StrongARM** EBSA-285 Evaluation Board

Reference Manual

October 1998

Order Number: 278136-001

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The EBSA-285 may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at http://www.intel.com.

Copyright © Intel Corporation, 1998

*Third-party brands and names are the property of their respective owners.

**ARM and StrongARM are trademarks of Advanced RISC Machines, Ltd.

Contents

Figures

Tables

Introduction 1

The EBSA-285 is supplied as a plug-in card. This chapter provides a physical description of the card and then describes:

- How to unpack the card and give it a visual inspection
- The different modes that the card can be used in
- How to configure the card to suit your application
- How to power up the card for the first time
- How to connect the card to a host system and run its onboard diagnostics

The EBSA-285 is an evaluation board for the SA-110 microprocessor and 21285 Core Logic controller. It is designed to:

- Provide a software test and debug environment
- Allow benchmarking of prototype software algorithms
- Act as a reference platform for operating system ports
- Demonstrate the performance of the 21285's PCI interface, memory controller and internal functional blocks
- Provide a building block that can be used to build software-accurate models of target applications
- Act as a proven hardware design that can be modified for use in other applications
- Allow all the major features of the 21285 to be demonstrated and tested

This document is a single point-of-reference both for configuring and using the board and for engineers wishing to copy parts of its design. As such, it has the following scope:

- Functional specification
- Theory of operation (to be read in conjunction with the circuit schematics)
- Configuration guide (memory options, speed options, jumper and link options)
- Programmers' guide (memory maps, boot process, references to programmable I/O devices on the board)

This document aims not to duplicate material that can be found elsewhere. Specifically, it does not duplicate material that can be found in vendor data sheets for components used in the design, nor does it document the ARM** software development environment.

1.1 How to Use This Document

All readers should turn to Chapter 1 for information about how to connect and power on the board, how to verify that it is working correctly, and how to connect it to a terminal or host system.

All readers are advised to read Chapter 2 to get an understanding of the overall functionality of the board. Subsequent chapters assume a familiarity with the material in that chapter.

Thereafter, software engineers will probably want to refer to the following chapters:

- Chapter 3, a guide to the memory map of the board and the address decoding of all I/O devices.
- Chapter 4, a guide to configuration of the memory and other devices on the board.
- Chapter 5, a brief introduction to the software development environment.
- Chapter 6, describes the onboard software, including the power-on sequence of the board, and the power-on diagnostics.
- Chapter 7, describes the Flash Management Utility that is provided with the EBSA-285.

Hardware engineers will probably want to refer to the following chapters:

- Chapter 8, a detailed technical description of card hardware, including the theory of operation.

A number of appendixes provide general reference material:

- Appendix A, describes all of the link and jumper options present on the card, and all of the cables that may be required for connection to the card.
- Appendix B, describes the machine-readable design databases for the EBSA-285 hardware and software.

1.2 Notation

All numbers are shown in decimal unless otherwise stated.

All hexadecimal numbers have an 0x prefix. 32-bit hex values have dots for ease of reading. Examples are: 0xfe0b.3004, 0xfb.

All binary number have an 0b prefix; long numbers include dots for ease of reading. Examples are: 0b00, 0b0000.0000.1010.0000.

This document refers to an 8-bit data unit as a byte, a 16-bit data unit as a half-word and a 32-bit data unit as a longword.[1]

Electrical signal names are shown thus: cpu_wait_l. An _l at the end of a signal name indicates that the signal is asserted (active) when it is low (close to 0V).

Displayed messages are printed in 9-point Courier format. For example:

```
Test Passed
```

1. Standard ARM notation is to use the terms byte, half-word and word, respectively. Intel's convention is to use the terms byte, word and longword. Therefore, this document avoids use of the term 'word', which is ambiguous to different audiences.

1.3 References

This section provides a selective bibliography and a reference to relevant manufacturers' data sheets. ARM-specific and SA-110-specific information is referenced in the support page at the end of this manual.

1. *High-Speed Digital Design* - a handbook of black magic. (Howard W Johnson, Martin Graham, 1993 Prentice Hall ISBN 0-13-395724-1)
2. Intel 28F008 data sheet (visit http://developer.intel.com)
3. Samsung *1996 16M Sync DRAM* databook

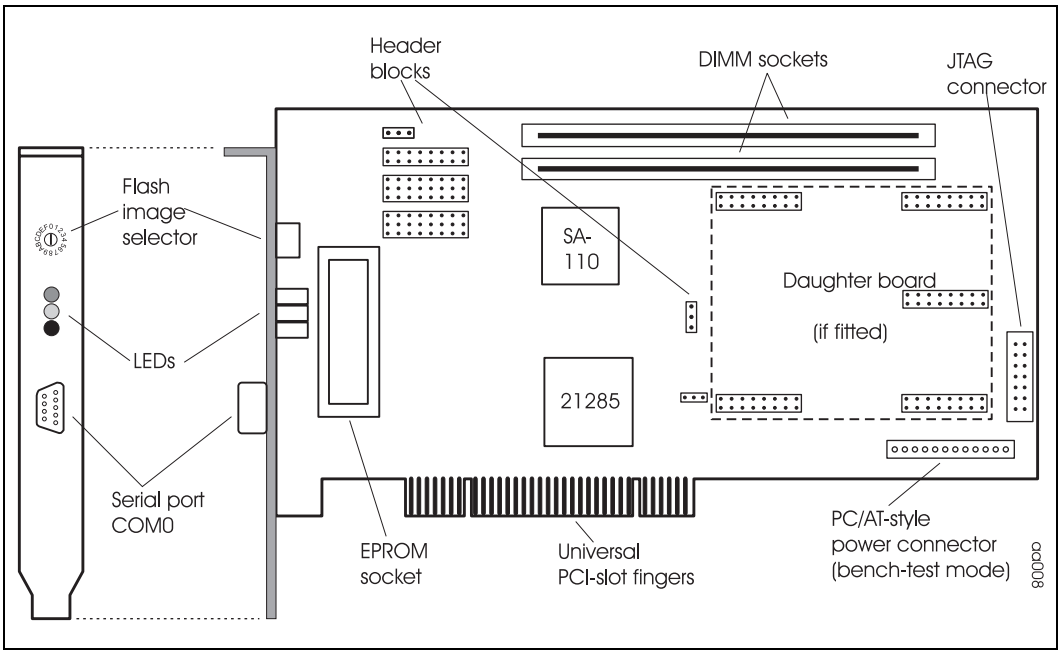
1.4 Physical Description

The physical layout of the EBSA-285 is shown in Figure 1-1. It is a single-board computer with the form-factor of a PCI add-in card.

The EBSA-285 contains processor, system controller, memory and input/output devices. There are a number of header blocks on the card that accept 2-pin jumpers, allowing the card to be configured in different ways, so that all of the major features of the 21285 can be used.

The bulkhead mounting bracket of the EBSA-285 holds a female 9-way D-type connector, three LED indicators and a rotary switch. The D-type connector provides an RS232 terminal connection to a host system. The LEDs provide status information during power up and self-test. The rotary switch is used by onboard software to allow a particular image stored on the board to be executed automatically at power up.

Figure 1-1. EBSA-285



1.5 Unpacking the Card

Caution: The EBSA-285 contains electronic components that are susceptible to permanent damage from electrostatic discharge (static electricity). To prevent electrostatic damage it is supplied in an antistatic bag. When handling the card, risk of damage can be alleviated by following a few simple precautions:

- Do not remove the card from the bag unless you are working on an antistatic, earthed surface and wearing an earthed antistatic wrist strap.
- Keep the antistatic bag that the card was supplied in; if you remove the card from a system, store it back in the bag.

The EBSA-285 is normally supplied with a single, 168-pin plug-in DIMM module containing 16 MB of SDRAM memory. If the DIMM is not fitted when you receive your EBSA-285, install it by following these steps:

1. Identify the DIMM socket closest to the edge of the EBSA-285.
2. Slide the DIMM into the socket taking account of the polarity slots. Do not touch the gold contacts. You can see that there are two polarization slots cut in the DIMM; this ensures that the DIMM is oriented correctly.
3. Support the underside of the EBSA-285 and press the DIMM down into the socket. It should mate with a gentle ‘click’.

Before you install and power up your EBSA-285, you should perform a quick visual inspection:

1. Inspect the card for physical damage.
2. Ensure that each of the 2-pin jumpers is pushed down firmly onto its mounting posts. If you remove any of the jumpers, refer to Appendix A to ensure they are replaced correctly.
3. Ensure that one DIMM is fitted, in the socket closest to the edge of the card, and that the main portion of the gold contacts on the DIMM has disappeared into the socket along the whole length of the DIMM.
4. Check the position of the Flash Image selector switch. There is a dot or an arrow on the switch showing which image is selected. If necessary, use a small screwdriver to rotate the switch so that image 0 is selected.

1.6 Understanding the Different Modes

There are two distinct ways in which the EBSA-285 can be configured. In order to understand the differences between them, it is necessary to review some aspects of the PCI-based system architecture.

The PCI bus has a multi-master capability, allowing any PCI master in the system peer-to-peer access to any other PCI master/target.

In general, the PCI bus is symmetrical, so that any device on the bus can have the same set of capabilities as any other device. However, one device in the system is responsible for generating a software-driven initialization and configuration of all devices on the bus after power up or reset.

Initialization and configuration is performed using Configuration Cycles on the PCI bus and the device that interfaces to the PCI bus to perform these cycles is called the Host Bridge. The processor that accesses the PCI bus through the Host Bridge is called the host processor, or simply the host.

In this document, all devices on the PCI other than the Host Bridge are referred to as PCI devices or add-in cards, or agents.

In a motherboard-based system like a PC, the host processor and the Host Bridge are built onto the motherboard. The motherboard may also contain some PCI devices (for example, an Ethernet network interface) and further PCI add-in cards may be plugged into PCI expansion connectors on the motherboard.

The EBSA-285 can act either as a Host Bridge (in which case the EBSA-285's SA-110 processor is the host processor) or as an add-in card (in which case the SA-110 processor would normally be termed a co-processor).

The 21285 is configured in a mode called 'Central Function' (CFN) mode when it is acting as the Host Bridge for a system.

In addition to the generation of PCI configuration cycles, there are a number of other functions that are normally associated with the Host Bridge:

- Bus arbitration
- PCI clock generation
- Interrupt controller
- Reset generation
- The provision of pullups or 'keepers' on some bus signals

Some of these functions are referred to in the PCI specification as 'Central Resource Functions'.

1.6.1 Add-in Card

This is the default mode for the card. When you receive a new EBSA-285, the jumpers on the card will be configured for this mode.

If you already own a PC with a PCI expansion bus then using this mode is the simplest way to power up and use the EBSA-285.

When the EBSA-285 is configured in this mode it can be plugged into an existing PC motherboard or into a PCI backplane which already contains a Host Bridge. The EBSA-285 is supplied with power, reset and PCI clock from the PCI connector. The motherboard or PCI backplane provides bus arbitration, interrupt controller and pullups for the system.

When the host system is powered up, software running on the host processor will read the configuration registers of the 21285 on the EBSA-285 and allocate system resources for it accordingly. On a standard PC, this function is performed by the BIOS.

1.6.2 Host Bridge

This is ***not*** the default mode for the card. When you receive a new EBSA-285, refer to Appendix A to configure the jumpers on the card for correct operation in this mode.

When the EBSA-285 is configured in this mode it must be plugged into a special slot (the 'System' or 'Host' slot) in a PCI backplane. Intel can supply the EBSA-BPL-5V and EBSA-BPL-3V3 as suitable backplanes. The EBSA-285 acts as the Host Bridge and also provides the PCI interrupt controller for the system. It is supplied with power and PCI clock from the backplane. The EBSA-BPL backplane also provides PCI bus arbitration and pullups.

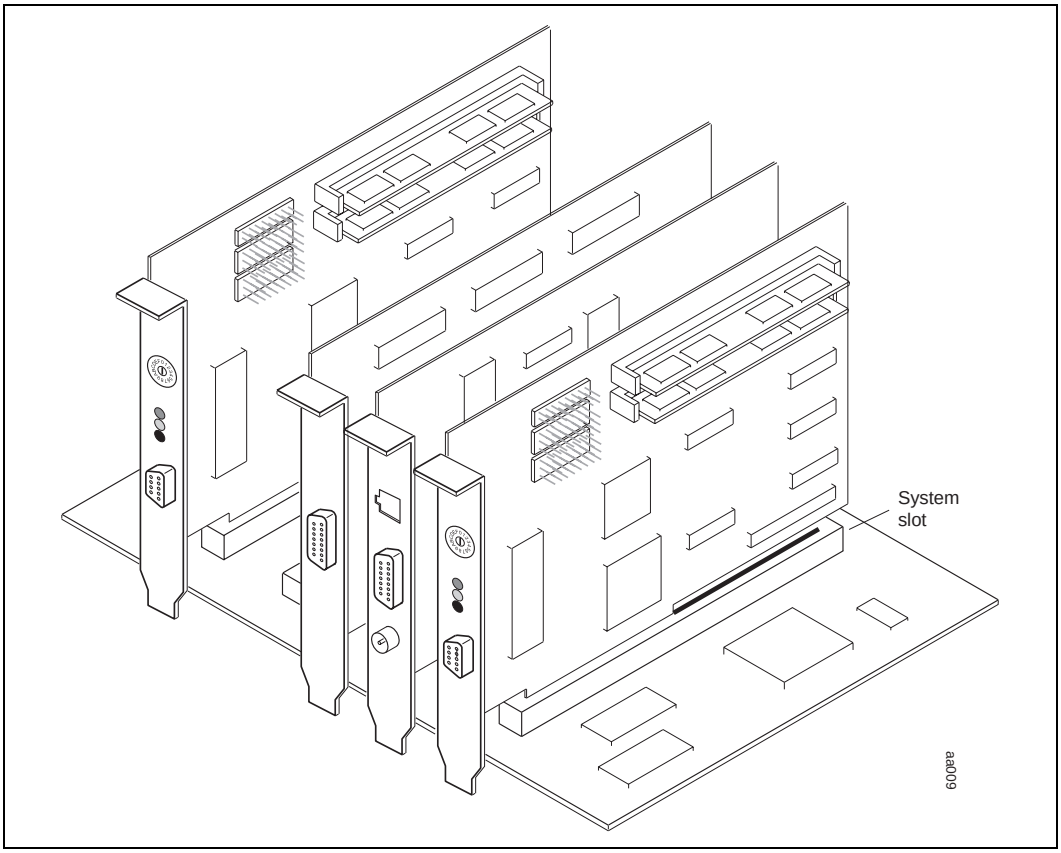
If you wish to use a PCI backplane other than the EBSA-BPL-5V/EBSA-BPL-3V3, refer to the *StrongARM** EBSA–BPL Reference Manual* to confirm that the backplane is suitable before powering up for the first time.

1.6.3 Example Installation

Figure 1-1 shows an advanced installation using the EBSA-285 and one of the Intel PCI Development Backplanes.

In this arrangement there are two EBSA-285 cards, a PCI network card and a PCI video card. The EBSA-285 in the backplane's System slot is configured as a Host Bridge and the other EBSA-285 is configured as an add-in card.

Figure 1-2. Example Configuration in an EBSA-BPL-5V or EBSA-BPL-3V3 Backplane



1.6.4 Other Configuration Options

The EBSA-285 also supports configuration options that are beyond the scope of this introductory section. These include:

- Setting the SA-110 processor core frequency
- Using the 21285 internal PCI arbiter instead of the backplane arbiter (only applicable in Host Bridge mode)
- Allowing a plug-in EPROM emulator to be used in place of the onboard flash ROM
- Holding the SA-110 processor in reset so that a new image can be programmed into the onboard flash ROM (only applicable in add-in card mode)

Refer to Appendix A for details on selecting and using these modes.

1.7 Powering Up for the First Time

Use Section 1.6 to decide which mode to use your card in, and use Appendix A to ensure the jumpers are set appropriately. If you need more details on how to install the card or attach a power supply, refer to Chapter 8.

