ADSP-2106x SHARC

EZ-KIT Lite Reference Manual

Hardware Rev. 1

ANALOG DEVICES

ADSP-2106x SHARC EZ-KIT Lite™ Reference Manual

Analog Devices, Inc.
Computer Products Division
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
(617) 329-4700

If you have comments or suggestions about this manual of find any errors in it, please contact us via email at:

`dsp_techpubs@analog.com`

For product marketing information or technical support, contact any Analog Devices sales office or authorized distributor. For applications engineering assistance, contact:

DSP Applications Engineering
Phone: (617) 461-3672
Fax: (617) 461-3010
email: `dsp_applications@analog.com`

PRINTED IN THE U.S.A **82-001802-01** **May 1997 EDITION**

Contents

Contents

Contents

FIGURES

Contents

TABLES

Introduction 1

1.1 OVERVIEW

Your SHARC EZ-KIT Lite™ is one of the best values in development systems available today. Your small investment gives you everything you need to access to the most powerful floating point digital signal processor in the world!

The Analog Devices ADSP-21061 processor used in the SHARC EZ-KIT Lite has many features integrated onto a single chip:

- The industry's fastest general-purpose 32-bit single-precision (or 40-bit extended precision) IEEE floating-point and 32-bit fixed-point DSP core with three independent, parallel computational units: ALU, multiplier, and shifter (40-MIPS, with 120 MFLOPS peak, 80 MFLOPS sustained)
- On-chip, configurable memory banks: dual-ported 1-megabit internal SRAM for fast, independent local memory access for DSP core, DMA controller and I/O processor
- Two 40 Mbit/s synchronous serial ports
- A sophisticated DMA controller (6 simultaneous channels with zero impact on performance of DSP core)

The SHARC EZ-KIT Lite provides an easy way for you to investigate the power of the SHARC family of processors and develop your own applications based on these high-performance DSPs. It is a complete development system package with a price that makes it ideal for getting started in DSP. The SHARC EZ-KIT Lite was designed to help you accomplish these goals:

- Evaluate Analog Devices' floating-point DSPs
- Learn about DSP applications
- Develop DSP applications
- Simulate and debug your application
- Prototype new applications

1 Introduction

The SHARC EZ-KIT Lite consist of a small ADSP-21061 based development / demonstration board with full 16-bit stereo audio I/O capabilities. The board's features include:

- Analog Devices ADSP-21061 DSP running at 40 MHz
- Analog Devices AD1847 16-bit Stereo SoundPort® Codec
- RS-232 interface
- Socketed EPROM
- User push-buttons
- User programmable LEDs
- Power supply regulation
- Expansion connectors

The board can run standalone or can simply connect to the RS-232 port of your PC. A monitor program running on the DSP in conjunction with a host program running on the PC lets you interactively download programs as well as interrogate the ADSP-21061. The board comes with a socketed EPROM so that you can run the monitor program and demonstrations provided or you can plug in an EPROM containing you own code.

The SHARC EZ-KIT Lite also comes with all the software you need to develop sophisticated, high-performance DSP applications. A C compiler, assembler, run-time libraries and librarian, linker, PROM splitter utility, and simulator are all included.

You can also connect an optional EZ-ICE® In-Circuit Emulator to the SHARC EZ-KIT Lite. The emulator allows you to load programs, start and stop program execution, observe and alter registers and memory, and perform other debugging operations. The EZ-ICE emulator is available from Analog Devices and other third party resellers.

The ADSP-21061 SHARC EZ-KIT Lite was designed and manufactured by BittWare Research Systems, Inc., of Concord, NH to Analog Devices' specifications.

1.2 EZ-KIT LITE SYSTEM ARCHITECTURE

Figure 1-1 is a block diagram of the SHARC EZ-KIT Lite system. You can access the ADSP-21061 SHARC processor from the PC through the RS-232 interface. The boot PROM provides code for execution when the SHARC EZ-KIT Lite is operating in standalone mode and loads a kernel that manages the RS-232 interface.

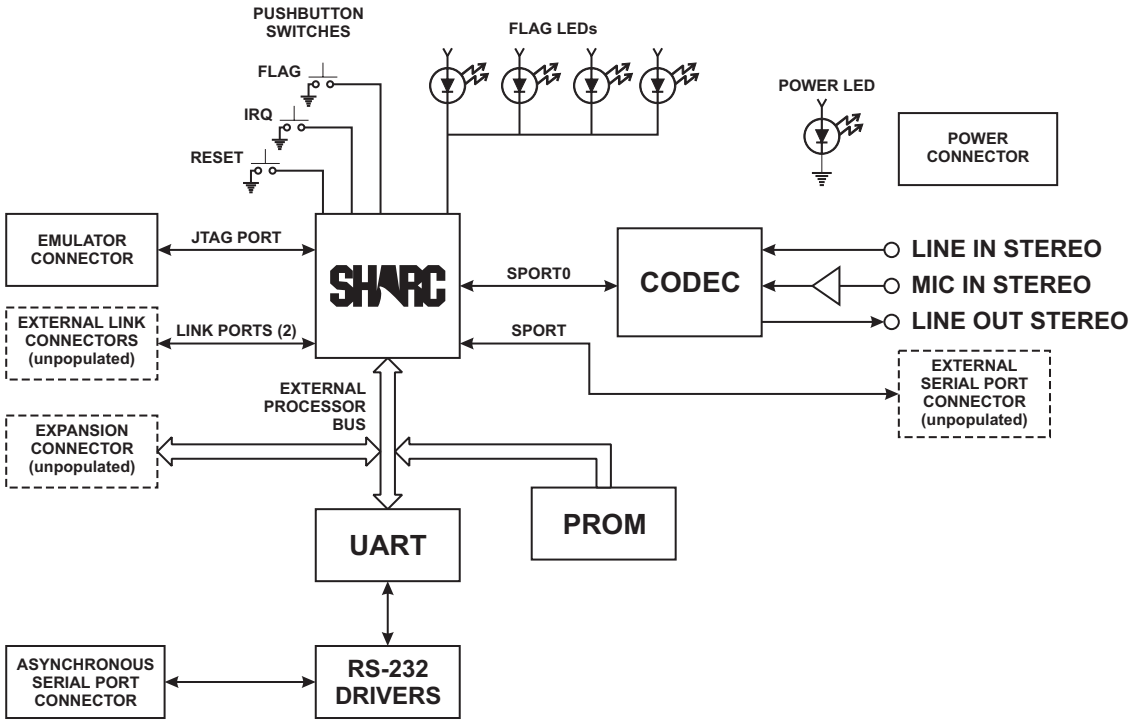


Figure 1-1 SHARC EZ-KIT Lite System Block Diagram

In-circuit emulation is possible with an EZ-ICE probe connected to the JTAG port. The PC controls the probe via an ISA card attached to the probe.

The AD1847 SoundPort Stereo Codec is accessed through a serial port that connects directly to the SHARC processor.

The SHARC EZ-KIT Lite board has several sites for connectors that allow you to expand the board's capabilities. These sites are not populated with connectors when you receive the board from Analog Devices.

1 Introduction

1.3 CONTENTS OF THIS MANUAL

This manual provides all the information you need to:

- Install the SHARC EZ-KIT Lite software onto an IBM compatible Personal Computer
- Connect your SHARC EZ-KIT Lite to your PC and the power supply
- Connect an input source (such as a microphone or CD player) and an output device (such as an amplified / powered speaker) to your SHARC EZ-KIT Lite
- Start the SHARC EZ-KIT Lite
- Use the provided demonstration programs
- Write your own ADSP-21061 programs to run on the SHARC EZ-KIT Lite board

Chapter 2 tells you how to get started. We recommend that you read this manual completely and thoroughly, especially if you are new to programming a DSP. If you are anxious to start using your new development system, this chapter provides you with the basics to get your SHARC EZ-KIT Lite up and running quickly. It also provides a more detailed set of instructions for software installation and hardware connections.

Chapter 3 provides a detailed description of the software that is included with your development system, including:

- SHARC EZ-KIT Lite Host program
- Demonstration programs
- Diag21k command-line utility

Chapter 4 shows you how to write your own ADSP-21061 programs that you can run on the SHARC EZ-KIT Lite board.

Chapter 5 contains a detailed description of the SHARC EZ-KIT Lite hardware.

Chapter 6 is a complete programming reference for the SHARC EZ-KIT Lite. It gives you all of details you will need when write programs that run on the

ADSP-21061. It also provides you with information on how to write programs for your PC that can interface with the SHARC EZ-KIT Lite board.

Chapter 7 is a reference to the DspHost library. It will teach you how to include standard I/O functions in your DSP programs.

Chapter 8 contains the electrical schematic diagrams for the SHARC EZ-KIT Lite board.

1.4 DOCUMENTATION AND RELATED PRODUCTS

This manual is a complete user's guide and reference to the ADSP-2106x SHARC EZ-KIT Lite Development System. We assume that you are already familiar with the SHARC architecture, operation, and programming as described in the *ADSP-2106x SHARC User's Manual*. To understand the SHARC EZ-KIT Lite system at a detailed level, you can refer to the following documents:

- *ADSP-21061 SHARC Data Sheet*
- *AD1847 Serial Port 16-Bit SoundPort Stereo Codec Data Sheet*
- *PC16550D Universal Asynchronous Receiver/Transmitter with FIFOs Data Sheet (National Semiconductor)*

The ADSP-21000 family of processors is supported by a complete set of development tools. Your SHARC EZ-KIT Lite development system comes with a restricted-capabilities version of these tools, and includes a C compiler, assembler, run-time libraries and librarian, linker, simulator, and PROM splitter. Features of these programs are summarized in the *ADSP-21000 Family Hardware and Software Development Tools Data Sheet*.

If you plan to use the SHARC EZ-KIT Lite in conjunction with the EZ-ICE emulator, refer to the documentation that accompanies that products as well as the information in Chapter 3 of this manual.

1 Introduction

Setting Up 2

2.1 OVERVIEW

This chapter describes how to set up your SHARC EZ-KIT Lite™ board so that you can get started quickly. It covers:

- Installation of the EZ-KIT Lite software
- External connections to the EZ-KIT Lite board

2.2 QUICK START

This section contains a very brief description of the EZ-KIT Lite software installation procedure. It is written for experienced PC users familiar with installing software on their PC.

The installation utility for the SHARC EZ-KIT Lite must be run from Windows™ 3.1 or later.

1. Make sure Windows is running.
2. Insert the SHARC EZ-KIT Lite CD into your CD-ROM drive.
3. From the Windows Program Manager, choose the Run command from the File menu (or select Run from the Start button in Windows 95). In the Command Line box, type `d:install` if your CD-ROM reader is drive "d:".
4. Click on the OK button or press enter.
5. Follow the instructions on the screen.

2 Setting Up

2.3 HARDWARE CONNECTIONS

Figure 2-1 highlights the external hardware connections to the SHARC EZ-KIT Lite. The following sections describe each of the connections.

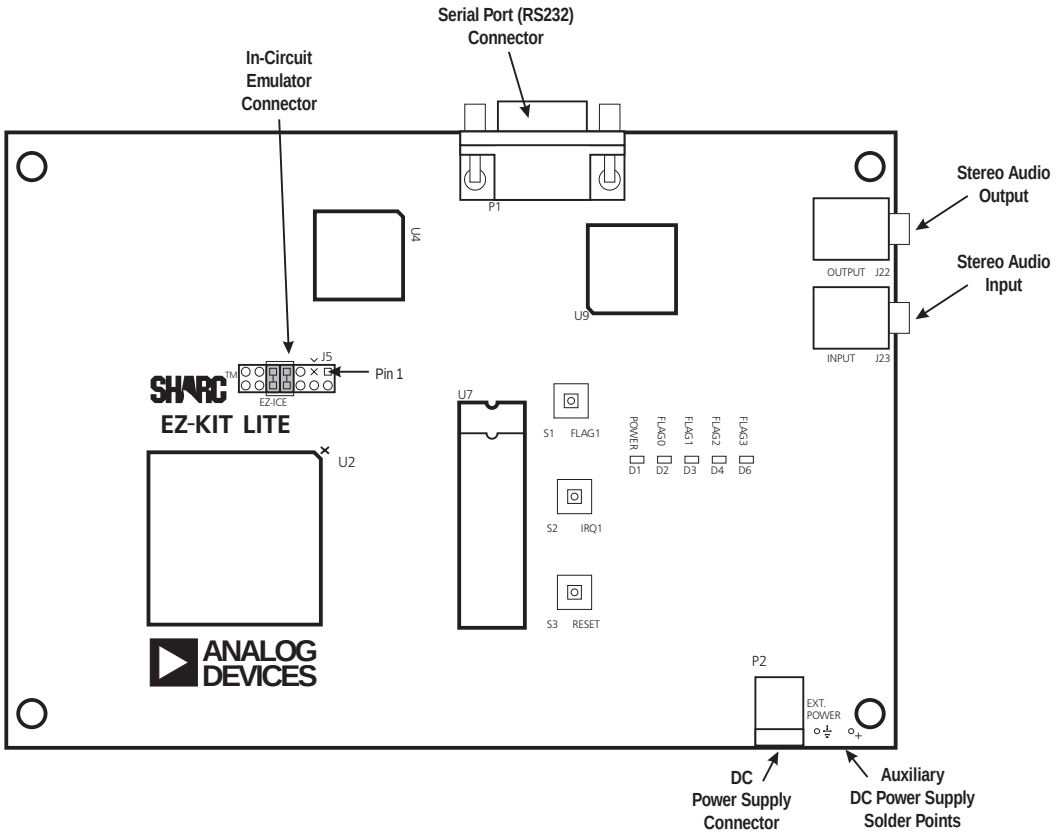


Figure 2-1 Hardware Connections to the SHARC EZ-KIT Lite

2.3.1 In-Circuit Emulator Connector

The In-Circuit Emulator connector is the mounting location for an EZ-ICE® in-circuit emulator probe. Please note that one of the pins is missing (pin 3) to provide keying. The socket in the mating connector (on the EZ-ICE probe) should have a plug inserted at that location.

The SHARC EZ-KIT Lite is shipped with two jumpers installed across pins 7&8 and 9&10. These jumpers must be removed before installing the EZ-ICE probe. When the probe is removed, be sure to replace these jumpers to ensure that the SHARC processor initializes correctly on power-up.

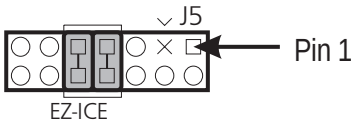


Figure 2-2 In-Circuit Emulator Connector (with jumpers installed)

2.3.2 Serial Port (RS-232) Connector

P1 is female 9-pin D-Sub connector. It is used to communicate with a host computer using RS-232 signal levels and asynchronous serial protocols. You can connect this to your PC using the supplied cable. The cable provides a straight-through connection from the DCE port on the EZ-KIT Lite to the DTE port on your PC. The DCD, DTR, and DSR signals are connected on the EZ-KIT Lite circuit board.

Table 2-1 Serial Port Pin Descriptions

Pin No.	Signal Name
1	DCD
2	Transmit Data (output)
3	Receive Data (input)
4	DTR
5	Signal Ground
6	DSR
7	Request to Send (input)
8	Clear to Send (output)
9	Not Connected

2.3.3 Stereo Audio Output

The Stereo Audio Output jack connects to the left (L) and right (R) LINE OUTPUT pins of the AD1847 codec. You can use standard audio cables with 1/8 inch (3.5mm) stereo plugs to connect these signals to a set of amplified speakers.

2.3.4 Stereo Audio Input

The Stereo Audio Input jack connects directly to the left (L) and right (R) LINE 1 INPUT pins of the AD1847 codec. You can use standard audio cables with 1/8 inch (3.5mm) stereo plugs to supply these inputs with line-level signals. You can also connect a microphone level signal by changing the Input Source Selector jumpers (see 5.3.1).

2.3.5 DC Power Supply Connector

The power supply connector is used to supply DC voltages to the SHARC EZ-KIT Lite board. The DC power supply included with your board should mate directly to this connector.

2.3.6 Auxiliary DC Power Supply Solder Points

If your SHARC EZ-KIT Lite did not come with a power supply, you can solder power cables to these connection points. You should provide 9 to 12 V DC up to 1 A.

2.4 DETAILED SOFTWARE INSTALLATION PROCEDURE

The SHARC EZ-KIT Lite comes with a special release of DSP21k Toolkit from BittWare Research Systems. The tools include:

- EZSHARC Host Program – a Windows-based program that you can use to download and execute demo programs and your own applications
- DSPHost – a DSP library and server that allows you to use standard I/O extensions (such as `printf()` and `write()`) in your DSP programs
- Diag21k – a command-line diagnostic utility
- demo programs with source code

2.4.1 Installing DSP21k

The DSP21k tools require approximately 2 MB of hard disk space. Some of the tools run from DOS and can be executed in a DOS box under Microsoft® Windows™ 3.1 or Windows 95, others require Windows 3.1 or later.

The installation utility for the SHARC EZ-KIT Lite must be run from Windows™ 3.1 or later.

1. Make sure Windows is running.
2. Insert the SHARC EZ-KIT Lite CD into your CD-ROM drive.
3. From the Windows Program Manager, choose the Run command from the File menu (or select Run from the Start button in Windows 95). In the Command Line box, type `d:install` if your CD-ROM reader is drive “d:”.
4. Click on the OK button or press enter.

5. Follow the instructions on the screen.

2.4.2 Setting the Environment Variables

DSPHost and Diag21k require an environment variable to provide necessary hardware-specific information. The environment variable is `ADSPx`, where `x` is the logical board number that is being described. Each processor requires its own environment variable, even if multiple processors are located on the same board. `ADSP0` describes processor (or board) number zero, `ADSP1` describes processors number one, and so on. Each `ADSPx` variable specifies a "board type." The board type parameter specifies the type of DSP board that is installed and how to access the processor. The board type for SHARC EZ-KIT Lite (Rev. 1) is 23. Each board type has a specific format for the variable; the format for the SHARC EZ-KIT Lite Rev. 1 is as follows (parameters in square brackets "[]" are optional):

```
ADSPx=0,23,<dsptype>,<com_port>[,<port_speed>[,<id>[,<msize>[,<wait>
   [,<ms0depth>,<ms0width>]]]]]
```

Where the parameters are defined as follows:

Table 2-2 ADSP Parameter Descriptions for Board Type 23

Argument	Description	Value / Range	Default
<dsptype>	SHARC processor type	3 for ADSP-21060, 4 for ADSP-21062, 5 for ADSP-21061	REQUIRED
<com_port>	PC serial port number	1 – 4	REQUIRED
<port_speed>	PC serial port bit rate	9600, 19200, 38400, 57600, 115200	9600
<id>	Multiprocessor ID number	0 or 1	1
<msize>	Size of external memory banks	$\log_2$(bank size) – 13	0
<wait>	WAIT register contents	See SHARC User's Manual	0x21AD6B5A
<ms0depth>	Depth of External Memory Bank 0 (K words)	0 – 10^{22}	0
<ms0width>	Width of External Memory Bank 0 (bits)	32 or 48	32

A proper ADSP variable for a SHARC EZ-KIT Lite connected to COM1 is:

```
set ADSP0=0,23,5,1,9600
```

2 Setting Up

Operation 3

3.1 OVERVIEW

This chapter describes how you can use the various tools included with the SHARC EZ-KIT Lite to learn more about the SHARC processor. You will also learn how to create a SHARC application that you can download and run. The last section in this chapter presents an overview of using the SHARC EZ-KIT Lite with the EZ-ICE emulator.

3.2 USING THE EZ-KIT LITE HOST PROGRAM

The SHARC EZ-KIT Lite Host Program is a Windows-based application following standard Windows Graphical User Interface conventions. This is the main program that you will use to communicate with the SHARC EZ-KIT Lite board. With it you can:

- run the demonstration programs
- upload, download, or view program and data memory contents
- download and execute user DSP programs

You can start the program by double-clicking on the program's icon that was installed during setup. Figure 3-1 shows the Host Program's main screen.

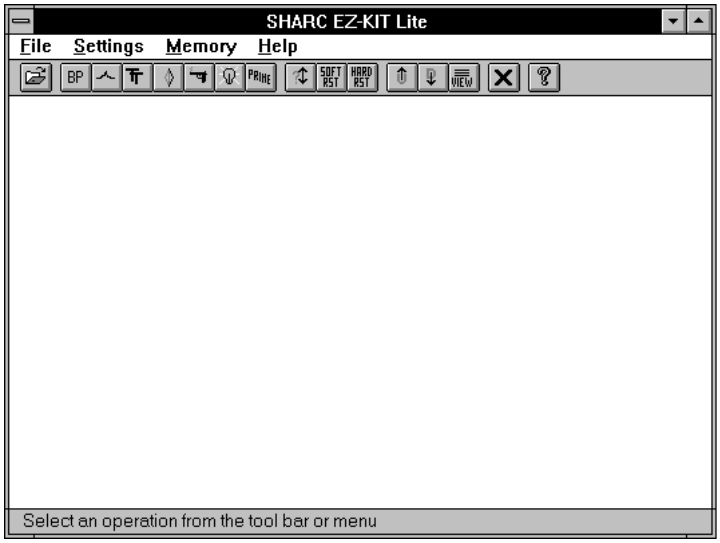


Figure 3-1 Host Program Main Window

3 Operation

Near the top of the window, you will find the menu bar. This contains the four main menu headings: File, Settings, Memory, and Help. Each of these menu headings are described later in this chapter. You can access menu items below each menu heading by clicking on a heading with your mouse or by holding the <alt> key while typing the letter that is underlined in each heading name.

Just below the menu bar, you will see a toolbar that contains shortcut buttons for most of the program's functions (if a toolbar is not visible below the menu bar, see *View Toolbar* in section 3.2.1.2). The following illustration describes each button's function. For a detailed discussion of each function, read the descriptions under the appropriate menu heading below.


File Open

Bandpass Filter Demo


Talkthru Demo


Peter Gunn Demo

Primes Demo


FFT Demo


Pluck String Demo


Blink Demo


Test Communications

Soft Reset SHARC

Hard Reset SHARC


Upload from SHARC memory


Download to SHARC memory

View SHARC memory


Exit Program


About Program

3.2.1 Command Descriptions

This section describes the each of the menu selections available on the host program's main window.

3.2.1.1 File Menu

When you select the File menu, shown in Figure 3-2, you will be presented with options that allow you to download and execute programs on the EZ-KIT Lite board.

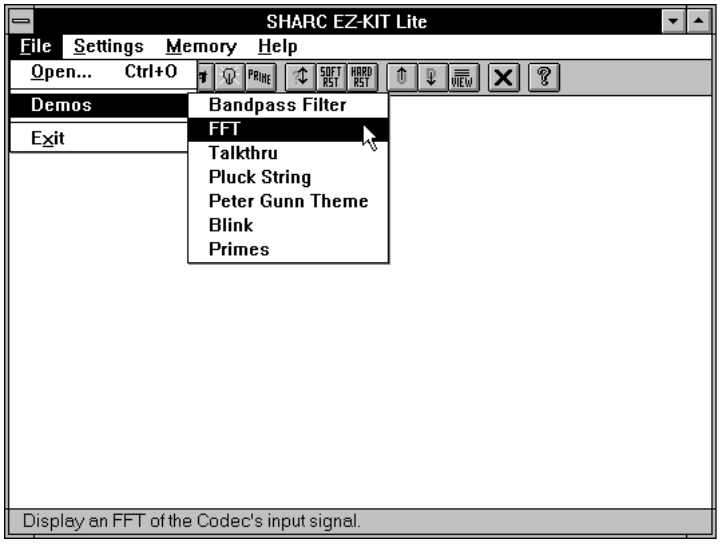


Figure 3-2 File Menu

⇒ Open... 

If you choose Open… (or press Ctrl-O, or click on the toolbar button shown), you will see a dialog box that allows you to choose a DSP program. The host program can open and download COFF files generated by the Analog Devices tools. By default, the linker produces executable COFF files with an `exe` file name extension. You should use the linker's -o option to rename the output file with a `21k` file name extension to avoid confusion with executable files that run on your PC. The file open dialog defaults to a `*.21k` file selection mask.

⇒ Demo

You can also select and start any of the demonstration programs from the fly-out menu that appears when you select Demos. You will learn more about each demonstration program in section 3.2.2.

3 Operation

After you select a user program or a demonstration program, you will see the dialog shown in Figure 3-3. While the program is being downloaded to the EZ-KIT Lite board, you can read the name of the file that is being downloaded in the status line located at the bottom of the main window. You can select the Abort Download button to stop the download.

