayed.

Selecting push button 6 redispays the program information, unless the stream contains more than five programs. In that case, selecting push button 6 displays more program information.

13.4 Using the VT 100 User Interface

The VT 100 terminal user interface is available only in the OS-9000 version of the demo program.

The VT 100 terminal user interface provides maximum flexibility in playing MPEG streams and debugging.

When terminal user input is selected, a simple command is entered from the terminal attached to the STBRP. The following table lists the terminal user interface commands.

Command	Parameters	Description
1	# 0 1	Inserts an audio PID, which must be entered in hexadecimal without a 0x prefix. If 0 is specified after the PID, the PID is not treated as a PCR PID. After this command finishes, audio information is forwarded from the demultiplexer to the decoder.
2	# 0 1	Inserts a video PID, which must be entered in hexadecimal without a 0x prefix. If 0 is specified after the PID, the PID is not treated as a PCR PID. After this command finishes, video information is forwarded from the demultiplexer to the decoder.
3		Creates the audio descriptor. This command must run before audio plays.
4		Creates video descriptor. This command must run before video plays.
5		Starts playing audio and video. OS-9000 waits until a video play command is issued before playing audio and video.
6		Starts playing audio and video. OS-9000 waits until an audio play command is issued before playing audio and video.
7		Starts playing audio.
8		Starts playing video.
9		Flushes all currently inserted PIDs.
a		Stops playing audio.
b		Stops playing video.
d		Issues a channel switch command to the MPEG decoder.
e	#	Changes to the specified channel. The channel number must be entered in hexadecimal without the 0x prefix.
f	#	Deletes the specified PID. The PID number must be entered in hexadecimal without the 0x prefix.
g	#	Inserts the specified PID as a PCR PID. The PID number must be entered in hexadecimal without the 0x prefix.
h		Display characteristics of the currently playing stream.
c		Flips Demultiplexer Framing Control Register bit 10.
p		Retrieves PAT and PMT information from the stream.
l		Displays PAT and PMT information.

Command	Parameters	Description
r	#	Reads and displays a demultiplexer register. The register number must be entered in hexadecimal without the 0x prefix.
w	# #	Writes a demultiplexer register. The register number must be entered in hexadecimal without the 0x prefix. Data to be written must be specified in hexadecimal without the 0x prefix.
i	#	Reads and displays a decoder register. The register number must be entered in hexadecimal without the 0x prefix.
o	# #	Writes a decoder register. The register number must be entered in hexadecimal without the 0x prefix. Data to be written must be specified in hexadecimal without the 0x prefix.
s	#	Inserts the PID associated with the specified program. The program number must be entered in hexadecimal without the 0x prefix.
t		Enables PCR processing.
x		Disables PCR processing.
q		Quits the demo program.

Appendix A. Programmable Logic Equations

This appendix lists the programmable logic equations for the following devices:

- ispGAL22V10C, on the STB Reference Platform (STBRP)
- ispLSI 1024-90LJ, on the Ethernet daughterboard attached to the STBRP

A.1 ispGAL22V10C Equations

The ispGAL22V10C equations follow.

```
MODULE ready_eq
"Rev A. Qualify WBEN0 WITH R_W DURING WRITES TO ASIC
"Rev B. Chip select change CS0 FOR CS2, EXCHANGED PIN 11 FOR PIN9
"Rev C. Seperate WBE0# signal into Transport chip

"INPUTS
    ASIC_27CLK      PIN 2;
    !ACK_DS         PIN 3;
    ASIC_READY      PIN 4;
    !ACK_EXP        PIN 5;
    !VES_ACK        PIN 6;
    R_W             PIN 7;
"
    !CS0            PIN 11;
    !CS1            PIN 10;
    !CS2            PIN 9;
    !CS3            PIN 12;
    !CS6            PIN 13;
"
    !RESET          PIN 16;
    !NMI_IN         PIN 21;
    !WBEN0          PIN 17;

"OUTPUTS
    X_READY         PIN 23 istype 'COM';
    !NMI            PIN 24 istype 'REG_D';
    NMID            PIN 25 istype 'REG_D';
    OUT_EN          PIN 26 istype 'COM';
    !WBEN0Q         PIN 27 istype 'COM';
    !WBEN0_TP       PIN 18 istype 'COM';

equations
    X_READY.OE =    OUT_EN;
    OUT_EN =      CS2
                # CS1
                # CS3
                # CS6;
```

```

X_READY = ACK_DS & CS2
          # ASIC_READY & CS1
          # ACK_EXP & CS3
          # VES_ACK & CS6;

NMID.CLK = ASIC_27CLK;
NMID := NMI_IN;

NMI.CLK = ASIC_27CLK;
NMI := NMID;

WBEN0Q = WBEN0 & !R_W;
WBEO_TP = WBEN0 & CS6;

END

```

A.2 ispLSI 1024-90LJ Equations

The ispLSI 1024-90LJ equations follow.

```

MODULE eth_con
"inputs
    PRQ      PIN 22;
    PRD      PIN 23;
    E_ACK    PIN 24; "FROM ETHERNET
    RX       PIN 25;
    TX       PIN 26;
    COL      PIN 27;
"    D0      PIN 28;
"    D1      PIN 29;
"    D2      PIN 30;
"    D3      PIN 31;
    D4      PIN 32;
    D5      PIN 33;
    D6      PIN 37;
    D7      PIN 38;
    ROW0     PIN 39;
    ROW1     PIN 40;
    ROW2     PIN 41;
    CS       PIN 42;
    R_W      PIN 43;
"    WBEO    PIN 44;
    A26      PIN 45;
    A27      PIN 46;
"    A28      PIN 47;
"    A29      PIN 48;
"    A30      PIN 13;
"    A31      PIN 14;
"    RESET   PIN 20;
    CLK27MHZPIN 54; "REGISTERED CLK
    BREQ     PIN 12;
"    SYSCLK   PIN; LOGIC CLK27MHZ

```

```

"OUTPUTS
    ETHERNETCSPIN ISTYPE 'COM';
    WACK    PIN 57 ISTYPE 'COM';
    RACK    PIN 58 ISTYPE 'COM';
    E_DIR   PIN 59 ISTYPE 'COM';
    G       PIN 60 ISTYPE 'COM';
    SAB     PIN 61 ISTYPE 'COM';
    SBA     PIN 62 ISTYPE 'COM';
    SRD     PIN 63 ISTYPE 'COM';
    SWR     PIN 6  ISTYPE 'COM';
    RX_LED  PIN 64 ISTYPE 'COM';
    TX_LED  PIN 65 ISTYPE 'COM';
    COL_LED PIN 66 ISTYPE 'COM';
    ACK     PIN 67 ISTYPE 'COM'; "TO PROCESSOR
    BACK    PIN 11 ISTYPE 'COM';
    COL0    PIN 3  ISTYPE 'REG_D';
    COL1    PIN 4  ISTYPE 'REG_D';
    COL2    PIN 5  ISTYPE 'REG_D';

"NODES
    PRQD    NODE ISTYPE 'REG_D';
    OEN     NODE ISTYPE 'COM';
    PTCLK   NODE ISTYPE 'COM';
    ETHERCSDNODE ISTYPE 'REG_D'; "ETHERNET DELAYED CHIP SELECT

DECLARATIONS
"A26  A27
"-----
"0    0      REGISTER ACCESS
"0    1      DMA OPERATION
"1    0      SWITCH
"1    1      POLLPRQ
DMACS = !CS & !A26 & A27;
SWITCHCS = !CS & A26 & !A27;
POLLPRQ = !CS & A26 & A27;

EQUATIONS
    RX_LED = !RX;
    TX_LED = !TX;
    COL_LED = !COL;
    !ETHERNETCS = !CS & !A26 & !A27 & !BACK;

    SAB = ETHERNETCS;
    SBA = ETHERNETCS;

    !RACK = DMACS & R_W;
    !WACK = DMACS & !R_W;

    !G = !ETHERNETCS
        # !G & ETHERCSD
        # DMACS & R_W
        # !PRD;