Use an RS232 null-modem cable to attach the COM0 port on the EBSA-285 to an RS232 port on a terminal or terminal emulator. For example, you could connect to a PC running Windows and use the Windows Terminal or Hyperterminal application. Configure the terminal to operate at 9600 baud, 8-bit data, 1 stop bit, no parity, no flow control. If you need more details on choosing an appropriate cable, refer to Appendix A.

Note: If you are using the card as an add-in card, you cannot use a terminal emulator running on the host for this initial test because the card will have issued messages before the host has booted.

Identify the group of three LEDs on the bulkhead mounting bracket of the EBSA-285. Power-cycle the system and watch the LEDs.

The LEDs should all be illuminated initially and then be extinguished after about half a second. At the same time, the terminal screen should display a message similar to this:

```
Angel Debug Monitor for EBSA 285 (FIQ), MMU off, Clock Switching on (serial)
1.00 (Advanced RISC Machines 2.11) rebuilt on Jun 9 1997 at 23:57:00
```

If you fail to see the behavior described then use this checklist to identify the problem:

- If you have configured the EBSA-285 as an add-in card and it stops the host PC from booting correctly, verify the jumper settings on the card.
- If the LEDs behave correctly but the terminal doesn't produce any output, check the terminal cable and terminal settings. You can test the cable by connecting it between two PC COM ports and running terminal emulation on each port; if the cable is correctly wired you will be able to type characters on either terminal emulator and display them on the other.
- If neither the LEDs nor the terminal behaves correctly then check that the jumpers are set correctly and that the flash image selector switch is set to 0.
- Attempt to run the onboard diagnostics by following the instructions in the next section.

1.8 Running the Onboard Diagnostics

You can get an additional level of confidence that the card is working correctly by running the onboard diagnostics that are programmed into the flash ROM.

Before starting the diagnostics:

1. Attach the card to a terminal as described in Section 1.8.
2. Use a small screwdriver to rotate the Flash Image selector switch so that the dot or arrow on the switch points to the number 1. This selects the image that, by default, contains the onboard diagnostics.
3. Power-cycle the system. The diagnostics should start up automatically and report progress on the terminal. The first thing that the diagnostics do is to flash all three LEDs once.

Chapter 6 describes the output that the diagnostics should produce and describes what to do if the diagnostics fail.

1.9 Using the ARM** SDT with your EBSA-285

The ARM Software Development Toolkit (SDT) includes a remote debugger. When running the remote debugger, one part runs on the host (this part includes the user interface) and the other part runs on the target (the EBSA-285). The host and target communicate across a communications channel. By default, the EBSA-285 uses its COM0 RS232 port to communicate with the host.

The software that runs on the Target is called the *remote debug agent* or *remote debug stub*. By default, the remote debug agent used with the EBSA-285 is a program called Angel.*

Use an RS232 null-modem cable between the COM0 port on the EBSA-285 and the RS232 port on the machine on which the SDT has been installed.

If you are using the EBSA-285 as an add-in card, the SDT can run on the same host. This requires COM0 to be connected to one of the COM ports on the host.

Start up the ARM debugger in the SDT. Use the ‘remote_a’ option from the options menu to select remote debug across a serial port.

For more details on using the SDT, refer to the *ARM Software Development Toolkit Reference Manual*. Chapter 6 describes how to use the SDT to build images that can be executed and debugged on the EBSA-285.

1.10 Support for Angel Over the Ethernet

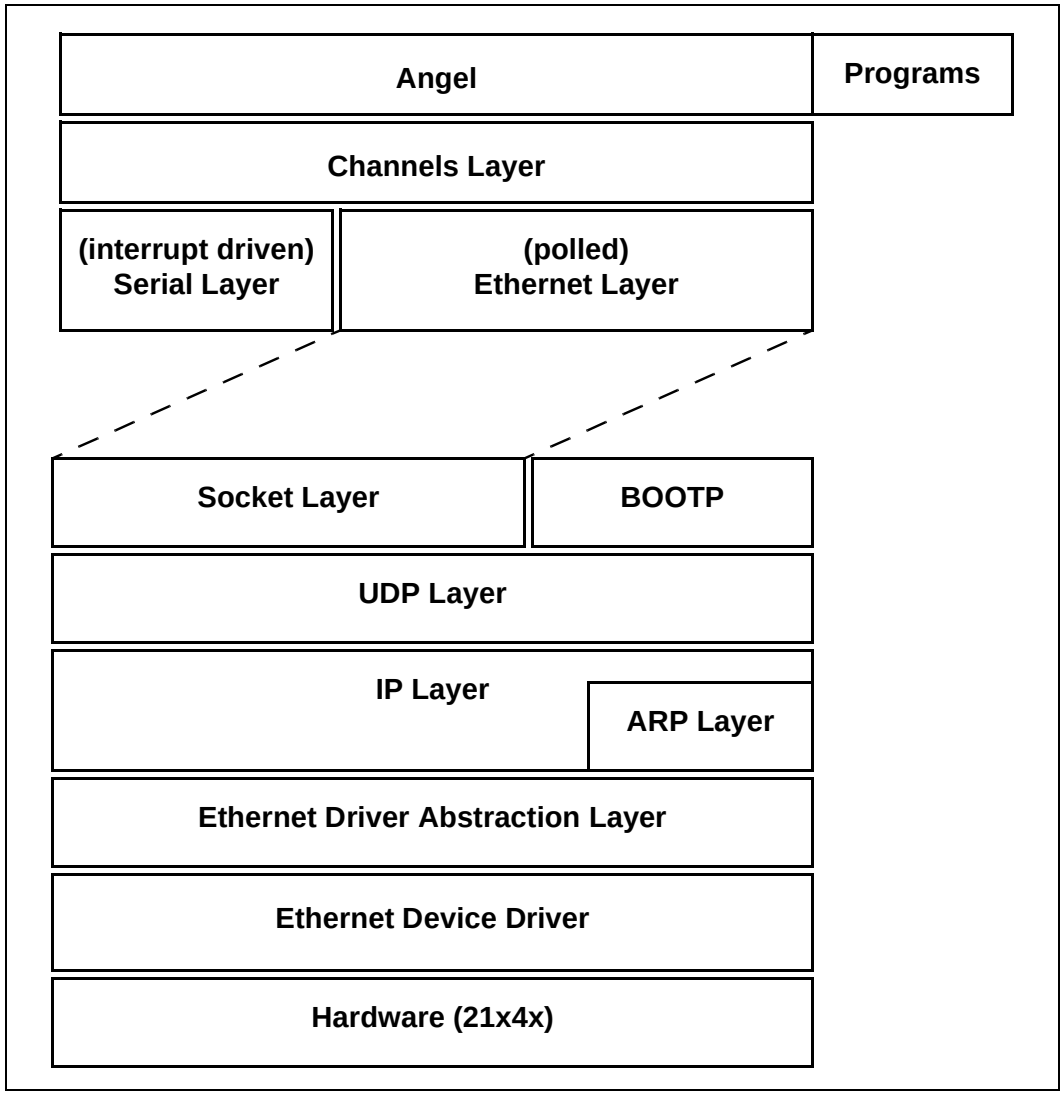
A TCP/IP Ethernet stack has been added to the Angel debugger for faster downloads and debugging of programs targeted for the EBSA-285. The system requires a Bootp server to be running on the Ethernet to which the EBSA-285 is connected, for it to resolve its own IP address. Beyond this the differences, other than download speed, between using the ARM debugger over serial and Ethernet are largely transparent.

For a description of IP stacks and the general terminology please refer to, *Internetworking with TCP/IP, Principles, Protocols and Architecture*. Author: Douglas Comer. ISBN: 0-13-470188-7

1.10.1 Description

Figure 1-3 shows the layers involved with communications over the Ethernet, and how Angel and user program relate. The program communicates its print/read statements and semi-hosting commands via Angel, just as for serial. The channels layer within Angel then decides which is the active device and uses that device to send the message to the debugging host. The stack is a minimal implementation to satisfy the needs of Angel only.

Figure 1-3. Angel Communication Overview



1.10.2 Low-Level Angel Interface

The area of interaction between Angel and the stack is via a socket layer interface. This is a minimal implementation of a socket layer, allowing use of:

Socket	Initialize a socket
bind	Bind socket to a port
sendto	Send to a specific address
recv	Receive
recvfrom	Receive and indicate source of data
close	Close the socket

All of the options and flags in these functions are ignored.

1.10.3 Initialization

Upon initialization the stack will attempt to find out its own IP address using the bootp protocol. It will send out four bootp requests, approximately one to two seconds apart, but will stop when it receives a valid reply. If no bootp request is received then only serial will be usable. A broadcast message is sent to indicate that the debugger is ready to connect when the bootp is complete.

Note: During this bootp period it will not be possible to connect to the debugger.

1.10.4 Host/Client Interaction

Angel will now poll the Ethernet stack and listen for the host debugger trying to connect. The communications occur over UDP/IP, the Angel connect happens on a known port. When a packet with a ‘magic’ word is received on this known UDP port it initiates a negotiation of other communication channels and various Angel parameters. The host may also connect via serial, in which case the Ethernet stack is no longer polled.

Angel can now communicate with the host over the negotiated channels. The ARM remote debugger usage from this point should be the same as for serial debugging. However, it should be noted that Angel must be allowed to run occasionally in order for it to examine the Ethernet stack – this is entirely dependent upon the application that is being run calling ‘angel_yield’ from time to time. This is not necessary for serial because serial uses interrupts to transfer data so is able to halt the application in the foreground. It is not detrimental to yield when serial is being used. Although there is a method built into Angel to start the polling mechanism, polling is always controlled by the program.

Packet transmission via Ethernet is immediate.

1.10.5 Areas of Difference

The stack requirements are different. Angel uses more stack because of the layered approach to IP - see devconf.h for the sizes.

Functional Specification 2

This chapter describes each functional element of the EBSA-285. More detailed information describing how the board works and how to program it can be found in later chapters of this document. Figure 2-1 is a basic block diagram of the board, showing how the major elements interconnect.

2.1 CPU

The EBSA-285 uses the SA-110 microprocessor as its CPU. The board allows the processor to be operated at any one of its 16 core clock frequencies (between 88.3 MHz and 287 MHz with the upper limit determined by the speed grade of the CPU fitted) at a fixed core voltage of 2 V.

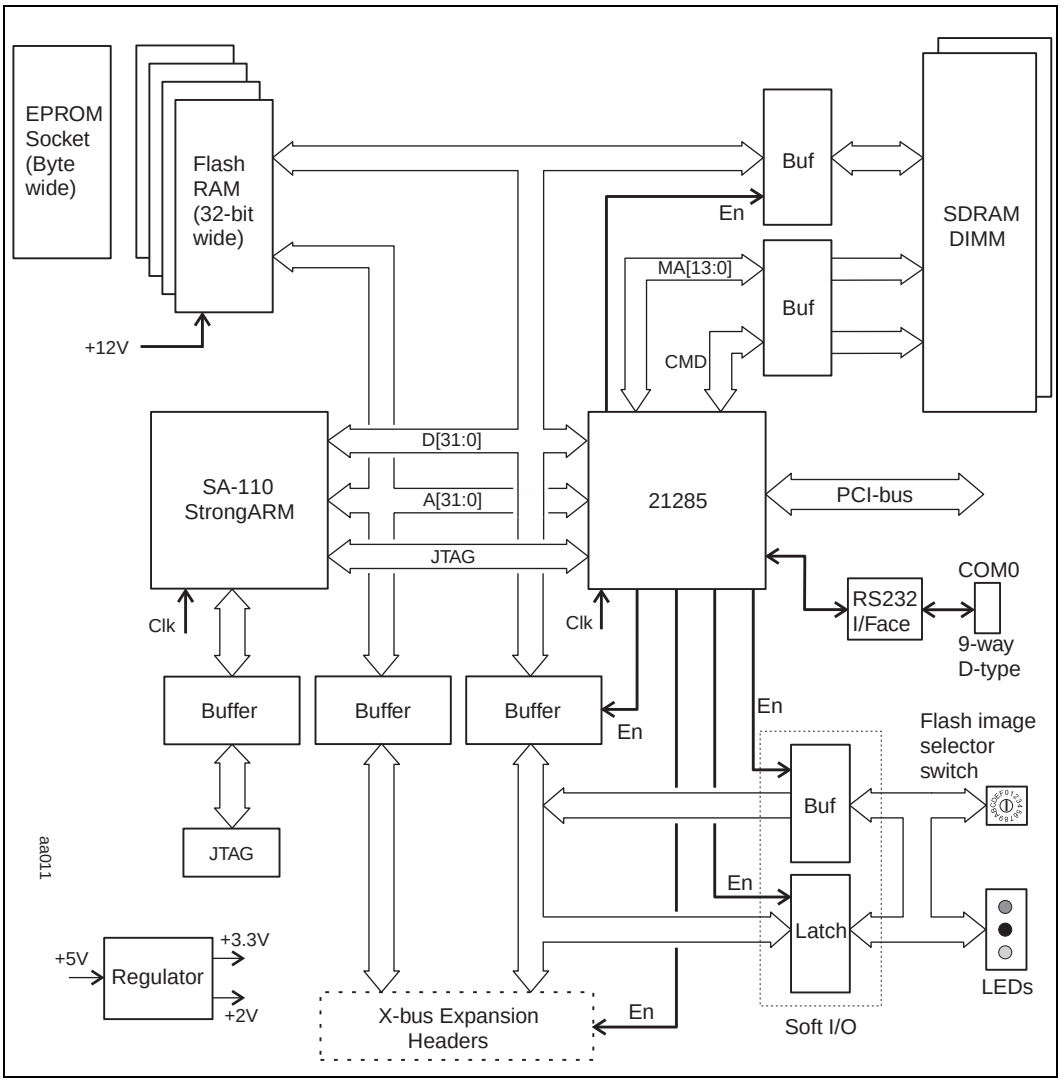
The CPU is packaged in a 144-pin thin quad flat pack (TQFP).

2.2 21285

The 21285 is a Core Logic controller for the SA-110 microprocessor. The 21285 performs all of the control functions on the EBSA-285.

The 21285 is packaged in a 256-point plastic ball grid array (BGA).

Figure 2-1. EBSA-285 Block Diagram



2.3 The Memory Subsystem

The EBSA-285 provides synchronous DRAM (SDRAM) for its main memory and flash ROM for its boot path and non-volatile storage. It also supports the use of a plug-in ROM emulator to aid software debug.

2.3.1 SDRAM

SDRAM chips usually contain multiple logical banks of memory (typically 2 or 4) within a single chip. The existence of these multiple banks is invisible to software but allows the memory controller to extract greater performance from the memories. To avoid confusion with these internal banks, the term *array* is used to describe a physical group of memory devices that share a common chip select and provide a 32-bit data path.

The 21285 supports four SDRAM arrays. The EBSA-285 has sockets for two 168-pin 64-bit SDRAM DIMMs. The 64-bit data path of the DIMM is treated as two separate arrays (since the 21285 requires a 32-bit data path). One of the sockets can accommodate a DIMM of up to four arrays, the other can accommodate a DIMM of up to two arrays. The standard configuration of the EBSA-285 contains a single 2-array DIMM, providing a total of 16 MB of memory. Appendix A.5 explains how to choose suitable DIMMs with which to upgrade the EBSA-285.

2.3.2 Flash ROM

Non-volatile storage is provided by four byte-wide 1 MB flash ROMs, arranged to provide a 32-bit ROM path. This provides a total of 4 MB of ROM.

The ROM is used for two purposes:

- In all configurations of the EBSA-285 it provides the boot code for the SA-110.
- When the EBSA-285 is used as an add-in card, the SA-110 software can make a region of the ROM visible on the PCI bus so that it appears in PCI space as a PCI expansion ROM. The expansion ROM makes code available to the host processor that can be executed for device-specific initialization and, possibly, a system boot function.

The 21285 supports 8-bit, 16-bit and 32-bit ROMs, but the EBSA-285 design only allows the 32-bit mode to be used for accesses to the flash ROM.