Figure 3-3 Program Download Status Dialog

Once the file has been successfully downloaded to the EZ-KIT Lite board, the monitor program will automatically start it.

⇒ Exit 

Choosing Exit from the File menu (or clicking on the toolbar button shown) will terminate the host program.

3.2.1.2 Settings Menu

When you select the Settings menu, shown in Figure 3-4, you will see menu items that affect operation of the host program and the EZ-KIT Lite board.

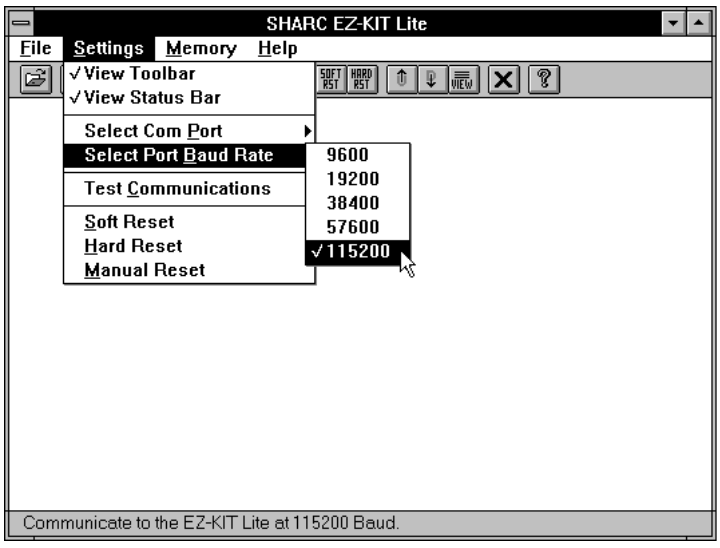


Figure 3-4 Settings Menu

⇒ View Toolbar

Selecting this menu item will remove the toolbar from the main window if it is currently visible. The toolbar will be restored if you select this item when it is not visible.

⇒ View Status Bar

The status bar is the area at the bottom of the main window that shows status of operations that you have initiated and descriptions of menu items that you have highlighted. Selecting this menu item will remove the status bar from the main window if it is currently visible. The status bar will be restored if you select this item when it is not visible.

⇒ Select Com Port

This menu item allows you to define the serial port (also known as a *com* port) to which you have connected your SHARC EZ-KIT Lite communications cable. When you choose Select Com Port, you will see a fly-out menu that gives you four choices: COM1, COM2, COM3, or COM4. The com port that is currently selected will be marked with a check. Once you select a new com port setting, the host program will try to reestablish communications with the EZ-KIT Lite board. When you exit the host program, this setting is saved in a file named `ezsharc.ini` that is stored in the directory from which you started the host program.

⇒ Select Port Baud Rate

This menu item allows you to define the speed (also known as the baud rate) at which your selected serial port will operate. When you choose Select Port Baud Rate, you will see a fly-out menu that gives you five speed choices: 9600, 19200, 38400, 57600, and 115200. The baud rate that is currently selected will be marked with a check. Once you select a new speed setting, the host program will tell the EZ-KIT Lite monitor program to use the new speed and then test the communications link. When you exit the host program, this setting is saved in a file named `ezsharc.ini` that is stored in the directory from which you started the host program.

> **NOTE:** When you reset or power-up the EZ-KIT Lite board, the monitor program communicates at 9600 baud. If the host program is already running at a different speed when this happens, it will not be able to communicate with the board. If you reset or power-up the EZ-KIT Lite board while the host program is running at a speed other than 9600 baud, you must restart the host program or use the *Manual Reset* function described below.

3 Operation

⇒ Test Communications 

This command allows you to verify that the host program can successfully communicate with the SHARC EZ-KIT Lite board. If the test is successful, you will see the message box shown here:

If the host program is not able to communicate with the board, you will see the following message box:

If this happens, you should do the following:

- Check to make sure that the serial cable is connected securely at both ends
- Verify that power is applied to the EZ-KIT Lite board
- Check the com port and baud rate settings as described above

After you have checked those items, choose Manually Reset The Board from the settings menu (described below). If this does not work, try restarting the host program.

⇒ Soft Reset 

When you choose this command, the demo or user program that is currently running will terminate. The monitor program will regain control of the SHARC, but will not disturb user program or data memory.

⇒ Hard Reset 

This command will reset the SHARC processor, which will initiate a boot from the EPROM. The monitor program will perform its power-on self tests and then start the default demo program (Peter Gunn Theme).

⇒ Manual Reset
You should use this command when you are experiencing difficulties with communications or if the board is not behaving normally. When you select this command from the Settings menu, you will see the following message box:

Press the reset switch on the SHARC EZ-KIT Lite board and then click on the OK button. The host program will try to reestablish communications with board and then it will report the results.

3.2.1.3 Memory Menu

When you select the Settings menu, shown in Figure 3-5, you will see menu items that allow you to read from and write to the SHARC processor's memory. You can use these commands while the board is idle and also when a demo or user program is running.

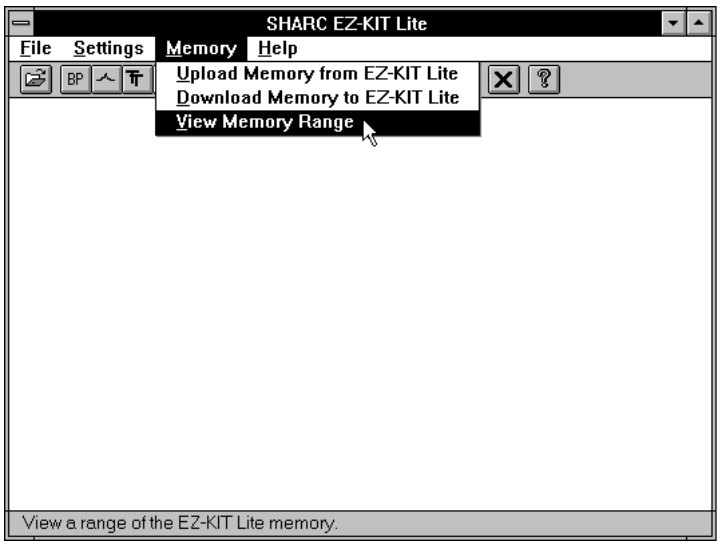


Figure 3-5 Memory Menu

3 Operation

⇒ Upload Memory from EZ-KIT Lite 
This command lets you read from the SHARC's internal memory and then save the results in a file on your PC. When you choose this menu item (or click on the toolbar button), you will see the dialog box shown in Figure 3-6.

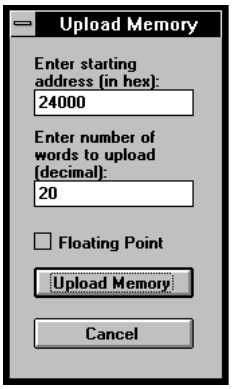


Figure 3-6 Upload Memory Dialog

Enter a valid SHARC memory address (using hexadecimal notation) into the first entry box and the number of words that you want to read (using decimal numbers) into the second entry box. If you want the data to be interpreted as floating point values, check on the Floating Point checkbox. When you click on the Upload Memory button, you will see a file dialog that you can use to specify the file name and location to save your data. Once you specify a file and select OK, the data will be read from the SHARC and written to the file in ASCII format.

⇒ Download Memory to EZ-KIT Lite 
This command lets you write to the SHARC's internal memory using data retrieved from a file on your PC. When you choose this menu item (or click on the toolbar button), you will see the dialog box shown in Figure 3-7.

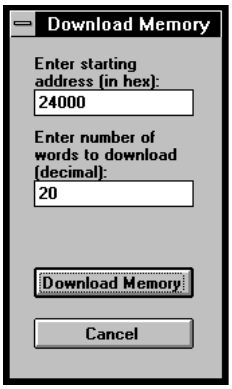


Figure 3-7 Download Memory Dialog

Enter a valid SHARC memory address (using hexadecimal notation) into the first entry box and the number of words that you want to write (using decimal numbers) into the second entry box. When you click on the Download Memory button, you will see a file dialog that you can use to specify the file name and location of your data source. Once you specify a file and select OK, the data will be read from the file and written to the SHARC.

⇒ View Memory Range 

This command lets you examine the SHARC's internal memory. When you choose this menu item (or click on the toolbar button), you will see the dialog box shown in Figure 3-8.

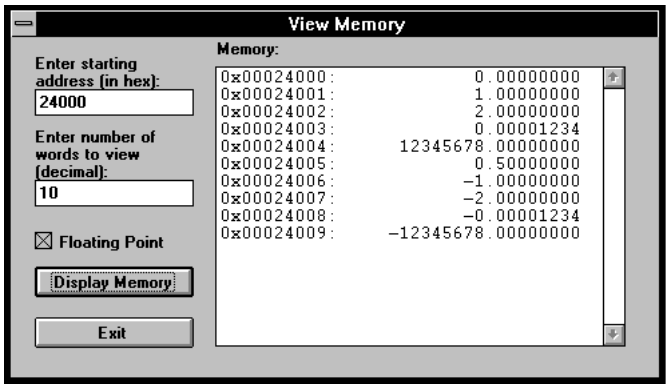


Figure 3-8 View Memory Dialog

Enter a valid SHARC memory address (using hexadecimal notation) into the first entry box and the number of words that you want to display (using decimal numbers) into the second entry box. If you want the data to be displayed as floating point values, check on the Floating Point checkbox. When you click on the Display Memory button, the data will be read from the SHARC and displayed on the Memory section of the dialog box.

3 Operation

3.2.1.4 Help Menu

This menu only has one menu item: About EZSHARC.

⇒ About EZSHARC 

When you select this menu item (or click on the toolbar button), the host program will display the About EZSHARC message box, shown below in Figure 3-9. The information that you see includes the firmware version and processor information that is retrieved from the EZ-KIT Lite board.

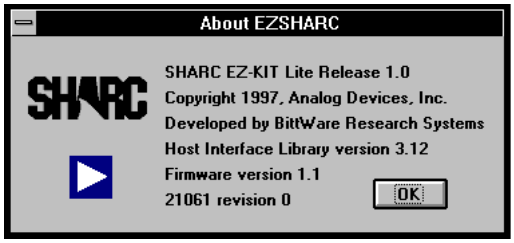


Figure 3-9 The About Dialog

3.2.2 Running the Demos

As described in the previous sections, you can start the included EZ-KIT Lite demonstration programs from either the File menu or the toolbar buttons. Each of the following sections describe what the demonstration programs do and how you should run them.

3.2.2.1 Bandpass Filter Demo 

This program demonstrates the effect of four bandpass filters against no filter on a codec input source or an internally generated noise source. After the host program downloads the program image, it will display the following dialog:

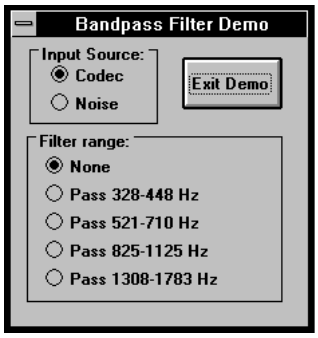


Figure 3-10 Bandpass Filter Demo Dialog

This demonstration starts with a talk-through program, with the Input Source set to Codec, and the Filter range set to None. The AD1847 codec digitizes the analog input signal and transmits the data to the SHARC's serial port. The SHARC reads data from the serial port and retransmits the data back to the codec. The codec converts the data to an analog signal that drives the output device. The sample rate for the digital data is 8 kHz.

When you choose a *Filter range* setting other than None, the SHARC inserts an FIR filter algorithm to process the digital signal data before it is retransmitted to the codec.

If you choose Noise as the *Input Source*, the SHARC will supply a fabricated noise source to the FIR filter instead of the live digital data from the analog input.

Select the Exit Demo button to terminate the demo program on the EZ-KIT Lite and close the dialog box.

The C source code for this program is located in the `ezsharc\bp` directory.

3.2.2.2 FFT Demo

This program demonstrates the application of a Fast-Fourier Transform (FFT) to a codec input source or an internally generated noise source. After the host program downloads the program image, it will display the following dialog:

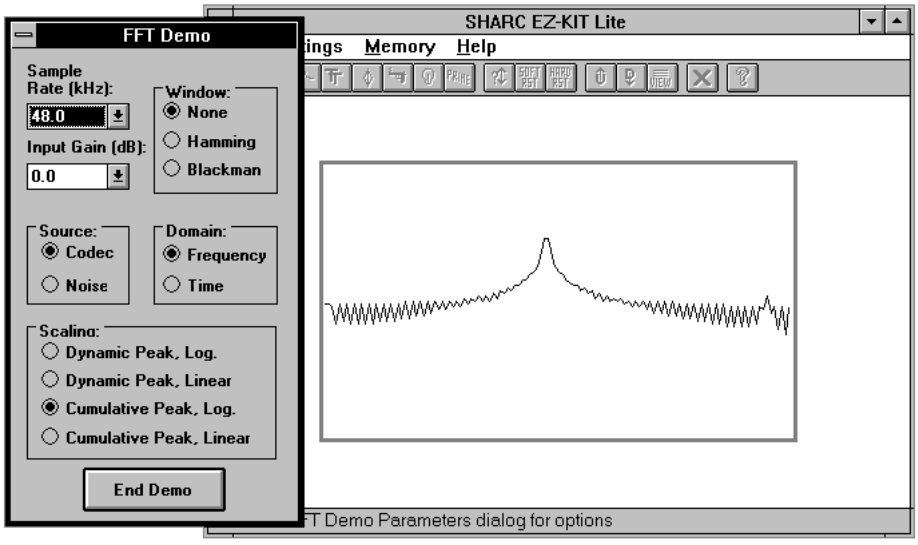


Figure 3-11 FFT Demo Dialog

3 Operation

This demonstration starts with a talk-through program, with the Input Source set to Codec. The AD1847 codec digitizes the analog input signal and transmits the data to the SHARC's serial port. The SHARC reads data from the serial port, performs an FFT calculation (which does not affect the data), and then retransmits the data back to the codec. The codec converts the data to an analog signal that drives the output device.

The results of the FFT are displayed in the client area of the main window. You can display either the time domain data or the frequency domain data by choosing the corresponding radio button in the *Domain* section of the dialog box.

You can change the codec's sample rate by choosing a value from the drop-down list in the *Sample Rate* section of the dialog box.

The *Input Gain* section allows you to amplify the input signal before it is digitized. Use the drop-down list to choose your desired gain value.

If you choose Noise as the *Source*, the SHARC will supply a fabricated noise source to the FFT algorithm instead of the live digital data from the analog input.

The *Window* section allows you to optionally apply a Hamming or Blackman window to the data before the FFT is calculated.

The *Scaling* section of the dialog box affects how the data is presented in the main window. By choosing one of the four options, you can make the horizontal scale follow either linear or logarithmic and the vertical scale either Dynamic (full scale every screen update) or Cumulative Peak (full scale over all updates).

Select the End Demo button to terminate the demo program on the EZ-KIT Lite and close the dialog box.

The C source code for this program is located in the `ezsharc\fft` directory.

3.2.2.3 Talk-through Demo 

The talk-through program is a simple program that exercises the stereo audio codec. To successfully experience this demo, you should connect an audio source and speaker. After the host program downloads the program image, it will display the following dialog:

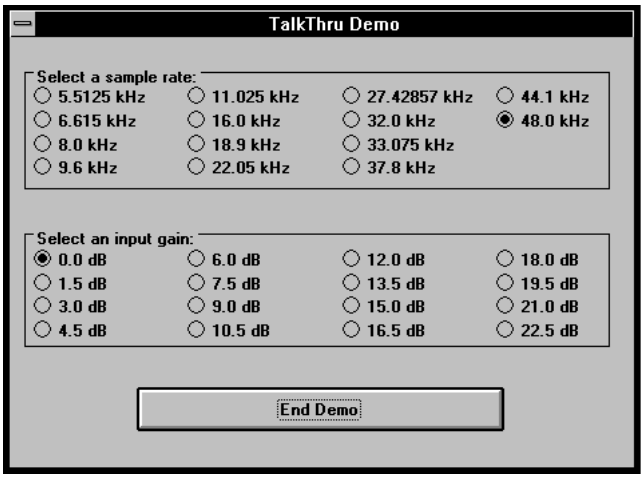


Figure 3-12 TalkThru Demo Dialog

When the talk-through program starts, the AD1847 codec digitizes the analog input signal and transmits the data to the SHARC's serial port. The SHARC reads data from the serial port and then retransmits the data back to the codec. The codec converts the data to an analog signal that drives the output device.

You can alter parameters of the AD1847 codec's operation while the program is running. You do this by choosing from the options presented in the demo's dialog box.

The radio buttons in the *Select a sample rate* section gives you fourteen choices for the codec's input and output conversion rate.

You can change the input gain that is applied to the input signal before it is digitized. Choose one of sixteen values presented in the *Select an input gain* section.

Select the End Demo button to terminate the demo program on the EZ-KIT Lite and close the dialog box.

The C source code for this program is located in the `ezsharc\tt` directory.

3.2.2.4 Pluck String Demo

This program demonstrates a simple algorithm for simulating the sound of a "plucked" string. The SHARC uses the algorithm to generate digital audio samples according to data that specifies the notes to be played. The audio samples are transmitted to the AD1847 codec over a serial port. The codec converts the data to an analog signal that drives the output device.

3 Operation

Since the program does not have a dialog box, you can select the Software Reset function from the Settings menu or click on the button on the toolbar to end the demo.

The C source code for this program is located in the `ezsharc\pluck` directory.

3.2.2.5 Peter Gunn Demo 

This program demonstrates the Karplus-Strong algorithm for simulating the sound of a "plucked" string. The SHARC uses the algorithm to generate digital audio samples according to data that specifies the notes to be played. The audio samples are transmitted to the AD1847 codec over a serial port. The codec converts the data to an analog signal that drives the output device.

The demo is also contained in the boot PROM and executes when the power is first applied or when you press and release the RESET button. Since the program does not have a dialog box, you can select the Software Reset function from the Settings menu or click on the button on the toolbar to end the demo.

The assembly source code for this program is located in the `ezsharc\gunn` directory.

3.2.2.6 Blink Demo 

This simple program demonstrates how to use the SHARC's built-in timer to toggle two LED's on the EZ-KIT Lite board.

Since the program does not have a dialog box, you can select the Software Reset function from the Settings menu or click on the button on the toolbar to end the demo.

The C source code for this program is located in the `ezsharc\blink` directory.

3.2.2.7 Primes Demo 

This program calculates the first twenty prime numbers. When the calculation is complete, the host program reads the calculated values and displays them in the dialog box shown in Figure 3-13.

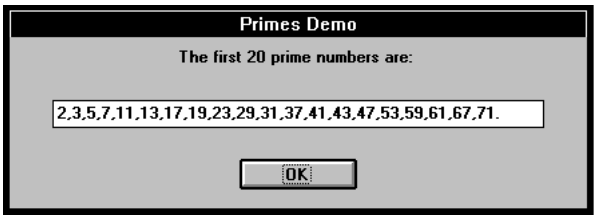


Figure 3-13 Primes Demo Dialog

The C source code for this program is located in the `ezsharc\primes` directory.

3.3 USING THE DIAG21K PROGRAM

Diag21k is a simple utility with a character-based interface that you can use to download and run DSP programs, perform memory tests, and "peek" & "poke" specific memory locations in a variety of formats.

3.3.1 Starting Diag21k

Diag21k accepts command-line switches that control various start-up options. The general syntax for Diag21k is

```
diag21k [switches]
```

The command-line switches are case-sensitive and are preceded with – or /. The following table describes each switch.

Table 3-1 Diag21k Command-line Switches

Switch	Description
-h or **-?**	Displays Diag21k command-line syntax and options.
-b*n* or **-B***n*	Instructs Diag21k to open board number *n* at start-up. Diag21k will look for the environment variable ADSP*n* to determine the board type. You should specify multiple **-b** switches to use Diag21k with more than one board. If you do not specify a board number, Diag21k defaults to board number 0.
-p*filename*	Diag21k will automatically load the DSP program *filename* after the board is opened. If you specify multiple **-b** switches, Diag21k will download the program to the board described by the first **-b** switch.
-P*prompt*	Diag21k will use the string *prompt* at the command line instead of the standard "diag21k" prompt. The active board number and ">" will be appended to the end of *prompt*.
-x*filename*	Diag21k will execute the command file *filename* after all the boards are opened.

3.3.2 Command Descriptions

This section describes all of the Diag21k commands. Table 3-2 lists the conventions used in this guide to describe the arguments to the commands. Table 3-3 summarizes the commands which are detailed in the text that follows the table.

Table 3-2 Diag21k Command Argument Conventions

Notation	Description
`[ ]`	The argument inside the brackets is optional
`[c]`	Continuous Mode: Appending 'c' to the command will cause it to repeat continuously—until a key is hit. An example is memory read (mrc), which will poll a memory location until a key is hit.
`[p]`	Paged Mode: Appending 'p' to the command will limit the command's output to a single screen. Pressing the <esc> will terminate the command; pressing any other key will allow the next screen to be output.
`<bank>`	A single letter indicating which memory segment should be affected by the command: p=program, s=data-sram, d=dram, e=external, l=label*, a=all (test command only).
`<fmt>`	A single letter indicating the data size and format: c=char, f=float (single-precision), h=hex, i=integer, l=long, s=string
`<test>`	A single letter indicating a memory test that should be performed: s=sequential, r=random, c=checkerboard, a =all.

**When label 'l' is given as a bank argument, the supplied address is assumed to be the name of a global variable.*

Table 3-3 Summary of Diag21k Commands

Command	Description
--	Comment (ignore remainder of line)
?	Display command list (help)
bi	Board Info: display information about active board
br	Board Reset: hard-reset entire DSP board
bs	Board Select: specify active board
echo	Print a message to the screen
fl	File Load: download DSP executable (COFF)
fx	File Execute: Run a command file
ga	Get Address: Display address of global variable
ih	IOP Help: display IOP register descriptions
in	Input word from I/O port
ir	IOP Read: display IOP register value
iw	IOP Write: set IOP register value
md	Memory Dump: read DSP memory and write output to file
mh	Memory Help: display memory command help screen
ml	Memory Load: Read data from file and write to DSP memory
mr	Memory Read
mt	Memory Test
mv	Memory View: display memory range graphically
mw	Memory Write
os	Operating System: Open a command shell
out	Output word to I/O port
pc	Processor Configure
pf	Processor Flag read
pfw	Processor Flag write
pr	Processor reset
ps	Processor start
q	Quit Dsp21k, reset active board
x	Exit: quit DSP21k, leave processor running

3 Operation

? – Command Help

Syntax:

? [<cmd>]

Description:

Displays a list of Diag21k commands with descriptions. You can get help about a specific command and its syntax by specifying an optional **<cmd>** parameter.

Example:

```
diag21k[0]>? mr
mr[c|p] bank[fmt] addr(hex) [count(dec)]        Read memory.
```

bi – Board Information

Syntax:

bi

Description:

Displays information about the active board.

```
diag21k[0]>bi
+-------------------------------------------------------------------------------+
| Board/Processor Information for ADSP#0   (Not Started)                        |
+-------------------------------------------------------------------------------+
|   Board Type: (23) SHARC EZ-KIT Lite                                          |
|     COM Port: #1                          COM Port Speed: 115200 baud         |
|     DSP Type: (5) ADSP-21061                                                  |
+-------------------------------------------------------------------------------+
| Int. Mem: 2 Mbit        IMDW0: 32-bit data               IMDW1: 32-bit data   |
|   MMS WS: 1     Ext Bank Size:     8 KW (MSIZE =  0) DRAM PgSz:     0 W       |
| Bank 0: Start = 0x00400000  Width = 32 bits  Depth =    0 KW  WS/WM = 6 / 2   |
| Bank 1: Start = 0x00402000  Width = 32 bits  Depth =    0 KW  WS/WM = 6 / 2   |
| Bank 2: Start = 0x00404000  WS/WM = 6 / 2                                     |
| Bank 3: Start = 0x00406000  WS/WM = 6 / 2                                     |
| Unbnkd: Start = 0x00408000  WS/WM = 6 / 2                                     |
+-------------------------------------------------------------------------------+
|  Program loaded: (none)                                                       |
|  Labels: *not defined*                                                        |
+-------------------------------------------------------------------------------+
```

br – Board Reset

Syntax:

br

Description:

Resets the SHARC processor, which will initiate a boot from the EPROM. The monitor program will perform its power-on self tests and then start the default demo program (Peter Gunn theme).