```

```

    !ACK = DMACS & PRQ
    # !E_ACK
    # SWITCHCS
    # POLLPRQ;

"PRQD.AR = !RESET;
  PRQD.CLK = CLK27MHZ;
  PRQD := PRQ;

E_DIR = !R_W;

  BACK = BREQ & ETHERNETCS;
  !PTCLK = !R_W & SWITCHCS;

  !SRD = !ETHERNETCS & R_W;
  !SWR = !ETHERNETCS & !R_W;

"SWITCH READ/WRITE LOGIC

  COL0 := D6;
  COL1 := D5;
  COL2 := D4;

  [COL0, COL1, COL2].CLK = PTCLK;
"[COL0, COL1, COL2].AR = !RESET;
  D7 = POLLPRQ & PRQD;
  D6 = SWITCHCS & ROW0;
  D5 = SWITCHCS & ROW1;
  D4 = SWITCHCS & ROW2;
  D7.OE = OEN;
  D6.OE = OEN;
  D5.OE = OEN;
  D4.OE = OEN;

  OEN = SWITCHCS & R_W
    # POLLPRQ & R_W;

  ETHERCSD.CLK = CLK27MHZ;
"ETHERCSD.AR = !RESET;
  ETHERCSD := !ETHERNETCS;

END

```

Appendix B. Bills of Materials

This appendix contains the bills of material for the STB Reference Platform (STBRP) and the Ethernet daughterboard.

B.1 STBRP Bill of Materials

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
48	C4, C12, C17, C18, C19, C20, C21, C22, C23, C24, C25, C30, C31, C32, C39, C40, C42, C46, C47, C48, C49, C50, C51, C54, C55, C58, C59, C61, C63, C64, C69, C71, C72, C74, C75, C76, C77, C78, C79, C80, C81, C82, C85, C86, C93, C94, C95, C96, C99, C102, C103, C105, C106, C107, C108, C109, C110, C111, C112, C113, C114, C115, C116, C117, C118, C119, C120, C121, C122, C123, C124, C125, C127, C128, C129, C130, C131, C133, C134, C135, C137, C138, C139, C140, C141, C142, C143, C144, C146, C147, C148, C150, C152, C153, C154, C155, C156, C158, C159, C160, C161, C163, C164, C165, C168, C169, C170, C171, C172, C173, C174, C175, C176, C177, C179, C180, C181, C182, C183, C184, C186, C187, C188, C189, C190, C191, C192, C193, C194, C195, C196, C198, C199, C200, C202, C203, C204, C205, C206, C208, C210, C212, C213, C214, C216, C218, C219, C222	Capacitor, 0.1 μ F	08055E104MAT050	AVX

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
33	C1, C5, C14, C35, C41, C45, C62, C65, C66, C73, C83, C84, C97, C98, C126, C132, C136, C145, C149, C151, C157, C162, C166, C167, C178, C185, C197, C201, C207, C209, C217, C220,	Capacitor, 10 μ F	TAJC106M010	AVX
5	C56, C87, C90, C91, C100	Capacitor, 22pF	08055A220MAT050	AVX
7	C36, C37, C38, C88, C89, C101, C104	Capacitor 470pF	08055A471MAT050	AVX
3	C26, C27, C28	Capacitor, 68pF	8055A680MAT050	AVX
3	C8, C9, C1	Capacitor, 560pF	08055A561MAT050	AVX
3	C13, C15, C16	Capacitor, 32pF	08055A320MAT050	AVX
2	C33, C34	Capacitor, 1.0 μ F	TAJA105M01	AVX
2	C6, C7	Capacitor, 3300pF	08055E332MAT050	AVX
2	C215, C211	Capacitor, 22 μ F/6V	TAJD226M006	AVX
11	C2, C3, C29, C43, C44, C52, C53, C67, C68, C70, C92	Capacitor, 0.33 μ F	8055E334MAT050	AVX
2	C11, C57	Capacitor, 0.68 μ F	08055E334MAT050	AVX
1	C223	Capacitor, 47 μ F	AJD476M006	AVX
1	C60	Capacitor, 100 μ F	TAJD107M006	AVX
5	D3,D4,D5,D6,D7	LED, Green	LU60355CT-ND	Lumex
1	D1	IR, LED	TSSF4500	Temic
1	D2	Photo diode	PV22NF	Temic
7	JP1, J6, J16, J21, J24, J25, J28	1 x 3, header	S1011-03-ND	Sullins
16	J7, J9, J10, J12, J13, J14,15, J17, J19, J22, J23, J27, J29, J30, J31, J35	2 x 10 header	SH-20DB-S1-TG	Robinson Nugent

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
1	J36	X8 header	NSH-16DB-S1-TG	Robinson Nugent
3	J3, J4, J5	RCA jack	16PJ097	Mouser
1	J1	S-Video	749264-1	AMP
1	J37	Smart Card connector	145207-3	AMP
1	J20	VMECON	535043-4	AMP
1	J18	Connector, 50 pin	04338-9	AMP
2	J11, J34	8-pin in-line header	103185-8	AMP
1	J2	SCART connector	DEI-0022	Dih Tain Precision
1	J32	4-pin power connector	15-24-4049	Molex
1	J3	SIMM Connector	822019-4	AMP
1	J8	4-pin header	S1011-04-ND	Sullins
4	J38, J39, J41, J42	2 x 1 header	S1011-02-ND	Sullin
1	J40	DIN power connector	SDS-50J	CP-2350-ND
8	L4, L5, L6, L7, L8, L9, L10, L11	Ferrite bead	BLM21A121SPT	Murata Erie
3	L1, L2, L3	Inductor, 1.8UH	ELJ-FAIR8KF2	Panasonic
			PCD1011CT-ND	Digikey
1	P3, straight	DB9 connector	747871-1	AMP
1	P2, right angle	DB9 connector	747840-6	AMP
1	P1, right angle	DB25 connector	747842-6	AMP
1	Q1	NPN transistor	MMBT3904L	Motorola