The flash ROM is divided into a number of separate blocks, which can be erased and reprogrammed independently. The EBSA-285 is supplied with code programmed into some of the flash blocks. In particular, block 0 of the flash contains the bootstrap code for the SA-110, including the remote debug agent.

If block 0 becomes corrupt, the SA-110 will be unable to execute code after a reset. In this situation, the flash can be reprogrammed using one of these techniques:

- Select the 21285's *blank ROM programming mode* via a jumper on the EBSA-285 and plug the board into an Intel PC, then reprogram block 0 across the PCI bus.
- Program the flash ROM via the 21285 JTAG port.

The EBSA-285 is supplied with a software utility that allows images to be programmed into Flash ROM either across the PCI bus, as described above, or under the control of the ARM** toolkit's remote debugger. This utility, the Flash Management Utility (FMU), is described in Chapter 7.

Software for reprogramming flash ROM via the 21285 JTAG port may be provided in future releases of the EBSA-285 software.

2.3.3 EPROM Emulator

An EPROM emulator is a debugging tool that connects to a target as though it were an EPROM but allows fast download and modification of code. The EBSA-285 has a 32-pin 0.6-inch DIL socket that can be used to connect a byte-wide 512 KB EPROM emulator head. A conventional EPROM **cannot** be fitted to this socket. This is because the socket provides 3.3 V power, and a $\overline{WE}$ line (not found on EPROMs). Similarly, only 3.3 V EPROM emulators can be used.

Warning: Fitting a 5 V emulator may result in damage to your EBSA-285.

Jumpers on the EBSA-285 are used to reconfigure the ROM width and disable the flash ROM in this mode; access to the flash ROM and the EPROM emulator socket are mutually exclusive. When the EBSA-285 is configured correctly, the 21285 makes SA-110 and PCI accesses to the EPROM emulator appear as 32-bit accesses.

The PROMJet* EPROM emulator, from EmuTec, has been used successfully on the EBSA-285. Details from:

Telephone number in the United States of America: +1 425 267 9604

The company web page: http://www.emutec.com

2.3.4 Memory-Map Switching

Immediately after reset, the 21285 decodes the flash ROM at two locations; at its normal base address in high memory and also at an alias of 0. Decoding at this alias allows the SA-110 to fetch its reset vector. The alias is disabled by the first store (write) instruction executed by the SA-110.

2.4 I/O Subsystem

All local I/O (within the EBSA-285 module) is performed as programmed I/O under the control of the SA-110. The I/O subsystem provides the following resources:

- An RS-232 console port (data leads only) accessed via a 9-way D-type on the bulkhead. This is referred to as COM0.
- An 8-bit I/O port used to control LEDs and read the state of jumpers and a switch.

The only other I/O facilities on the board are those provided by the 21285 itself.

2.5 Interrupts

When the EBSA-285 is used as a Host Bridge (21285 configured as Central Function), logic in the 21285 acts as an interrupt controller for interrupts generated locally (on the module and within the 21285) and for interrupts generated by other devices on the PCI.

When the EBSA-285 is used as an add-in card, logic in the 21285 acts as an interrupt controller for interrupts generated locally (on the module and within the 21285). In this mode, the SA-110 can generate an interrupt to the Host Bridge (across the PCI bus) under software control. The interrupt is routed out of the EBSA-285 as INTA#.

2.6 PCI Interface

The EBSA-285 has a 32-bit PCI interface that is compliant with the *PCI Local Bus Specification*, Revision 2.1. It supports both 3.3 V and 5 V signalling. The EBSA-285 has the capability either to generate or to respond to configuration cycles on the PCI bus. These cycles are normally performed by a Host Bridge, however, they can be generated by an add-in card, for example, to determine the PCI access to system memory.

2.7 PCI Bus Arbiter

The 21285 contains a PCI bus arbiter. When the EBSA-285 is configured as a Host Bridge, and plugged into a suitable backplane, the EBSA-285 can provide arbitration for the 21285 itself plus up to four devices on the PCI. The PCI Bus Arbiter cannot be used at the same time as the X-Bus (since they share signal pins on the 21285). A set of jumpers on the EBSA-285 enables one or the other.

Note: When the PCI Bus Arbiter is enabled the X-Bus is disabled, so the EBSA-285 LEDs and flash image selector switch cannot be used.

2.8 JTAG

The SA-110 and the 21285 both contain JTAG ports that allow test access to the I/O pins of the device. The EBSA-285 daisy-chains the two JTAG ports and provides access to the port through a 7x2 0.1-inch pitch header connector. It is recommended that all ROM communications performed by the programming software tool, be done through the DS21285 JTAG Boundary Scan chain. The microprocessor should be held in the reset state by asserting nRESET signal low, so as to grant access to address and data buses.

The Test Access Port (TAP) is buffered from the connecting source device by a standard 74ACT244 device. The physical connection to the TAP is made to the board by a 14 way IDC connector. Table 2-1 shows the IDC connector pinout.

Table 2-1. TAP IDC Connector Pinout

Pin	Type	Use	Polarity
1	–	33R to +5 V	–
2	–	GND	–
3	Input to board	TRST_L	Active low
4	–	NC	–
5	Input to board	TDI	Normal
6	–	GND	–
7	Input to board	TMS	Normal
8	–	GND	–
9	Input to board	TCK	Normal
10	–	GND	–
11	Output from board	TDO	Normal
12	Input to board	SRST_L ─ Board reset	Active low
13	–	Connected to pin1	–
14	–	GND	–

This pinout is compatible with ARM PCBs and microprocessors.

Table 2-2 shows the signals on Test Access Ports (TAPs).

Table 2-2. Signals on the TAP

Signal Name	Pull-up	Sampled/Updated on Clock Transitions From-to	Optional
Test Data In (TDI)	Yes	Low to high	No
Test Data Out (TDO)	No	High to low	No
Test Clock input (TCK)	No	–	No
Test Mode Signal (TMS)	Yes	Low to high	No
Test Reset input (TRST)	Yes	Asynchronous	Yes

The JTAG registers available are shown in Table 2-3.

Table 2-3. JTAG Registers

Register	Purpose
Instruction register	Holds the current instruction op-code value for the following task.
Boundary scan	Reads and writes data to physical device connection pins.
Bypass	Shortcuts the route of the JTAG daisy chain. Register is one cell long per device.
Device ID	Read out a preprogrammed compounded device information number.
Design specific test data register	Test data register used by IC manufacturers

The JTAG commands available are shown in Table 2-4.

Table 2-4. JTAG Commands

Command	Nature	Op-Code	285 Value
EXTEST	Public	All 0’s	0000
SAMPLE	Public	User defined	0001
BYPASS	Public	All 1’s	1111
INTEST	Public	User defined	—
RUNBIST	Public	User defined	—
IDCODE	Public	User defined	0100
USERCODE	Public	User defined	—
CLAMP	Public	User defined	0011
HIGHZ	Public	User defined	0010

2.9 Expansion

The I/O capabilities of the EBSA-285 can be expanded in two ways:

- The PCI interface
- The expansion headers of the buffered 21285 X-Bus. These allow a small mezzanine PCB to be attached for connection to the X-Bus. The X-Bus provides a simple way of providing access to low-performance I/O.

2.10 Clocks

The EBSA-285 uses these oscillators:

- 3.6864 MHz oscillator: This oscillator is used for two purposes. Firstly, it is used to drive the SA-110 phase-locked loop (PLL) input, from which the SA-110 generates its core clock. Secondly, it is used to provide a fixed frequency input to one of the timers in the 21285.
- 50 MHz oscillator: This oscillator provides the osc input clock for the 21285. The 21285 buffers and redrives this clock to generate the SA-110 bus clock, the SDRAM clocks and the 21285 feedback clock, fclk. The local buses and the majority of the 21285's internal logic run synchronously at this clock frequency.
- When the EBSA-285 is plugged into a backplane (either as an add-in card or a Host Bridge) it receives PCI clock from the backplane.

2.11 Resets

There are three sources of reset on the EBSA-285:

- Power-on reset
- Reset from PCI
- Reset from the 21285 watchdog timer

Power-on reset is generated automatically when power is applied to the EBSA-285. It can also be initiated by:

- A push-button switch attached to a 2-pole 0.1-inch pitch connector on the board
- Circuitry connected to the JTAG test connector

Resets generated by any of these methods are equivalent and indistinguishable.

When the EBSA-285 is run in Central Function mode, the circuitry on the board is reset by power-on reset and the 21285 generates PCI reset output under software control. In Central Function mode, the 21285 watchdog timer can be used to reset the whole system in a way that is equivalent to a power-on reset.

When the EBSA-285 is run in non-Central Function mode, it receives reset from the PCI and this reset is used to reset all of the circuitry on the board.

2.12 Power Requirements

The EBSA-285 has the following power requirements:

- + 5 V +/-5%, @ 1 A
- + 12 V +/-5%, @ 250 mA

When the EBSA-285 is plugged into a backplane (either as a Host Bridge or an add-in card), it draws power from the PCI edge connector. The EBSA-285 will function without the + 12 V supply; that supply rail is only used to allow reprogramming of the flash ROMs.

2.13 Onboard Power Generation

The EBSA-285 generates + 3.3 V and + 2 V using onboard circuitry. The +3.3 V is generated from + 5 V using a DC/DC converter. The + 3.3 V is used to supply the flash, the SDRAMs, the 21285 and various data and address buffers. The +2 V is generated from + 3.3 V using a linear regulator. The + 2 V is used to supply the core voltage for the SA-110.

Note: Although the EBSA-285 supports both + 3.3 V and + 5 V signalling on the PCI bus, it never draws + 3.3 V power from a PCI connector; it always requires + 5 V.

2.14 Onboard Software

The EBSA-285 onboard software is programmed into the flash ROM. The flash ROM can contain a number of independent images. At a minimum, the flash ROM contains a program called the Primary Boot Loader (PBL). The PBL can load and start a specific image that is stored in flash ROM and selected using the rotary flash image selector switch on the bulkhead mounting bracket. By default, the PBL starts up the ARM remote debug agent.

An onboard diagnostic suite is also programmed into flash ROM and can also be selected by means of the rotary switch.

Programmer's Guide 3

Much of the functionality on the EBSA-285 is fixed by the design of the 21285. This chapter describes the facilities that are specific to the EBSA-285. These are:

- The flash memory
- The SDRAM memory
- The X-Bus memory-map
- Interrupt assignment
- Timer assignment
- The soft input/output port
- The reset state of the system

This chapter, in conjunction with the 21285 Data Sheet, should act as a complete reference for programmers of the EBSA-285.

3.1 Flash Memory

Non-volatile code storage is provided by four 1 MB flash ROMs. These are Intel 28F008 parts. They are arranged to provide a 32-bit boot path for the SA-110.

Each flash ROM provides sixteen 64 KB blocks of memory. The blocks are contiguous but can be erased and programmed separately. Because the flash is arranged as 32-bit memory, it can be treated as sixteen 256 KB blocks decoded at addresses 0x4100 0000 - 0x413F FFFF.

A new EBSA-285 contains debugger and self-test images in the first two blocks of the flash ROM. User code can be programmed into other blocks. A utility for managing images in flash ROM is described in Chapter 7.

The EBSA-285 can be configured to accommodate an 8-bit ROM emulator, which attaches via a 32-pin 0.6-inch DIL socket on the board. Refer to Section 2.3.3 for information on choosing and using a ROM emulator.

3.2 SDRAM Memory

The EBSA-285 can accommodate two 168-pin 3.3 V SDRAM DIMMs. The standard configuration is to fit a single 16 MB DIMM, which appears as two 8 MB arrays decoded at addresses 0x0 - 0xFF FFFF. Appendix A.5 describes how to upgrade or change the configuration of the DIMM. Section 4.7 describes how to initialize the 21285's memory controller to enable access to the SDRAM.

3.3 X-Bus Memory-Map

The X-Bus is used for accesses to external low-speed I/O devices. There are four separate address spaces. On the EBSA-285 the only address space that is used is the XCS2 address space, 0x4001 2000 - 0x4001 2FFF. All accesses in this region alias to the soft input/output register, which is described in Section 3.6. The remaining X-Bus address regions could be used by a mezzanine daughtercard attached to the EBSA-285.

Initialization of the X-Bus is described in Section 4.4.

3.4 Interrupt Assignment

The 21285 allows some of its external pins to be used as interrupt inputs. The assignment of these signals on the EBSA-285 is shown in Table 3-1 on page 3-2.

Table 3-1. Interrupt Assignment

Signal	Bit in IRQ Status	Assignment
pci_irq_l	18	When the EBSA-285 is configured as the Host Bridge and plugged into a PCI backplane, this bit supplies INTA# from the PCI.
xcs_l[2]	14	This bit would never normally be enabled as an interrupt on the EBSA-285.
xcs_l[1]	13	This bit is unassigned on the EBSA-285. It could be used on an X-Bus daughtercard.
xcs_l[0]	12	This bit is unassigned on the EBSA-285. It could be used on an X-Bus daughtercard.
irq_in_l[3]	11	When the EBSA-285 is configured as the Host Bridge and plugged into a PCI backplane, this bit supplies INTD# from the PCI.
irq_in_l[2]	10	This input is driven from a fixed 3.68 MHz oscillator. It is normally used to increment Timer 3, as described in Section 3.5. It would never normally be enabled as an interrupt.
irq_in_l[1]	9	When the EBSA-285 is configured as the Host Bridge and plugged into a PCI backplane, this bit supplies INTC# from the PCI.
irq_in_l[0]	8	When the EBSA-285 is configured as the Host Bridge and plugged into a PCI backplane, this bit supplies INTB# from the PCI.

Each of these interrupts has programmable polarity.

3.5 Timer Assignment

The 21285 has four internal timers. These can be incremented by clocks internal to the 21285 or by an external input.

If timer 3 is configured to increment from an external input, it will be clocked from the SA-110 PLL input oscillator, and will count at a rate of 3.68 MHz. This is the only independent fixed oscillator frequency on the board and it can be used to infer the system bus speed, the PCI clock speed, and the SA-110 core clock speed.

3.6 Soft Input/Output Register

The bulkhead mounting bracket of the EBSA-285 holds three LED indicators (one each red, amber and green) and a 16-position switch. The soft input/output port allows software to control the state of the LEDs and read the state of the switch. Software can also read the state of three onboard jumpers. The bit assignment of this register is shown in Table 3-2.

Table 3-2. Bit Assignment of Soft Input/Output Register

Bit	Name	Type	Description
7	TOGGLE	Read/Write	This bit acts as a read/write bit and has no other effect. Its intention is to provide a bit that can be toggled under software control to provide some indication that the X-Bus can be accessed successfully.
6:3	Unused	Write-only	These bits are unused on writes; data is don't-care.
2	RED_L	Write-only	Write a 0 to illuminate the red LED, write a 1 to extinguish the red LED.
1	GREEN_L	Write-only	Write a 0 to illuminate the green LED, write a 1 to extinguish the green LED.
0	AMBER_L	Write-only	Write a 0 to illuminate the amber LED, write a 1 to extinguish the amber LED.
6	IBUF6	Read-only	Read the state of jumper J17 pins 9-10. Read '1' if the jumper is removed, '0' if the jumper is fitted. This bit is used by the supplied initialization software to determine whether or not the EBSA-285 should permit access to PCI. PCI accesses are enabled when fitted.
5	IBUF5	Read-only	Read the state of jumper J17 pins 11-12. Read '1' if the jumper is removed, '0' if the jumper is fitted.
4	IBUF4	Read-only	Read the state of jumper J17 pins 13-14. Read '1' if the jumper is removed, '0' if the jumper is fitted.
3:0	SWITCH_L	Read-only	Read the state of the 16-position switch. The data is the inverse of the value selected on the switch, so that this nibble will read 0xf when the switch is set to '0', 0xe when the switch is set to '1', and so forth.

Onboard software adopts a consistent policy for the use of the LEDs, the rotary switch and the jumpers. This is described in Chapter 6.