Example:

```
diag21k[0]>br
       Board reset
```

bs – Board Select

Syntax:

bs [boardnum]

Description:

Makes *boardnum* the active board. All subsequent commands to Diag21k will be interpreted and applied in the context of the board defined by the given board number. The *boardnum* parameter must be one of the boards that was opened on the command line with the **-b** switch. If you do not specify a *boardnum* parameter, a list of opened boards is displayed. If you specify a board that is not open, the active board number will not change.

Note: The active board number is always displayed in the command prompt.

Example:

```
diag21k[0]>bs 2
       Current board is 2
diag21k[2]>
```

3 Operation

echo – Print message to screen

Syntax:

echo <message string>

Description:

Displays **<message string>** on the standard output device (the screen by default). This command is most useful in command files to describe execution steps.

Example:

```
diag21k[0]>echo Reset and load processor with test code.
Reset and load processor with test code.
```

fl – File Load

Syntax:

fl filename[.21K]

Description:

Resets the active processor and then downloads a DSP executable file in the format (COFF) generated by the Analog Devices linker, ld21k. By default, the linker produces a '.exe', but we recommend using a '.21k' extension to avoid confusion with PC executables. Therefore, if you do not specify a file extension, '.21k' will be appended. This command will not start the program; you must use the **ps** (Processor Start) command to start the processor. Diag21k will store the addresses of global symbols that appear in the COFF file; you can use the **ga** (Get Address) command or the 'l' bank modifier in memory access commands to access the symbols.

Example:

```
diag21k[0]>fl primes60
       "primes60.21k" loaded
```

fx – File eXecute

Syntax:

fx filename[.CMD]

Description:

Redirects input from a text file that should contain a list of commands to be executed by Diag21k. The commands in the file use the same syntax as commands that are entered at the keyboard. If you do not specify a file extension, '.CMD' will be appended.

Example:

```
diag21k[0]>fx memtest.cmd
```

ga – Get Address

Syntax:

ga label

Description:

Returns the address and memory bank of a global symbol that is referenced in the most recently downloaded executable (using the **fl** command). Remember that the C compiler will prepend underscores.

Example:

```
diag21k[0]>ga _primes
       Label "_primes" = 0003:0004, DATA_SRAM
```

3 Operation

ih – IOP Help

Syntax:

ih [<regname | regaddr>]

Description:

Displays IOP register descriptions and addresses. You can specify the IOP register by its name or its address. Typing **ih** without an argument lists all IOP registers.

Example:

```
diag21k[0]>ih syscon
 "SYSCON"       (0x00)  System configuration register

diag21k[0]>ih 0xe0
 "STCTL0"       (0xe0)  Serial Port 0 Transmit Control Register
```

in – Input word from I/O port

Syntax:

in portnum [count]

Description:

Diag21k will input a 16-bit value from the I/O port given by **portnum**. The optional parameter **count** will cause Diag21k to read from the same I/O port **count** times.

Example:

```
diag21k[0]>in 0x700
  In (0x700) = 0xff1a = -230
```

ir – IOP Read

Syntax:

ir[c] <regname | regaddr> [count]

Description:

Reads and displays the value of an IOP register. You can specify the IOP register by its name or its address. Typing **irc** will read the same IOP register continuously until a key is pressed. The optional **count** parameter will cause Diag21k to read subsequent IOP register addresses.

Example:

```
diag21k[0]>ir syscon 4
  SYSCON  (0x00) = 0x00000410 = 000000002020 = 1040
  VIRPT   (0x01) = 0x00020014 = 000000400024 = 131092
  WAIT    (0x02) = 0x21ad6b5a = 004153265532 = 565013338
  SYSTAT  (0x03) = 0x00000112 = 000000000422 = 274
```

iw – IOP Write

Syntax:

iw[c] <regname | regaddr> value

Description:

Writes **value** to an IOP register. You can specify the IOP register by its name or its address. Typing **iwc** will write the same IOP register continuously until a key is pressed.

Example:

```
diag21k[0]>iw wait 0x21ad6b5a
```

3 Operation

md – Memory Dump

Syntax:

md <bank>[<fmt>] addr(hex) count(dec) filename

Description:

Reads **count** locations from memory bank **<bank>** and address **addr** and then writes the data in **<fmt>** format to an ASCII file (**filename**). The syntax for **<bank>** and **<fmt>** are given in Table 3-2. The address parameter (**addr**) should be hexadecimal; the **count** should be decimal.

Example:

```
diag21k[0]>md li _primes 20 primes.dmp
```

mh – Memory Help

Syntax:

mh

Description:

Displays detailed syntax for the memory commands.

Example:

```
diag21k[0]>mh
+-------------------------------------------------------------------------------+
¦ MEMORY ACCESS COMMAND HELP                                                    ¦
+-------------------------------------------------------------------------------¦
¦-COMMAND----SYNTAX-----------------------------------------------------------¦
¦ Mem-Bank   mb      prog [sram [dram [offset]]]                                ¦
¦ Mem-Dump   md      <bank>[<fmt>] addr(hex) count(dec) filename                ¦
¦ Mem-Load   ml      <bank>[<fmt>] addr(hex) filename                           ¦
¦ Mem-Read   mr[c|p] <bank>[<fmt>] addr(hex) [count(dec)]                       ¦
¦ Mem-Test   mt[c]   <bank><test> [<max_err>]                                   ¦
¦ Mem-Write  mw[c]   <bank>[<fmt>] addr(hex) value [count [delta]]              ¦
¦                                                                              ¦
¦-MODIFIERS------------------------------------------------------------------¦
¦       c = continuous mode (until key hit)                                    ¦
¦       p = paged mode (<esc> key quits, any other key advances page)          ¦
¦  <bank> = [p|s|d|e|l|a] = [Program/Data-SRAM/Data-DRAM/Extern/Label/ALL]     ¦
¦   <fmt> = [c|d|f|h|i|l|s] = [Char/Disasm/Float/Hex/Integer/Long/String]      ¦
¦  <test> = [s|r|c|a] = [Sequential/Random/Checkerboard/ALL]                   ¦
¦                                                                              ¦
¦- EXAMPLES: -----------------------------------------------------------------¦
¦      Read: diag21k[0]>mr p 0 10                                              ¦
¦      Dump: diag21k[0]>md si 0 100 dump.mem                                   ¦
¦     Write: diag21k[0]>mw sf 1a6 1.23 5 0.01                                  ¦
¦ DRAM Test: diag21k[0]>mt da                                                  ¦
+-------------------------------------------------------------------------------+
```

ml – Memory Load

Syntax:

ml <bank>[<fmt>] addr(hex) filename

Description:

Reads **<fmt>** formatted ASCII data from **filename** (probably created with memory dump command—**md**) and writes it to DSP memory bank **<bank>** and address **addr**. The syntax for **<bank>** and **<fmt>** are given in Table 3-2. The address parameter (**addr**) should be hexadecimal.

Example:

```
diag21k[0]>ml si 1000 primes.dmp
```

mr – Memory Read

Syntax:

mr[c|p] <bank>[<fmt>] addr(hex) [count(dec)]

Description:

Reads **count** locations from memory bank **<bnk>** and address **addr** and displays the values in the **<fmt>** format. The syntax for **<bank>** and **<fmt>** are given in Table 3-2. A single location can be polled continuously (until a key is hit) by appending 'c' to the command. If you specify a large **count**, you can display a single page at a time by appending 'p' to the command. The default format for program memory disassembles assembly-level code; the default format for other memory banks is hexadecimal. The address parameter (**addr**) should be hexadecimal; the **count** should be decimal.

Examples:

(Read program memory from 20080-20084 and display in hex)

```
diag21k[0]>mr ph 20080 5
       PROG-SRAM [0002:0080] = 0x0f00:0000:0000
       PROG-SRAM [0002:0081] = 0x1100:0002:8000
       PROG-SRAM [0002:0082] = 0x1100:0002:8001
       PROG-SRAM [0002:0083] = 0x140a:0001:8000
       PROG-SRAM [0002:0084] = 0x142c:0008:0000
```

(Disassemble program memory from 20080-20084)

```
diag21k[0]>mr p 20080 5
 20080=0f00:0000:0000 r0=0;
 20081=1100:0002:8000 dm(0x28000)=r0;
 20082=1100:0002:8001 dm(0x28001)=r0;
 20083=140a:0001:8000 bit set mode2 0x18000;
 20084=142c:0008:0000 bit clr astat 0x80000;
```

(Read 10 integers starting at location "_primes")

```
diag21k[0]>mr li _primes 10
       DATA_SRAM [0003:0004] =      2
       DATA_SRAM [0003:0005] =      3
       DATA_SRAM [0003:0006] =      5
       DATA_SRAM [0003:0007] =      7
       DATA_SRAM [0003:0008] =     11
       DATA_SRAM [0003:0009] =     13
       DATA_SRAM [0003:000A] =     17
       DATA_SRAM [0003:000B] =     19
       DATA_SRAM [0003:000C] =     23
       DATA_SRAM [0003:000D] =     29
```

mt – Memory Test

Syntax:

mt[c] <bank><test>[max_err]

Description:

Performs specified test(s) **<test>** on selected memory banks **<bank>**. The syntax for **<bank>** and **<test>** are given in Table 3-2. The parameter **max_err** specifies the maximum number of errors that are reported for each test (the default is 5). If you append 'c' to the command, the memory test will repeat continuously (until a key is pressed). You can abort the current memory test and all remaining tests by pressing the <esc> key. Pressing any other key will abort only the current memory test.

Example:

```
diag21k[0]>mt aa
Power-on self-test found no errors.
```

mw – Memory Write

Syntax:

mw[c] <bank>[<fmt>] addr(hex) value [count [delta]]

Description:

Writes **value** to **count** locations starting at memory bank **<bank>** and address **addr**. A single location can be written to continuously (until a key is hit) by appending 'c' to the command. The parameter **count** is always decimal (default count is 1). The format for **value** is determined by **<fmt>**. The syntax for **<bank>** and **<fmt>** are given in Table 3-2. If you specify **delta**, it will be cumulatively added to **value** each time it is written.

Examples:

(Fill program memory locations 20000-20009 with hex value)

```
diag21k[0]>mw p 20000 1234:5678:abcd 10
```

(Write the value 1.23 to "my_float_var")

```
diag21k[0]>mw lf my_float_var 1.23
```

3 Operation

os – Operating System

Syntax:

os [command string]

Description:

Execute operating system command or shell. If you do not enter a command string, a temporary command shell is opened; you must type "exit" to return to Diag21k.

Examples:

(Look for dsp21k executables in current directory)

```
diag21k[0]>os dir *.21k
```

(Open command shell to the operating system)

```
diag21k[0]>os
Type EXIT to return.
c:\dsp21k\bin>
```

out – Output word to I/O port

Syntax:

out portnum portval [count]

Description:

Diag21k will output a 16-bit value **portval** to the I/O port given by **portnum**. The optional parameter **count** will cause Diag21k to write to the same I/O port **count** times.

Example:

```
diag21k[0]>out 0x300 0xff1a
```

pc – Processor Configure

Syntax:

pc

Description:

Configures a SHARC processor prior to accessing memory or loading a program. Its primary function is to program the SHARC's SYSCON and WAIT registers so that memory can be accessed properly through the IOP. The ADSPx environment variable controls how external memory is accessed. This command should be issued *after* the processor has been reset since the reset will return these registers to their default values.

Example:

```
diag21k[0]>pc
       processor configured
```

pf – Processor Flag Read

Syntax:

pf

Description:

Returns the state of the active board's processor flags (all that are available for reading). Bit 0 of the returned value represents FLAG0, bit 1 represents FLAG1, etc.

Example:

```
diag21k[0]>pf
       Flags[3-0] = 0111
```

3 Operation

pfw – Processor Flag Write

Syntax:

pfw flag value

Description:

Sets **flag** to the specified **value**. The flags that can be set vary depending on the type of board. The parameter **value** can be 0 or 1.

Example:

```
diag21k[0]>pfw 3 1
```

pr – Processor Reset

Syntax:

pr

Description:

Soft resets the processor. The demo or user program that is currently running will terminate. The monitor program will regain control of the SHARC but will not disturb user program or data memory.

Example:

```
diag21k[0]>pr
       processor reset
```

ps – Processor Start

Syntax:

ps

Description:

Starts the currently loaded program; execution will begin at the reset vector.

Example:

```
diag21k[0]>ps
       processor running
```

q – Quit

Syntax:

q

Description:

Leave Diag21k after resetting the active processor.

Example:

```
diag21k[0]>q
exiting...reset processor

c:\dsp21k\bin>
```

x – eXit

Syntax:

x

Description:

Leave Diag21k with DSP running.

Example:

```
diag21k[0]>x

c:\dsp21k\bin>
```

3 Operation

3.4 USING THE EZ-ICE EMULATOR

The optional EZ-ICE In-Circuit Emulator gives you a powerful tool for debugging programs running on the SHARC. The SHARC EZ-KIT Lite's operation can be completely controlled from the emulator's user interface. Through the emulator you can download programs, start and stop program execution, set breakpoints, and observe and change register and memory contents.

The EZ-ICE consists of an ISA bus card and a cable that extends from the EZ-ICE card to a probe that can be connected to the SHARC EZ-KIT Lite. The probe mounts to the development board using the In-Circuit Emulator connector. Refer to the hardware connections diagram (Figure 2-1) for the location of the EZ-ICE connector (J5). The connector is keyed to prevent an incorrect installation.

To set up the SHARC EZ-KIT Lite for use with the emulators, follow these steps:

1. Apply power to the SHARC EZ-KIT Lite with the two jumpers on the JTAG connector installed (see Figure 3-14).

2. Remove the two jumpers from the JTAG connector (these jumpers must be reinstalled when the emulator is removed) and then connect the EZ-ICE probe. As long as the emulator software is not running, you can safely attach and remove the EZ-ICE probe while the SHARC EZ-KIT Lite is powered on.

3. Start the emulator software on the PC.

Follow the instructions in the emulator documentation to download and run programs.

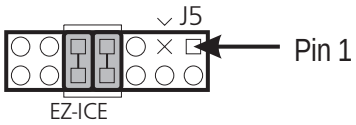


Figure 3-14 EZ-ICE Connector (with jumpers installed)

Table 3-4 EZ-ICE Connector Pin Out

Pin	Name	Description
1	GND	
2	EMU*	Emulator Status
3	KEY PIN	No Pin
4	CLKIN	21061 CLKIN
5	BTMS	Target's TMS
6	TMS	Test Mode Select
7	BTCK	Target's TCK
8	TCK	Test Clock
9	BTRST*	Target's TRST
10	TRST*	Test Reset
11	BTDI	Target's TDI
12	TDI	Test Data In
13	GND	
14	TDO	Test Data Out

3 Operation

Developing Applications 4

4.1 OVERVIEW

If you'd like to develop your own DSP programs, you can use the software development tools provided with the SHARC EZ-KIT Lite. If you have limited experience in developing code for a DSP-based system, you should review the steps in this chapter. You should also read all the related product documentation listed in Chapter 1.

The following development steps serve as a guideline for creating your own programs. Keep in mind that the development process varies depending upon the style of the particular developer. Follow this guideline as a starting point and feel free to modify it to suit your own work style. In many cases you will be able to skip a step because of the components shipped with EZ-KIT Lite. For example, the hardware architecture description process is described below, yet it is not necessary to follow this step since an architecture file for the ADSP-21061 development board is included in the EZ-KIT Lite package. The following sections explain these topics in more detail.

All commands that are mentioned throughout the following sections are to be typed at the DOS prompt (`C:\>`). You should check that the Analog Devices software development tools are installed and that the executable binaries directory (usually `c:\ez-kit\bin`) is listed in your PATH variable.

4.2 DEFINING YOUR SYSTEM

Before you can start developing your own application, you must define what it will do and how you can best use your available hardware resources. The following sections will take you through these steps.

4.2.1 Step 1: System Requirements

The first step in developing a DSP system is to determine what capabilities the system will need. These capabilities will depend on the types of algorithms being implemented, the types of signals being used, and the types of I/O devices that need to be connected to the DSP. An evaluation of the size requirement for data memory and program memory is made based on the amount of data being acquired by the I/O and the amount of processing being performed. The estimated size of the program created by implementing the algorithms used also determines the required program memory.

4.2.2 Step 2: System Design

Once the system requirements are determined, a hardware system can be designed. In this case, an ADSP-21061 based system has been designed for you. This design utilizes an AD1847 audio codec to perform the A/D and D/A conversions. The AD1847 is connected to serial port 0 (SPORT0) on the SHARC. The internal memory of the ADSP-21061 (1 Megabit) is sufficient so that no external memory is connected. The system's 16550 UART is used to communicate via the RS-232 interface to the host PC. If you wish, you can study the electrical schematics in Chapter 8.

4.2.3 Step 3: Architecture Description File

When using the ADSP-21000 family development tools, you must describe the hardware system in an architecture description file. The primary purpose of this file is to define the SHARC's memory map for the linker. Since the ADSP-21061 system is already defined, this step has been done for you. A file called `ezkit.ach` is included in the EZ-KIT Lite software. The text file shown below is the contents of this architecture file.

```
!-----------------------------------------------------------------------
.SYSTEM         SHARC_EZKIT_Lite;
!
! This architecture file is required for use with the SHARC EZ-KIT
! Lite development software. It is structured for use with the C
! compiler but also can be used with assembly code.
!
! This architecture file allocates:
!    Internal 133 words of 48-bit run-time header in memory block 0
!              16 words of 48-bit initialization code in memory block 0
!             619 words of 48-bit kernel code in memory block 0
!            7424 words of 48-bit C code space in memory block 0
!              4K words of 32-bit PM C data space in memory block 0
!
!              8K words of 32-bit C DM data space in memory block 1
!              4K words of 32-bit C heap space in memory block 1
!            3712 words of 32-bit C stack space in memory block 1
!             384 words of 32-bit kernel data in memory block 1

.PROCESSOR = ADSP21061;

!   ------------------------------------------------------------
!   Internal memory Block 0
!   ------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00020000 /END=0x00020084 /PM/WIDTH=48        seg_rth;
.SEGMENT/RAM/BEGIN=0x00020085 /END=0x00020094 /PM/WIDTH=48        seg_init;
.SEGMENT/RAM/BEGIN=0x00020095 /END=0x000202ff /PM/WIDTH=48        seg_knlc;
.SEGMENT/RAM/BEGIN=0x00020300 /END=0x00021fff /PM/WIDTH=48        seg_pmco;
.SEGMENT/RAM/BEGIN=0x00023000 /END=0x00023fff /PM/WIDTH=32        seg_pmda;

!   ------------------------------------------------------------
!   Internal memory Block 1
!   ------------------------------------------------------------
.SEGMENT/RAM/BEGIN=0x00024000 /END=0x00025fff /DM/WIDTH=32        seg_dmda;
.SEGMENT/RAM/BEGIN=0x00026000 /END=0x00026fff /DM/WIDTH=32 /cheap seg_heap;
.SEGMENT/RAM/BEGIN=0x00027000 /END=0x00027e7f /DM/WIDTH=32        seg_stak;
.SEGMENT/RAM/BEGIN=0x00027e80 /END=0x00027fff /DM/WIDTH=32        seg_knld;

!   ------------------------------------------------------------
!   External Memory Select 1 is reserved for the UART.
!   ------------------------------------------------------------

.ENDSYS;
```

This architecture file assumes that you will be using the EZ-KIT Lite's built in kernel to download and start your program. The kernel uses certain regions of memory (`seg_krnlc` and `seg_krnld`) that your program should avoid. The remaining memory segments are available to your program. Refer to section 6.2.1 for more details on the memory map.

The development tools included with the SHARC EZ-KIT Lite will always use this architecture file. You must buy the complete version of the ADSP-21000 Family Software Development Tools to specify a different architecture file.

4.3 DEVELOPING CODE

4.3.1 Step 4: Writing Source Code

Once the hardware is determined, you can begin to develop the software. First, determine all the memory requirements for variables and arrays along with all the needed interrupts for the ADSP-21061 system. Any hardware or registers that need to be initialized should also be planned out.

Your program can be written in C or using the SHARC's assembly language instructions. You will write a program by entering text (C code or assembly language instructions) into a text file and then processing the text file with either the C compiler or the assembler. The C compiler converts standard C expressions into native SHARC assembly instructions. The Assembler translates the processor's algebraic, easy-to-read instruction set from your source file or the C compiler's output into a relocatable object file.

4.3.2 Step 5: Running The Compiler or Assembler

After you have finished creating the text file which contains your program, you must use either the compiler or the assembler to create the object file.

Compiling a C program actually requires several steps. Analog Devices provides a tool that can perform all of the required steps for you. To compile a C program called `myprog.c`, you can enter the following command:

```
g21k myprog.c
```

The tool will automatically perform all of the required steps to compile the source into assembly language, assemble the compiler's output into a relocatable object file, and then link the object file into an executable binary file. By default, the resulting file will be named `myprog.exe`.

If you want to separate the compile and link stages, the following command will generate a relocatable object file by stopping the process before invoking the linker:

```
g21k -c myprog.c
```

You can get a complete list of all of the available switches by entering the following command:

```
g21k -?
```

If your source code uses assembly language, you can run the assembler with the following command:

```
asm21k -adsp21060 my_prog.asm
```

There are a number of assembler switches that can be optionally used for functions like list file creation and object file naming. Here is another example which specifies the creation of a listing file which will be called `my_prog.lst`.

```
asm21 -adsp21060 -l my_prog.asm
```

You can get a complete list of all of the available switches by entering the following command:

```
asm21k -?
```

By default, the assembler creates an object file with the extension `.obj` on the file name. The example command shown above creates the file `my_prog.obj`.

The tool that you used to compile C programs can also be used to invoke the assembler. The filename extension (`.asm`) tells the tool to skip the C compiler stage and just invoke the assembler. For examplem, the following command will assemble my_prog.asm and create a relocatable object file:

```
g21k -c myprog.asm
```

By default, this tool creates an object file with the extension `.o` on the file name. The example command shown above creates the file `my_prog.o`.

4.3.3 Step 6: Running The Linker

The linker creates an executable file from the object modules created by the assembler (or the assembly phase of the C compiler tool g21k). The following example creates a file called `demo.exe`.

```
ld21k my_prog -o demo.exe
```

There are a number of other switches that are used to create a symbol table, create a map file ,and to specify the object files indirectly (specify a file name where the file contains a list of all the object files to be linked). If you type the following command, the proper use of the command and all the switches will be listed on the screen.

```
ld21k -?
```

4.4 VERIFYING THE APPLICATION

4.4.1 Step 7: Running The Simulator

The simulator lets you run your code in a simulation environment to test your software without using an actual hardware system. This optional step is used to make sure your software works before you run it on your hardware.

Many times a problem can arise where you load your software on to the hardware and it doesn't work. Without verifying your program's operation on the simulator, you can not be sure whether the failure is related to hardware or software. If you have verified the operation of your software on the simulator then download the code to your hardware and it doesn't work, it most likely is due to faulty hardware.

The simulator requires Microsoft Windows. You can start the emulator by clicking on the icon that was installed with the development tools. Please refer to the development tools documentation for instructions on running the simulator.

4.4.2 Step 8: Running on the SHARC EZ-KIT Lite Board

The SHARC EZ-KIT Lite board has an EPROM on it which contains ADSP-21061 code. When the board is powered up (or reset) the code is automatically transferred from the EPROM into the internal memory of the ADSP-21061. The code shipped with the EZ-KIT Lite includes a monitor (kernel) program which allows the ADSP-21061 to communicate with the RS-232 interface. The code performs a self test and then sends an audio signal to the audio output connector. You will need to have a set of powered speakers attached to the audio output connector to hear the sound produced. The signal at this connector is a line level so you will need a set of speakers that have an amplifier in them.

Once you install the host software onto your PC you should be able to run the host program under Windows. You can then download an executable file (`.exe` file produced by the linker). You should refer to section 3.2 of this manual for details on using the host program. You can also use the character-based program Diag21k from DOS. Refer to section 3.3 of this manual for detailed instructions.

4 Developing Applications

Programs that you write must conform to certain restrictions to ensure that your program does not interfere with the operation of the kernel. These restrictions are detailed in section 6.2.6.

If you prefer, you can program your own EPROM and insert it into the EPROM socket on the EZ-KIT Lite board to run your own program in stand alone mode.