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
57	R7, R8, R36, R40, R41, R42, R43, R44, R45, R47, R48, R49, R51, R52, R53, R62, R63, R73, R77, R78, R79, R80, R82, R83, R91, R101, R102, R105, R106, R108, R118, R119, R123, R124, R125, R126, R127, R129, R130, R132, R133, R134, R135, R141, R142, R144, R146, R147, R148, R154, R157, R158, R159, R160, R161, R170, R173	10K ohm resistor	RM73B2A103J	KOA
12	R90, R93, R97, R114, R115, R116, R117, R136, R137, R138, R139, R140	33 ohm resistor	RM73B2A330J	KOA
1	R172	330 ohm resistor	RM73B2A331J	KOA
1	R155	100 ohm resistor	RM73B2A101J	KOA
40	R46, R54, R55, R58, R59, R60, R64, R67, R68, R69, R70, R81, R84, R85, R86, R87, R88, R92, R94, R96, R99, R100, R103, R104, R107, R109, R110, R111, R112, R113, R120, R121, R122, R131, R143, R145, R149, R156, R162, R171	1K ohm resistor	RM73B2A102J	KOA
1	R50	27K ohm resistor	RM73B2A273J	KOA
6	R26, R27, R56, R57, R61, R65	15K ohm resistor	RM73B2A153J	KOA
	R66	4.99K ohm resistor	RK73H2A4991F	KOA
	R24	12 ohm resistor	RM73B2A120J	KOA
2	R25, R23	20 ohm resistor	20RM73B2A200J	KOA
3	R28, R29, R30	47 ohm resistor	47RM73B2A470J	KOA
	R1, R2	499, 1%	RK73H2A4990F	KOA

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
6	R163, R164, R165, R166, R167, R169	0 ohm resistor	RM73B2A000J	KOA
2	R31, R32	56.2K ohm resistor	RK73H2A56221F	KOA
1	R151	10 ohm resistor	RM73B2A100J	KOA
1	R39	90.9, 1%	RK73H2A909RF	KOA
1	R38	150, 1%	RK73H2A1500F	KOA
5	R89, R95, R98, R128, R168	22 ohm resistor	RM73B2A220J	KOA
1	R37	24 ohm resistor	RM73B2A240J	KOA
8	R20, R21, R22, R33, R34, R35, R152, R153	75 ohm resistor	RM73B2A750J	KOA
1	R19	56 ohm resistor	RM73B2A750J	KOA
1	R150	2.2K ohm resistor	RM73B2A222J	KOA
12	R3, R4, R5, R6, R9, R10, R11, R12, R13, R14, R15, R16	4.7K ohm resistor	RM73B2A472J	KOA
1	RN1	Resistor network, 100	Y4101CT-ND	Panasonic\
2	RN2, RN3	Resistor network 2.2K	745-101-R222CT-ND	Digikey
2	S1, S2	SW pushbutton	EVQ-PJS04K	Panasonic
			P8048SCT-ND	Digikey
1	U32	PowerPC 403GC	PPC403GC	IBM
1	U2	Buffer, 3.3V	SN74LVT16244ADGG	TI\
1	U18	MPEG Transport Chip	VES2020X	VLSI
1	U10	VCXO, 27MHz 25ppm	K1525-27MHZ	Champion Technology
1	U22	Oscillator, 27MHz 25ppm	MA025HAH-27 MHz	MMD
1	U3	Oscillator, 14.31818Mhz	CTX115-ND	CTS

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
1	U17	MPEG Decoder	IBMCD21	IBM
10	U13, U15, U19, U24, U27, U28, U30, U33, U34, U35	DRAM, 5V, 256 x 16	TMS45160-60DZ	TI
1	U5	EuroDENC	SAA7182WPA	Philips
1	U4	Audio, DAC	CS4331-KS	Crystal
2	U1,U7	Line TX/RX, RS232	MAX562CWI	Maxium
1	U12	STB peripheral	STBP	IBM
1	U3	Transceiver	SN74LVT245ADBR	TI
1	U37	Smart Card interface	TDA8002AT/5	Philips
1	U39	Multi-standard infrared transceiver	CS8130-CS	Crystal
1	U36	Reset, power supervisor	MAX708TCSA	Maxium
1	U6	Regulator, 3.3V	LT1085CT	Linear Technology
3	U9, U20, U31	Transceiver, 5V	DT74FCT162245AT V	IDT
2	U16, U26	Buffer, 5V	DT74FCT162244AT V	IDT
1	U8	Descrambler	VES0010A2	VLSI
1	U11	2KB EEPROM	X24165S	Xicor
1	U23	2MB flash memory	AM29F016-90EC	AMD
2	U25, U29	FET buffer	QS3244S0	Quality Semicon
1	U14	PAL 22V10	ispGAL22V10	Lattice
1	Y1	Oscillator 3.6864 Mhz	CTX071	CTS
			CTX071-ND	Digikey

B.2 Ethernet Daughterboard Bill of Materials

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
20	C1, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C27	Capacitor, 0.1 μ F	08055E104MAT050	AVX
3	C2, C3, C4	Capacitor, 10 μ F/10V	TAJC206M010	AVX
4	C23,C24,C25,C29	Capacitor, 0.01 μ F	08055E103MAT050	AVX
2	C26,C28	Capacitor, 22 μ F/10V	TAJD226M010	AVX
1	C30	Capacitor, 0.001 μ F	08055E102MAT050	AVX
9	D1, D2, D3, D4,D5, D6, D7, D8, D9	Schottky Diode	—	Zetec
2	D10, D13	LED, green	—	Lumex
1	D11	LED, yellow	—	Lumex
1	D12	LED, red	—	Lumex
1	J1	1 x 8 header	103185-8	AMP
1	J2	VME connector	650908-5	AMP
1	J3	BNC connector	227161-7	AMP
1	J4	RJ-45 connector	520252-4	AMP
1	J5	3-pin header	—	Sullins
1	RN1	Resistor network, 330 ohm	EXB-V8V331JV	Panasonic
1	RN2	Resistor network, 1.5K ohm	EXB-V8V152JV	Panasonic
1	RN3	Resistor network, 4.7K ohm	EXB-V8V472JV	Panasonic

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
5	R1, R2, R3, R5, R21	Resistor, 10K ohm	RM73B2A103J	KOA
4	R6, R7, R10, R17	Resistor, 1K ohm	RM73B2A102J	KOA
2	R8, R9	Resistor, 499 ohm	RK73H2A4990F	KOA
2	R11, R12	Resistor, 50.1, 1%	RK73H2A501RF	KOA
2	R13, R16	Resistor, 271, 1%	RK73H2A2710F	KOA
2	R14, R15	Resistor, 67, 1%	RK73H2A670RF	KOA
1	R18	Resistor, 1M	—	KOA
1	R19	Resistor, 800, 1%	PK73H2A8000F	KOA
1	R20	Resistor, 22.1, 1%	RK73H2A221RF	KOA
9	S1, S2, S3, S4, S5, S6, S7, S8, S9	Push button switch	EVQ-QHF02W	Panasonic
1	T1	Transformer	PE64103	Pulse Engineering
1	T2	Transformer	FD12-101G	Halo Electronics
1	U1	FPGA	ispLSI1024-90LJ	Lattice
1	U2	Transceiver latch	74F646SC	National Semiconductor
1	U3	SRAM	HM6264AFP-70	Hitachi
1	U4	Latch	74F573SC	National Semiconductor
1	U5	Coax transceiver	DP8392CN	National Semiconductor
1	U6	Ethernet controller	DP83902AV	National Semiconductor