3.7 The Reset State of the System

After reset, the 21285 SDRAM controller and the X-Bus are disabled. The 21285 decodes the flash ROM at two locations; at its normal base address and also at an alias of 0.

Since the 21285 SDRAM controller is disabled by reset, SDRAM contents is UNDEFINED after reset.

Software Configuration and Initialization 4

Software must initialize the system hardware after a power up or reset. This chapter provides guidelines for the various initialization steps that need to be taken.

The software that is loaded into the system by the Primary Boot Loader or by a remote debug agent such as Angel can make assumptions about what in the system has already been configured. The state of the run-time environment in these different situations is described in Chapter 6.

The correct sequence for the initialization steps required after reset is:

1. Disable the flash ROM alias at address 0.
2. Determine the board configuration.
3. Initialize the X-Bus or PCI bus arbiter.
4. Initialize the SDRAM.
5. Configure the 21285 UART (optional).
6. Configure the PCI interface (optional).
7. Configure the SA-110 memory-management unit (MMU), turning on virtual memory, clock switching and caches (optional).

4.1 Disabling the Flash ROM Alias

After reset, the 21285 decodes the flash ROM at two locations; at its normal base address of 0x4100 0000 and also at an alias of 0. Decoding at this alias allows the SA-110 to start executing from the normal reset vector address of 0. Once the first write operation has been executed by the SA-110, the alias is disabled. Software should ranch to the high-order alias of the flash ROM before executing the first write operation.

4.2 Accessing the Flash ROM

After reset and in normal operation, the flash ROM appears as a 32-bit read-only device. It is actually implemented using four, 8-bit devices that are accessed in parallel. Flash operations other than simple ROM operations are selected by writing specific commands into the Command User Interface. Refer to the Intel 28F008SA data sheet for details of the commands.

The four flash parts can be accessed individually or simultaneously, depending upon whether commands are sent to all parts simultaneously or to a subset of the parts. For example, programming a flash location is achieved by writing 0x40 to a byte location followed by writing a data byte to the same location. A write of 0x40404040 to any EBSA-285 flash location followed by a write of 0xabcd.ef01 to the same location will result in 0xabcd.ef01 being programmed into the location (one byte programmed into each of the four flash devices). However, a write of 0x40000040 to an EBSA-285 flash location followed by a write of 0xabcd.ef01 to the same location will result in the 0xab and 0x01 bytes being programmed and the other two bytes of the longword being unchanged.

4.2.1 Programming the Flash from the SA-110

The SA-110 must not be executing from flash ROM when it executes a flash programming algorithm. The reason for this is that flash programming requires a defined sequence of reads and writes to the flash. Code fetches would disrupt the sequence.

The Angel debug monitor relocates itself into SDRAM after power-up. The standard flash management utility also executes from RAM, thus avoiding any code fetches when programming flash.

4.2.2 Programming the Flash from the PCI Interface

Before programming the flash from the PCI interface, the 21285 *ROM Write Byte Address Register* must be set to 0.

The 21285 allows the flash to be reprogrammed from the PCI interface while the SA-110 is running (though this will only work correctly if the SA-110 is not accessing the flash). However, the 21285 also provides a special mechanism for reprogramming flash, called ‘Blank ROM Mode’. On the EBSA-285, this mode is selected by moving a jumper on J15 (refer to Appendix A.2.2.3. for details).

4.3 Determining the Card Configuration

The EBSA-285 can be used in various modes, all selected by jumper, and it may be necessary for software running on the board to behave differently in the different modes. Two bits identify the configuration of the board:

- Bit 23 of the 21285 *X-Bus Cycle/Arbiter Register* – the X-Bus/Arbiter bit:

  The 21285 is configured, at power up, to enable ***either*** its X-Bus ***or*** its internal PCI bus arbiter. Depending upon which mode is selected, software should initialize the appropriate registers, as described in Section 4.4 and Section 4.5. If the X-Bus is disabled, the soft input/output register described in Section 3.6 cannot be used.

- Bit 31 of the 21285 *SA-110 Control Register* – the CFN bit:

  The 21285 is configured, at power up, to act either as a Host Bridge (CFN asserted) or an add-in card.

4.4 Initializing the X-Bus

If the X-Bus is enabled, it should be configured by writing to the 21285 *X-Bus Cycle/Arbiter Register* and *X-Bus I/O Strobe Mask Register.* The *X-Bus I/O Strobe Mask Register* is also used to set the polarity of some of the 21285 interrupt inputs. A suitable initialization sequence for the EBSA-285 is:

1. Write 0x1000 16db to the 21285 *X-Bus Cycle/Arbiter Register.*
2. Write 0xfcfc fcfc to the 21285 *X-Bus I/O Strobe Mask Register.*
3. Write 0x6000 0000 to the 21285 *SA-110 Control Register* (enables X-Bus chip selects 2 and 1).

4.5 Initializing the PCI Bus Arbiter

If the PCI bus arbiter is enabled, it may be configured by writing to the 21285 *X-Bus Cycle/Arbiter Register.* The arbiter will work correctly with the default (power up) values in this register.

4.6 Setting the INITIALIZE_COMPLETE Bit

If the EBSA-285 is configured as an add-in card, plugged into a PC, the PC will attempt to access the EBSA-285's PCI configuration registers as part of the PC's power up self-test (POST) sequence. After reset, the 21285 will cause a PCI retry in response to a PCI configuration cycle. Unless software on the EBSA-285 configures the PCI interface, this will cause the PC to retry forever, so that it appears to 'hang'. The minimum initialization required to avoid this is to set bit 0 (INITIALIZE_COMPLETE) in the 21285 *SA-110 Control Register.* A more sophisticated initialization is described in Section 4.9.

4.7 Initializing the SDRAM

Two sets of operations are required in order to allow access to the SDRAM:

- Configure the 21285 memory controller registers.
- Configure the mode registers in the SDRAM arrays.

This section provides a sample configuration, coded in ARM** assembler. The code assumes that all four SDRAM arrays are populated with 2Mx8 parts that can run with a latency of two.

1. Start with a set of equates for registers and register values.

```
CSR_BASE           EQU        &42000000
SDRAM_TIMING       EQU        &10C
SDRAM_ADDR_SIZE_0  EQU        &110
SDRAM_ADDR_SIZE_1  EQU        &114
SDRAM_ADDR_SIZE_2  EQU        &118
SDRAM_ADDR_SIZE_3  EQU        &11C
CMD_DRIVE          EQU        &800
PARITY_ENABLE      EQU        &1000
Tref_min           EQU        &010000
Tref_norm          EQU        &1A0000
Trp                EQU        &1
Tdal               EQU        &4
Trcd               EQU        &20
Tcas               EQU        &80
Trc                EQU        &300
```

2. After reset, the SDRAM arrays are in an unknown state. To put them into a known state, force an all-banks precharge to each of the four possible arrays. You must access all four arrays for this even if all four are not fitted. This is necessary because the 21285 counts these precharge accesses, and inhibits access to the SDRAM until all four have been completed. Failure to perform four precharge accesses will result in unpredictable operation. An all-banks precharge is initiated by a read from any address in the mode register address space.

```
ldr    r0,=&40000008 ; SDRAM array 0
ldr    r0,[r0]
ldr    r0,=&40004008 ; array 1
ldr    r0,[r0]
ldr    r0,=&40008008 ; array 2
ldr    r0,[r0]
ldr    r0,=&4000C008 ; array 3
ldr    r0,[r0]
```

3. Write to the *SDRAM Mode Register* in the SDRAMs. This requires one write operation for each SDRAM array. The address is important, not the data. The offset from the start of the mode space for each SDRAM array controls what data is written to the SDRAM mode register. The mode register should be configured for a burst size of 4 and for linear addressing.

```
ldr    r0,=&40000008:OR:Tcas
str    r0,[r0]
ldr    r0,=&40004008:OR:Tcas
str    r0,[r0]
ldr    r0,=&40008008:OR:Tcas
str    r0,[r0]
ldr    r0,=&4000C008:OR:Tcas
str    r0,[r0]
```

4. Write to the *SDRAM Timing Register* in the 21285. Set the refresh interval to the minimum because we have to wait for 8 refresh cycles to complete before we can rely on the SDRAMs operating normally.

```
ldr    r1,=CSR_BASE
ldr    r0,=Trp:OR:Tdal:OR:Trcd:OR:Tcas:OR:
           Trc:OR:CMD_DRIVE:OR:Tref_min
str    r0,[r1,#SDRAM_TIMING]
```

5. Wait for 8 refresh cycles to complete. The minimum refresh interval is 32 cycles and we are currently running with the Icache off, so the complete process will take 256 cycles.

```
ldr    r0,=&100
wait       subs   r0,r0,#1
bgt    wait
```

6. Write to the four 21285 *SDRAM Address and Size Registers*. This simple code assumes four arrays are fitted and that they are all the same size and type. More sophisticated code would automatically detect and size each array.

```
ldr     r0,=&14
str     r0,[r1,#SDRAM_ADDR_SIZE_0]
ldr     r0,=&800014
str     r0,[r1,#SDRAM_ADDR_SIZE_1]
ldr     r0,=&1000014
str     r0,[r1,#SDRAM_ADDR_SIZE_2]
ldr     r0,=&1c00014
str     r0,[r1,#SDRAM_ADDR_SIZE_3]
```

7. Finally, reset the refresh interval to a sensible value. Continuing to run with a very short interval would waste memory bandwidth. The refresh interval is calculated to refresh 4096 rows in 64 ms.

```
ldr     r0,=Trp:OR:Tdal:OR:Trcd:OR:Tcas:OR:
            Trc:OR:CMD_DRIVE:OR:Tref_norm
str     r0,[r1,#SDRAM_TIMING]
```

4.8 Re-initializing the SDRAM

The 21285 only allows SDRAM mode writes to be performed when refresh is disabled. Therefore, the initialization sequence shown in the previous section cannot be re-executed. Either of the following two modifications will allow the code to be re-executed:

- At the start of the sequence, read the 21285 *SDRAM Timing Register* and check whether refresh is enabled (refresh interval set to a non-zero value). If it is enabled, you can infer that the SDRAM initialization has already been performed and that no further action is needed. If refresh is not enabled, you can execute the initialization sequence described in Section 4.7.
- At the start of the sequence, disable refresh by setting the refresh interval to 0, then wait for 15 bus clock cycles to ensure that any pending or in-progress refresh completes successfully. You can then execute the initialization sequence described in Section 4.7.

4.9 Initializing the PCI Interface

This section describes the minimum set of PCI registers that must be configured to allow the PCI interface to be used.

1. Start by setting some important registers to a known state as follows:
   - Write 0xc in the 21285 *Outbound Interrupt Mask Register* (disable outbound interrupts).
   - Write 0x0 in the 21285 *Doorbell PCI Mask Register* (clear doorbell interrupts to PCI).
   - Write 0x0 in the 21285 *Doorbell SA-110 Mask Register* (clear doorbell interrupts to the SA-110).
   - Write 0x0 in the 21285 *PCI Address Extension Register* (set it to a known state).
   - Write 0x1 in the 21285 *Interrupt Line Register* (some PCI systems do not correctly recognize an interrupt ID of 0).
2. Negate the PCI reset signal (this only has an effect if the EBSA-285 is the Host Bridge - it is ignored otherwise):
   - Set bit 9 (PCI not reset) in the 21285 *SA-110 Control Register.*
3. Open up a window from PCI memory space into the EBSA-285 SDRAM address space. If the EBSA-285 is configured as an add-in card, configuring this window will allow the host PC to allocate PCI memory address space for the EBSA-285 when it performs its POST. To create an 8-MB window:
   - Write 0x007c.0000 to the 21285 *SDRAM Base Address Mask Register.*
4. The next step must only be performed if the CFN bit is set and the bench test bit is clear (refer to Section 4.3):
   - Write 0x0 to the 21285 *Command Register*. This stops the 21285 from responding to any PCI transactions.
   - Write 0x4000.0000 to the 21285 *CSR Memory Base Address Register.*
   - Write 0xf000 to the 21285 *CSR I/O Base Address Register.*
   - Write 0x0 to the 21285 *SDRAM Base Address Register*.
   - Write 0x17 to the 21285 *Command Register*. This enables the 21285 as a bus master and allows it to respond to I/O space and memory space transactions as target.
5. Finally, set bit 0 (INITIALIZE_COMPLETE) in the 21285 *SA-110 Control Register.* This will allow the EBSA-285 to respond to PCI configuration cycles, as described in Section 4.6.

4.10 Initializing the 21285 UART

The EBSA-285 runs with an **fclk_in** frequency of 50 MHz. The frequency of **fclk_in** determines what divisors are appropriate when configuring the 21285's internal UART. Table 4-1 shows baud rate divisors for this bus frequency.

Table 4-1. 21285 Baud Rate Divisors for 50 MHz fclk_in

Baud Rate	Divisor	Error
50	15624	0.00%
75	10416	0.00%
110	7101	0.00%
134.5	5808	0.01%
150	5207	-0.01%
300	2603	-0.01%
600	1301	-0.01%
1200	650	-0.01%
1800	433	-0.01%
2000	390	0.10%
2400	325	0.15%
3600	216	-0.01%
4800	162	0.15%
7200	108	0.45%
9600	80	-0.47%
19200	40	0.76%
38400	19	-1,73%
56000	13	0.35%
128000	5	-1.73%

4.11 Configuring Cacheable/Non-Cacheable Space

To get maximum performance from the SA-110 you must enable clock switching and turn on the internal Icache and Dcache. The Dcache can only be enabled when the MMU is enabled. The page-tables used by the MMU control, on a page-by-page basis, whether or not the contents of the memory page is Dcacheable and whether or not writes to the page can use the SA-110 write buffer. Refer to the *StrongARM** SA-110 Microprocessor Technical Reference Manual* for more details.

For correct operation, the following rules must be followed:

- The SDRAM address region may be marked Icacheable, Dcacheable, Bufferable.
- The flash ROM address region may be marked Icacheable and Dcacheable for reads. During writes to flash ROM (reprogramming) the Dcache and write buffer should be disabled.
- Some regions of I/O space may be marked as bufferable, non-cacheable, which will improve the performance of write operations. The PCI memory space in particular (0x8000 0000 - 0xFFFF FFFF) should be marked as bufferable, non-cacheable. This will allow SA-110 writes to this region to be merged within the 21285 resulting in write bursts on the PCI whenever possible.
- The 21285 CSR address region should be marked non-cacheable, non-bufferable.

If you implement the Software Dcache Flush Algorithm described in Chapter 6 of the *StrongARM** SA-110 Microprocessor Technical Reference Manual* you can use accesses to the 21285's SA-110 Cache Flush region (0x5000 0000 - 50FF FFFF). Read accesses to this region complete in the minimum amount of time and return dummy data. This minimizes the time it takes to flush the caches.

Software Development Environment 5

This chapter describes the types of image that may be built for the EBSA-285, and how to use the ARM** software development toolkit to build the images. The toolkit itself is described in the *ARM Software Development Toolkit Reference Manual*. Two types of image are described:

- Loadable debuggable images
- Standalone flash images

Flash images may be programmed into flash using the FMU utility described in Chapter 7.

Note: This chapter assumes the EBSA-285 is using the Angel debug agent.

5.1 Loadable Debuggable Images

These images are run under the control of the Angel debug agent held in flash ROM, communicating with either the ARM command line Symbolic Debugger (armsd) or the ARM Windowing Debugger.

5.1.1 Building

Debuggable images can be written in either C or assembler. As well as describing the target CPU as StrongARM** (-cpu StrongARM1), you should assemble or compile with the -g option. This adds symbolic information to the executable image file. If your program uses any standard C library calls, for example **printf()**, you should link with the Angel (semi-hosted) C library. As symbolic debug information is included in the default, there is no need to use any extra options when linking debuggable images. Images should be linked using either the -AIF or the -AIF -BIN options. Images linked with the -BIN option can still be debugged at the machine code level.