4.4.3 Step 9: Programming An EPROM

Once you have verified that your software works you can format the executable so that it can be programmed into an EPROM. The EPROM can then be inserted into the EPROM socket on the board to run your program. You can invoke the boot loader tool with the following command:

```
ldr21k -bprom -o demo.ihx -id1exe=demo.exe
```

This will take the executable file `demo.exe` and create a PROM file called `demo.ihx`. The PROM format is the Intel Hex record format. This file can be downloaded to a PROM programmer to program an EPROM which can be inserted into the EPROM socket on the SHARC EZ-KIT Lite board.

Upon power up reset or when you hit the reset button on the board, the contents of the EPROM are automatically loaded into the internal program and data memories of the ADSP-21061 and coded execution begins.

Hardware Description 5

5.1 OVERVIEW

This chapter describes the hardware characteristics of the SHARC EZ-KIT Lite board. It includes a discussion of the board's major features and section that describes the user-configurable items.

5.2 BOARD LAYOUT

Figure 5-1 shows the layout of the SHARC EZ-KIT Lite board, which consists of a printed circuit board measuring 4.5 inches by 6.5 inches. This figure highlights the locations of the major components which are described in the following sections.

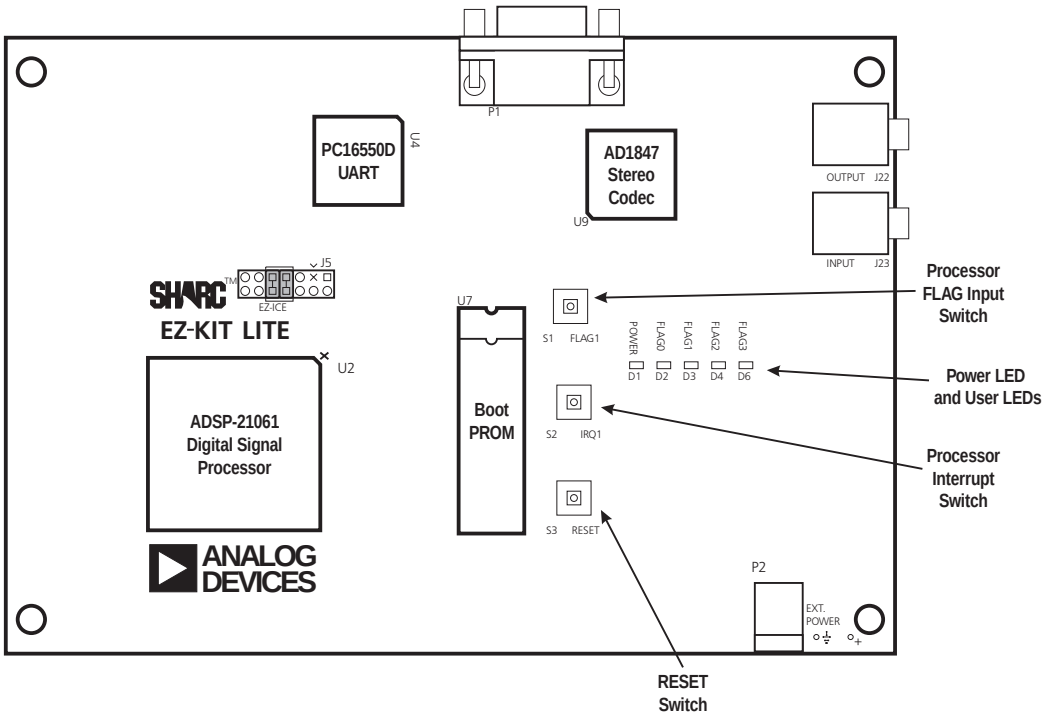


Figure 5-1 Major Features of the SHARC EZ-KIT Lite Rev. 1 Board

5.2.1 SHARC Processor

This is the ADSP-21061 processor which operates at 40 MHz. The Pin 1 Index is located in the upper right-hand corner.

5.2.2 Boot PROM

The boot PROM (U7) provides 8-bit wide program storage that can be loaded by the SHARC at startup. The socket that is mounted on this board is designed to accept EPROMs from 256K bits up to 8M bits. Jumpers JP1 - JP4 provide the necessary adjustments required to accommodate the different sizes of EPROM. When the SHARC is configured for PROM booting, the first 256 instructions (1536 bytes) are automatically loaded by the SHARC when reset is released. The remaining program image must be loaded by the program that is installed in those first 256 instructions. The `ldr21k` utility can do this for you. Refer to the *ADSP-2106x SHARC User's Manual* for more information on program booting.

5.2.3 PC16550D UART

The UART (U4) and the line driver (U5) provide the RS-232 interface that is used to communicate with the PC. The PC16550D is similar to devices used in most PCs. It has a programmable bit rate and has transmit and receive FIFO registers.

The UART is attached to the SHARC's external memory bus and is selected by MS1 (external memory bank 1). The UART can generate an interrupt to the SHARC on IRQ2.

5.2.4 AD1847 Stereo Codec

The AD1847 codec (U9) provides the stereo audio input (A/D) and output (D/A) interface. It is connected to the SHARC via SPORT0. This high speed synchronous serial port carries all of the data, control, and status information between the DSP and the codec.

5.2.5 Power LED

The Power LED, when on, indicates that +5 V_{DC} used by the DSP and digital circuitry is present.

5.2.6 User LEDs

There are four LEDs on the SHARC EZ-KIT Lite board. Your DSP program can control them to indicate certain conditions in the software or to provide feedback. The LEDs are controlled by processor FLAG outputs of the DSP and are labeled according to the flag that enables them.

5.2.7 Processor Interrupt Switch

The IRQ1 switch lets you send an interrupt (IRQ1) to the DSP. This lets you manually cause this interrupt when executing a program.

5.2.8 RESET Switch

The RESET switch lets you initiate a power-on reset to the DSP. There are no restrictions on when the switch can be used, so do not press the switch unless you want a complete DSP reset.

5.2.9 Processor FLAG Input Switch

The FLAG1 switch lets you toggle the status of a flag pin (FLAG1) to the DSP. This lets you manually trigger the flag, providing an "event" while executing software.

5.2.10 Expansion Connectors

There are seven expansion connector sites that provide the signals for adding optional custom hardware. The interface contains the SHARC processor bus as well as six link ports (ADSP-21060 or ADSP-21062 only), a synchronous serial port, interrupts, flags, and various control signals.

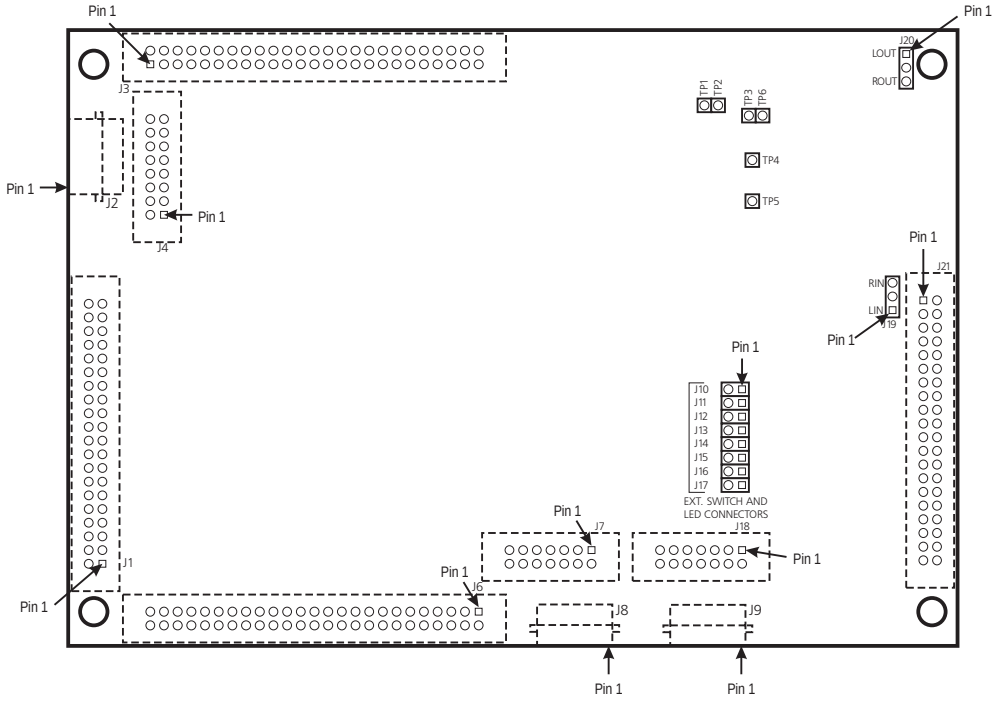


Figure 5-2 Expansion Connector Locations

5.3 BOARD CONFIGURATION

Figure 5-3 shows the locations of the configuration jumpers on the SHARC EZ-KIT Lite board. These jumpers should be checked before using the board to

5 Hardware Description

ensure proper operation. Each of the jumper selection blocks are described in the following sections.

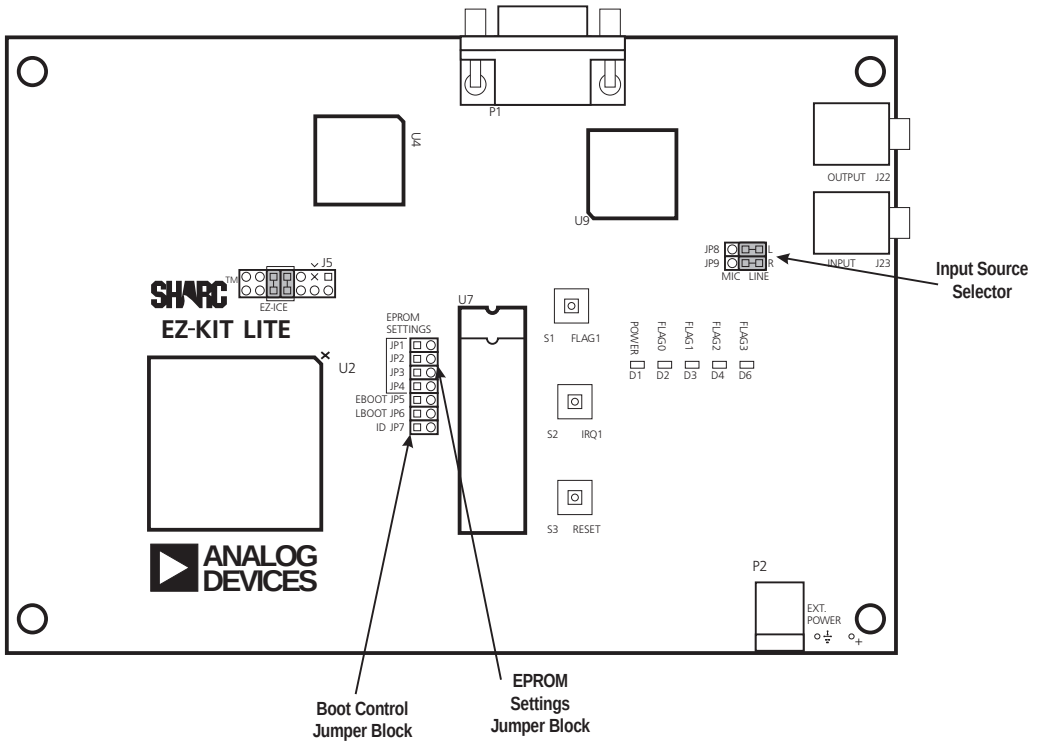


Figure 5-3 Configuration Jumpers

5.3.1 Input Source Selector

This pair of jumpers selects whether your input signal is standard line level or microphone level. If you select microphone level, the input signal is amplified before it is sampled by the AD1847 codec.

Line-level input audio (factory default).

Microphone-level input audio.

5.3.2 Boot Control Jumper Block

These jumpers control three of the SHARC processor's pins: EBOOT (JP5), LBOOT (JP6), and ID0 (JP7). The first two define the processor's boot mode according to the following table:

Table 5-1 Boot Mode Selection

Mode	JP5	JP6
EPROM	OUT	IN
Host Port	IN	IN
Link Port	IN	OUT

The SHARC EZ-KIT Lite is shipped with EPROM boot mode selected by default. The pins at location JP6 are shorted on the circuit board. If you want to use one of the other boot modes, you must modify the traces on the board.

The ID0 pin determines the SHARC's multiprocessor ID. If JP7 is shorted the multiprocessor ID number will be 000. If JP7 is left open (the factory-shipped default), the multiprocessor ID number will be 001.

5.3.3 EPROM Settings Jumper Block

These four jumpers (JP1-JP4) define the size of EPROM that you have installed in the Boot PROM socket. The SHARC EZ-KIT Lite is shipped with the 27C010 selected by default. The pins at locations JP2 and JP3 are shorted on the circuit board. If you want to use an EPROM of a different size, you must modify the traces on the board.

Table 5-2 EPROM Jumper Selection Chart

U7	JP1	JP2	JP3	JP4
27C256 (32K × 8)	OUT	IN	OUT	IN
27C512 (64K × 8)	OUT	IN	IN	OUT
27C010 (128K × 8)	OUT	IN	IN	OUT
27C020 (256K × 8)	IN	OUT	IN	OUT
27C040 (512K × 8)	IN	OUT	IN	OUT
27C080 (1M × 8)	IN	OUT	IN	OUT

5 Hardware Description

Programming Reference 6

6.1 OVERVIEW

This chapter gives you the technical details that you will need when you are writing programs for the SHARC EZ-KIT Lite. One section focuses on memory models, addresses, and other resources for programs that run on the SHARC processor. The other section focuses on the serial host interface so that you can write programs on the PC that talk directly to the monitor program running on the EZ-KIT Lite board.

6.2 DSP PROGRAMS

The section describes the model for EZ-KIT Lite DSP programs by focusing on the resources that are available to the SHARC. These resources include the memory map, the flags, the interrupts, and the serial ports. Table 6-1 summarizes these resources as they are implemented on the EZ-KIT Lite board.

Table 6-1 Summary of EZ-KIT Lite SHARC Resources

	MS (Memory Select)	FLAG	IRQ	Serial Port
0	Expansion Connector	LED D2 / 1847 CODEC RESET	Expansion Connector	1847 CODEC
1	16550 UART	From Push-button / LED D3	From Push-button	Expansion Connector
2	Expansion Connector	LED D4	From 16550 UART	
3	Expansion Connector	LED D6		

6.2.1 Memory Map

The SHARC memory model defines three main memory spaces. **Internal** memory space addresses a SHARC processor's own on-chip dual-ported SRAM. **Multiprocessor** memory space addresses the on-chip SRAM of other SHARC processors in the same cluster (i.e., SHARC processors that share a common processor bus). **External** memory space addresses other devices on the shared bus such as SRAM or DRAM.

6.2.1.1 Internal Memory Space

Since the SHARC EZ-KIT Lite has no external memory, you must store all code instructions and data in the built-in SRAM. The ADSP-21061 processor has one megabit of internal dual-ported SRAM that is divided into two 512-kilobit blocks. The blocks are designed so that you can configure regions of memory to be either 32 or 48 bits wide. The *ADSP-2106x SHARC User's Manual* contains

detailed information about the configuration and limitations of this on-chip SRAM.

If you are writing programs that will be loaded by the built-in kernel, you should be aware of how it uses memory. The architecture file for the EZ-KIT Lite (`ezkit.ach`) defines memory segments that are compatible with programs you write for the C compiler and the assembler. Table 6-2 lists the internal memory regions that are defined in the architecture file.

Table 6-2 Internal Memory Map

Segment Name	Description	Memory Block	Length	Width	Start Address	End Address
seg_rth	Run-time header	0	133	48	0x00020000	0x00020084
seg_init	Initialization data	0	16	48	0x00020085	0x00020094
seg_knlc	Kernel Code	0	619	48	0x00020095	0x000202FF
seg_pmco	User Code	0	7424	48	0x00020300	0x00021FFF
seg_pmda	User PM Data	0	4096	32	0x00023000	0x00023FFF
seg_dmda	User DM Data	1	8192	32	0x00024000	0x00025FFF
seg_heap	User C Heap	1	4096	32	0x00026000	0x00026FFF
seg_stak	User C Stack	1	3712	32	0x00027000	0x00027E7F
seg_knld	Kernel Data	1	384	32	0x00027E80	0x00027FFF

6.2.1.2 Multiprocessor Memory Space

The multiprocessor memory space (MMS) is consumed by any and all SHARC processors that are connected to the external processor bus (up to six as defined by the maximum cluster size). SHARC processors appear in specific portions of the MMS according to their multiprocessor ID. The default multiprocessor ID is one (001) for the EZ-KIT Lite processor. You can change the ID to zero (000) with a jumper setting (see 5.3.2).

Since the SHARC EZ-KIT Lite is a single-processor board, the only way to view other SHARC processors through the MMS is through the expansion connectors to your custom hardware.

6.2.1.3 External Memory Space

External memory space is further divided into four banked sections of memory and an non-banked section of memory. Each of the four banked sections (numbered 0 to 3) are sized the same and are defined by the MSIZE bits of the SHARC processor's SYSCON register (refer to the *ADSP-2106x SHARC User's Manual* for more details). Bank 0 starts at 0x40 0000 and the bank size determines the starting address of each of the other banks. Non-banked memory space starts

after Bank 3 and covers the remainder of external memory space (up to 0xFFFF FFFF).

Table 6-3 shows examples of settings for MSIZE and how it affects the bank addresses. Please note that programming the MSIZE bits may affect the location of other resources available to the SHARC processor. For example, the 16550 UART is selected by accessing external bank 1 (see Table 6-4); changing the MSIZE setting from its default value of 0 (8K banks) to a value of 4 (128K banks) will cause the starting address of the UART memory space to move from 0x0040 2000 to 0x0042 0000.

Table 6-3 Example MSIZE Settings

Memory Depth	MSIZE value	Memory Bank	Starting Address (hex)	Ending Address (hex)
128K	4	MS0	0040 0000	0041 FFFF
		MS1	0042 0000	0043 FFFF
		MS2	0044 0000	0045 FFFF
		MS3	0046 0000	0047 FFFF
512K	6	MS0	0040 0000	0047 FFFF
		MS1	0048 0000	004F FFFF
		MS2	0050 0000	0057 FFFF
		MS3	0058 0000	005F FFFF

External Memory Bank 1 addresses the 16550 UART registers (see Table 6-4). Please note that the UART only decodes the three least significant address bits which results in aliases in the remainder of the bank. All of the UART registers are 8 bits wide. Refer to the PC16550D UART data sheet for detailed programming information.

Table 6-4 External Memory Bank 1 Map

Offset from base of bank 1 (hex)	16550 Register Name
0000 0000	Receiver Buffer (read only, DLAB=0) Transmitter Holding (write only, DLAB=0) Divisor Latch LS (DLAB=1)
0000 0001	Interrupt Enable (DLAB=0) Divisor Latch MS (DLAB=1)
0000 0002	Interrupt Identification (read only) FIFO Control (write only)
0000 0003	Line Control
0000 0004	MODEM Control
0000 0005	Line Status
0000 0006	MODEM Status
0000 0007	Scratch

6.2.2 Flags

The ADSP-21061 SHARC processor has four I/O flags that you can program as either inputs or outputs. Bits in the MODE2 register control the direction. At reset, all of the flags are configured as inputs. Bits in the ASTAT register contain the value of each of the flag pins.

FLAG0 controls the $\overline{\text{RESET}}$ pin on the AD1847 codec. Clearing FLAG0 to zero holds the AD1847 in reset. Setting FLAG0 to one releases RESET and restarts the AD1847. The flag should be configured as an output and set high during program initialization. FLAG0 is also connected to LED D2 on the SHARC EZ-KIT Lite. LED D2 will light while FLAG0 is zero.

FLAG1 is connected to the push-button switch labeled "FLAG1" (S1) and to LED D3. The flag should be configured as an input during program initialization. The DSP's FLAG1 input value is "0" (and LED D3 will light) when the push-button is depressed and the value is "1" when the push-button is released.

FLAG2 is connected to LED D4. The flag should be configured as an output during program initialization. The FLAG2 LED lights while FLAG2 is zero.

FLAG3 is connected to LED D6. The flag should be configured as an output during program initialization. The FLAG3 LED lights while FLAG3 is zero.

6.2.3 Interrupts

The SHARC has three external interrupt pins. They are prioritized, individually maskable (IMASK register), and can be configured to be either edge sensitive or level sensitive (bits in the MODE2 register). At reset, all external interrupts are level sensitive and masked. Other relevant SHARC registers are MODE1 (NESTM and IRPTEN bits), IRPTL, and IMASKP.

Two of the SHARC's three external interrupt inputs are allocated on the SHARC EZ-KIT Lite. Table 6-1 summarizes their use.

IRQ0 is not connected to any devices on the SHARC EZ-KIT Lite; however, it is connected to one of the expansion connectors.

IRQ1 is connected to a push-button switch on the SHARC EZ-KIT Lite board. Depressing the switch generates the interrupt.

IRQ2 is connected to the 16550 UART. The UART can be programmed to generate an interrupt when it requires attention.

6.2.4 Serial Ports

The SHARC has two high-speed synchronous serial ports (SPORTs). They can operate in point-to-point connection full-duplex mode, with independent transmit and receive data lines and clocks, or they can be wired together with multiple SPORTs to operate in a time-division multiplexed (TDM) mode. As summarized in Table 6-1, only one of the SHARC's SPORTs is allocated on the SHARC EZ-KIT Lite.

SPORT0 is connected to the serial port on 1847 SoundPort CODEC. The port is configured for multichannel TDM operation.

SPORT1 is not connected to any devices on the SHARC EZ-KIT Lite; however, it is connected to one of the expansion connectors.

6.2.5 Stereo Audio Codec Programming

The stereo audio interface built into the SHARC EZ-KIT Lite is based on the AD1847 SoundPort® Stereo Codec. You can find detailed programming information in the AD1847 data sheet.

The SHARC EZ-KIT Lite uses SPORT0 to communicate with the AD1847's control and data interface. The SHARC's FLAG0 port controls the AD1847's $\overline{\text{RESET}}$ and BM pins. You can disable the CODEC by clearing FLAG0 to zero.

6.2.6 Kernel Compatibility

When you write programs that run on the EZ-KIT Lite board's SHARC processor, there are certain programming restrictions you should consider to ensure that the on-board kernel program will continue to operate. If you violate any of these restrictions, the kernel may become disabled or unstable. If this happens, you must reset the EZ-KIT Lite to reload the kernel from the boot EPROM.

- **Avoid using kernel memory regions.** The kernel uses two regions of the SHARC's internal SRAM. These regions are defined as separate segments (`seg_knlc` and `seg_knld`) in the EZ-KIT Lite architecture description file (see Table 6-2). If your program is written entirely in C, the linker will automatically use the appropriate memory segments.

- **Avoid the UART.** Since the kernel uses the RS-232 interface to communicate with the host computer, it must manage the UART chip. The UART is accessed in external memory space within external memory bank 1 (see Table 6-4). Your program can freely change the MSIZE setting in the SYSCON register (potentially changing the base address of external memory bank 1)

since the kernel recalculates the UART base address every time it needs to access the chip.

- **Don't interfere with the UART interrupt.** The kernel depends on the hardware interrupt signal IRQ2 to communicate with the UART. There are several ways that your program can affect the kernel's ability to receive interrupts from the UART:
  - **Do not disable interrupts globally.** Bit 12 (IRPTEN) in the MODE1 register must remain set.
  - **Do not mask IRQ2.** The IMASK register allows your program to enable or disable individual interrupts. Bit 6 (IRQ2I) in the IMASK register must remain set.
  - **Do not change IRQ2's sensitivity.** The UART uses the interrupt signal to indicate multiple conditions. If the kernel satisfies one condition, there may be other conditions requiring attention; however, the IRQ2 signal will not transition to the inactive state and then back to the active state. Therefore, the SHARC's interrupt controller must treat IRQ2 as a level-sensitive signal. Bit 2 (IRQ2E) in the MODE2 register must remain cleared.

6.3 SERIAL HOST INTERFACE

You can write programs for the PC that talk directly to the monitor program that loads from PROM when the EZ-KIT Lite board is reset. This section details the protocol for the serial interface between the PC and the monitor program.

The interface uses the RS-232 standard for serial communications that is common to all personal computers. This section does not cover the details of interfacing to the serial hardware on your computer. You can use the software programming interfaces that are offered by the operating system that you are using, or you can use one of many third-party packages that are available.

6.3.1 Message Packet Format

Serial communications between your PC program and the SHARC EZ-KIT Lite monitor program consists of message packets that are transmitted in both directions. Each message packet consists on an integral number of 32-bit words. Since the serial interface uses an 8-bit data format, consecutive bytes are combined to produce each word. Figure 6-1 shows the format for a typical message packet. Each packet has two words dedicated for header information. The remaining words contain message data (if necessary).