Quantity	Reference	Description	Manufacturer P/N	Manufacturer
1	U7	Voltage converter, 5V/9V	PM7202	Valor
1	U8	Oscillator, 20 MHz	—	CTS
2	J6, J7	20-pin header	NSH-20DB-S1-TG	Robinson Nugent
1	J8	2-pin header	—	Sullins

Appendix C. Acronyms and Abbreviations

ADC. analog-to-digital conversion
ANSI. American National Standards Institute
API. application programming interface
ASIC. application-specific integrated circuit
ASK. amplitude shift key
ASSP. application-specific standard products
ARM. alternate refresh mode
ATM. asynchronous transfer mode
BME. burst mode enable
BNC. Bayonet Niell-Councilman connector
CAS. column address strobe
CCIR. International Radio Consultive Committee
CIF. common interface format
CIP. common isochronous packet
CMOS. complementary metal oxide semiconductor
CMP. composite video
CPM. clock and power management
CPU. central processing unit
CRC. cyclic redundancy check
CSMA/CD. carrier sense multiple access with collision detection;
CSSP. customer-specific standard products
CTS. clear to send
CVBS. combined video blanking and sync
D/A. digital to analog converter
DAC. digital to analog converter
DAVIC. Digital Audio-Visual Council
DCD. data carrier detect
DCE. data communications equipment
DCR. device control register
DCT. discrete cosine transform
DDI. digital data interface
DENC. digital video encoder
DIN. Deutsches Institut für Normung (German Institute for Standardization)
DMA. direct memory access
DMSD. digital multi-standard decoder
DRAM. dynamic random-access memory
DSF. designated special function
DSP. digital signal processor
DSR. data set ready
DTO. discrete time oscillator
DTR. data transmit ready

DTSC. data three-state control
DVB. digital video broadcast; Digital Video Broadcasting System
DVBE. DVB Exerciser
DVD. Digital Versatile Disk
DVI. digital video interactive
EBIU. extended bus interface unit
EDO. extended data output
EEPROM. electronic erasable programmable read-only memory
EOT. end of transfer
ERM. early RAS mode
FIFO. first in, first out
FIR. finite impulse response
FIT. fixed interval timer
FM. frequency modulation
FPM. fast page mode
FPGA. field programmable gate array
GB. one billion (one thousand million) bytes
Gb. one billion bits
GPIO. general-purpose input/output
GPR. general-purpose register
GUI. graphical user interface
HSI. hue, saturation, and intensity
HSL. hue, saturation, and lightness
HSV. hue, saturation, and value
HSMC. high-speed memory controller
HSYNC. horizontal sync
IBM. International Business Machines
IEC. International Electrotechnical Commission
IEEE. Institute of Electrical and Electronics Engineers
IFD. interface device
I²C. Inter-Integrated Circuit
I/O. input/output
IP. Internet Protocol
IR. infrared receiver
IrDA. Infrared Data Association
ISO. International Standards Organization
ISR. Interrupt Status Register
JPEG. Joint Photographic Experts Group
JTAG. Joint Test Action Group
LL. link layer
LSB. least significant byte
lsb. least significant bit
LSSD. level-sensitive scan design
MAC. media access control
MAUI. multimedia application user interface

MB. one million (one thousand thousands) bytes
Mb. one million bits
MHz. one million Hertz (cycles per second)
MMIO. memory-mapped input/output
MMU. memory management unit
MPEG. Moving Pictures Experts Group
msb. most significant bit
MSB. most significant byte
NC. network computer
NIC. network interface card
NIM. network interface module
NIU. network interface unit
NTSC. National Television Systems Committee
OCM. on-chip memory
OE. output enable
OPB. on-chip peripheral bus
OSD. on-screen display
PAL. phase alternate line
PAT. program association table
PCM. pulse code modulation
PCR. program clock reference
PES. packetized elementary stream
PIA. peripheral interface adapter
PID. packet identifier
PIT. programmable interval timer
PLB. PowerPC local bus
PLL. phase-lock loop
PLLC. plastic leadless chip carrier
PMT. program map table
POF. plastic optical fiber
PSI. program-specific information
PSP. platform support package
PWM. pulse width modulation
QAM. quadrature amplitude modulation
QPSK. quadrature phase shift key
RAM. random-access memory
RAS. row address strobe
RE. ready enable
RGB. red, green, blue
RISC. reduced instruction set computer
ROM. read-only memory
RTOS. real-time operating system
RTS. ready to send
RZ. return to zero
SAP. secondary audio program

SAR. segmentation and reassembly
SBB. system building blocks
SCART. European standard for composite video broadcast signal
SCC. serial communications controller
SCI. smart card interface
SDRAM. synchronous dynamic random access memory
SGRAM. synchronous graphics random access memory
SI. service information
SIF. standard (or source) input format
SICC. serial and infrared communications controller
SIMM. single in-line memory module
SMPTTE. Society of Motion Picture and Television Engineers
SNR. signal-to-noise ratio
SOC. system-on-a-chip
S/PDIF. Sony/Philips Digital Interface Format
SPR. special purpose register
SRAM. static random access memory
STB. set-top box
STBP. STB peripheral (chip)
STBRP. STB Reference Platform
STC. system time clock
STP. shielded twisted pair
STU. set-top unit
S-Video. separate video
TAP. transport assist processor
TC. terminal count
TCP/IP. Transmission Control Protocol/Internet Protocol
TLB. translation lookaside buffer
TS. transport stream
UART. universal asynchronous receiver/transmitter
UIC. universal interrupt controller
UTP. unshielded twisted pair
VBI. vertical blanking interval
VCO. voltage-controlled oscillator
VCXO. voltage-controlled crystal oscillator
VESA. Video Electronic Standards Association
VITC. vertical interval time code
VMI. video module interface
VPS. video program system
VSYNC. vertical sync
WDT. watchdog timer
WSS. wide-screen signalling
WE. write enable
Y/C. luma/chroma
YUV. luminance (Y), red minus Y (U), and blue minus Y (V)

Index

A

- A/V decoder, MPEG 7-11
- assembling the STB Reference Design Kit
 - components 5-1
- audio connector 8-5
- audio DAC 7-12
- auxiliary power connector 8-15

B

- base OS-9000 installation, modifying 12-2
- base pSOS installation, modifying 11-2
- baud rate, setting for boots over SP1 10-10
- book
 - conventions used
 - highlighting xiv
 - numeric xiv
 - syntax diagrams xv
 - organization xiii
 - who should use this book xiii
- boot devices, selecting 10-4
- boot ROM 7-5
- Booting
 - over SP1 10-11
- booting
 - setting the baud rate for boots over SP1 10-10
- buffering
 - address (STBRP) 7-8
 - data (STBRP) 7-8
- buffering hardware, STBRP 7-8