Images that are to be loaded across the serial line using the debugger's load command may be linked to use any base address in SDRAM except addresses that are below 0x8000. Addresses 0 - 0x8000 are used by Angel for context, stacks and so on.

5.1.2 Run-Time Environment

5.1.2.1 Memory Map

All SDRAM except address range 0 to 0x8000 is available to the program. The X-Bus and SDRAM will have been initialized before entry to the program. The SA-110 MMU, write buffer, and caches will not have been initialized unless you have done this by running a previous program or by writing to the system coprocessor using debugger commands.

The C heap will be placed directly above the text segment of the program. By default, Angel runs with the Dcache and the write buffer disabled but with clock switching and the Icache enabled. The C library initialization functions will place the user stack at the top of SDRAM.

5.1.2.2 ARM C Library Support

The ARM C library is described in the toolkit manual. All standard C functions are supported. All reference to files (including references to standard input and output) refers to these files on the host. This means that, for example, a call to **printf()** prints a string to the host that is running the debugger.

5.1.2.3 Exception Vectors

The Angel debug monitor uses the Undef, SWI and FIQ exception vector entries. The program can safely modify any other exception vector to jump to its own exception handlers. The program can also install its own handlers using SWI_InstallHandler. This is described in the Angel documentation.

5.1.2.4 Access to I/O Devices

Angel uses the COM0 serial port. The program must not access this device. All other devices may be used by the program.

5.2 Standalone Flash Images

These are images that are written into one or more consecutive flash ROM blocks. At boot time the Primary Boot loader selects the image to run and then transfers control to it. Flash images can either execute in place or from memory. If the flashed image is to be executed from memory, the Primary Boot Loader first initializes the memory and then copies the image into memory before passing control to it.

As a side effect of initializing the X-Bus, the Primary Boot Loader always disables the flash ROM alias at address 0x0 and executes from the high-order alias.

5.2.1 Building

Images may be written in C or assembler. No special options are needed when assembling or compiling. As well as providing startup code to swap the initial memory map and so on, you must supply the code of any library functions used (refer toSection 5.2.2.2). There are two ways of linking such images:

- -AIF -BIN -BASE n
  - — If the base address is outside of the address range of the flash ROM, the PBL will copy the image to its base address in system SDRAM (removing the header in the process) and execute it from its entry point; the image will execute from SDRAM.

    In this case, the image may occupy non-contiguous blocks in flash ROM.
  - — If the base address is equal to the flash block address + 0xc0, the PBL will execute the image by branching to its entry point; the image will execute from flash ROM (for example, use address 0x410C 00C0 for an image that will execute out of flash block 3).

    In this case, the image must occupy contiguous blocks in flash.
  - — If the address does not meet either of these requirements, the FMU will report an error and will not program the image into flash.

    Images linked with this option may use any base address in SDRAM.

- -AIF -BASE n

  The image will execute from flash ROM. Requirements are:

  — The image must occupy contiguous blocks in flash ROM

  — The image must not contain any writable initialized data

  — The address of the first flash block to be used for the image must be known at link time

  — The base ‘n’ must be the address of the flash block + 0x40 (for example, 0x410C 0040 for flash block 3)

  In this case, the image is started by branching to the BL instruction that is the first longword of the AIF header. The FMU does not validate the entry point.

  This option should normally be avoided (except for programs that relocate themselves to SDRAM during initialization) since accesses to flash ROM are much slower than access to SDRAM.

5.2.2 Run-Time Environment

5.2.2.1 Memory Map

All of SDRAM is available to the program. If the program is run from SDRAM, then SDRAM will have been initialized before entry to the program. If it is run directly from flash, then the X-Bus has been initialized and the initial memory map has been swapped. The boot time memory map will still be in use although the PC will be in the first alias above 4100 0000 of the flash block (not in a low alias). The MMU and caches will not have been initialized. The C library initialization functions place the user stack at the top of SDRAM.

5.2.2.2 C Library Support

ARM’s software development toolkit includes sources and porting information for two run-time libraries; a minimum standalone library and an ANSI C library. EBSA-285 ports of these libraries may be supplied as part of the firmware database in the hardware developer’s kit.[1]

5.2.2.3 Exception Vectors

The program may modify and use the exception vectors without restriction.

5.2.2.4 Access to I/O Devices

If a C library is used, it will provide routines to access some devices (for example, the COM0 serial port) and it will expect exclusive access to the associated underlying hardware. Other than this, the program may access any device.

1. Early versions of the HDK are unlikely to provide this.

Onboard Software 6

The EBSA-285 is shipped with the following programs blown into its flash ROM:

- Primary Boot Loader (PBL)
- Angel remote debug agent
- Diagnostics

When the EBSA-285 is reset or powered up, code execution commences with a fetch from the reset vector at location 0, the first image in flash. This will start execution of the PBL.

6.1 Primary Boot Loader

The Primary Boot Loader (PBL) is part of a special Angel image programmed into the first block (block 0) of the flash. The PBL is the first code executed when the EBSA-285 comes out of reset.

The flash can contain a number of different images; the main function of the PBL is to determine which image to execute and then to execute it. If necessary, the PBL will load the selected image from flash into system memory.

Images are programmed into flash ROM using the Flash Management Utility (FMU) described in Chapter 7. The format of the images in flash is described in Section 6.2.

When the PBL is executed, it performs these tasks:

- Reads the value of the flash image selector switch to determine which image to branch to after initial boot (see Table 6-1)

Table 6-1. Boot Image Selection

Selection	Contents	Action
0	Image 0:	Enters ARM**/Angel* remote debug stub within PBL image
1	Image 1	Contains diagnostics image when board first supplied
2 to F	Images 2 to F	Available for user programs

- If Image 0 is selected, enters the ARM/Angel remote debug stub within the PBL image
- If any other image is selected:

1.Switches the memory map

2.Searches for the image in flash, and verifies that the checksum is correct.

3.If image is not found or is corrupt (bad checksum), behaves as though the selected image is image 0.

4.If the image is in executable AIF format, jumps to the image (the system memory map has not been changed and the DRAM has not been initialized).

5.If the image is in non-executable AIF format, then:

a. If the image executes from SDRAM, then it:
   i. Initializes DRAM
   ii. Loads the image into memory at the addresses defined in the AIF header

b. Jumps to the image's entry point

6.2 Format of Images in Flash ROM

The flash ROM is in four 1 MB parts, organized to provide 16, 256 KB, 32-bit wide, blocks. Block 0 (at address 0x0000 0000, after reset) is reserved for the PBL/Angel. The remaining 15 blocks can be used to hold other images.

Each image, apart from the PBL/Angel, has an image header that allows it to be stored across non-contiguous blocks. Only the first block used by the image has an image header. Any individual block is only used by one image or no image. Any block that is not in use will be in its erased state.

The format of an image stored in the flash ROM is basically AIF (ARM Image Format), with a few additional bytes prepended. The format is shown in Table 6-2. When the FMU is used to program an image into flash, the FMU will create and prepend the header information onto the image.

Note: You may write an alternative flash programming utility, but it should follow the defined flash structure so that the PBL can load the image.

Table 6-2. Flash Image Header

Offset (bytes)	Size (bytes)	Name	Description
0	4	Type	BL to AIF header (for executable AIF) or BL to image entry point (for non-executable AIF on image to be executed from flash) or NOP (for non-executable AIF executed from RAM).
4	1	Number	Unique image number (0 to 0xff).
5	3	Sig	0x55 0xaa 0x00.
8	4	Map	Allocation map. Bit 0 represents block 0, bit 31 represents block 31 (only bits 15:0 are required for the current flash part).
12	4	Checksum	Checksum of image including headers, using the algorithm described below.
16	4	Length	Image length (including all headers) - used to determine what gets checksummed.
20	16	Name	ASCII string identifying name of image. Unused characters should be set to 0x20 (ASCII space).
36	4	Bootflags	Bit 0 is NoBoot. When set for an image, the PBL will load the image but then pass control to the ARM remote debug stub within the PBL.
40	24	Reserved	Reserved for future use.
64	128	AIF header	AIF header for image.

The headers use a total of 192 bytes. The first free byte is at offset 192 (0xc0).

The checksum is formed by taking the 2’s complement of the 32-bit sum (ignoring carry) of all longwords of the header and image, excluding the checksum itself, as specified by the length field. If the length is not an integral number of longwords, the ‘missing’ bytes are set to 0xff (the unprogrammed state of bytes in flash ROM).

When the checksum is correct, a 32-bit sum (ignoring carry) of all longwords of the header and image, including any bytes required to round the length up to an integral number of longwords, will be 0. Block 0 of the flash will always contain image 0, the PBL/Angel image. It is not defined whether this image contains an image header. Images can have an image number between 0 and 0xff, but the PBL/Angel can only load and start image numbers 0-F.

Software that deletes an image in flash ROM should erase all the blocks used by that image. Software that programs an image in flash ROM should determine which blocks are free by checking each block for an image header and then ORing the allocation maps of all the valid image headers.

6.3 Angel

The version of Angel used on the EBSA-285 is a part of ARM's code. The full source code of Angel is supplied as part of ARM's SDT. The EBSA-285-specific source code is supplied on disk with the EBSA-285, along with the files needed to rebuild Angel from source. The standard version of Angel programmed into flash block 0 enables the SA-110 Icache and clock switching. Read the README.TXT file on the disk provided with the EBSA–285 for a description of the images supplied.

6.4 Diagnostics

These perform diagnostic tests that check that the system is operating correctly. The onboard diagnostics are normally programmed into flash block 1 and executed when image 1 is selected.

6.4.1 Preparing to Run the Diagnostics

The diagnostics expect to output results to the COM0 port, so this must be set up correctly. Set it to 9600 baud, 8 data bits, 1 stop bit, no parity and no flow control. Select image 1 and then reset or power-cycle the system.

6.4.2 Description of Tests

The diagnostics test a series of system functions and, when complete, print a summary of the tests run and their results. If any tests fail, the red LED is lit. If all of the tests are successful, the green LED is lit.

The tests run through the following steps:

1. The initial state at power up of the LEDs is all on. The diagnostic tests turn them all off, on and then off again with a short pause between. This proves that the I/O path through the X-Bus is working. After the tests, either the red LED will be lit to show that an error occurred or the green LED will be lit to show that the tests were successful.
2. A banner is written to the COM port announcing that the tests have started. This demonstrates that the COM port is working.
   ```
   ========================================================
   EBSA285 PowerOn Selftests
   ========================================================
   ```
3. Clock switching and the Icache are now turned on. At this point the diagnostics are running from flash; enabling clock switching and turning on the Icache will allow the diagnostics to run as quickly as possible.
   ```
   Enabling Clock Switching...Done
   Enabling I-Cache...Done
   ```
4. The diagnostics now initialize memory. If memory cannot be initialized, the red LED is lit and the tests halt.
   ```
   *** Initialising DRAM ***
   ```

5. Now that the memory is initialized, it can be tested:

```
*** Starting DRAM tests ***
Word write each word's address to itself
Test Passed
Word write each word's address to its top halfword
Test Passed
Byte write and read each byte; contents of each byte should be address mod 255
Test Passed
Word read the data written by the previous test
Test Passed
Store multiple tests starting
Test Passed
Load multiple tests starting
Test Passed
*** DRAM tests complete ***
```

6. Up to this point, diagnostic tests have been run using registers only, but now that the memory has been initialized and tested, it is available for use by test software. For this reason, the rest of the diagnostics is mostly written in C rather than in assembler. This part of the diagnostics is actually a separate image designed to be run from main memory:

```
DRAM size is 0x800000
EBSA285 Stub Code
Copying 0x13b8 bytes, from 0x410409b4 to 0x200000
####################
Jumping to C code...
============================================================
EBSA285 PowerOn Selftests (built Jun 06 1997, at 10:08:02)
============================================================
```

7. We are now running the C based diagnostic tests from memory and using a region of memory to hold the test results.

```
Using memory at 0x700000 for results
```

8. The diagnostics now run through a series of tests, each testing a different aspect of the system. The first test checks that the 21285's on board timers are functioning:

```
-------
Timer Tests [v1.0]
.....................................................................
Done
```

9. The next test checks that the flash ROM part is functioning correctly. It selects an unused flash block and writes a test pattern to it. It then reads back the test pattern to check that it was correctly written.

```
-------
Flash Tests [v1.0]
Searching for flash device
Flash found at 0x41000000 (0x10 blocks of size 0x10000
Scanning Flash blocks for usage
Listing Flash Blocks
0x0 *  0xeb00002e  0xaa5500  0x1  0xbfea8f6b Bootloader
0x1 *  0xeb00002e  0xaa5501  0x2  0x95a7bd94 post
0x2 *  0xeb00002e  0xaa5502  0x4  0x929c9fed Angel
0x3    0x3020100  0x7060504  0xb0a0908  0xf0e0d0c
0x4    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0x5    0x3020100  0x7060504  0xb0a0908  0xf0e0d0c
0x6    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0x7    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0x8    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0x9    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xa    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xb    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xc    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xd    0x3020100  0x7060504  0xb0a0908  0xf0e0d0c
0xe    0xffffffff  0xffffffff  0xffffffff  0xffffffff
0xf    0x3020100  0x7060504  0xb0a0908  0xf0e0d0c
Testing Block 0x3
Writing test pattern
Reading test pattern
Flash test worked
```

10. The final test confirms that the COM port can receive characters as well as print them:

```
-------
COM Port Tests [v1.0]
Type some characters followed by C/R
This is a test string
Did you see the characters echoed correctly? [yY/nN]? y
Done
```

11. Finally the test results and information about the system are printed. If any of the tests have failed, the red LED will be lit. If all tests were successful, the green LED will be lit.

```
=========================================================
EBSA285 PowerOn Selftests (built Jun 06 1997, at 10:08:02)
=========================================================
Memory size is 0x800000, CPU id is 0x4401a103
21285 device id is 0x1065, 21285 revision is 0x0
CPU Frequency is 228.1 MHz
Central Function Mode, PCI disabled
Flags: J17 pins 9-10 not fitted, J17 pins 11-12 not fitted,
       J17 pins 13-14 not fitted
Flash device at 0x41000000, 0x10 of size 0x10000
Image switch is 0x4
Test Results Summary
    Memory tests    Successful
    Timer tests     Successful
    Flash tests     Successful
    COM Port tests  Successful
=========================================================
Selftests complete, change boot selection before rebooting
=========================================================
```

Flash Management Utility 7

This chapter describes the Flash Management Utility (FMU). This program is supplied in source form and as both an ARM** Image Format (AIF) file and a DOS executable file.

7.1 Using the FMU

Images are programmed into flash using the Flash Management Utility (FMU). There are two versions of the FMU; an Angel* remote debugger loadable version (**fmu.aif**) and a DOS executable version (**fmu.exe**).

- The Angel loadable version uses the ARM debugger I/O services to provide a command-line interface.
- The DOS loadable FMU uses PCI BIOS services to access the EBSA-285 onboard flash device. During this access, the SA-110 must be held in blank programming mode (jumper on J15 pins 5-6).