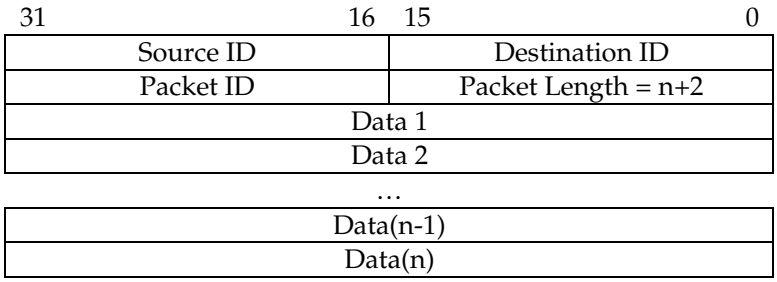


Figure 6-1 Serial Message Packet Format

The first word of the header contains the Source and Destination IDs (16 bits each) for message packet routing. For the SHARC EZ-KIT Lite, these values should always be zero.

The second word of the header contains the Packet ID and Packet Length. The **Packet ID** is a 16-bit value that identifies the type of message. The range of values for this field is from 0x0000 to 0x3FFF. This range is for command messages from the PC to the EZ-KIT Lite monitor program. When the monitor program responds to a command message, the Packet ID will contain the same value with most significant bit (31) set. If the monitor program detects an error while processing the command message, it will also set the second-most significant bit (30). The **Packet Length** is a 16-bit value that is the number of 32-bit words in the message packet including two-word header.

Table 6-5 lists all of the commands that are implemented in the SHARC EZ-KIT Lite monitor program. You can read a detailed description of each command in the sections that follow the table.

Table 6-5 Message Packet ID Summary

0x0000 - 0x00FF series: Download Commands	
0x0005	Write DM32
0x000e	Write PM48
0x0100 - 0x01FF series: Upload Commands	
0x0105	Read DM32
0x010E	Read PM48
0x010F	Read Core
0x0200 - 0x2FF series: Control Function Commands	
0x0201	Start processor
0x0203	Reset Processor
0x0204	Reset Board
0x0206	Set Serial Port Speed
0x0207	Verify Communications
0x0208	Resynchronize Communications

6.3.2 Download Commands

These commands instruct the monitor program to write data to SHARC memory.

Name: Write DM32 *Packet ID:* 0x0005

Description: Download 32-bit data memory

Format:

Source ID	Destination ID
0x0005	n+4
Starting address	
n = Number of 32-bit data locations to fill.	
D1	
D2	
…	
D(n-1)	
Dn	

Normal Response:

Source ID	Destination ID
0x8005	2

Name: Write PM48 *Packet ID:* 0x000E

Description: Download 48-bit program memory (unpacked format)

Format:

Source ID	Destination ID
0x000E	(n × 2) + 4
Starting address	
n = Number of 48-bit data locations to fill.	
D1 upper 32	
0	D1 lower 16
D2 upper 32	
0	D2 lower 16

…

D(n-1) upper 32	
0	D(n-1) lower 16
Dn upper 32	
0	Dn lower 16

Normal Response:

Source ID	Destination ID
0x800E	2

6.3.3 Upload Commands

These commands instruct the monitor program to read SHARC memory and return the requested data.

Name: Read DM32 *Packet ID:* 0x0105

Description: Upload 32-bit data memory

Format:

Source ID	Destination ID
0x0105	4
Starting address	
Number of 32-bit data locations to read.	

Normal Response:

Source ID	Destination ID
0x8105	n+4
Starting address	
n = Number of 32-bit data locations read.	
D1	
D2	

…

D(n-1)
Dn

Name: Read PM48 *Packet ID:* 0x010E

Description: Upload 48-bit program memory (unpacked format)

Format:

Source ID	Destination ID
0x010E	4
Starting address	
Number of 48-bit data locations to read.	

Normal Response:

Source ID	Destination ID
0x810E	(n × 2) + 4
Starting address	
n = Number of 48-bit data locations read.	
D1 upper 32	
0	D1 lower 16
D2 upper 32	
0	D2 lower 16

…

D(n-1) upper 32	
0	D(n-1) lower 16
Dn upper 32	
0	Dn lower 16

Name: Read Core *Packet ID:* 0x010F

Description: Return some SHARC core register values

Format:

Source ID	Destination ID
0x0105	2

Normal Response:

Source ID	Destination ID
0x810F	16
MODE1	
MODE2	
ASTAT	
IMASK	
IRPTL	
STKY	
PCSTK	
LADDR	
PCSTKP	
spare	
spare	
spare	
spare	
spare	

6.3.4 Control Function Commands

These commands instruct the monitor program to perform a program- or board-level control function.

Name: Start processor *Packet ID:* 0x0201

Description: Start processor executing user code

Format:

Source ID	Destination ID
0x0201	2

Normal Response:

Source ID	Destination ID
0x8201	2

Name: Reset Processor *Packet ID:* 0x0203

Description: Reset Processor by restarting kernel

Format:

Source ID	Destination ID
0x0203	2

Normal Response:

Source ID	Destination ID
0x8203	2

Name: Reset Board *Packet ID:* 0x0204

Description: Reset board

Format:

Source ID	Destination ID
0x0204	2

Normal Response:

Source ID	Destination ID
0x8204	2

Name: Set Serial Port Speed *Packet ID:* 0x0206

Description: Order kernel to change serial port speed and respond at the new speed

Format:

Source ID		Destination ID	
0x0206		4	
0	0	0	Divisor LSB
0	0	0	Divisor MSB

Divisor (MSB × 256 + LSB) = 1,152,000 ÷ New Speed

Normal Response:

Source ID	Destination ID
0x8206	2

6 Programming Reference

Name: Verify Communications *Packet ID:* 0x0207

Description: Verify communications with the board by having it echo back the message

Format:

Source ID	Destination ID
0x0207	n+2
D1	
D2	
…	
D(n-1)	
Dn	

Normal Response:

Source ID	Destination ID
0x8207	n+2
D1 = Power-on error code (0=no error)	
D2 = Kernel Version	
…	
D(n-1)	
Dn	

Name: Resynchronize Communications *Packet ID:* 0x0208

Description: Respond and then ignore all data until 0xBADC0DB4 is received.

Format:

Source ID	Destination ID
0x0208	2

Normal Response:

Source ID	Destination ID
0x8208	2

Host Response:

BA	DC	0D	B4

6.3.5 Error Response Message

When the monitor program detects an error while processing a command that it received from the host, it will return an error response message with the format show below.

Format:

Source ID	Destination ID
0xc000 + Original Packet ID	3
Error code	

Error codes:

Code	Meaning
1	Unknown packet type
2	Unknown destination
3	*RESERVED*
4	Communications Interface Error

6 Programming Reference

DspHost Reference 7

7.1 OVERVIEW

DspHost is a utility for DSP developers using add-in boards based on the Analog Devices ADSP-210xx floating point DSP processors. DspHost extends the capability of the ADI C Compiler to provide the DSP board with the I/O resources of the host PC, including the keyboard, the console, and the file system.

DspHost provides a library that contains standard C I/O functions. Standard functions include `printf`, `scanf`, `putc`, `getc`, and many more. These functions allow programs that include standard I/O function calls to be executed on the DSP.

7.2 DSPHOST EXAMPLE

Imagine being able to easily create a new application or port an existing one, and then execute that program on a DSP that does screen I/O. To show you how easy it is using DspHost, we'll step you through it. Here's a simple program named `hello.c`. This source is installed in your `\ezsharc\hello` subdirectory.

Listing 7-1 hello.c

```
#include <stdio.h>

main()
{
  printf("hello, world...");
  printf("this is your dsp speaking!\n");
}
```

Note that there is nothing special about this program—it will compile with any standard C compiler.

Now we want to compile the program and include the DspHost library. A batch file that compiles `hello.c` as a DspHost program is located in the same directory; from the DOS prompt, type:

```
mk
```

The batch file will invoke the g21k compiler and ensure that the DspHost library is linked with your program. You may notice that the resulting executable file is called `hello.21k` and not `hello.exe` as would ordinarily be generated by the compiler/linker. The batch file explicitly names the output file with a `.21k`

extension so as to avoid confusion with executable files that run on the PC. This convention is used throughout this and other chapters when referring to executable programs that are compiled for the 210xx family of processors.

The next step is to run the program on the DSP. This is accomplished by running the DspHost server, `dh21k.exe` from the DOS prompt. DspHost *requires* one command line argument: the name of the DSP executable file. So to run `hello.21k`, enter the following from the DOS prompt:

```
dh21k hello.21k
```

DspHost will load the program `hello.21k` into DSP memory and then execute it. All I/O calls made by the DSP program are handled by DspHost. When the DSP program has terminated, the PC will return to the DOS prompt.

7.3 COMPILING PROGRAMS FOR DSPHOST

To compile a DspHost program, two things must happen:

1. One or more of the DspHost header files must be included into your source.

2. The DspHost library (`libdh.a`) must be linked with your program.

The DspHost header files are stored in the `\ezsharc\include` directory during installation. You can use the compiler's `-I` option to specify where to search for them. The header files have standard names such as `stdio.h` and `stdlib.h`. See section 7.5 for a complete list of header files and the functions that they prototype.

The DspHost library is stored in the `\ezsharc` directory during installation. You can use the linker's `-L` option to specify where to search for it.

The easiest way to deal with all of this is to start with the `mk.bat` file provided in the `\ezsharc\hello` directory. You can modify it so that it compiles your program source instead of `hello.c`. You should specify where you installed the EZ-KIT Lite files (e.g. `c:\ezsharc`) so that your program source can be located anywhere. It's also convenient to use an MS-DOS batch file parameter reference (%1) so that you can vary the name of the source file without changing the batch file.

Listing 7-2 Example DspHost Compilation Batch File

```
g21k -a c:\ezsharc\ezkit.ach -Ic:\ezsharc\include -o %1.21k %1.c c:\ezsharc\libdh.a
```

So instead of having to type:

```
g21k -a c:\ezsharc\ezkit.ach -Ic:\ezsharc\include -o myprog.21k myprog.c
   c:\ezsharc\libdh.a
```

you simply have to type:

```
mk myprog
```

The DspHost library (`libdh.a`) must be linked with your program *before* the standard C runtime library (`libc.a`). This is done automatically when you use `g21k` to compile and link your program, however, if you compile and then link separately with `ld21k`, you must list the DspHost library before the C runtime library in the command line. For example, separate compile and link commands look like this for the Hello program:

```
g21k -c -Ic:\ezsharc\include hello.c
```

This command compiles the source into the object module `hello.o`, and then:

```
ld21k -o hello.21k -a c:\ezsharc\ezkit.ach -Lc:\ezsharc
   c:\adi_21k\21k\lib\060_hdr.obj hello.o -ldh -lc
```

to link the object modules and libraries. Note how `libdh.a` and `libc.a` are abbreviated with the linker's `-l` option. You also could have named them explicitly.

7.4 RUNNING DSPHOST

DspHost's runtime server is called `dh21k.exe`. It has the following command syntax:

```
dh21k [-e][-s][-bN][-q] filename.21k [arg1 ...]
```

The `dh21k` program downloads the DSP executable file `filename.21k` into your board's memory and starts the processor. It then waits for I/O requests from the DSP program and services them. When the DSP program terminates, the DspHost server displays any error codes or messages and then quits.

The `-e` switch causes DspHost to display the exit code of the DSP program upon termination.

The `-s` switch causes DspHost to run the specified DSP program in standalone mode. This means that the DSP program will be downloaded and started and then DspHost will terminate, leaving the DSP program running. Since the DspHost server is not running, I/O requests are not supported.

The `-b` switch should be followed by a board number. DspHost downloads and runs DSP programs on board number 0 unless this switch is used.

The `-q` switch runs DspHost in quiet mode. This will suppress the initial banner that is displayed when DspHost starts.

Note that any other text after the name of the DSP program is considered to be a command line argument and is passed to the DSP program for normal `argc` and `argv` processing.

7.5 STANDARD HEADER FILE DESCRIPTIONS

Each of the following tables list the DspHost functions that are available for a specified standard header file. The functions are described in section 7.7.

Table 7-1 Function Listing for conio.h

Console and port I/O functions.

cgets	cscanf	kbhit
cprintf	getch	putch
cputs	getche	ungetch

Table 7-2 Function Listing for direct.h

Directory handling and creation functions.

chdir	mkdir
getcwd	rmdir

Table 7-3 Function Listing for io.h

Low-level file handling functions.

access	isatty	setmode
chmod	lseek	tell
chsize	mktemp	umask
close	open	unlink
creat	read	write
dup	read8	write8
dup2	read16	write16
eof	remove	
filelength	rename	

Table 7-4 Function Listing for process.h

Process control functions	
abort	exit

Table 7-5 Function Listing for stdio.h

Standard I/O functions			
clearerr	fopen	fwrite	rename
fclose	fprintf	fwrite8	rewind
fcloseall	fputc	fwrite16	scanf
fdopen	fputchar	getc	sprintf
feof	fputs	getchar	sscanf
ferror	fread	gets	tmpfile
fflush	fread8	getw	tmpnam
fgetc	fread16	printf	ungetc
fgetchar	freopen	putc	unlink
fgetpos	fscanf	putchar	vfprintf
fgets	fseek	puts	vprintf
fileno	fsetpos	putw	vsprintf
flushall	ftell	remove	

7.6 READING AND WRITING DOS FILES

The functions fread, fwrite, read, and write deal with 32-bit entities. A file accessed with these functions will be four times as large as the number of items specified (four bytes for every memory location).

The functions fread16, fwrite16, read16, and write16 deal with 16-bit entities (the sixteen least-significant bits of a 32-bit memory location). A file accessed with these functions will be twice as large as the number of items specified (two bytes for every memory location).

The functions fread8, fwrite8, read8, and write8 deal with 8-bit entities (the eight least-significant bits of a 32-bit memory location). A file accessed with these functions will have a size the same as the number of items specified (one byte for every memory location).

7.7 STANDARD FUNCTION DESCRIPTIONS

The DspHost library provides a large selection of standard C input/output functions. The standard functions allow existing programs that contain standard I/O function calls to be easily ported to the DSP. They are also very easy to use for DSP program development since most programmers will already know how to use these standard functions. Functions in the DspHost library are called the same as they would be in any C program. The following is a complete alphabetical list—with descriptions—of the DspHost library functions.

7 DspHost Reference

FUNCTION

access

SYNOPSIS

```
#include <io.h>
int access(char * pathname, int mode);
```

DESCRIPTION

Determines wether a specified file or directory exists and (in the case of a file) whether it can be accessed in the specified mode.

ARGUMENTS

`pathname`	File of directory path name
`mode`	00(exist) 02(write) 04(read) 06(read/write)

RETURNS

0 if the file has the given access mode or if the directory exists; -1 if the directory or file does not exist or does not have the given mode.

FUNCTION

chdir

SYNOPSIS

```
#include <direct.h>
int chdir(char * dirname);
```

DESCRIPTION

Changes the current working directory.

ARGUMENTS

`dirname`	Path name of new working directory

RETURNS

0 if the working directory is successfully changed; -1 if the specified path hane could not be found

7 DspHost Reference

FUNCTION

chmod

SYNOPSIS

```
#include <sys\types.h, sys\stat.h, io.h>
int chmod(char * filename, int pmode);
```

DESCRIPTION

Changes file permission settings.

ARGUMENTS

`filename`	Path name of existing file
`pmode`	Permission setting for file

RETURNS

0 if the permission setting is successfully changed; nonzero if the specified file could not be found

FUNCTION

chsize

SYNOPSIS

```
#include <io.h>
int chsize(int handle, long size);
```

DESCRIPTION

Changes the size of a file.

ARGUMENTS

`handle`	Handle referring to open file
`size`	New length of file in bytes

RETURNS

0 if the file size is successfully changed; -1 if not

7 DspHost Reference

FUNCTION

clearerr

SYNOPSIS

```
#include <stdio.h>
void clearerr(FILE * stream);
```

DESCRIPTION

Resets the error and end-of-file indicators for a stream.

ARGUMENTS

`stream`	Pointer to FILE structure

RETURNS

No return value

FUNCTION

close

SYNOPSIS

```
#include <io.h>
int close(int handle);
```

DESCRIPTION

Closes a file.

ARGUMENTS

`handle`	Handle referring to open file

RETURNS

0 if the file was successfully closed; -1 if the file-handle argument is invalid

7 DspHost Reference

FUNCTION

creat

SYNOPSIS

```
#include <sys\types.h, sys\stat.h, io.h>
int creat(char * filename, int pmode);
```

DESCRIPTION

Creates a new file or opens and truncates an existing file.

ARGUMENTS

`filename`	Path name of new file
`pmode`	S_IWRITE, S_IREAD, S_IREAD | S_IWRITE

RETURNS

A handle for the created file if successful; -1 if not

FUNCTION

dup

SYNOPSIS

```
#include <io.h>
int dup(int handle);
```

DESCRIPTION

Associates a second file handle with the currently open file.

ARGUMENTS

`handle`	Handle of an open file

RETURNS

A new file handle if successful; -1 if not

7 DspHost Reference

FUNCTION

eof

SYNOPSIS

```
#include <io.h>
int eof(int handle);
```

DESCRIPTION

Determines whether the end of the file has been reached.

ARGUMENTS

`handle`	Handle referring to open file

RETURNS

1 if the current position is end-of-file; 0 if not; -1 if an error has occurred

FUNCTION

exit

SYNOPSIS

```
#include <process.h or stdlib.h>
void exit(int status);
```

DESCRIPTION

Terminates the program.

ARGUMENTS

`status`	Exit status

RETURNS

No return value

7 DspHost Reference

FUNCTION

fclose

SYNOPSIS

```
#include <stdio.h>
int fclose(FILE * stream);
```

DESCRIPTION

Closes an open stream.

ARGUMENTS

`stream`	Target stream

RETURNS

0 if the stream is successfully closed; EOF if not

FUNCTION

fcloseall

SYNOPSIS

```
#include <stdio.h>
int fcloseall();
```

DESCRIPTION

Closes all open streams.

ARGUMENTS

RETURNS

The total number of streams closed; EOF if an error occurs

7 DspHost Reference

FUNCTION

fdopen

SYNOPSIS

```
#include <stdio.h>
FILE * fdopen(int handle, char * mode);
```

DESCRIPTION

Associates a stream with a file handle, allowing a file opened for low-level I/O to be buffered and formatted.

ARGUMENTS

`handle`	Handle of open file
`mode`	Access permissions: r, w, a, r+, w+, a+, t, b

RETURNS

A pointer to the open stream if successful; NULL if not

FUNCTION

feof

SYNOPSIS

```
#include <stdio.h>
int feof(FILE * stream);
```

DESCRIPTION

Determines whether the end of stream has been reached.

ARGUMENTS

`stream`	Pointer to a stream

RETURNS

A nonzero value after the first read operation that attempts to read past the end of the file; 0 if the current position is not end-of-file

7 DspHost Reference

FUNCTION

ferror

SYNOPSIS

```
#include <stdio.h>
int ferror(FILE * stream);
```

DESCRIPTION

Tests for a reading or writing error on stream.

ARGUMENTS

`stream`	Pointer to a stream

RETURNS

A nonzero value if an error occurred; 0 if not

FUNCTION

fflush

SYNOPSIS

```
#include <stdio.h>
int fflush(FILE * stream);
```

DESCRIPTION

Writes the contents of the buffer associated with the stream to the associated file.

ARGUMENTS

`stream`	Pointer to target stream

RETURNS

0 if successful or where the specified stream has no buffer or is open for reading only; EOF otherwise

7 DspHost Reference

FUNCTION

fgetc

SYNOPSIS

```
#include <stdio.h>
int fgetc(FILE * stream);
```

DESCRIPTION

Reads a character from stream.

ARGUMENTS

`stream`	Pointer to target stream

RETURNS

The character read; EOF if an error occurred or at end-of-file

FUNCTION

fgetchar

SYNOPSIS

```
#include <stdio.h>
int fgetchar();
```

DESCRIPTION

Reads a character from the standard input stream, stdin.

ARGUMENTS

RETURNS

The character read; EOF if an error occurred or at end-of-file

FUNCTION

fgetpos

SYNOPSIS

```
#include <stdio.h>
int fgetpos(FILE * stream, fpos_t pos);
```

DESCRIPTION

Gets the current value of the stream file-position indicator.

ARGUMENTS

`stream`	Pointer to target stream
`pos`	Position-indicator buffer

RETURNS

0 if successful; a nonzero value if not

FUNCTION

fgets

SYNOPSIS

```
#include <stdio.h>
char * fgets(char * string, int n, FILE * stream);
```

DESCRIPTION

Reads a string of, at most, n characters from the input stream and stores it in string.

ARGUMENTS

`string`	Storage location for data
`n`	Maximum number of characters read and stored
`stream`	Pointer to target stream

RETURNS

string if successful; NULL to indicate an error or end-of-file

7 DspHost Reference

FUNCTION

filelength

SYNOPSIS

```
#include <io.h>
long filelength(int handle);
```

DESCRIPTION

Gets the length in bytes of a file.

ARGUMENTS

`handle`	Target file handle

RETURNS

The file length in bytes if successful; -1 if not

FUNCTION

fileno

SYNOPSIS

```
#include <stdio.h>
int fileno(FILE * stream);
```

DESCRIPTION

Returns the file handle currently associated with stream.

ARGUMENTS

`stream`	Pointer to target stream

RETURNS

The file handle

7 DspHost Reference

FUNCTION

flushall

SYNOPSIS

```
#include <stdio.h>
int flushall();
```

DESCRIPTION

Writes the contents of all buffers for open output streams to their files and clears all buffers for open input streams of their contents.

ARGUMENTS

RETURNS

The number of open streams

FUNCTION

fopen

SYNOPSIS

```
#include <stdio.h>
FILE * fopen(char * filename, char * mode);
```

DESCRIPTION

Opens a specified file for stream I/O.

ARGUMENTS

`filename`	Path name of file
`mode`	Access permissions: r, r+, r+b, r+t, w, w+, w+b, w+t, a, a+, a+b, a+t

RETURNS

A pointer to the open file if successful; NULL if not

7 DspHost Reference

FUNCTION

fprintf

SYNOPSIS

```
#include <stdio.h>
int fprintf(FILE * stream, char * format [, argument,
     ...]);
```

DESCRIPTION

Formats and prints a series of characters and values to the output stream.

ARGUMENTS

`stream`	Pointer to target stream
`format`	Format control string
`[, argument, ...]`	Optional

RETURNS

The number of characters printed if successful; a negative value if not

FUNCTION

fputc

SYNOPSIS

```
#include <stdio.h>
int fputc(int c, FILE * stream);
```

DESCRIPTION

Writes a single character to the target stream.

ARGUMENTS

`c`	Character to be written
`stream`	Pointer to target stream

RETURNS

The character written if successful; EOF if not

7 DspHost Reference

FUNCTION

fputs

SYNOPSIS

```
#include <stdio.h>
int fputs(char * string, stream Output string);
```

DESCRIPTION

Copies a string to the target stream.

ARGUMENTS

`string`	FILE *
`Output string`	Pointer to target stream

RETURNS

A nonnegative value if successful; EOF if not

FUNCTION

fread

SYNOPSIS

```
#include <stdio.h>
int fread(void * buffer, int size, int count, FILE *
     stream);
```

DESCRIPTION

Reads up to count items of size bytes from the input stream and stores them in buffer.

ARGUMENTS

`buffer`	Storage location for data
`size`	Item size in bytes
`count`	Maximum number of items to be read
`stream`	Pointer to target stream

RETURNS

The number of items actually read, which may be less than count if an error occurs

7 DspHost Reference

FUNCTION

freopen

SYNOPSIS

```
#include <stdio.h>
FILE * freopen(char * filename, char * mode, FILE *
     stream);
```

DESCRIPTION

Closes the file currently associated with stream and reassigns stream to the file specified by filename.

ARGUMENTS

`filename`	Path name of new file
`mode`	Access permissions: r, r+, r+b, r+t, w, w+, w+b, w+t, a, a+, a+b, a+t
`stream`	Pointer to target stream

RETURNS

A pointer to the newly opened file if successful; NULL if not

FUNCTION

fscanf

SYNOPSIS

```
#include <stdio.h>
int fscanf(FILE * stream, char * format [, argument,
     ...]);
```

DESCRIPTION

Reads data from the current position of stream into the locations given by argument (if any).