C

- cables, STB Reference Design Kit 1-4
- components
 - STB Reference Design Kit
 - hardware 5-1
 - software 3-1
 - STBRP 7-1
- composite video connector 8-5
- conditional access 7-5
- conditional access support
 - OS-9000 PSP 12-33
 - pSOS PSP 11-34
- configuring a SingleStep debug session 11-3

- configuring pROBE on the STBRP 11-3
- connecting STBRP A/V outputs to a TV 6-1
- connecting the STBRP and DVBE serial ports to a host PC 6-1
- connecting the STBRP and DVBE to power 6-2
- connecting the STBRP, DVBE, and host PC to Ethernet 6-1
- connectors, STBRP
 - audio 8-5
 - auxiliary power 8-15
 - composite video 8-5
 - DVB transport stream 8-10
 - expansion 8-6
 - I2C 8-16
 - IEEE 1284 port 8-3
 - RISCTrace 8-13
 - RISCWatch 8-13
 - SCART 8-4
 - serial port 8-3
 - S-video 8-6
 - test headers 8-16
- contacting the IBM Embedded Systems Solution Center xvi
- conventions used
 - highlighting xiv
 - numeric xiv
 - syntax diagrams xv
- critical interrupt switch, STBRP 8-24

D

- DAC, audio 7-12
- data communications requirements, host PC 2-1
- decoder, MPEG A/V 7-11
- demo program
 - list/select programs menu 13-6
 - main menu 13-2
 - MPEG player menu 13-4
 - OSD demo menu 13-3
 - using the GUI 13-2
 - using the VT 100 user interface 13-6
- DENC 7-12
 - device driver functions
 - OS-9000 PSP 12-17
 - pSOS PSP 11-13
 - device drivers
 - OS-9000 PSP 12-17
 - pSOS PSP 11-12

- diagnostics
 - OS-9000 PSP 12-17
 - pSOS PSP 11-13
- DENC device drivers
 - modules, OS-9000 PSP 12-18
 - source code, OS-9000 PSP 12-17
 - source code, pSOS PSP 11-14
- descrambler, DVB 7-11
- device drivers
 - installing 3-1
 - OS-9000 PSP
 - DENC 12-17
 - EEPROM 12-20
 - Ethernet 12-22
 - flash memory 12-4
 - I2C bus 12-18
 - IEEE 1284 port 12-29
 - IR remote control 12-23
 - keypad 12-25
 - MFM 12-15
 - MPFM 12-12
 - serial port 12-27
 - smart card 12-30
 - transport demultiplexer 12-7
 - pSOS PSP
 - DENC 11-12
 - EEPROM 11-17
 - Ethernet 11-19
 - flash memory 11-4
 - I2C bus 11-14
 - IEEE 1284 port 11-25
 - IR receiver 11-21
 - keypad 11-23
 - MPEG A/V decoder 11-10
 - serial port 11-24
 - smart card 11-27
 - transport demultiplexer 11-7
- diagnostics
 - MPEG A/V decoder, pSOS PSP 11-11
 - MPFM, OS-9000 PSP 12-12
 - OS-9000 PSP
 - DENC 12-17
 - EEPROM 12-20
 - flash memory 12-5
 - I2C bus controller 12-19
 - IEEE 1284 port 12-29
 - IR transceiver 12-23
 - keypad 12-26
 - MFM 12-16
 - SCI 12-30
 - serial port 12-27
 - STBP chip functions 12-32
 - transport demultiplexer 12-8
 - pSOS PSP
 - DENC 11-13
 - EEPROM 11-18
 - Ethernet 11-20
 - flash memory 11-5
 - I2C bus controller 11-16
 - IEEE 1284 port 11-26
 - IR transceiver 11-21
 - keypad 11-23
 - Level 1 11-34
 - Level 1–Level 3 11-34
 - Levels 2 and 3 11-35
 - Levels 2 and 3 UI task 11-36
 - SCI 11-27
 - serial port 11-24
 - transport demultiplexer 11-9
- DRAM
 - system 7-4
 - transport chip 7-10
- DVB
 - I/O 7-10
- DVB descrambler 7-11
- DVB transport chip 7-10
- DVBE
 - connecting to the DVBE buffer card 5-1
 - power supply 6-2
- DVBE buffer card, connecting the DVBE 5-1
- E**
- EEPROM
 - device driver functions
 - OS-9000 PSP 12-20
 - pSOS PSP 11-19
 - device drivers
 - OS-9000 PSP 12-20
 - pSOS PSP 11-17
 - diagnostics
 - OS-9000 PSP 12-20
 - pSOS PSP 11-18
- EEPROM device driver
 - reading data (pSOS PSP) 11-17

- writing data (pSOS PSP) 11-18
- EEPROM device drivers
 - modules, OS-9000 PSP 12-22
 - source code, OS-9000 PSP 12-21
 - source code, pSOS PSP 11-19
- EEPROM, serial 7-5
- equations
 - ispGAL22V10C PLD A-9
 - ispLSI 1024-90LJ PLD A-10
- Ethernet
 - daughterboard bill of materials B-7
 - device driver functions
 - pSOS PSP 11-20
 - device drivers
 - OS-9000 PSP 12-22
 - pSOS PSP 11-19
 - diagnostics
 - pSOS PSP 11-20
- Ethernet daughterboard
 - header J1 signals 7-17
 - headers 7-16
 - indicator lamps 7-14
 - jumper J5 settings 7-19
 - jumper J8 settings 7-19
 - jumpers 7-16
 - keypad 7-15
 - network interfaces, Ethernet 7-14
 - switch matrix 7-15
 - test header J6 signals 7-17
 - test header J7 signals 7-18
- Ethernet device drivers
 - source code, pSOS PSP 11-20
- expansion connector signals 8-6
- expansion connector, STPRP 8-6

F

- FasTrak, using the OS-9000 PSP version 12-2
- features
 - basic STB Reference Design Kit 1-1
- Federal Communications Commission (FCC)
 - Statement xvi
- flash boot ROM, system 7-5
- flash memory
 - device driver functions
 - OS-9000 PSP 12-5
 - pSOS PSP 11-5
 - device drivers

- OS-9000 PSP 12-4
- pSOS PSP 11-4
 - source code, OS-9000 PSP 12-6
 - source code, pSOS PSP 11-7
- diagnostics
 - OS-9000 PSP 12-5
 - pSOS PSP 11-5
 - system boot ROM 7-5
- flash memory device drivers
 - modules, OS-9000 PSP 12-7
- flash_read() 11-5
- functions
 - OS-9000 PSP
 - DENC device driver 12-17
 - EEPROM device driver 12-20
 - flash memory device driver 12-5
 - I2C bus device driver 12-19
 - IEEE 1284 port device driver 12-29
 - IR remote control device driver 12-24
 - keypad device driver 12-26
 - MFM device driver 12-16
 - MPFM device driver 12-12
 - serial port device driver 12-27
 - smart card device driver 12-31
 - STBP chip 11-29, 12-33
 - transport demultiplexer 12-8
 - pSOS PSP
 - DENC device driver 11-13
 - EEPROM device driver 11-19
 - Ethernet device driver 11-20
 - flash memory device driver 11-5
 - I2C bus device driver 11-16
 - IEEE 1284 port device driver 11-26
 - IR receiver device driver 11-22
 - keypad device driver 11-23
 - MPEG A/V decoder device driver 11-11
 - serial port device driver 11-24
 - smart card device driver 11-28
 - STBP chip 11-29
 - transport demultiplexer 11-9