When the FMU is started, it checks for the presence of a flash ROM, issues a start-up message and then prompts for user input:

```
Intel EBSA-285 Flash Management Utility [1.2] (Angel)
Searching for flash device
Flash found at 0x41000000 (16 blocks of size 0x40000)
Scanning Flash blocks for usage
FMU>
```

The FMU provides these commands:

- Help – List all of the available commands

```
FMU> help
FMU command summary:

List               - List images in flash
ListBlocks         - List how each Flash block is being used
TestBlock <block-number>
                   - Write a test pattern to a particular flash block
Delete <image-number>
                   - Delete an image in flash
DeleteBlock <block-number>
                   - Deletes a block that appears not to be in an image
DeleteAll          - Deletes all blocks except block 0
Program <image-number> <image-name> <file-name> [<block-number>] [NoBoot]
                   - Program the given image into flash
Quit               - Quit
Help               - Print this help text
```

- List – List the images in flash. For example:

```
FMU> list
Listing images in Flash
Image 0    “Bootld“   Length 45232 bytes, Map 0x00000001
Image 1    “Post“     Length 536 bytes, Map 0x00000002
```

  — You supply the image number and name when you program the image

  — The length shown is the size of the image including all headers

  — The map is a bit map showing which blocks of the flash are occupied by the image; bit 0 of the map corresponds to block 0 of the flash, and the image’s header is in the lowest block occupied by the image

  — You optionally supply the NoBoot option when you program the image

- ListBlocks – List how each flash block is being used. The first few bytes of the flash block are listed. If the block contains an image, its image number is given. For example:

```
FMU> listblocks
0: (Image 0) 0x2e 0x00 0x00 0xeb 0x00 0x55 0xaa 0x00
1: (Image 1) 0x02 0x00 0x00 0x00 0xe0 0xdd 0x21 0xc6
2: (Image 2) 0xd8 0x10 0x01 0x00 0x65 0x46 0x6f 0x72
3: (Image 2) 0x4c 0x0a 0x00 0x40 0x10 0x03 0x00 0x00
4: (Unused)   0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
5: (Unused)   0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
6: (Unused)   0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
7: (Unused)   0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
8: (Unused)   0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
9: (Unused)   0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
10: (Unused)  0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
11: (Unused)  0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
12: (Unused)  0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
13: (Unused)  0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
14: (Unused)  0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
15: (Unused)  0xff 0xff 0xff 0xff 0xff 0xff 0xff 0xff
```

- TestBlock <block-number> – Test a particular flash block by writing a test pattern to the block and then verifying it. For example:

```
FMU> testblock 15
Do you really want to do this (y/N)? y
Writing test pattern to block 15
Reading test pattern from block 15
Flash test of block 15 worked
```

- Delete <image-number> – Delete an image in flash. You cannot normally delete the flash image that starts in flash block 0 (the primary boot loader). The only time that the FMU utility permits you to do this is if the ARM remote debugger stub is executing from EPROM rather than flash. For example:

```
FMU> delete 3
Do you really want to do this (y/N)? y
Deleting flash blocks: 4
Scanning Flash blocks for usage
FMU>
FMU> delete 0
WARNING: Deleting flash boot block
Do you really want to do this (y/N)? y
Deleting flash blocks: 0
Scanning Flash blocks for usage
```

  If you are running an ARM remote debugger stub from an image other than image 0, you can delete that image, but the FMU will be terminated during the delete. If you restart the system, it will execute the PBL and run correctly.

- DeleteBlock <block-number> – Delete a block that is not part of an image. This may be used to clean out corrupt blocks, or blocks that have been programmed by the TestBlock command. The FMU will not allow you to delete a block that is part of a valid image. For example:

```
FMU> deleteblock 15
Do you really want to do this (y/N)? y
Delete flash block 15
Scanning Flash blocks for usage
```

- DeleteAll – Delete all blocks except blocks 0 and 1.
- Program <image-number> <image-name> <file-name> [<block-number>] [NoBoot] – Program the image with name <image-name> into the flash as image number <image-number>. The image is read from the host from file <file-name> (which may include a directory name). Refer to Section 7.1.1 for details of the block-number option and to Section 7.1.2 for details of the NoBoot option. The Program command will fail with an error if:
  - The image number is already in use.
  - There is insufficient free space in the flash.
  - The specified blocks are not free.
  - The file does not exist or cannot be opened.

  For example:

```
FMU> program 3 ledloop2 d:\users\crook\ledloop.aif noboot
Writing d:\users\crook\ledloop.aif into flash block 4
Deleting blocks ready to program:
Deleting block 4
Calculating checksum
Writing flash image header
Image is non-executable AIF file
The bootloader will copy this image to 200000 before executing it
Writing image file
Scanning Flash blocks for usage
```

- Quit – exit from the FMU. When this command is executed, the FMU will return control to the debugger.
- Exit – a synonym for Quit.

7.1.1 When to Specify the Block Number

By default, the FMU ‘Program’ command will program an image into flash using any free blocks allocated in ascending block order. This can result in an image occupying non-contiguous blocks within the flash.

When an image is a non-executable image (an image that will be loaded into system memory by the PBL prior to execution) the PBL will load an image from non-contiguous flash blocks into contiguous system memory. Therefore, allowing an image to occupy non-contiguous flash blocks makes efficient use of the flash by avoiding fragmentation problems.

When an image is an executable image, it must occupy contiguous blocks within the flash. In general, it must also have been linked to execute from a specific address (and therefore block) in the flash. Therefore, when using the FMU to program an executable image, you must specify the block-number when you issue the ‘Program’ command.

When a block-number is specified, the FMU will program the image into contiguous flash blocks, starting from the specified block. The command will fail if insufficient unused contiguous blocks are available.

Refer to Section 6.1 for information on the PBL, and to Section 5.2 for information on building images that can be executed from flash.

7.1.2 When to Specify the ‘NoBoot’ Option

The usual reason to program an image into flash is so that it can be automatically executed after reset or power up. If the image number is less than 0xF, the flash image select switch can be set so that the PBL will load and execute the image after a reset or power up.

Sometimes, it is desirable to have the PBL load the image into system memory but then drop into the ARM remote debug stub. This process allows the image to be started up under the control of the debugger, to use the I/O facilities of the debugger and ultimately to pass control back to the debugger when the image terminates.

If you use the NoBoot option when programming an image into flash, the PBL will load the image into system memory but will not execute it; instead, control will pass to the ARM remote debug stub within image 0.

There is no way to change the state of the NoBoot flag for an image once it has been programmed; you must delete the image and reprogram it with the NoBoot flag changed.

Refer to Section 6.1 for information on the PBL and the flash image select switch.

Theory of Operation/Hardware Design 8

8.1 General

This chapter provides a technical description of the EBSA-285 hardware and explains various trade-offs made in the design. It should be read in conjunction with the 21285 data sheet and the EBSA-285 schematic set (the schematic set is provided as part of the EBSA-285 design database - refer to Appendix B). You should read this chapter if you wish to gain a detailed understanding of the operation of the card or if you wish to design a board based on the 21285. You are assumed to:

- Have a background in high-speed digital design
- Have some familiarity with the ARM** architecture, the SA-110 bus interface and the 21285 data sheet
- Have access to the manufacturer's data sheets for the flash, SDRAM and other components used on the card

This chapter is organized so that each hardware-related section of the Functional Specification (Chapter 2) has a corresponding section here in which the topic is covered in more detail. In addition, this chapter includes:

- A topic-by-topic tour of the EBSA-285 schematics, including a description of the principal buses
- A discussion of how an expansion card could be designed for the EBSA-285
- A summary of the design rules used for the PCB layup and routing

8.2 An Introduction to the Schematics

Specific sheets of the schematic set are referenced by sheet number (for example, SHT6). The sheet number is shown in the bottom right corner of the schematic.

The first sheet of the schematics (SHT1) is an index to the remaining sheets. The block diagram (SHT2 of the schematics, (simplified in the body of the manual as Figure 2-1) shows all the major components of the design, and provides a cross-reference to the location of any particular functional block within the schematic set.

On the schematics, every signal has a three-letter prefix[1] that indicates the origin (driver) of the signal. For bi-directional signals, the 'most important' driver of the signal determines the prefix.

1. There are a few exceptions, but they should not cause confusion.

8.3 Voltage Domains

The integrated circuits on the EBSA-285 use a mixture of 5 V and 3.3 V switching levels.

The parts of the circuitry requiring 3.3 V are:

- SA-110 (SHT3)
- SDRAM DIMMs (SHT11, 12)
- EPROM emulator socket (SHT8)

The parts of the circuitry requiring 3.3 V but are 5 V-tolerant are:

- 21285 (SHT5)
  This allows the 21285 to be used on universal cards and the oscillator to be interfaced directly.
- Flash ROM (SHT8)
- LVT buffers (SHT6, 7)

The parts of the circuitry requiring 5 V are:

- Oscillators (SHT3, 5 13)
- JTAG (SHT18)
- PCI (SHT14)
- X-Bus Expansion (SHT4)
- Reset Circuitry (SHT18)

Signals that are generated with 5 V switching levels must be level converted before they can be used as inputs to the SA-110. This affects the following signals:

- JTAG port (3v3_tdi, 3v3_tms, 3v3_trst_l, 3v3_tck)
- SA-110 oscillator (un_3v3_osc3)
- Reset (flash_rst_l)

The level conversion is performed using a QuickSwitch* QS3384 (SHT3).

8.4 Interfacing Techniques

The following interfacing techniques are used on the card:

- 74LVT devices are used as level converters, these parts have a 3.3 V supply but are 5 V-tolerant. Their output switching range is within the TTL switching threshold, hence they can drive the TTL level devices powered from the 5 V rail.
- Output signals from 3.3 V devices can be used to drive TTL directly. This method is used for the bus control logic and on the X-Bus header control logic.
- A Quality Semiconductors ‘QuickSwitch’ device is used for level conversion (equivalent pin-compatible devices are available from Texas Instruments and National Semiconductors). The QS3384 acts as a set of bi-directional FET switches. It introduces negligible delay (250 ps). Since the FET switches saturate, the switching level can be controlled by controlling the saturation (supply rail) voltage. With the QS3384 powered at 4.3 V, the driven output will be limited to 3.3 V, even under light loading.

8.5 Principal Buses

The principal internal buses in the EBSA-285 design are:

- cpu_a[31:2] – CPU Address bus

  This bus connects the SA-110 to the 21285 and also connects to the flash ROM. A sub-set of this bus is buffered to generate the X-Bus address bus. The SA-110 uses this bus to drive addresses for all of its accesses (to SDRAM, flash ROM, X-Bus and the 21285). The 21285 uses this bus to drive addresses for PCI accesses to the flash ROM. Byte resolution is provided by the byte lane enables cpu_be[3:0] (SHT3) for CPU accesses. This bus has 3.3 V switching levels and is not 5 V-tolerant (because the SA-110 pins are not 5 V-tolerant).

- buf_a[11:2] - X-Bus address

  This bus is generated from cpu_a[11:2] via a 74LVT16244 (SHT6). The buffer is permanently enabled. This bus has 3.3 V switching levels but is 5 V-tolerant.

- cpu_d[31:0] - CPU data bus

  This bus is connected to the SA-110, the 21285 and the flash ROM. This bus is buffered to generate the SDRAM and X-Bus data buses. This bus has 3.3 V switching levels and is not 5 V-tolerant.

- buf_d[31:0] - Buffered Data bus

  This bus is generated from cpu_d[31:0] via a 74LVT16245 (SHT6). The buffer has the output permanently enabled with the direction of flow governed by fbg_dwren_l. The direction pin on the 74LVT16245 has the A-B data path active when the signal is high. This means the CPU data bus must connect to the ‘B’ side because fbg_dwren_l is an active-low signal. This bus has 3.3 V switching levels but is 5 V-tolerant.

- xbuf_buf_d[15:0] - Buffered X-Bus Data bus

  This bus is generated from cpu_d[15:0] via a 74LVT16245 (SHT7). The buffer has the output permanently enabled with the direction of flow governed by fbg_xd_wren_l. The direction pin on the 74LVT16245 has the A-B data path active when the signal is high. This means the CPU data bus must connect to the ‘B’ side because fbg_xd_wren_l is an active-low signal. This bus has 3.3 V switching levels but is 5 V-tolerant.

- fbg_ma[12:0] - SDRAM address bus

  This bus is driven by the 21285 to provide addresses for all SDRAM accesses. At power up some of these signals act as inputs to the 21285 to allow configuration information to be latched by the 21285. The signals have 3.3 V switching levels and are not 5 V-tolerant. This bus is buffered before driving the SDRAM arrays.

- buf_ma[12:0] - Buffered SDRAM address bus

  This bus is generated from fbg_ma[12:0] via a 74LVT16244 (SHT6). The buffer is permanently enabled. The signals have 3.3 V switching levels and are not 5 V-tolerant (because the SDRAM pins are not 5 V-tolerant). This bus drives the two SDRAM DIMM sockets.

8.6 CPU

The EBSA-285 uses the SA-110 microprocessor (SHT3). The card allows the processor to be operated at any one of its 16 core clock frequencies (between 88.3 MHz and 287 MHz with the upper limit determined by the speed grade of the CPU fitted) at a fixed core voltage of 2 V. Core frequency is selected by a series of jumpers (SHT17).

The SA-110 has 3.3 V I/O but is not 5 V-tolerant so requires that any 5 V signals be level shifted. The SA-110 is used in the following modes:

- Asynchronous bus mode (SnA low and MCLK as input)
- Enhanced bus mode (CONFIG high)
- APE high mode (Non-Fastbus mode)

The CPU is packaged in a 144-pin thin quad flat pack (TQFP).

8.7 21285

The 21285 (SHT5) is a 3.3 V device in a 256-pin plastic BGA package, it has 3.3 V switching levels but has 5 V-tolerant inputs. It interfaces to the PCI, SA-110, SDRAM and X-Bus, providing control signals for buffer control for SDRAM and X-Bus accesses and control signals for CPU accesses to flash ROM or EPROM emulator. Due to the large number of outputs with high speed edges the 21285 is sensitive to the amount of power plane decoupling it receives. The EBSA-285 uses eighteen 0.1 µF, two 47 µF, one 10 µF and seven 0.01 µF decoupling capacitors. These are arranged as close as possible to the 21285 on the bottom side of the EBSA-285 PCB.

8.8 SDRAM Interface

The SDRAM interface consists of:

- Multiplexed address bus
- Data bus
- CMD interface
- Chip selects
- Bank selects
- Byte selects
- Clocks

All these signals have relatively high switching frequencies with fast rise and fall times, and each signal drives a number of devices. To ensure that the maximum number of devices (32 SDRAMs on two DIMMs) can be driven while maintaining the integrity of the signals, each of the signals is buffered and has a 33.2 Ω series termination resistor (SHT20). The series termination resistors are placed close to the output of the driver (so that there is less than 3 cm of etch between the output of the buffer and the resistor).

8.8.1 Multiplexed Address Bus

The multiplexed address bus provides the SDRAMs with multiplexed row and column data. A 13-bit bus is provided by the 21285 ma[12:0], each DIMM (SHT11, 12) has a 14-bit bus ma[13:0] so the most significant bit is tied to ground via a pull-down resistor (SHT11). The bus is buffered by a 74LVT16244 (SHT6) and has a 33.2 W series termination resistor on each line (SHT20).

8.8.2 Bank Address Bus

Each SDRAM DIMM can have multiple arrays, each comprising two or four banks. The 21285 can address four banks of SDRAM by using the Bank Address bus fbg_ba[1:0]. Both signals can be used on the primary DIMM socket J12, so this socket can contain a four bank DIMM. The secondary DIMM socket only uses fbg_ba[1], so can only contain a two bank DIMM, and only if there is not a four bank DIMM in the primary socket.

8.8.3 Data Bus

The data bus is the buffered CPU data bus buf_d[31:0] (SHT6). The data bus is not series-terminated because it is a bi-directional bus.

8.8.4 CMD

The CMD outputs from the 21285 generate command information to the SDRAMs. Each line is buffered via a 74LVT16244 (SHT7) and has a 33.2 W series termination resistor (SHT20).

buf_cmd0: This line is connected to /WE on the DIMMs.

buf_cmd1: This line is connected to /CAS on the DIMMs.

buf_cmd2: This line is connected to /RAS on the DIMMs.

The 21285 defines the CMD signals to be active-high, but their behavior is such that they can connect to the SDRAM /WE, /RAS and /CAS signals directly (no inversion is required).

8.8.5 Chip Selects

There are four chip selects, buf_cs[3:0], each is buffered via a 74LVT16244 (SHT7) and has a 33.2 W series termination resistor (SHT20). Each chip select is used to select a single SDRAM array, so a maximum of four arrays may be used in a system. Byte Selects buf_dqm[3:0] are used to provide byte lane information to an SDRAM array. These signals allow each byte within a 32-bit longword to be accessed, either individually or together. Each signal is buffered via a 74LVT16244 (SHT7) and has a 33.2 W series termination resistor (SHT20).