ARGUMENTS

`stream` Pointer to target stream

`format` Format control string

`[, argument, ...]` Optional

RETURNS

The number of fields that were successfully converted and assigned, or EOF for end-of-file

7 DspHost Reference

FUNCTION

fseek

SYNOPSIS

```
#include <stdio.h>
int fseek(FILE * stream, long offset, int origin);
```

DESCRIPTION

Moves the file-position indicator associated with stream to a new location that is offset bytes from origin. It has limited use in text mode.

ARGUMENTS

`stream`	Pointer to target stream
`offset`	Number of bytes from origin
`origin`	Initial position; SEEK_CUR, SEEK_END, SEEK_SET

RETURNS

0 if successful; a nonzero value if not

FUNCTION

fsetpos

SYNOPSIS

```
#include <stdio.h>
int fsetpos(FILE * stream, fpos_t * pos);
```

DESCRIPTION

Sets the file-position indicator for stream to the value of pos.

ARGUMENTS

`stream`	Pointer to target stream
`pos`	Position-indicator storage

RETURNS

0 if successful; a nonzero value if not

7 DspHost Reference

FUNCTION

ftell

SYNOPSIS

```
#include <stdio.h>
long ftell(FILE * stream);
```

DESCRIPTION

Gets the current position of the file-position indicator associated with stream. Use with fseek.

ARGUMENTS

`stream`	Pointer to target stream

RETURNS

The current position if successful; -1L if not

FUNCTION

fwrite

SYNOPSIS

```
#include <stdio.h>
int fwrite(void * buffer, int size, int count, FILE *
     stream);
```

DESCRIPTION

Writes up to count items of length size from buffer to the output stream.

ARGUMENTS

`buffer`	Pointer to data to be written
`size`	Item size in bytes
`count`	Maximum number of items to be written
`stream`	Pointer to target stream

RETURNS

The number of full items actually written, which may be less than count if an error occurs

7 DspHost Reference

FUNCTION

getc

SYNOPSIS

```
#include <stdio.h>
int getc(FILE * stream);
```

DESCRIPTION

Reads a single character from the current stream position and increases the associated file-position indicator to point to the next character.

ARGUMENTS

`stream`	Pointer to target stream

RETURNS

The character read if successful; EOF to indicate an error or end-of-file

FUNCTION

getch

SYNOPSIS

```
#include <conio.h>
int getch();
```

DESCRIPTION

Reads without echoing a single character from the console.

ARGUMENTS

RETURNS

The character read

7 DspHost Reference

FUNCTION

getchar

SYNOPSIS

```
#include <stdio.h>
int getchar();
```

DESCRIPTION

Reads a single character from the current standard input stream, stdin, and increases the associated file-position indicator to point to the next character.

ARGUMENTS

RETURNS

The character read if successful; EOF to indicate an error or end-of-file

FUNCTION

getche

SYNOPSIS

```
#include <conio.h>
int getche();
```

DESCRIPTION

Reads a single character from the console and echoes the character.

ARGUMENTS

RETURNS

The character read

7 DspHost Reference

FUNCTION

getcwd

SYNOPSIS

```
#include <direct.h>
char * getcwd(char * buffer, int maxlen);
```

DESCRIPTION

Gets the full path name (including disk drive specification) of the current working directory and stores it at buffer.

ARGUMENTS

`buffer`	Storage location for path name
`maxlen`	Maximum length of path name

RETURNS

buffer if successful; NULL if not

FUNCTION

gets

SYNOPSIS

```
#include <stdio.h>
char * gets(char * buffer);
```

DESCRIPTION

Reads a line from the standard input stream, stdin, and stores it at buffer.

ARGUMENTS

`buffer`	Storage location for input string

RETURNS

The argument if successful; NULL if not, or if end-of-file is reached

7 DspHost Reference

FUNCTION

isatty

SYNOPSIS

```
#include <io.h>
int isatty(int handle);
```

DESCRIPTION

Determines whether handle is associated iwth a chacter device (a terminal, console, printer, or serial port).

ARGUMENTS

`handle`	Handle referring to device to be tested

RETURNS

A nonzero value if the device is a character device; 0 if not

FUNCTION

kbhit

SYNOPSIS

```
#include <conio.h>
int kbhit();
```

DESCRIPTION

Checks the console for an unread keystroke.

ARGUMENTS

RETURNS

A nonzero value if a key has been pressed; 0 if not

7 DspHost Reference

FUNCTION

lseek

SYNOPSIS

```
#include <io.h, stdio.h>
long lseek(int handle, long offset, int origin);
```

DESCRIPTION

Moves the file pointer associated with handle to a new location that is offset bytes from origin.

ARGUMENTS

`handle`	Handle referring to open file
`offset`	Number of bytes from origin
`origin`	SEEK_SET, SEEK_CUR, SEEK_END

RETURNS

The offset in bytes of the new position if successful; -1L if not

FUNCTION

mkdir

SYNOPSIS

```
#include <direct.h>
int mkdir(char * dirname);
```

DESCRIPTION

Creates a new directory with the specified directory name.

ARGUMENTS

`dirname`	Path name for new directory

RETURNS

0 if successful; -1 if not

7 DspHost Reference

FUNCTION

mktemp

SYNOPSIS

```
#include <io.h>
char * mktemp(char * template);
```

DESCRIPTION

Creates a unique file name by modifying the given file-name pattern. The argument template must be a null-terminated string with six trailing X's. These X characters are replaced with a unique collection of letters plus a period, so that there are two letters, a period, and three suffix letters in the new file name.

ARGUMENTS

`template`	File-name pattern (see description above)

RETURNS

A pointer to the modified template if successful; NULL if not

FUNCTION

open

SYNOPSIS

```
#include <sys\types.h, sys\stat.h, io.h, fcntl.h>
int open(char * filename, int oflag, int pmode);
```

DESCRIPTION

Opens the file specified by filename and prepares the file for subsequent reading or writing, as define by oflag and, optionally, pmode.

ARGUMENTS

`filename`	File path name
`oflag`	O_APPEND, O_BINARY, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_TEXT, O_TRUNC, O_WRONLY
`pmode`	S_IWRITE, S_IREAD, S_IREAD | S_IWRITE

RETURNS

A handle for the opened file if successful; -1 if not

7 DspHost Reference

FUNCTION

printf

SYNOPSIS

```
#include <stdio.h>
int printf(char * format [, argument, ...]);
```

DESCRIPTION

Formats and prints a series of characters and values to the standard output stream, stdout

ARGUMENTS

`format`	Format control string
`[, argument, ...]`	Optional arguments, variables of the type specified in the format-control string

RETURNS

The number of characters printed if successful; a negative value if not

FUNCTION

putc

SYNOPSIS

```
#include <stdio.h>
int putc(int c, FILE * stream);
```

DESCRIPTION

Writes the single character c to the output stream at the current position.

ARGUMENTS

`c`	Character to be written
`stream`	Pointer to target stream

RETURNS

The character written if successful; EOF may indicate failure

7 DspHost Reference

FUNCTION

putch

SYNOPSIS

```
#include <conio.h>
int putch(int c);
```

DESCRIPTION

Writes the character c directly to the console.

ARGUMENTS

c	Character to be written

RETURNS

c if successful; EOF if not

FUNCTION

putchar

SYNOPSIS

```
#include <stdio.h>
int putchar(int c);
```

DESCRIPTION

Writes the single character c to the standard output stream, stdout.

ARGUMENTS

c	Character to be written

RETURNS

The character written if successful; EOF may indicate failure

7 DspHost Reference

FUNCTION

puts

SYNOPSIS

```
#include <stdio.h>
int puts(char * string);
```

DESCRIPTION

Writes string to the standard output stream, stdout, replacing the string's terminating null character ('\0') with a newline character ('\n') in the output stream.

ARGUMENTS

`string`	String to be output

RETURNS

A nonnegative value if successful; EOF if not

FUNCTION

read

SYNOPSIS

```
#include <io.h>
int read(int handle, void * buffer, unsigned int count);
```

DESCRIPTION

Attempts to read count bytes into buffer from the file associated with handle.

ARGUMENTS

`handle`	Handle referring to open file
`buffer`	Storage location for data
`count`	Maximum number of bytes

RETURNS

The number of bytes read if successful; -1 if not

7 DspHost Reference

FUNCTION

remove

SYNOPSIS

```
#include <stdio.h or io.h>
int remove(char * filename);
```

DESCRIPTION

Deletes the file specified by filename.

ARGUMENTS

`filename`	Path name of file to be removed

RETURNS

0 if successful; a nonzero value if not

FUNCTION

rename

SYNOPSIS

```
#include <stdio.h or io.h>
int rename(char * oldname, char * newname);
```

DESCRIPTION

Renames the file or directory specified by oldname to the name given by newname.

ARGUMENTS

`oldname`	Pointer to old name
`newname`	Pointer to new name

RETURNS

0 if successful; a nonzero value if not

7 DspHost Reference

FUNCTION

rewind

SYNOPSIS

```
#include <stdio.h>
void rewind(FILE * stream);
```

DESCRIPTION

Repositions the file pointer associated with stream to the beginning of the file.

ARGUMENTS

`stream`	Pointer to target stream

RETURNS

No return value

FUNCTION

rmdir

SYNOPSIS

```
#include <direct.h>
int rmdir(char * dirname);
```

DESCRIPTION

Deletes the directory specified by dirname.

ARGUMENTS

`dirname`	Path name of directory to be removed

RETURNS

0 if successful; a nonzero value if not

FUNCTION

rmtmp

SYNOPSIS

```
#include <stdio.h>
int rmtmp();
```

DESCRIPTION

Cleans up all the temporary files in the current directory. The function removes only those files created by tmpfile and should be used only in the same directory in which the temporary files were created.

ARGUMENTS

RETURNS

The number of temporary files closed and deleted

FUNCTION

scanf

SYNOPSIS

```
#include <stdio.h>
int scanf(char * format [, argument, ...]);
```

DESCRIPTION

Reads data from the standard input stream, stdin, into the locations given by an argument.

ARGUMENTS

`format`	Format control string
`[, argument, ...]`	Optional arguments; pointers to variables of the type specified in the format-control string

RETURNS

The number of fields successfully converted and assigned; EOF if input failure occurs before any conversion

7 DspHost Reference

FUNCTION

tell

SYNOPSIS

```
#include <io.h>
long tell(int handle);
```

DESCRIPTION

Gets the current position of any file pointer associated with handle.

ARGUMENTS

`handle`	Handle referring to open file

RETURNS

The current position of the file pointer (if any) associated with handle if successful; -1L if not

FUNCTION

tmpfile

SYNOPSIS

```
#include <stdio.h>
FILE * tmpfile();
```

DESCRIPTION

Creates a temporary file and returns a stream pointer to that file.

ARGUMENTS

RETURNS

A stream pointer if successful; a null pointer if not

7 DspHost Reference

FUNCTION

tmpnam

SYNOPSIS

```
#include <stdio.h>
char * tmpnam(char * string);
```

DESCRIPTION

Generates a string that can be used as the name of a temporary file.

ARGUMENTS

`string`	Pointer to temporary name

RETURNS

A pointer to the name generated if successful; NULL if not

FUNCTION

umask

SYNOPSIS

```
#include <sys\types.h, sys\stat.h, io.h>
int umask(int pmode);
```

DESCRIPTION

Sets the file-permission mask of the current process to the mode specified by pmode. The file-permission mask is used to modify the permission setting of new files created by create, open, or sopen.

ARGUMENTS

pmode	S_IWRITE, S_IREAD

RETURNS

The previous value of pmode

7 DspHost Reference

FUNCTION

ungetc

SYNOPSIS

```
#include <stdio.h>
int ungetc(int c, FILE * stream);
```

DESCRIPTION

Pushes the character c back onto the input stream and clears the end-of-file indicator. A subsequent read operation on the stream begins with c.

ARGUMENTS

`c`	Character to be pushed
`stream`	Pointer to target stream

RETURNS

The character argument c if successful; EOF if not

FUNCTION

ungetch

SYNOPSIS

```
#include <conio.h>
int ungetch(int c);
```

DESCRIPTION

Pushes the character c back to the console, causing c to be the next character read by getch or getche.

ARGUMENTS

c	Character to be pushed

RETURNS

The character argument c if successful; EOF if not

7 DspHost Reference

FUNCTION

unlink

SYNOPSIS

```
#include <io.h or stdio.h>
int unlink(char * filename);
```

DESCRIPTION

Deletes the file specified by filename.

ARGUMENTS

`filename`	Path name of file to be removed

RETURNS

0 if successful; -1 if not

FUNCTION

write

SYNOPSIS

```
#include <io.h>
int write(int handle, void * buffer, unsigned int count);
```

DESCRIPTION

Writes count bytes from buffer into the file associated with handle. The write operation begins at the current position if the file. If the file is open for appending, the operation begins at the current end of file.

ARGUMENTS

`handle`	Handle referring to open file
`buffer`	Data to be written
`count`	Number of bytes

RETURNS

The number of bytes actually written if successful; -1 if not

7 DspHost Reference

Schematics 8

8.1 OVERVIEW

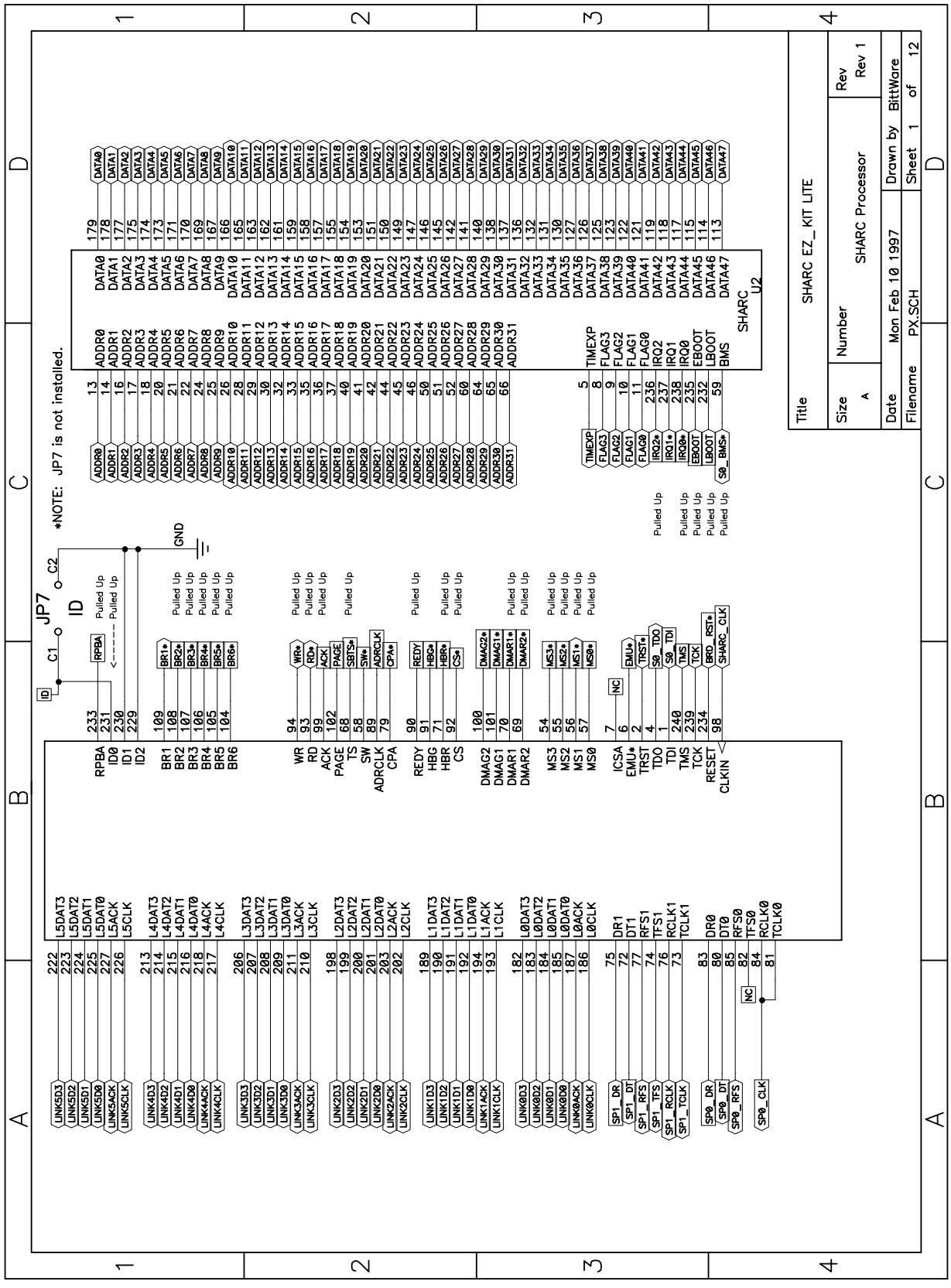
This chapter provides the design information that was used to create the SHARC EZ-KIT Lite board. The schematics are in section 8.2. The logic equations and state diagram for the 16V8 programmable logic device (U3) are listed starting on page 8-14.

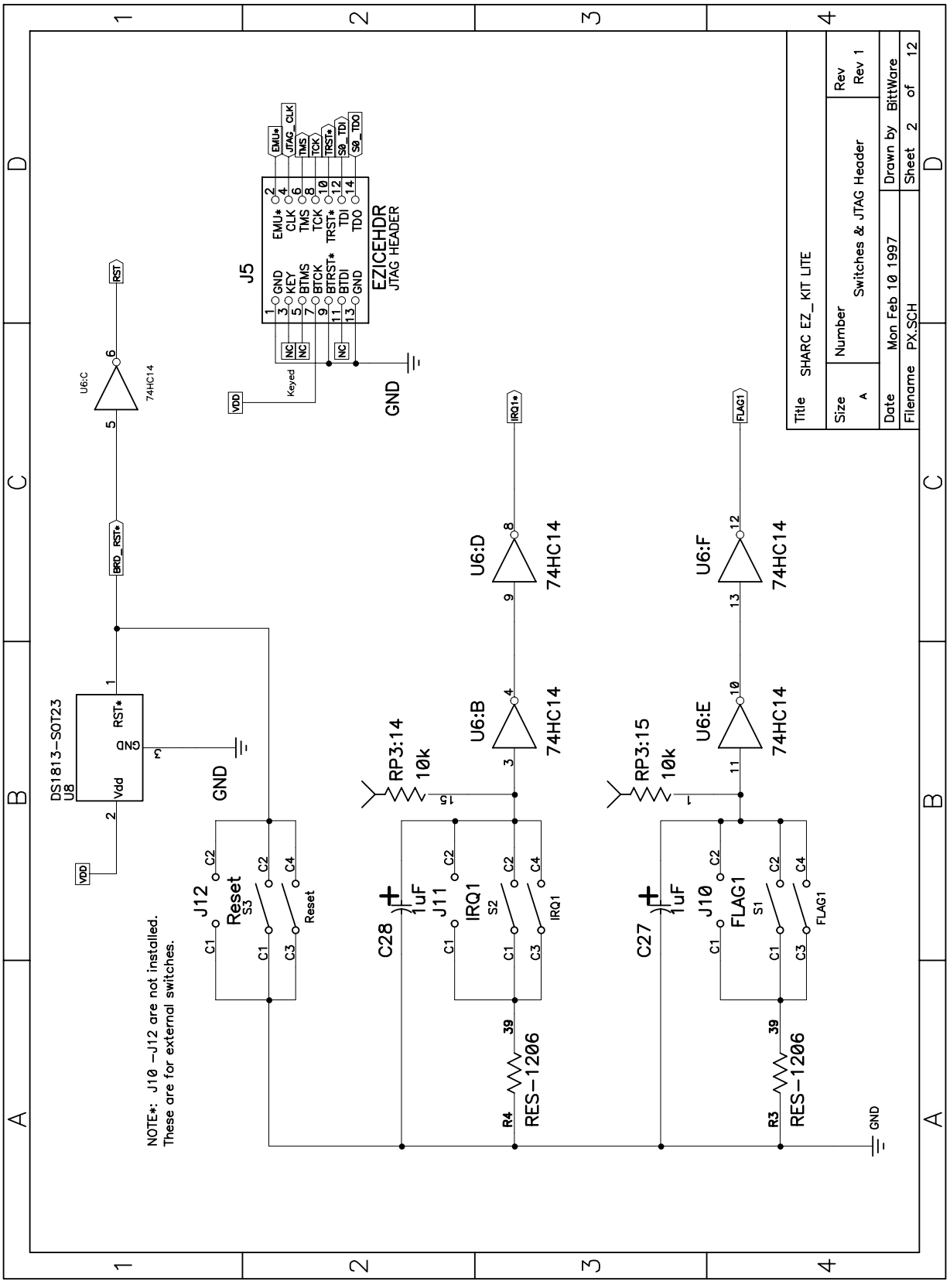
8.2 BOARD SCHEMATICS

The following pages provide complete electrical schematics for the SHARC EZ-KIT Lite Rev. 1 board. You can use the following table to quickly find a page that interests you.