G

- general purpose I/O (GPIO) 7-6
- GPIO pins
 - I/O
 - general-purpose (GPIO) 7-6
- GUI, demo program 13-2

H

- hardware components
 - cables and power supply 1-4
 - user-supplied 1-4
- hardware components, STB Reference Design Kit 1-3
- hardware requirements, host PC 2-1
- headers
 - Ethernet daughterboard 7-16
- host PC configuration
 - TCP/IP, Windows 3.n 4-3
 - TCP/IP, Windows 95 4-3
 - TCP/IP, Windows NT 4-5
 - terminal emulator
 - HyperTerminal 4-1
 - Windows Terminal 4-2
 - Trumpet Winsock 4-4
- host PC requirements
 - data communications and networking 2-1
 - hardware 2-1
 - software 2-1
- HyperTerminal (Windows 95) 4-1

I

- I/O
 - DVB 7-10
- I2C
 - expansion connector 8-16
- I2C bus
 - device driver functions
 - OS-9000 PSP 12-19
 - pSOS PSP 11-16
 - device drivers
 - OS-9000 PSP 12-18
 - pSOS PSP 11-14
- I2C bus controller
 - diagnostics
 - OS-9000 PSP 12-19
 - pSOS PSP 11-16
- I2C bus device driver
 - reading data (pSOS PSP) 11-15
 - writing data (pSOS PSP) 11-15
- I2C bus device driver (pSOS PSP)
 - reading data 11-15
- I2C bus device drivers
 - source code, OS-9000 PSP 12-19
 - source code, pSOS PSP 11-16

- I2C device drivers
 - modules, OS-9000 PSP 12-20
- IBM Embedded Systems Solution Center xvi
- IEEE 1284 port
 - device driver functions
 - OS-9000 PSP 12-29
 - pSOS PSP 11-26
 - device drivers
 - OS-9000 PSP 12-29
 - pSOS PSP 11-25
 - diagnostics
 - OS-9000 PSP 12-29
 - pSOS PSP 11-26
- IEEE 1284 port device drivers
 - modules, OS-9000 PSP 12-30
 - source code, OS-9000 PSP 12-29
 - source code, pSOS PSP 11-27
- IEEE 1284 port, STBP 7-5
- IEEE1284
 - connector 8-3
- indicator lamps, Ethernet daughterboard 7-14
- installation requirements, software 3-1
- installing the device drivers 3-1
- installing the PSP 3-1
- integrating the OS-9000 PSP 12-1
- integrating the pSOS PSP 11-1
- IR receiver
 - device driver functions
 - pSOS PSP 11-22
 - device drivers
 - pSOS PSP 11-21
- IR receiver device driver
 - source code, pSOS PSP 11-22
- IR remote control
 - device driver functions
 - OS-9000 PSP 12-24
 - device drivers
 - OS-9000 PSP 12-23
- IR remote control device driver
 - source code, OS-9000 PSP 12-25
- IR remote control device drivers
 - modules, OS-9000 PSP 12-25
- IR transceiver
 - diagnostics
 - OS-9000 PSP 12-23
 - pSOS PSP 11-21
- ispGAL22V10C PLD equations A-9

ispLSI 1024-90LJ PLD equations A-10

J

jumper

OSC/VXCO 9-1

jumper settings 8-23

jumpers

clocking 9-1

Ethernet daughterboard 7-16

jumpers, setting STBRP 8-23

K

keypad

device driver functions

OS-9000 PSP 12-26

pSOS PSP 11-23

device drivers

OS-9000 PSP 12-25

pSOS PSP 11-23

diagnostics

OS-9000 PSP 12-26

pSOS PSP 11-23

hardware 7-15

keypad device drivers

modules, OS-9000 PSP 12-27

source code, OS-9000 PSP 11-24, 12-27

L

Level 1 diagnostics, pSOS PSP 11-34

Level 1–Level 3 diagnostics, pSOS PSP 11-34

Levels 2 and 3 diagnostics, pSOS PSP 11-35

loading the OS-9000 boot code 12-1

loading the pSOS boot code 11-1

M

memory

serial EEPROM 7-5

STBRP 7-4

system DRAM 7-4

system flash boot ROM 7-5

menus, demo program

list/select programs 13-6

main 13-2

MPEG player 13-4

OSD demo 13-3

MFM

device driver functions, OS-9000 PSP 12-16

device drivers

OS-9000 PSP 12-15

diagnostics

OS-9000 PSP 12-16

MFM device driver

modules, OS-9000 PSP 12-16

MFM device drivers

source code, OS-9000 PSP 12-16

miscellaneous files and directories, OS-9000

PSP 12-34

miscellaneous files and directories, pSOS

PSP 11-36

modifying OS-9000 system files 12-3

rom.bl 12-4

rom_mods.ml 12-4

sysinit.c 12-3

systype.h 12-3

modifying the base OS-9000 installation 12-2

modifying the base pSOS installation 11-2

modules

OS-9000 PSP

DENC device driver 12-18

EEPROM device driver 12-22

flash memory device driver 12-7

I2C device driver 12-20

IEEE 1284 port device driver 12-30

IR remote control device driver 12-25

keypad device driver 12-27

MFM device driver 12-16

MPFM device driver 12-15

serial port device driver 12-28

smart card device driver 12-32

STBP chip functions 12-33

transport demultiplexer device

driver 12-12

pSOS PSP

transport demultiplexer device

driver 11-10

MPEG

A/V decoder 7-11

MPEG A/V decoder

device driver functions

pSOS PSP 11-11

device driver, pSOS PSP 11-10

diagnostics, pSOS PSP 11-11

MPEG A/V decoder device driver

source code, pSOS PSP 11-12

MPFM

- device driver function, OS-9000 PSP 12-12
- device driver, OS-9000 PSP 12-12
- diagnostics, OS-9000 PSP 12-12
- MPFM device driver
 - modules, OS-9000 PSP 12-15
 - source code, OS-9000 PSP 12-14

N

- networking requirements, host PC 2-1

O

- operating systems, supported 2-1
- OS-9000
 - loading boot code 12-1
 - modifying system files 12-3
 - modifying the base installation 12-2
 - problems with the base software 12-4
 - starting the demo program 13-1
- OS-9000 PSP
 - conditional access support 12-33
 - device drivers
 - DENC 12-17
 - EEPROM 12-20
 - Ethernet 12-22
 - flash memory 12-4
 - I2C bus 12-18
 - IEEE 1284 port 12-29
 - IR remote control 12-23
 - keypad 12-25
 - MFM 12-15
 - MPFM 12-12
 - serial port 12-27
 - smart card 12-30
 - transport demultiplexer 12-7
 - diagnostics
 - DENC 12-17
 - EEPROM 12-20
 - flash memory 12-5
 - I2C bus controller 12-19
 - IEEE 1284 port 12-29
 - IR transceiver 12-23
 - keypad 12-26
 - MFM 12-16
 - MPFM 12-12
 - SCI 12-30
 - serial port 12-27
 - STBP chip functions 12-32