8.8.6 SDRAM Clocks

All the SDRAM clocks, fbg_sdclk[3:0], have matched lengths and impedances with each other and with the 21285 SDRAM clock reference net, fbg_fclk_o. The overall length of the nets have been kept as short as possible to maintain signal integrity. The clock reference net is used to minimize the effect of skew on the SDRAM clock nets, to give the 21285 a reference for all SDRAM transactions. Each of the lines has series termination of 33.2 W. An active-high clock enable signal is provided for each pair of clocks. These pins should be used to enable the SDRAM clocks going to used SDRAM arrays and to disable the clocks going to unused or missing arrays.

The 21285 SDRAM clock outputs have high-current drivers and must not be buffered externally; this would introduce too much skew into the SDRAM timing.

8.8.7 DIMMs

The EBSA-285 uses 168-pin 64-bit DIMMs. The 64-bit memory on the DIMM is used to provide two arrays of 32-bit wide memory. This leads to an unorthodox wiring scheme for the DIMMs. The DIMM sockets are physically large and need to be near the 21285 because of the high speed signalling between them. The sockets are placed along the top edge of the EBSA-285 as a compromise between space and signalling constraints.

The serial presence detect, which normally uses the I^2C bus, is not used on this design. The unused I^2C signals are tied high and the I^2C address lines, SA[2:0], are tied low.

The DIMMs that are fitted must use 3.3 V switching levels.

The 4Mx64 part contains four arrays of 32-bit memory and can only be placed in the first DIMM socket. The 2Mx64 part contains two arrays of 32-bit memory and may be placed in either of the 2 DIMM sockets.

An array consists of the parts that make up a 32-bit wide section of memory selected by a single chip select signal. Each array must use a single chip select line that is unique to that array. Table 8-1 and Table 8-2 show the relationship between the chip select lines, the DQM lines, clk lines and the data byte for that combination.

Table 8-1. SDRAM Array Configuration: 2-Array Part

Array	Byte	CS	DQM	DQ	CLK
1	0	0	0	DQ0-DQ7	0
	1	0	1	DQ8-DQ15	1
	2	0	4	DQ32-DQ39	2
	3	0	5	DQ40-DQ47	3
2	0	2	2	DQ16-DQ23	0
	1	2	3	DQ24-DQ31	1
	2	2	6	DQ48-DQ55	2
	3	2	7	DQ56-DQ63	3

Table 8-2. SDRAM Array Configuration: 4-Array Part

Array	Byte	CS	DQM	DQ	CLK
Arrays 1 and 2 are as shown in Table 8–1.					
3	0	1	0	DQ0-DQ7	0
	1	1	1	DQ8-DQ15	1
	2	1	4	DQ32-DQ39	2
	3	1	5	DQ40-DQ47	3
4	0	3	2	DQ16-DQ23	0
	1	3	3	DQ24-DQ31	1
	2	3	6	DQ48-DQ55	2
	3	3	7	DQ56-DQ63	3

The SDRAM draws a large amount of operating and refresh current (300 to 400 mA) at switching frequencies of up to 55 MHz. This requires that careful consideration is paid to the decoupling capacitors around the DIMMs. Each DIMM has local decoupling on the DIMM. Four 22 µF capacitors placed close to each DIMM socket provide decoupling on the EBSA-285.

8.9 Flash ROM Interface

The EBSA-285 uses four Intel TSOPII 28F008 flash ROM parts (SHT8). The PCB footprint is designed so that they could also accommodate the larger 28F016 parts if desired. The flash ROMs are 8-bit devices, arranged to provide a 32-bit data path.

Flash ROMs are connected directly to the unbuffered CPU address and data buses (cpu_a[22:2] and cpu_d[31:0]). During all ROM accesses, the 21285 drives output-enable on cpu_a30 and write-enable on cpu_a31, so these address lines are wired to the appropriate signals on the flash ROMs. When the 21285 is accessing the flash ROM the CPU address bus drivers are placed in a high impedance state by the 21285 asserting the ABE signal.

A 12 V supply is required for programming the flash ROM. This is the only place that 12 V is used on the board, and it is only used during programming of the flash. If programming of the flash ROM is not required, the EBSA-285 can be powered without 12 V.

The flash ROM requires 3.3 V switching levels.

The socket is provided to allow an EPROM emulator to be connected to the card for rapid program development. It is a single 32-pin DIL part. Writing to this memory area is permitted (the write enable signal, cpu_a31, is routed to the socket) but the emulator must support this mode. The EPROM socket is wired to provide a byte wide memory; cpu_a[18:2] are wired to the socket lines a[18:2] and cpu_a[29:28] are wired to a[1:0].

The 512 KB of the EPROM socket is mapped in place of the flash ROM but is byte wide rather than 32-bit wide. The socket should not be used in half-word or longword modes and it cannot be used when the flash ROM is active.

Two jumpers (SHT17) control the selection of flash or the EPROM socket.

- J15 pins 22/23/24 route fbg_rom_ce_l to either the flash or EPROM (see Appendix A.2.2.3.). The unselected signal line is held high by a pull-up resistor.
- J15 7/8/9 either pull fbg_ma4 up (flash) or down (EPROM socket).

  Both sets of jumpers must be changed together. To effect the new selection, the board must then be power-cycled.

When the EBSA-285 is reset the flash ROM is also reset. This places the flash in a known state at reset or power up. Resetting the flash halts any automated write/erase cycles that the flash is performing. This avoids a read being made from flash during an automated cycle. If that were to occur, the flash would provide status information. This would prevent the CPU from booting correctly. Flash reset is covered in Section 8.17.

8.10 X-Bus Interface

The X-Bus interface allows the connection of low speed ISA style peripherals to the EBSA-285. It provides a 16-bit data bus, xbuf_buf_d[15:0], and 10-bit address bus, buf_a[11:2]. Control signals are provided for reset, read and write strobes, buffer direction control and chip selects. The bus is unclocked and does not provide a means to stall the bus.

8.10.1 Soft I/O

The 21285 XCS2 region is used to decode the onboard soft I/O. This (see SHT7) consists of an 8-bit output latch, 74ABT377 and an 8-bit input port, 74ABT541. The signal levels are 5 V.

The assignment of the input and output port is described in Section 3.6. Writing to anywhere within the XCS2 area on a 32-bit boundary will write to the 74ABT377. Reading from anywhere from within the XCS2 area on a 32-bit boundary will read from the 74ABT541.

With the clock enable signal active, the outputs of the 74ABT377 are latched on the trailing (rising) edge of the clock pulse. The clock enable pin is driven by fbg_xcs2_l and the clock pin is driven by the write strobe fbg_xiow_l. The latched states of obuf_d[2:0] drive the three, bulkhead LEDs, while obuf_d7 is looped back to the input port for diagnostic test of the soft I/O.

The 74ABT541 used for input is an octal buffer with 3-state outputs and two output enable signals. Both enables must be asserted for the buffer to drive its outputs. The output enable signals are wired to the fbg_xcs2_l and fbg_xior_l lines. In addition to the output monitoring function of obuf_d7, input lines lnk_soft[13:10] hold the state of the flash image selector switch, while ibuf_d[6:4] monitor jumpers on J17.

8.10.2 X-Bus Expansion Headers

The X-Bus expansion provides the X-Bus data, address and control signals to 5, 2 x8 0.1 inch pitch headers (SHT4). It also provides 5 V and ground to the header. This expansion option is intended to provide an interface for additional low performance PC-AT type, 8- or 16-bit peripherals. The signals provided are:

- buf_a[10:2]
- buf_be[1:0]
- xbuf_buf_d[15:0]
- flash_rst
- flash_rst_l
- xd_wren_l
- fbg_xior_l
- fbg_xiow_l
- fbg_xcs2
- xbus_xcs1_l
- xbus_xcs0_irq_l
- vdd (5V)
- gnd (0V)

The pinout of the connectors is shown in Appendix A.2.1.

8.11 The Serial Port

COM0 (SHT21) is a minimal serial port with just Rx and Tx lines. The data signals are shifted between RS232 and internal (5 V) signal levels by a MAX211 RS232 driver/receiver.

EMC radiation is inhibited by a 220 pF capacitor between the Tx line (**con_tx**) and ground.

8.12 Interrupts

The EBSA-285 is designed to work in a number of modes so that all major modes of the 21285 can be evaluated. This means there are a number of different sources of interrupt depending upon which mode the card is placed in:

- Central function mode requires that system wide interrupts be handled by the EBSA-285, which means that four interrupt lines (to service PCI interrupts d:a) are required when in this mode.

  The 21285 has four general-purpose interrupt pins (IRQ_IN[3:0]) and a single interrupt pin (PCI_IRQ) that is an input in central function mode and an output in non-central function mode. To service the four PCI interrupts, pci_int[d:b]_l are routed to IRQ_IN3, 1 and 0, and pci_irq_l performs the function of PCI_INTA#.

- When the 21285 is not the central function it must request an interrupt from the central function on the request line, pci_irq_l (the PCI_IRQ pin is an output in this mode).

- In both central function and non-central function modes the EBSA-285 must handle:

  — An interrupt from a fixed frequency timer (to provide ticks of fixed period for calibration)

  To provide the required timing reference, the 3.68 MHz oscillator signal (fbg_timer) is connected to the IRQ_IN2 pin, this is used to clock timer 3 of the 21285.

  — Interrupts from the X-Bus expansion interface

  The X-Bus chip select pins, XCS[2:0], can be configured as interrupt inputs. XCS0 (connected to xbus_xcs0_req_l) should be used as a COM port interrupt. Software should be written so that this line is used as an input and not as a chip select line. The interrupt outputs from both COM1 and COM2 should be OR’d together and wired to this pin on any expansion cards that provide serial communication ports.

8.13 PCI Interface

The PCI bus is a full *PCI Local Bus Specification*, Revision 2.1, 32-bit, 33 MHz compliant interface. The bus is provided on a standard set of PCB fingers to allow it to be used as a plug-in card in a standard PCI socket. The connector is of the universal type so can be plugged into both a 3.3 V system and 5 V system. Signalling levels used are 3.3 V but the logic is 5 V-tolerant. When used as a plug-in card the EBSA-285 will draw power from the PCI socket on the 5 V fingers. High speed decoupling (12 x 0.01 µF) is provided on the unused 3 V supply fingers to provide a good 3 V signal return path.

When the card is used as a central function, a number of the reserved pins are used to provide the four sets of interrupt, request and grant pins required. The use of these pins follows that proposed by the PICMG standard. The PCI pins are used as shown in Table 8-3.

Table 8-3. Use of Reserved PCI Pins

Pin	Not Central Function	Central Function
B9	PRSNT #1	pci_req3_l
B10	Reserved	pci_req1_l
B11	PRSNT #2	pci_gnt3_l
B18	pci_req_l	fbg_pci_req_l
A6	pci_irq_l (Output)	pci_inta_l (Input)
A14	Reserved	pci_gnt1_l
A17	fbg_pci_gnt_l	fbg_pci_gnt0_l
A19	Reserved	pci_req2_l
A26	idsel	pci_gnt2_l
A15	pci_rst_l (Input)	pci_rst_l (Output)

When the card is not acting as PCI-Bus arbiter it must be set in the X-Bus mode by setting the appropriate jumpers (SHT17).

8.14 PCI Bus Arbiter

The PCI arbiter can be used when the EBSA-285 is configured as the central function in a system. It interfaces via the PCI interface giving four system wide interrupts and four pairs of request and grant lines. For further details see Section 8.13.

8.15 JTAG

The JTAG port (SHT18) provides a connection for the JTAG interface. This is used for boundary scan testing of the SA–110 and the 21285, which are connected in the TDO/TDI ring. It can also be used to program flash memory. A system reset, srst_l, allows the system to be reset by a device connected to the JTAG port.

A pair of 74ACT244 buffers isolate the logic from direct connection to JTAG signals (except for srst_l, which is not buffered). The TTL outputs from the buffers are level-shifted to 3.3 V by a QS3384 (SHT3).

8.16 Clocks

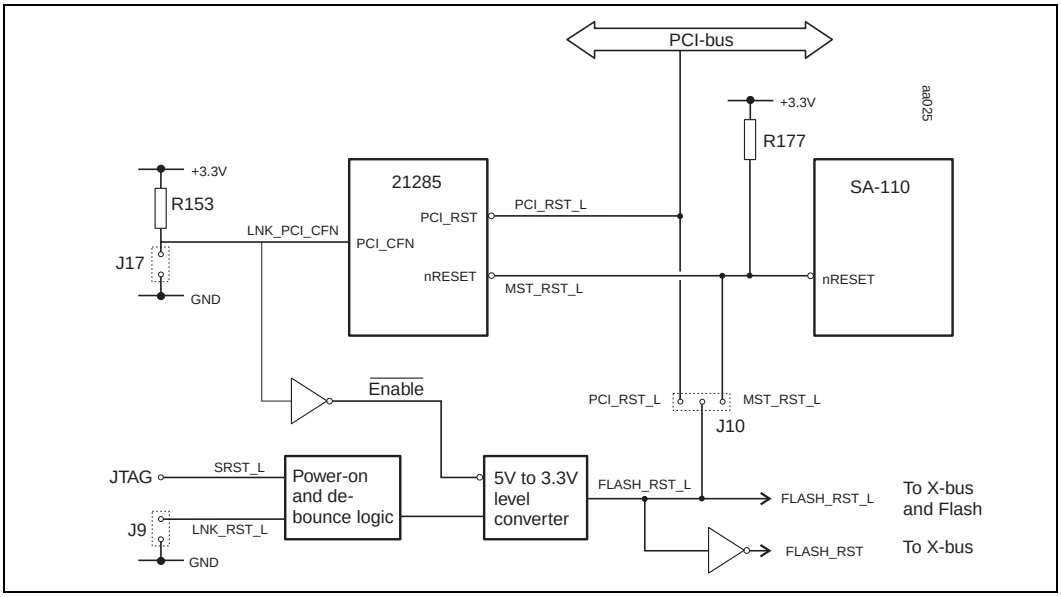
The EBSA-285 uses the following oscillators:

- 3.6864 MHz TTL compatible oscillator (SHT3): The oscillator is level converted to 3.3 V (SHT3) then drives the SA-110 PLL input and a timer in the 21285 through a pair of series termination resistors (SHT20). These series termination resistors should have impedance matched etch lengths. The PLL on the SA-110 generates the core clocks according to the settings on the CCCFG[3:0] links. The 21285 timer can be used to provide timer ticks to an operating system.
- 24 MHz TTL compatible oscillator (SHT13): When used in a backplane the link connecting this oscillator to pci_clk must be removed.
- 50 MHz TTL compatible oscillator (SHT5): This drives the 21285 core clock via a series resistor (SHT20). Provision has been made in the etch for a 50 W termination resistor to allow replacement of the oscillator with a frequency generator during device verification.
- The 21285 generates six clock outputs for driving FCLK (fbg_fclk_o), MCLK (fbg_mclk) and four SDRAM arrays (fbg_sdclk_[3:0]). The impedance and etch length of all these signals is identical, providing minimum skew for clock equalization.
  - — FCLK is looped to FCLK_IN.
  - — MCLK on the 21285 drives MCLK on the SA-110. The 21285 stops the clock in the high state in order to introduce stalls in SA-110 cycles.
  - — The SDCLK[3:0] pins are connected to CK[3:0] on the SDRAMs. All have matched lengths and loads, each signal being terminated by a series resistor of 33.2 Ω. SDCLK3 is used to clock the onboard array (parity or unbuffered memory depending on build option) and one of the DIMM arrays.

8.17 Reset

There are a number of sources of reset (see Figure 8-1) on the EBSA-285. Depending upon the mode of the card, resets behave differently.