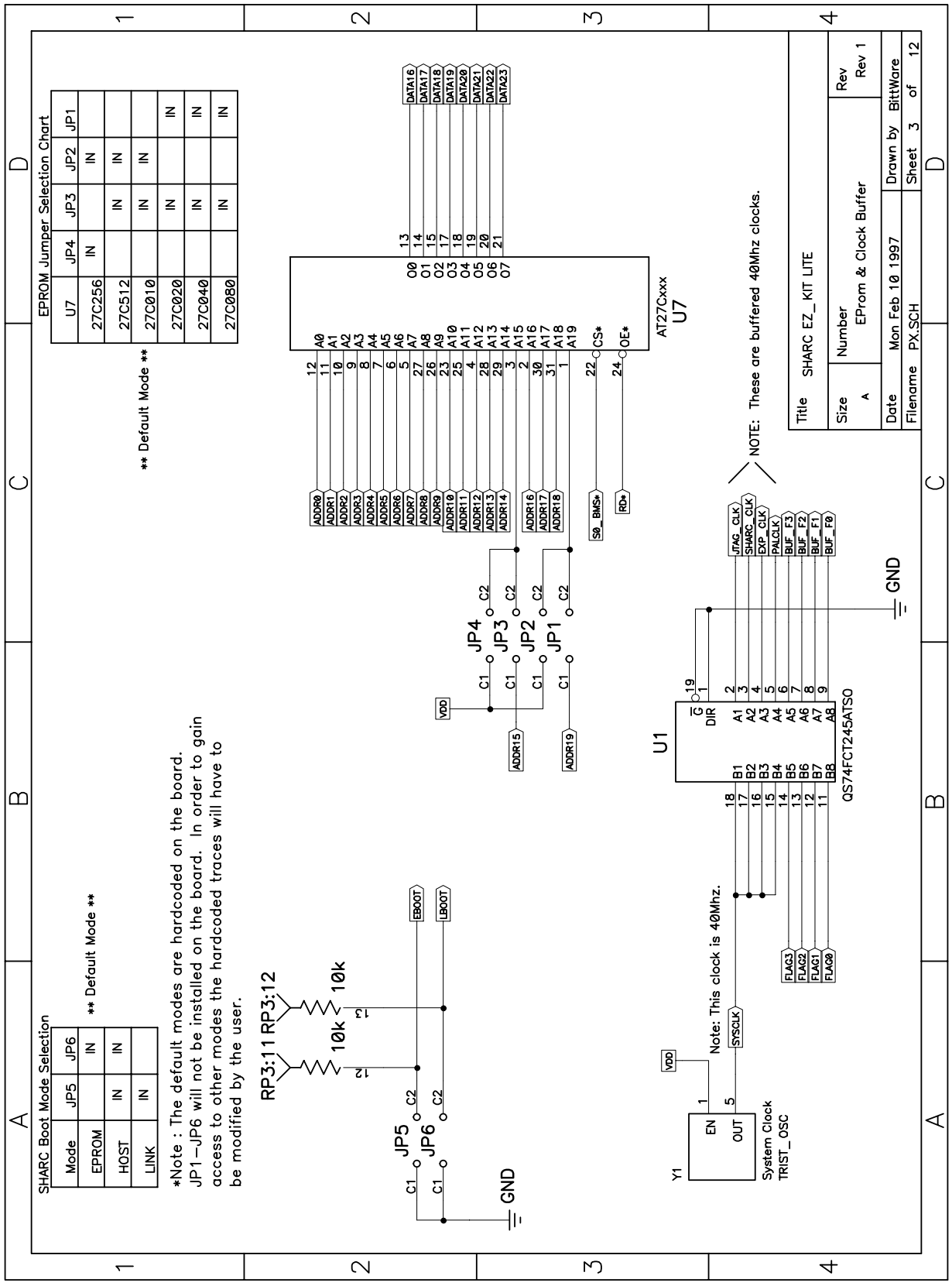
Table 8-1 SHARC EZ-KIT Lite Schematic Contents

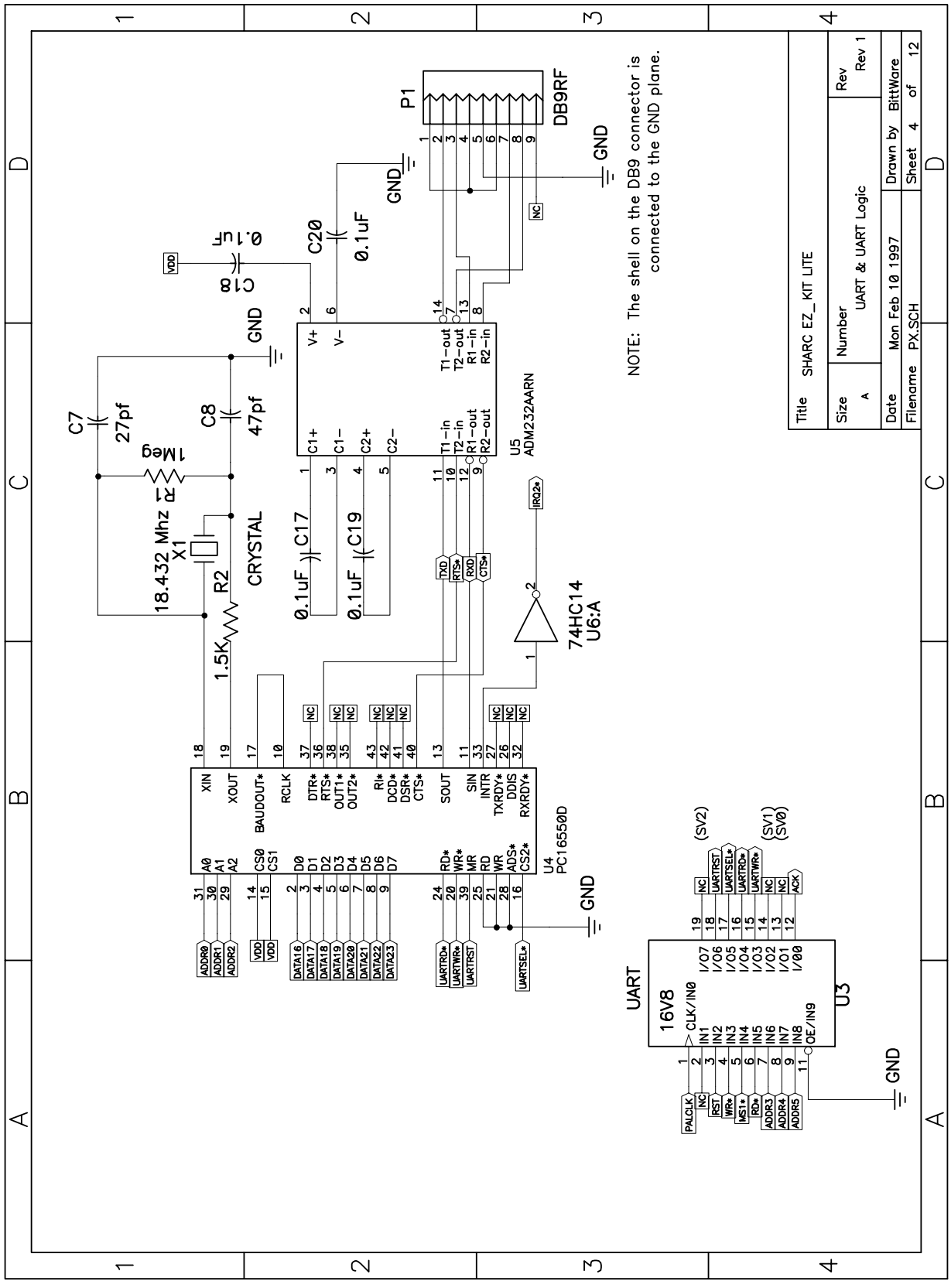
8 Schematics



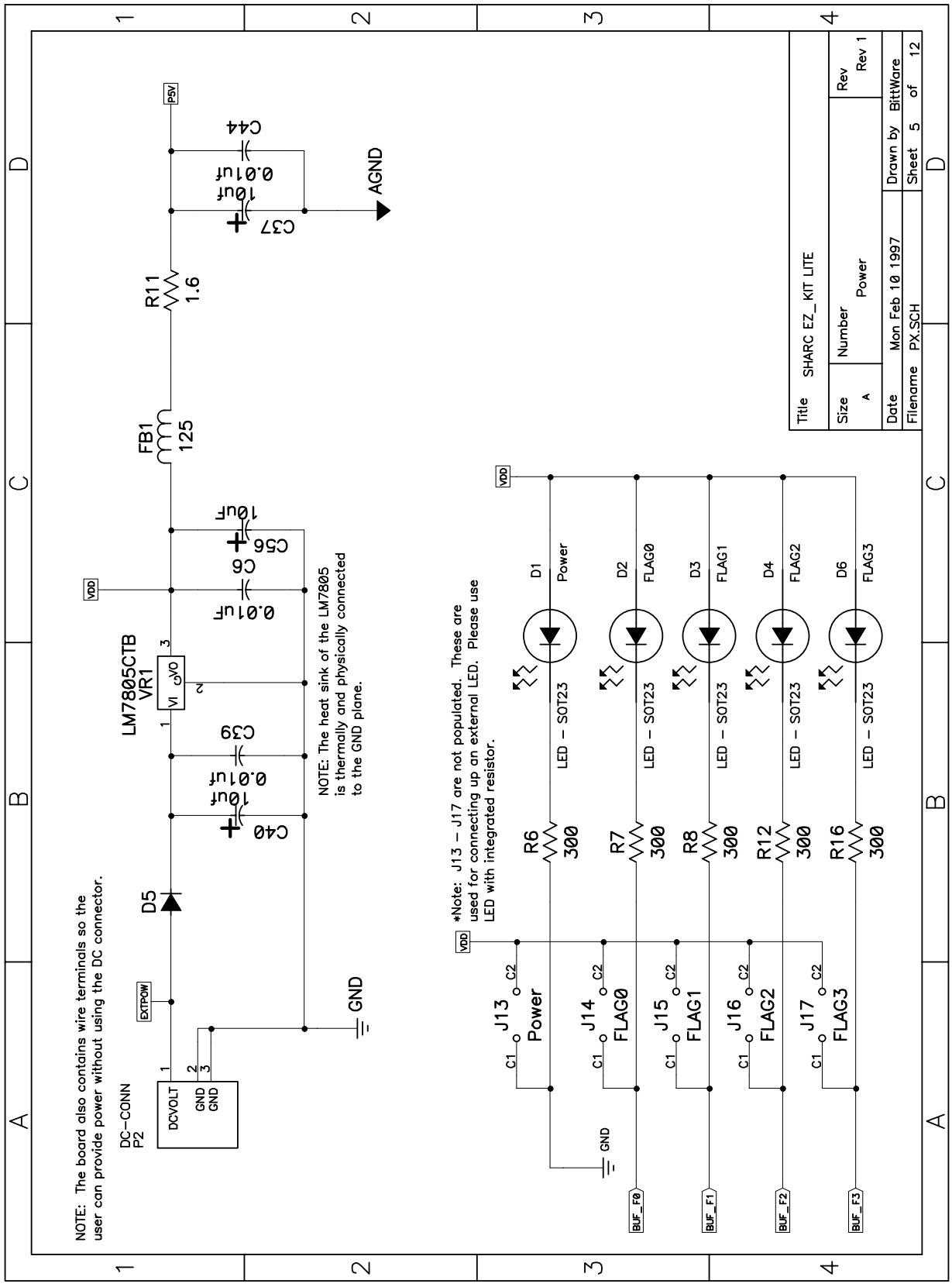

J5
EZICEHDR
JTAG HEADER
EMU*
CLK
TMS
TCK
TRST*
TDI
TDO
GND
KEY
BTMS
BTCK
BTRST*
BTDI
Keyed
NC
VDD
U6:C
74HC14
RST
BRD_RST*
DS1813-SOT23
U8
RST*
Vdd
J12
Reset
S3
J11
IRQ1
S2
J10
FLAG1
S1
C28
C27
1uF
RP3:14
RP3:15
10k
U6:B
U6:D
U6:E
U6:F
IRQ1*
FLAG1
R4
R3
RES-1206
39
NOTE*: J10 -J12 are not installed.
These are for external switches.
Title SHARC EZ_ KIT LITE
Size A
Number Switches & JTAG Header
Rev Rev 1
Date Mon Feb 10 1997
Drawn by BittWare
Filename PX.SCH
Sheet 2 of 12

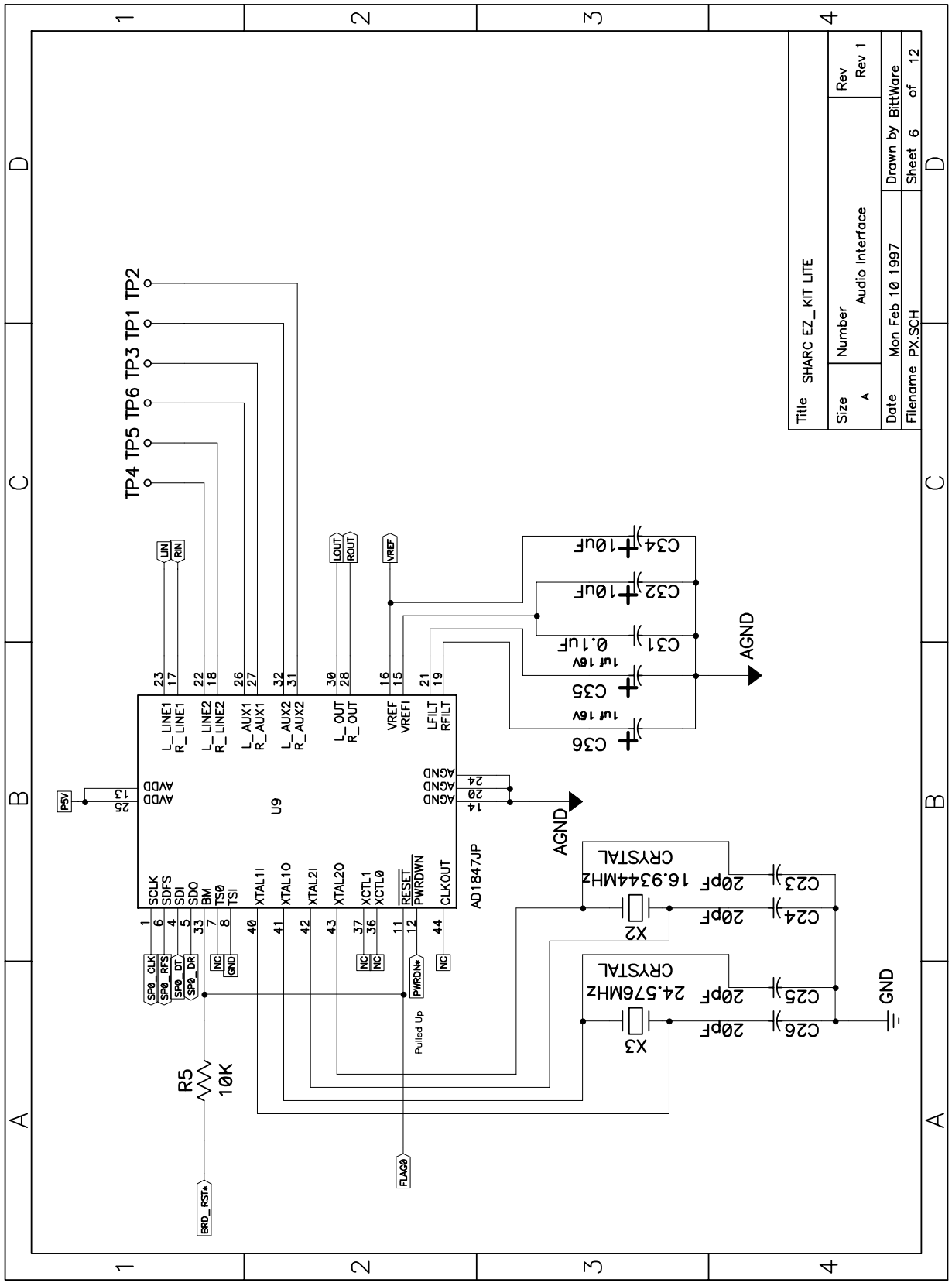
8 Schematics



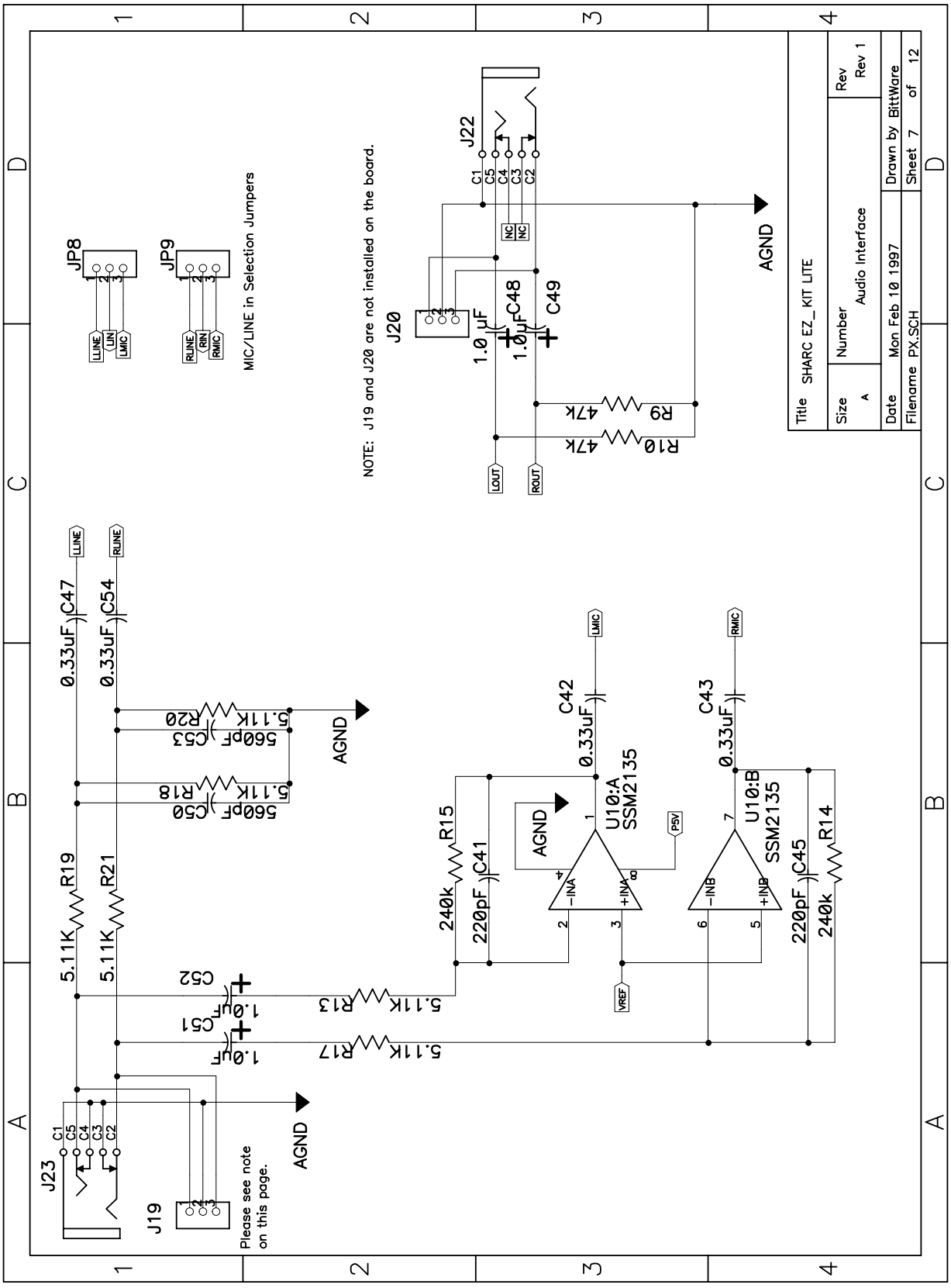

P1
DB9RF
GND
NOTE: The shell on the DB9 connector is connected to the GND plane.
C7 27pf
C8 47pf
1Meg R1
18.432 Mhz X1
CRYSTAL
R2 1.5K
C17 0.1uF
C19 0.1uF
C20 0.1uF
C18 0.1uF
VDD
U5 ADM232AARN
U4 PC16550D
74HC14 U6:A
IRQ2*
TXD
RTS*
RXD
CTS*
UART
16V8
U3
(SV2)
(SV1)
(SV0)
Title SHARC EZ_ KIT LITE
Size A
Number UART & UART Logic
Rev Rev 1
Date Mon Feb 10 1997
Drawn by BittWare
Filename PX.SCH
Sheet 4 of 12

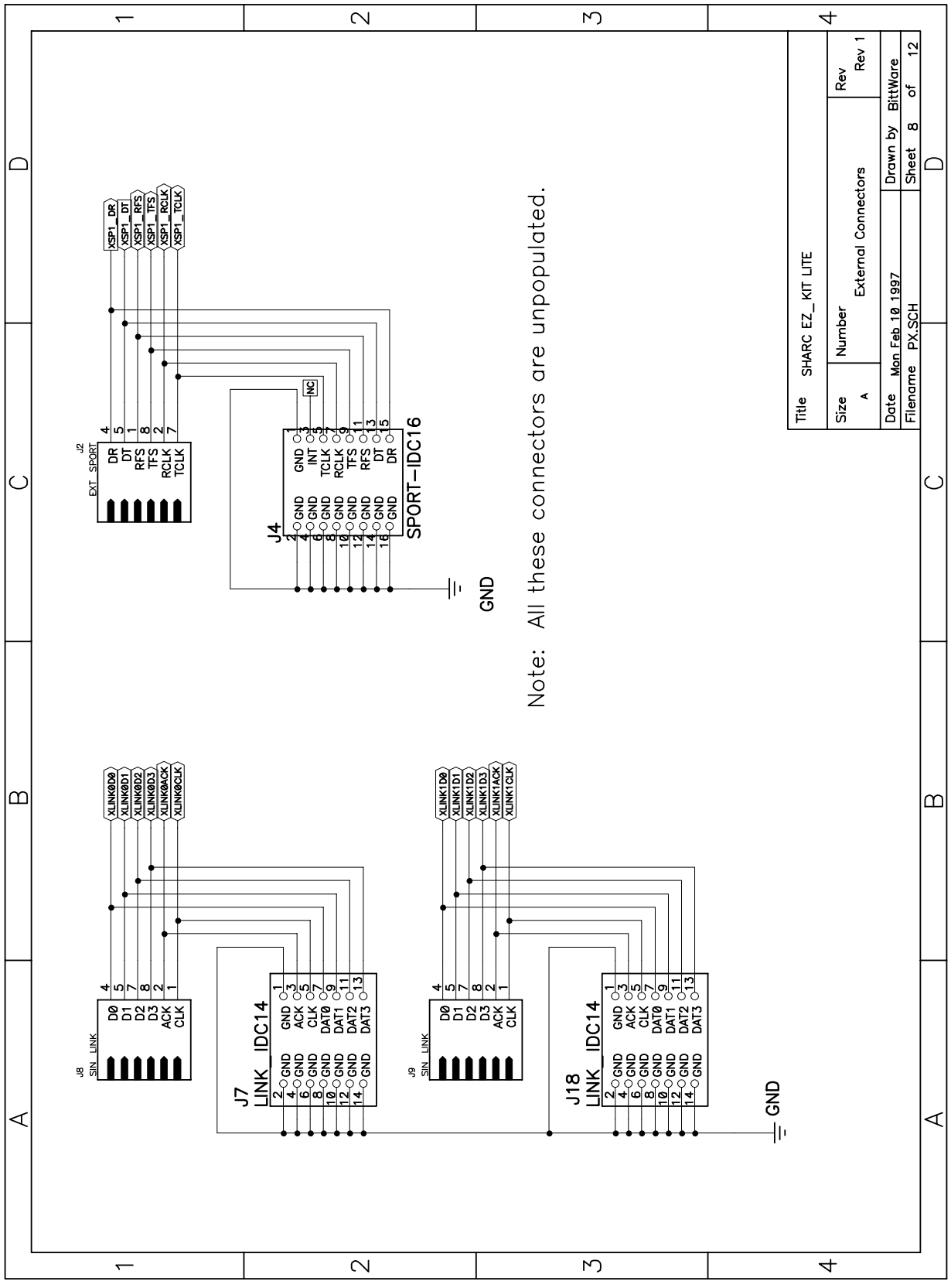
8 Schematics



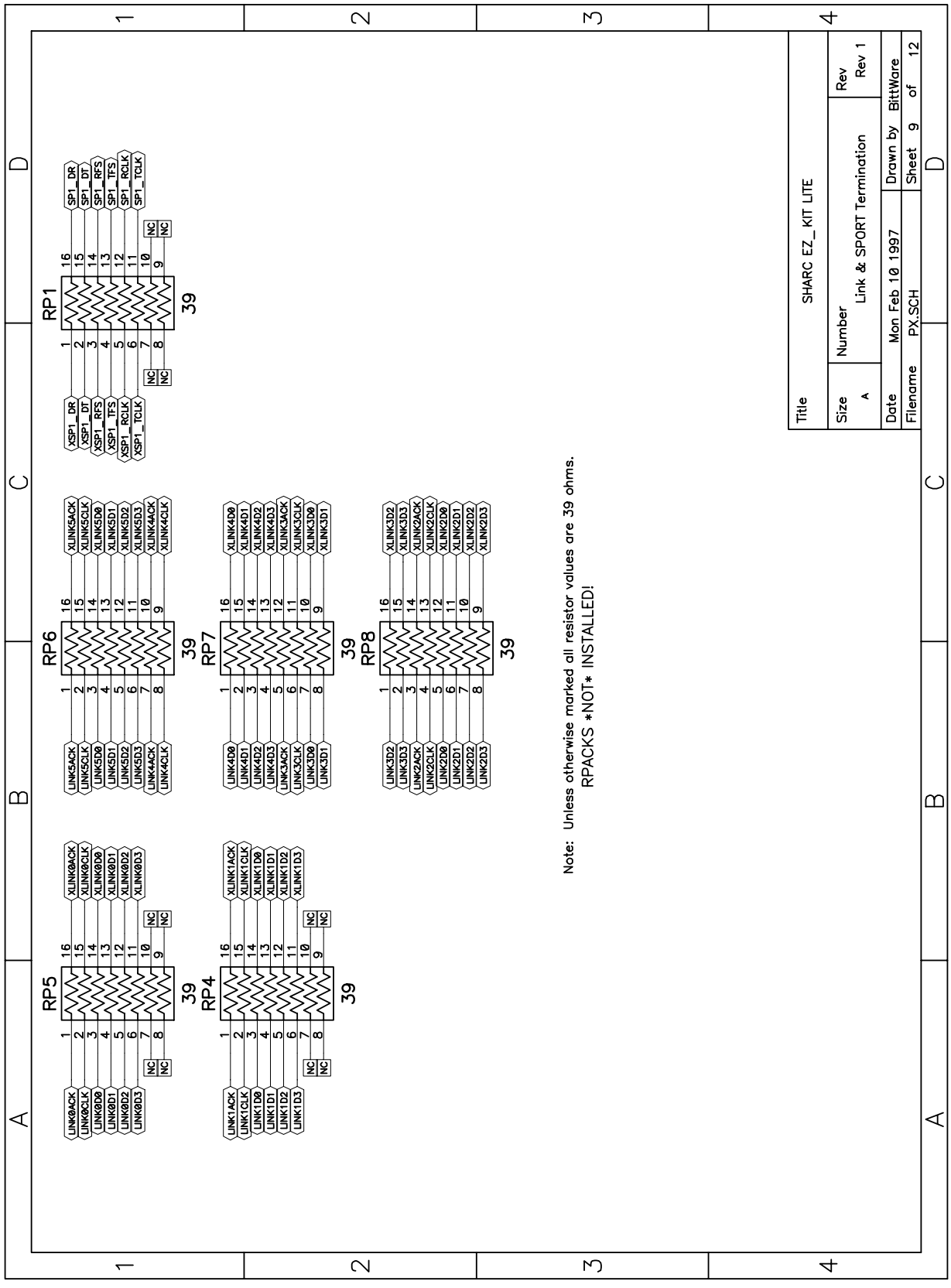

Title SHARC EZ_ KIT LITE
Size A
Number Audio Interface
Rev Rev 1
Date Mon Feb 10 1997
Drawn by BittWare
Filename PX.SCH
Sheet 6 of 12
TP4 TP5 TP6 TP3 TP1 TP2
U9
AD1847JP
L_ LINE1
R_ LINE1
L_ LINE2
R_ LINE2
L_ AUX1
R_ AUX1
L_ AUX2
R_ AUX2
L_ OUT
R_ OUT
VREF
VREFI
LFILT
RFILT
AVDD
AGND
SCLK
SDFS
SDI
SDO
BM
TS0
TSI
XTAL1I
XTAL1O
XTAL2I
XTAL2O
XCTL1
XCTL0
RESET
PWRDWN
CLKOUT
LIN
RIN
LOUT
ROUT
VREF
PSV
SPO_CLK
SPO_RFS
SPO_DT
SPO_DR
NC
GND
PWRDN#
FLAG0
BRD_RST#
Pulled Up
R5
10K
C34 10uF
C32 10uF
C31 0.1uF
C35 1uf 16V
C36 1uf 16V
AGND
X2
16.9344MHz
CRYSTAL
C23 20pF
C24 20pF
X3
24.576MHz
CRYSTAL
C25 20pF
C26 20pF
GND