- transport demultiplexer 12-8
- functions
 - DENC device driver 12-17
 - EEPROM device driver 12-20
 - flash memory device driver 12-5
 - I2C bus device driver 12-19
 - IEEE 1284 port device driver 12-29
 - IR remote control device driver 12-24
 - keypad device driver 12-26
 - MFM device driver 12-16
 - MPFM device driver 12-12
 - serial port device driver 12-27
 - smart card device driver 12-31
 - STBP chip 11-29, 12-33
 - transport demultiplexer device driver 12-8
- integrating 12-1
- miscellaneous files and directories 12-34
- modules
 - DENC device driver 12-18
 - EEPROM device driver 12-22
 - flash memory device driver 12-7
 - I2C device driver 12-20
 - IEEE 1284 port device driver 12-30
 - IR remote control device driver 12-25
 - keypad device driver 12-27
 - MFM device driver 12-16
 - MPFM device driver 12-15
 - serial port device driver 12-28
 - smart card device driver 12-32
 - STBP chip functions 12-33
 - transport demultiplexer device driver 12-12
- source code
 - demo program 12-34
 - DENC device driver 12-17
 - EEPROM device driver 12-21
 - flash memory device driver 12-6
 - I2C bus device driver 12-19
 - IEEE 1284 port device driver 12-29
 - IR remote control device driver 12-25
 - keypad device driver 11-24, 12-27
 - MFM device driver 12-16
 - MPFM device driver 12-14
 - serial port device driver 12-28
 - smart card device driver 12-32
 - STBP chip functions 12-33
 - transport demultiplexer device

- driver 12-11
- using DAVIDLite 12-4
- using the FasTrak version 12-2

P

- parallel port (IEEE 1284) 7-5
- power supplies, STB Reference Design Kit 1-4
- power supplies, STBRP and DVBE 6-2
- power, connecting the STBRP and DVBE to 6-2
- PPC403 Embedded Controller 7-1
- pSOS
 - loading boot code 11-1
 - modifying the base installation 11-2
- pSOS Infrared Receiver Device Driver
 - Functions 11-22
- pSOS PSP
 - conditional access support 11-34
 - configuring a SingleStep debug session 11-3
 - device drivers
 - DENC 11-12
 - EEPROM 11-17
 - Ethernet 11-19
 - flash memory 11-4
 - I2C bus 11-14
 - IEEE 1284 port 11-25
 - IR receiver 11-21
 - keypad 11-23
 - MPEG A/V decoder 11-10
 - serial port 11-24
 - smart card 11-27
 - transport demultiplexer 11-7
 - diagnostics
 - DENC 11-13
 - EEPROM 11-18
 - Ethernet 11-20
 - flash memory 11-5
 - I2C bus controller 11-16
 - IEEE 1284 port 11-26
 - IR transceiver 11-21
 - keypad 11-23
 - Level 1 11-34
 - Level 1–Level 3 11-34
 - Levels 2 and 3 11-35
 - Levels 2 and 3 UI task 11-36
 - MPEG A/V decoder 11-11
 - SCI 11-27
 - serial port 11-24

- transport demultiplexer 11-9
- EEPROM device driver
 - reading data 11-17
 - writing data 11-18
- functions
 - DENC device driver 11-13
 - EEPROM device driver 11-19
 - Ethernet device driver 11-20
 - flash memory device driver 11-5
 - I2C bus device driver 11-16
 - IEEE 1284 port device driver 11-26
 - IR receiver device driver 11-22
 - keypad device driver 11-23
 - MPEG A/V decoder device driver 11-11
 - serial port device driver 11-24
 - smart card device driver 11-28
 - STBP chip 11-29
 - transport demultiplexer device driver 11-9
- I2C bus device driver
 - reading data 11-15
 - writing data 11-15
- integrating 11-1
- miscellaneous files and directories 11-36
- modules
 - transport demultiplexer device driver 11-10
- pROBE
 - configuring on the STBRP 11-3
- server tasks 11-29
 - control task 11-31
 - database task 11-32
 - debug task 11-30
 - IR application task 11-33
 - root task 11-29, 11-30
 - transport task 11-31
 - user interface task 11-33
- source code
 - demo program 11-34
 - DENC device driver 11-14
 - EEPROM device driver 11-19
 - Ethernet device driver 11-20
 - flash memory device driver 11-7
 - I2C bus device driver 11-16
 - IEEE 1284 port device driver 11-27
 - IR receiver device driver 11-22
 - MPEG A/V decoder device driver 11-12
 - serial port device driver 11-25

- server tasks 11-33
- smart card device driver 11-28
- STBP chip functions 11-29
- transport demultiplexer device driver 11-10
- pSOS™ Platform Support Package 11-1
 - PSP
 - installing 3-1
 - publications, related xvi
- R**
- related publications xvi
- reset switch, STBRP 8-24
- ROM monitor
 - accessing 10-1
 - bootp and tftp configuration for loads 4-6
 - PC 4-6
 - menus 10-1
 - changing IP addresses 10-5
 - disabling the automatic display 10-8
 - displaying the current configuration 10-8
 - exiting the main menu 10-16
 - initial ROM monitor menu 10-1
 - saving the current configuration 10-9
 - selecting boot devices 10-4
 - selecting power-on tests 10-2
 - using the ping test 10-7
- root tasks, pSOS PSP 11-29

S

- SCART connector 8-4
- SCI
 - diagnostics
 - OS-9000 PSP 12-30
 - pSOS PSP 11-27
- SCI (smart card interface) 7-5
- serial EEPROM, STBRP 7-5
- serial port
 - device driver functions
 - OS-9000 PSP 12-27
 - pSOS PSP 11-24
 - device drivers
 - OS-9000 PSP 12-27
 - pSOS PSP 11-24
 - diagnostics
 - OS-9000 PSP 12-27
 - pSOS PSP 11-24

- serial port connector (STBRP) 8-3
- serial port device drivers
 - modules, OS-9000 PSP 12-28
 - source code, OS-9000 PSP 12-28
 - source code, pSOS PSP 11-25
- serial port, STBP 7-5
- serial ports, STBRP and DVBE, connecting to a host PC 6-1
- server tasks, pSOS PSP 11-29
 - A/V task, pSOS PSP 11-30
 - control task, pSOS PSP 11-31
 - database task, pSOS PSP 11-32
 - debug task, pSOS PSP 11-30
 - IR application task, pSOS PSP 11-33
 - transport task, pSOS PSP 11-31
 - user interface task, pSOS PSP 11-33
- setting the STBRP jumpers 8-23
- signals
 - CD21 control test header
 - test headers, CD21 control test header signals 8-19
 - CD21 DRAM data bus test header 8-18
 - CD21 DRAM data bus test header (J15) 8-19
 - CD21 DRAM data bus test header (J19) 8-20
 - CD21 DRAM test header 8-16
 - DVB transport stream connector 8-10
 - expansion connector 8-6
 - IEEE 1284 port 8-3
 - JTAG interface 8-13
 - PPC403 control signals test header 8-20
 - PPC403 controls/CD21 DRAM address bus test header 8-21
 - ready and interrupt signals test header 8-22
 - RISCTrace 8-15
 - SCART 8-5
 - serial port 8-3
 - system address bus and others test header test headers
 - system address bus and others signals 8-22
 - system bus test header 8-17
 - system data bus (J31) test header 8-23
 - system data bus test header (J13)
 - test headers, system data bus (J13) signals 8-18
 - transport DRAM data bus test header test headers

- transport DRAM data bus 8-19
- transport DRAM test header 8-21
- YUV bus test header 8-17
- SingleStep, configuring a debug session 11-3
- smart card
 - device driver functions
 - OS-9000 PSP 12-31
 - pSOS PSP 11-28
 - device drivers
 - OS-9000 PSP 12-30
 - pSOS PSP 11-27
- smart card device drivers
 - modules, OS-9000 PSP 12-32
 - source code, OS-9000 PSP 12-32
 - source code, pSOS PSP 11-28
- smart card interface (SCI) 7-5
- software
 - installation requirements 3-1
 - installing device drivers 3-1
 - installing the PSP 3-1
- software components, STB Reference Design Kit 1-3
- software requirements, host PC 2-1
- source code
 - OS-9000 PSP
 - demo program 12-34
 - DENC device driver 12-17
 - EEPROM device driver 12-21
 - flash memory device drivers 12-6
 - I2C bus device driver 12-19
 - IEEE 1284 port device driver 12-29
 - IR remote control device driver 12-25
 - keypad device driver 11-24, 12-27
 - MFM device driver 12-16
 - MPFM 12-14
 - serial port device driver 12-28
 - smart card device driver 12-32
 - STBP chip functions 12-33
 - transport demultiplexer device driver 12-11
 - pSOS PSP
 - demo program 11-34
 - DENC device driver 11-14
 - EEPROM device driver 11-19
 - Ethernet device driver 11-20
 - flash memory device drivers 11-7
 - I2C bus device driver 11-16
 - IEEE 1284 port device driver 11-27
 - IR receiver device driver 11-22
 - MPEG A/V decoder 11-12
 - serial port device driver 11-25
 - server tasks 11-33
 - smart card device driver 11-28
 - STBP chip functions 11-29
 - transport demultiplexer device driver 11-10
- standards publications xvi
- STB peripheral (STBP) chip 7-5
- STB Reference Design Kit
 - assembling 5-1
 - basic features and functions 1-1
 - components 5-1
 - hardware components 1-3
 - software components 1-3, 3-1
 - supplied cables and power supplies 1-4
- STBP
 - GPIO 7-6
 - IEEE 1284 port 7-5
 - SCI 7-5
 - serial port 7-5
- STBP chip
 - functions
 - OS-9000 PSP 11-29, 12-33
 - pSOS PSP 11-29
- STBP chip functions
 - diagnostics
 - OS-9000 PSP 12-32
 - modules, OS-9000 PSP 12-33
 - source code, OS-9000 PSP 12-33
 - source code, pSOS PSP 11-29
- STBRP
 - advanced features 1-1
 - bill of materials B-1
 - block diagram 1-2
 - clock distribution 9-1
 - components 7-1
 - connecting A/V outputs to 6-1
 - critical interrupt switch 8-24
 - hardware
 - buffering 7-8
 - DVB transport chip 7-10
 - IEEE 1284 port, STBP 7-5
 - IR transceiver 7-9
 - serial port, STBP 7-5

- STBP 7-5
 - system control logic 7-5
- indicator lamps 8-24
- power supply 6-2
- reset switch 8-24
- switches 8-24
- STBRP components
 - hardware
 - memory 7-4
 - serial EEPROM 7-5
 - system DRAM 7-4
 - system flash boot ROM 7-5
- STBRP connectors
 - audio 8-5
 - auxiliary power 8-15
 - composite video 8-5
 - DVB transport stream 8-10
 - expansion 8-6
 - I2C expansion 8-16
 - IEEE 1284 port 8-3
 - RISCTrace 8-13
 - RISCWatch 8-13
 - SCART 8-4
 - serial port 8-3
 - S-video 8-6
 - test headers 8-16
- STBRP jumpers, setting 8-23
- STBRP serial port
 - connector 8-3
- supported operating systems 2-1
- S-video connector 8-6
- switch matrix 7-15
- syntax diagrams xv
- system control logic, STBRP 7-5
- system DRAM, STBRP 7-4
- system flash boot ROM, STBRP 7-5

T

- TCP/IP
 - configuring for Windows 3.n 4-3
 - configuring for Windows 95 4-3
 - configuring for Windows NT 4-5
- terminal emulator
 - HyperTerminal 4-1
 - Windows Terminal 4-2
- test header
 - J17 transport DRAM data bus 8-19

- test header connectors 8-16
- test headers
 - CD21 DRAM data bus 8-18
 - CD21 DRAM data bus signals 8-19
 - CD21 DRAM signals 8-16
 - DRAM data bus signals 8-20
 - PPC403 control signals 8-20
 - PPC403 controls/CD21 DRAM address bus signals 8-21
 - ready and interrupt signals 8-22
 - system bus test signals 8-17
 - system data bus (J31) signals 8-23
 - transport DRAM 8-21
 - YUV bus signals 8-17

- transport chip
 - audio DAC 7-12
 - DENC 7-12
 - DRAM 7-10
 - DVB descrambler 7-11
 - DVB I/O 7-10
 - MPEG A/V decoder 7-11
- transport chip, DVB 7-10
- transport demultiplexer
 - device driver functions
 - OS-9000 PSP 12-8
 - pSOS PSP 11-9
 - device drivers
 - OS-9000 PSP 12-7
 - pSOS PSP 11-7
 - source code, OS-9000 PSP 12-11
 - source code, pSOS PSP 11-10
 - diagnostics, OS-9000 PSP 12-8
 - diagnostics, pSOS PSP 11-9
- transport demultiplexer device drivers
 - module, pSOS PSP 11-10
 - modules, OS-9000 PSP 12-12
- transport stream cable, connecting 5-2
- transport stream connector 8-10
- Trumpet Winsock
 - configuring on the host PC 4-4
 - downloading 4-3
 - installing 4-4
- TV, connecting STBRP A/V outputs to 6-1

U

- using a terminal emulator
 - HyperTerminal for Windows 95 4-1

Windows Terminal 4-2

V

VT 100 user interface, demo program 13-6

W

Windows 3.n

TCP/IP configuration 4-3

Windows 95

TCP/IP configuration 4-3

Windows NT

TCP/IP configuration 4-5

Windows Terminal (Windows 3.n and Windows
NT) 4-2