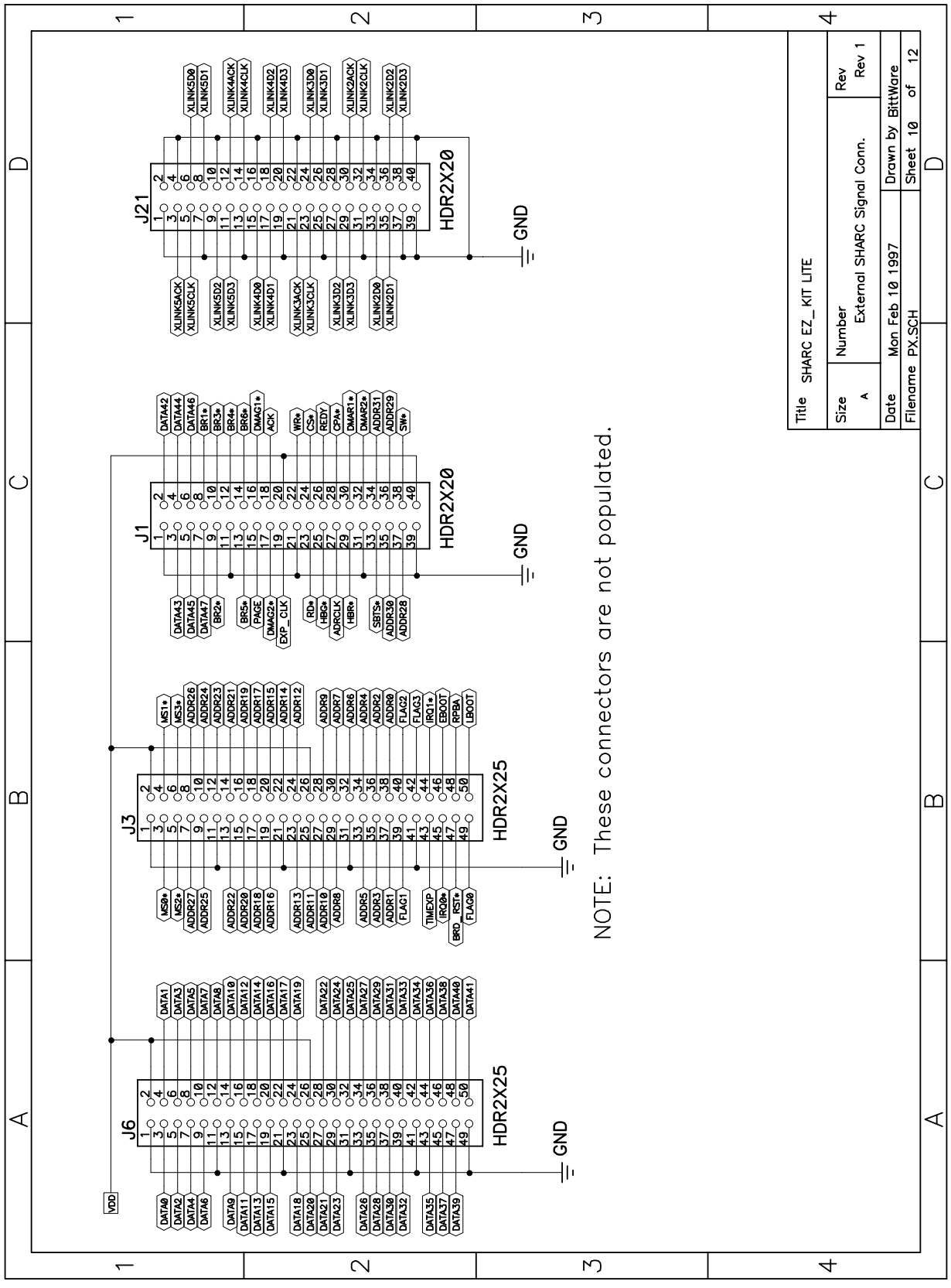
8 Schematics



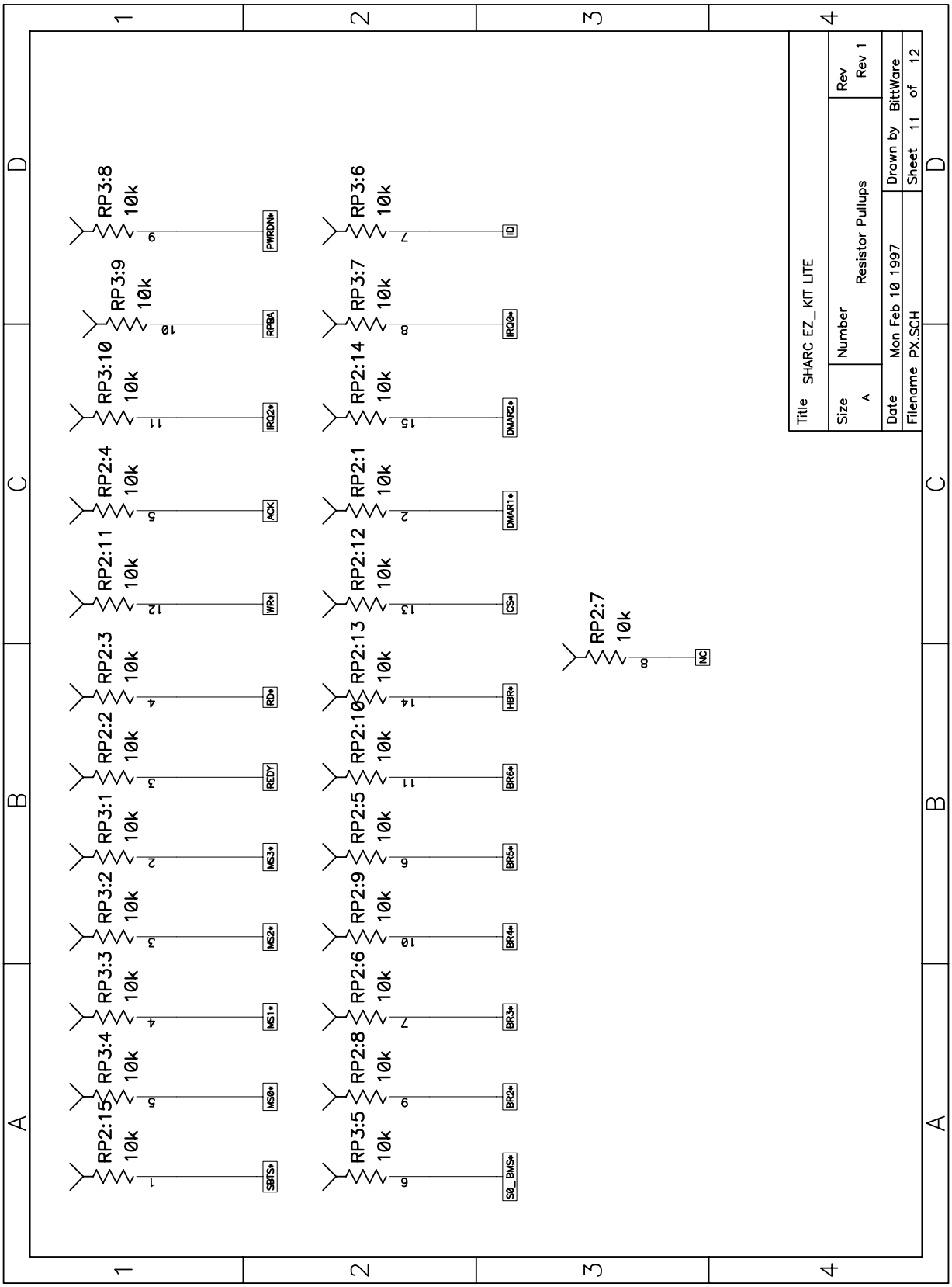

XSP1_DR
XSP1_DT
XSP1_RFS
XSP1_TFS
XSP1_RCLK
XSP1_TCLK
J2
EXT SPORT
DR
DT
RFS
TFS
RCLK
TCLK
J4
SPORT-IDC16
GND
INT
NC
DR
DT
RFS
TFS
RCLK
TCLK
GND
Note: All these connectors are unpopulated.
XLINK0D0
XLINK0D1
XLINK0D2
XLINK0D3
XLINK0ACK
XLINK0CLK
J8
SIN_LINK
D0
D1
D2
D3
ACK
CLK
J7
LINK_IDC14
GND
ACK
CLK
DAT0
DAT1
DAT2
DAT3
XLINK1D0
XLINK1D1
XLINK1D2
XLINK1D3
XLINK1ACK
XLINK1CLK
J9
J18
Title SHARC EZ_KIT LITE
Size A
Number External Connectors
Rev Rev 1
Date Mon Feb 10 1997
Drawn by BittWare
Filename PX.SCH
Sheet 8 of 12

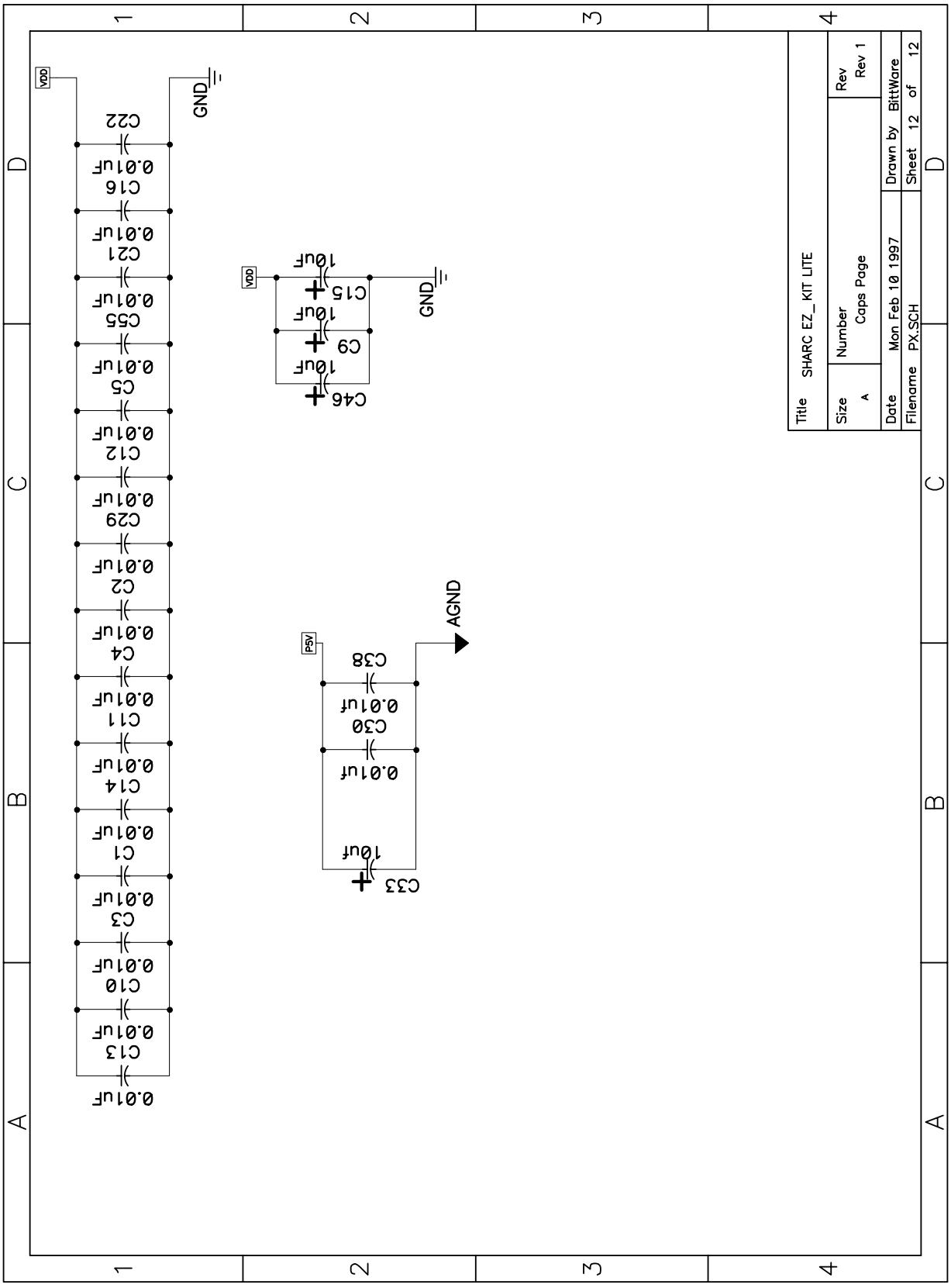
8 Schematics




J21
HDR2X20
J1
HDR2X20
J3
HDR2X25
J6
HDR2X25
GND
VDD
NOTE: These connectors are not populated.
Title SHARC EZ_KIT LITE
Size A
Number External SHARC Signal Conn.
Rev Rev 1
Date Mon Feb 10 1997
Drawn by BittWare
Filename PX.SCH
Sheet 10 of 12

8 Schematics




VDD
GND
C22
0.01uF
C16
0.01uF
C21
0.01uF
C55
0.01uF
C5
0.01uF
C12
0.01uF
C29
0.01uF
C2
0.01uF
C4
0.01uF
C11
0.01uF
C14
0.01uF
C1
0.01uF
C3
0.01uF
C10
0.01uF
C13
0.01uF
C46
10uF
C9
10uF
C15
10uF
P5V
AGND
C38
0.01uf
C30
0.01uf
C33
10uf
Title SHARC EZ_ KIT LITE
Size A
Number Caps Page
Rev Rev 1
Date Mon Feb 10 1997
Drawn by BittWare
Filename PX.SCH
Sheet 12 of 12

8 Schematics

8.3 PAL EQUATIONS

The following equations are used to program the 16v8 PAL device U3.

```
"  PSTATE.ABL
"  ABEL code created by Visual Software Solution's StateCAD Version 3.00
"  Thu Feb 06 09:23:40 1997

MODULE PSTATE

DECLARATIONS
"clock name
    CLK PIN 1;

"Input variables
    RD PIN 6;
    RESET PIN 3;
    SEL PIN 5;
    WR PIN 4;
    ROE PIN 11;

"Output variables
    ACK PIN 12 ISTYPE 'com';

"Logic variables
    UARTRST PIN 18 ISTYPE 'com';

"State variables
    SV0  PIN 13 ISTYPE 'reg';
    SV1  PIN 14 ISTYPE 'reg';
    SV2  PIN 19 ISTYPE 'reg';
    URD  PIN 16 ISTYPE 'reg';
    USEL  PIN 17 ISTYPE 'reg';
    UWR  PIN 15 ISTYPE 'reg';

"Vectors
DECLARATIONS
    SV=[
            USEL,
            URD,
            UWR,
            SV2,
            SV1,
            SV0
    ];

"State Register assignment
DECLARATIONS
    sreg=[ SV0,SV1,SV2,URD,USEL,UWR];

EQUATIONS
    sreg.clk=CLK;
    !sreg.oe = ROE;
    ACK.oe = !SEL;
```

```
DECLARATIONS
    STATE0=[1, 1, 1, 1, 1, 1];
    STATE1=[0, 0, 0, 1, 0, 1];
    STATE2=[1, 0, 0, 1, 0, 1];
    STATE3=[0, 0, 0, 0, 0, 1];
    STATE4=[1, 0, 0, 0, 0, 1];
    STATE5=[0, 0, 0, 1, 0, 0];
    STATE6=[1, 0, 0, 1, 0, 0];
    STATE7=[0, 1, 0, 1, 0, 0];
    STATE8=[0, 1, 0, 0, 0, 1];
    STATE9=[1, 1, 0, 0, 0, 1];
    STATE10=[0, 0, 1, 0, 0, 1];
    STATE11=[1, 0, 1, 0, 0, 1];
    STATE12=[0, 1, 1, 0, 0, 1];
    STATE13=[1, 1, 1, 0, 0, 1];
    STATE14=[1, 1, 0, 1, 0, 0];
    STATE15=[0, 0, 1, 1, 0, 0];
    STATE16=[1, 0, 1, 1, 0, 0];
    STATE17=[1, 1, 0, 1, 0, 1];
    STATE18=[0, 0, 1, 1, 0, 1];
    STATE19=[0, 1, 0, 1, 0, 1];

EQUATIONS
    WHEN (!(
              (sreg.FB==STATE0)
            # (sreg.FB==STATE1)
            # (sreg.FB==STATE2)
            # (sreg.FB==STATE3)
            # (sreg.FB==STATE4)
            # (sreg.FB==STATE5)
            # (sreg.FB==STATE6)
            # (sreg.FB==STATE7)
            # (sreg.FB==STATE8)
            # (sreg.FB==STATE9)
            # (sreg.FB==STATE10)
            # (sreg.FB==STATE11)
            # (sreg.FB==STATE12)
            # (sreg.FB==STATE13)
            # (sreg.FB==STATE14)
            # (sreg.FB==STATE15)
            # (sreg.FB==STATE16)
            # (sreg.FB==STATE17)
            # (sreg.FB==STATE18)
            # (sreg.FB==STATE19)
    ) &
            ( RESET )) THEN {
            [sreg] := [STATE0];
    }
```

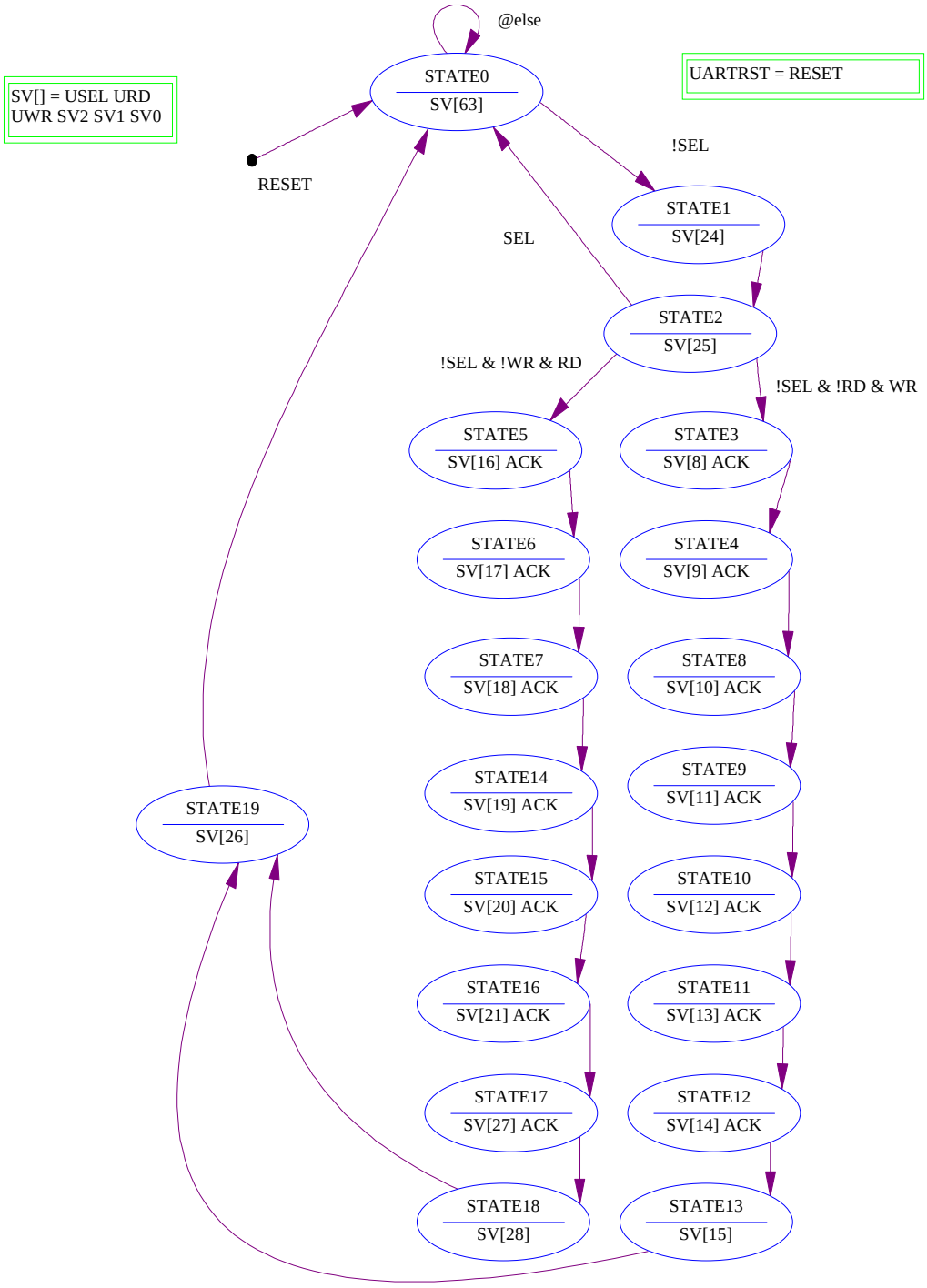
8 Schematics

```
    WHEN (!(
              (sreg.FB==STATE0)
            # (sreg.FB==STATE1)
            # (sreg.FB==STATE2)
            # (sreg.FB==STATE3)
            # (sreg.FB==STATE4)
            # (sreg.FB==STATE5)
            # (sreg.FB==STATE6)
            # (sreg.FB==STATE7)
            # (sreg.FB==STATE8)
            # (sreg.FB==STATE9)
            # (sreg.FB==STATE10)
            # (sreg.FB==STATE11)
            # (sreg.FB==STATE12)
            # (sreg.FB==STATE13)
            # (sreg.FB==STATE14)
            # (sreg.FB==STATE15)
            # (sreg.FB==STATE16)
            # (sreg.FB==STATE17)
            # (sreg.FB==STATE18)
            # (sreg.FB==STATE19)
    ) &
            !( RESET )) THEN {
            [sreg] := [STATE0];
    }

state_diagram sreg;

state STATE0:
    ACK=1;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !SEL ) THEN STATE1;
            IF ( SEL ) THEN STATE0;
state STATE1:
    ACK=1;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE2;
state STATE2:
    ACK=1;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !SEL & RD & WR  #  !SEL & !WR & !RD ) THEN STATE2;
            IF ( SEL ) THEN STATE0;
            IF ( !SEL & !RD & WR ) THEN STATE3;
            IF ( !SEL & !WR & RD ) THEN STATE5;
state STATE3:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE4;
```

```
state STATE4:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE8;
state STATE5:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE6;
state STATE6:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE7;
state STATE7:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE14;
state STATE8:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE9;
state STATE9:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE10;
state STATE10:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE11;
state STATE11:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE12;
state STATE12:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE13;
state STATE13:
    ACK=1;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE19;
```

```
state STATE14:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE15;
state STATE15:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE16;
state STATE16:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE17;
state STATE17:
    ACK=0;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE18;
state STATE18:
    ACK=1;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE19;
state STATE19:
    ACK=1;
    IF ( RESET ) THEN STATE0;
    ELSE
            IF ( !0) THEN STATE0;

"Logic Equations
EQUATIONS
UARTRST =  RESET ;
END PSTATE
```

This is the state diagram for PAL U3.



Page A0

8 Schematics