Figure 8-1. Reset Circuits



8.17.1 Host Bridge

In this mode, the 21285 is configured to be central function; the EBSA-285 is responsible for providing the system wide reset. All resources on the EBSA-285 must be reset and the PCI bus must be reset by asserting pci_rst_l. The resources to be reset on the EBSA-285 are:

- SA-110
- 21285
- Flash ROM
- X-Bus interface

The sources of reset in this mode are

- Automatically at power on (SHT18)
- By a push button (SHT17 and 18)
- Under remote control, by a debug box attached to the JTAG connector (SHT18)
- PCI reset under software control
- By the 21285: to reset the complete system via pci_rst_l
- On a time-out from the watchdog timer, WDT

8.17.1.1 Power-On Reset

The power on reset circuitry is used to ensure a graceful power up for the EBSA-285. It is generated from an R-C network and schmidt trigger arrangement. When power is applied to the EBSA-285, VDD is ‘soft started’ and de-bounced by the R-C network and schmidt triggers to generate a 5 V active-low signal 5v_flash_rst_l. This signal is then level converted to 3.3 V by the QS3384 (SHT3) to become flash_rst_l. This signal, in central function mode, will become mst_rst_l via a jumper (SHT17).

mst_rst_l is the master reset for the EBSA-285 in central function mode. It is used to reset the SA-110 (SHT3) and the 21285 (SHT5).

The flash is reset by flash_rst_l.

The X-Bus expansion uses an active-high version of flash_rst_l which is generated (SHT3) by using an inverting schmidt buffer. This is a 5 V level part so the active-high version flash_rst has 5 V signalling levels. This is necessary because some Super I/O chips have an active-high reset pin as a legacy from the ISA bus.

8.17.1.2 Switch Reset

A 2-pole 0.1 inch pitch connector is provided to allow an external normally-open switch to be added to generate lnk_rst_l. In a lab environment a reset can be generated by shorting the two poles. The signal is de-bounced by the same circuitry as the power up reset to ensure a clean reset pulse.

8.17.1.3 JTAG Connector Reset

The signal srst_l comes from an external device via the JTAG port. Unlike the standard JTAG signals, it is not buffered. This signal uses the same de-bounce circuitry as the switch reset.

8.17.1.4 Watch Dog Timer Reset

If the WDT times out and causes a reset, there needs to be a system wide reset. A WDT reset occurs when the WDT is enabled in the 21285 and the CPU allows the timer to time out, which should only happen during a software failure.

On a WDT reset the 21285 asserts mst_rst_l. In central function mode this will reset the SA-110 and reset the flash ROM via the jumper by asserting flash_rst_l. This signal will be inverted by a schmidt trigger and will reset any active-high peripherals on the X-Bus expansion headers.

The QS3384 output acts as an open drain output and so will not back-drive the reset circuitry.

8.17.1.5 PCI Reset

On power up the 21285 holds pci_rst_l asserted so holding the system, with the exception of the EBSA-285, in reset. To de-assert pci_rst_l the SA-110 must set a bit in a 21285 control register.

8.17.2 Add-in Card

In this mode the 21285 is configured to run in non-central-function mode; an external host bridge provides the system wide reset. The EBSA-285 resources must be reset on an assertion of pci_rst_l. The 21285, flash ROM and X-Bus interface are reset directly by pci_rst_l but the SA-110 is reset by a mst_rst_l generated by the 21285. This can be used to hold the CPU in reset while the flash ROM is being programmed.

8.17.2.1 PCI Master Reset

When a PCI master asserts reset the EBSA-285 should be held in reset. In this mode pci_rst_l is an input to the 21285. When it is asserted the 21285 asserts mst_rst_l which resets the SA-110. The flash ROM and X-Bus are reset by pci_rst_l, which is wired to the local flash_rst and flash_rst_l by jumper J10(SHT17).

8.17.2.2 Blank Programming Mode

When the EBSA-285 is placed in blank programming mode the CPU is held in reset by mst_rst_l, which is generated by the 21285, while the system programs the flash ROM with an image. When the programming is complete the system needs to be powered down and changed from blank programming mode.

8.18 Power

The EBSA-285 requires 5 V and 12 V from an external source. Power comes onto the card either through a PC-style 12-way connector (SHT13) or through the PCI fingers. The EBSA-285 never draws power from the 3.3 V PCI fingers, even when it is plugged into a 3.3 V PCI connector. Onboard regulators generate 3.3 V and 2 V from 5 V.

Most of the devices on the card use 3.3 V, which is the main power plane on the EBSA-285. 3.3 V is generated by a Maxim MAX767 regulator, providing 3 A at 3.3 V.

5 V is used by the soft I/O and the oscillators.

The current requirement from the 3.3 V rail is as follows:

- SA-110 (worst case 430 mA)
- The 21285
- SDRAM (worst case all refresh @ 3x480 mA)
- Flash ROM (operating current @ 4x50 mA)
- 7 LVT buffers (7x5 mA)

8.18.1 3.3 V Generation

The 3.3 V rail is regulated from the 5 V rail by a Maxim MAX767 (SHT15), which is a 5 V to 3.3 V synchronous step-down power supply controller.

The 3.3 V output of the MAX767 is generated by an internal, current mode, pulse width modulation (PWM) step-down regulator. The PWM regulator is configured to operate at 200 KHz, which provides maximum conversion efficiency for the MAX767. The MAX767 also has a 3.3 V, 5mA reference voltage on pin 8. Fault protection circuitry shuts off the output should the reference lose regulation or the input voltage go below a nominal 4 V.

The MAX767 requires a few external components to convert the 5 V supply into an accurate, regulated 3.3 V output. External components include two N-channel MOSFETs, a rectifier, and an LC output filter. The gate drive signal for the high side MOSFET, which must not exceed the input voltage, is provided by a boost circuit that uses a 0.1 µF capacitor. The synchronous rectifier keeps efficiency high by clamping the voltage across the rectifier diode. An external low-value current sense resistor sets the maximum current limit. An external capacitor of 0.01 µF sets the programmable soft start, reducing in-rush surge currents upon start up.

8.18.2 2.0-V Generation

The 2 V power is used to power the SA-110 core. 2 V is regulated from the 3.3 V rail by a Linear Technology LT1086 (SHT15). The LT1086 is a low dropout regulator with an adjustable output. The output voltage is set to 2 V by R166 and R178.

8.18.3 Power Sequencing

The SA-110 requires two voltage supplies; a 3.3 V supply to power its primary input/output buffers and a 2 V supply to power its core. The 3.3 V supply must be stable earlier than the core voltage. This requirement prevents any latch-up within parasitic structures on the SA-110, and is satisfied by deriving the core voltage from the 3.3 V supply. This is the only power sequencing restriction.

8.19 Decoupling

The EBSA-285 uses tantalum electrolytic capacitors for bulk decoupling of the power rails, and ceramic capacitors for decoupling of individual ICs. The bulk decoupling capacitors, which are a mixture of 10 µF and 47 µF parts, are evenly distributed around the card. The individual decoupling capacitors are 0.1 µF and are located close to the power and ground pins of the ICs that they decouple.

The 21285 high speed clocks and the PCI connector have local high speed decoupling in the form of 0.01 µF capacitors.

Each SDRAM DIMM has four 22 µF decoupling capacitors as well as the on-DIMM decoupling and bulk decoupling.

There are twelve 0.01 µF decoupling capacitors between the PCI 3.3 V fingers and ground. This is to provide an ac return and is required even though the 3.3 V fingers are not used to supply power.

8.20 Jumpers and Test Points

A range of jumpers and test points provide for configuration and monitoring of the EBSA-285. Details are supplied in Appendix A, and on SHT17 of the circuit diagrams.

8.21 Expanding the EBSA-285

The buffered X-Bus interface provides all the X-Bus signals from the 21285, a buffered version of the data bus, a buffered subset of the address bus A[11:2], plus ground rails and 5 V for the expansion card. The format of the connectors allows the use of a mezzanine card that plugs into the 8x2 0.1 inch headers. The interface can be used for PC type ISA-style peripherals. Address lines may be used for self-decoding devices and to decode internal register access. There is enough address information to place a PC style Super I/O chip on the expansion bus.

An X-Bus chip select, fbg_xcs2_l is used to decode the onboard Soft I/O. Chip select xbus_xcs0_irq_l is available as an interrupt for any COM ports placed on the X-Bus, while xbus_xcs1_l is programmed as a disable/enable line for self decoding devices.

The X-Bus does not have a clock, so if a peripheral requires a clock then one should be provided on the expansion card.

8.22 The Printed Circuit Board

The printed circuit board is an 8-layer board with six signalling layers, a 3.3 V power plane and a ground plane. It is routed using 0.005 inch track and gap rules with a nominal etch impedance of 88 Ω for the outer layers and 58 Ω for the inner layers.

The power plane is not split despite the different power requirements of the components. This was done to maintain signal integrity. Power other than 3.3 V was routed as a signal but with the largest copper area as possible, so ignoring the track sizing of other signals.

During layout, the critical components were placed first. The 21285 needed to be near the PCI fingers to satisfy the etch length requirements of the PCI specification. The SA-110 was placed close to the 21285 due to the large number of signals passing between the two components. The pinout of the 21285 is such that there is a logical flow of signals between the two devices.

The buffers for the main buses and control signals were placed as close to the 21285 and SA-110 as possible. This is because the signals are unterminated and should, therefore, be kept as short as possible to maintain signal integrity. Where series termination resistors were needed, they were placed as close to the outputs of these buffers as possible. The address, data and SDRAM buffers were placed first.

The SDRAM DIMMs had to be placed along the top edge of the card because of their physical size.

The oscillators were placed as close to the output destination as possible, so the 50 MHz crystal was placed near the 21285 and the 3.68 MHz crystal as close to the SA-110 as possible.

The switch mode power supply was placed as close to the PCI fingers as possible so that the 5 V from the PCI did not need to be routed too far. The power connector for an off card PC style power supply was then placed as close to the switch mode supply as possible. The placement of the components for the switch mode power supply followed the recommendations made in the data sheet for the Maxim MAX767 part.

LEDs, image selection switch and serial port, which must be accessible to the user, were placed along the bulkhead edge so that they could be accessed from outside a PC style chassis.

The remaining components were placed where they were most convenient.

The first nets to be routed were the various clocks, SDRAM and the SA-110, making sure that they were impedance and length matched. Any test points that were to be placed on these nets were placed as large vias at appropriate points. Test point etch lengths were kept to a minimum.

The oscillator outputs were series terminated at source and then routed to keep the etch length to a minimum.

The signals between SA-110 and 21285, and the signals between the 21285 and PCI fingers were routed next as they were short, and simple to route.

Next to be routed were the other high speed SDRAM signals, buf_dqm[3:0], buf_cmd[2:0], buf_cs[3:0] and buf_ma[12:0].

The power signals, 2 V, 12 V and Vdd, were placed on the outer layers and had the copper area used as large as possible.

All un-terminated signals were kept as short as possible, placing the termination resistors within 3cm of the output of the signal driver.

8.23 Design Improvements

A new design that was done with reference to the lessons learned from the EBSA-285 would have a number of changes.

The most fundamental change would be to use only 3.3 V, with the exception of the SA-110 core. The areas of the design affected would be the soft I/O and oscillators. The oscillators could be replaced by a 3.3 V variant and the soft I/O latches and buffers could also be changed for equivalents in 3.3 V families such as the LCX or LVX from National Semiconductor, which have 5 V-tolerant inputs and outputs.

The buffers for the SDRAM control and address lines (SHT7) could probably be omitted, the other buffers are required to either control flow or provide signal level conversion.

The decoupling on the EBSA-285 was conservative, providing more decoupling than was required to ensure correct function in all possible modes.

The series termination was also conservative, with the result that the termination resistors for the SDRAM control and address lines could probably be omitted.

Any design that did not need to provide the flexibility of the EBSA-285 could remove all the jumpers with the possible exception of the one that enables blank programming mode.

More Vdd lines should be routed to the X-Bus expansion headers.

A 3.3 V supply should also be routed to the X-Bus expansion headers. This would allow 3.3 V peripherals to be attached without requiring the 3.3 V to be generated on the expansion card.

Known limitations of the card/21285 are:

- X-Bus peripherals and the internal PCI arbiter are mutually exclusive functions (functions implemented with pin sharing on the 21285).
- The 3.3 V power rail will always be derived from a 5 V source (backplane or connector). The EBSA-285 cannot supply its 3.3 V rail from the 3.3 V pins on the PCI connector.
- The EBSA-285 cannot program flash ROM via the PCI when in system mode.
- flash_rst is a 5 V signal and flash_rst_l is a 3.3 V signal.

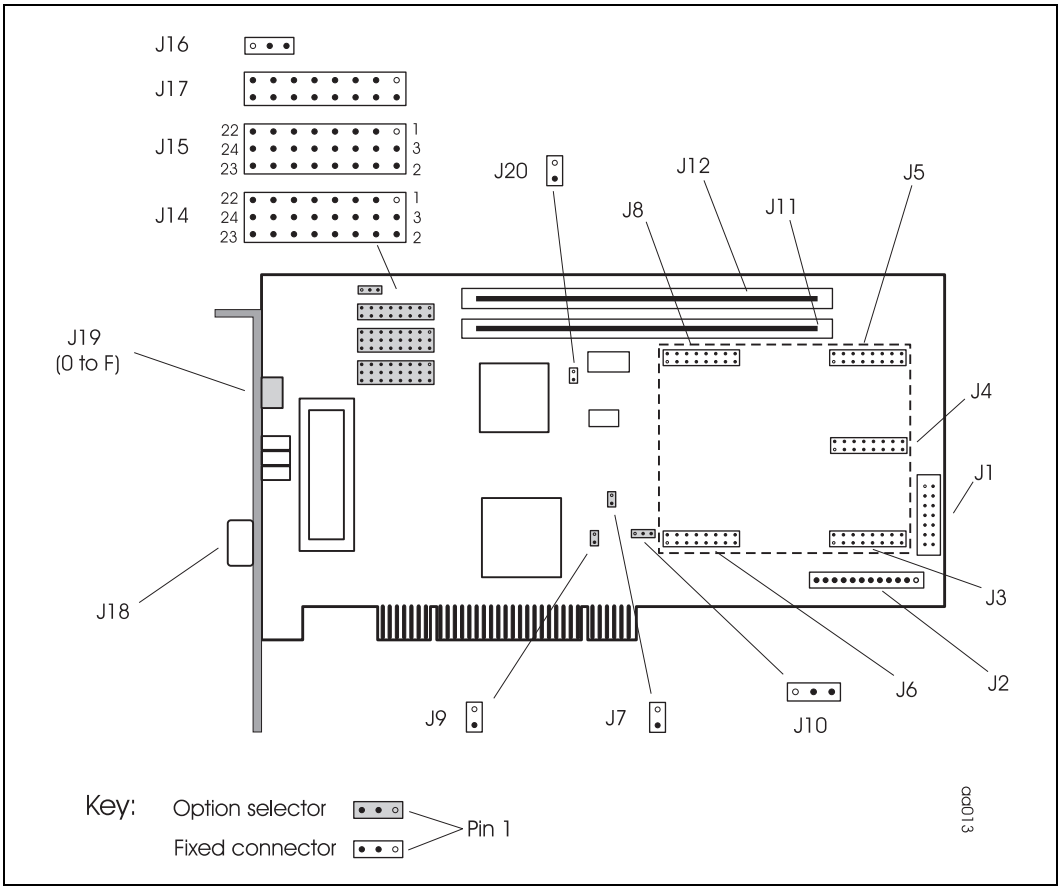
Configuration Guide A

This appendix describes:

- The default configuration of the board
- The settings for all links and jumpers
- The pinouts of all connectors
- The meaning of all LEDs.
- The cables required for connection to the board
- How to upgrade the SDRAM DIMMs

Locations of all jumpers and connections are shown in Figure A-1. Table A-1 provides a brief description of each jumper and connector.

Figure A-1. Jumper and Connector Locations



A.1 Default Configuration

This section covers configuration of the EBSA-285 for its major functions. The rationale behind the jumper settings for these functions is shown graphically in Figure A-2. Snapshots of jumper settings for each of the relevant functions are provided in:

- Add-in card: Figure A-3 (X-bus enabled)
- Host bridge: Figure A-4 (X-bus enabled)

Figure A-2. Primary Jumper Settings

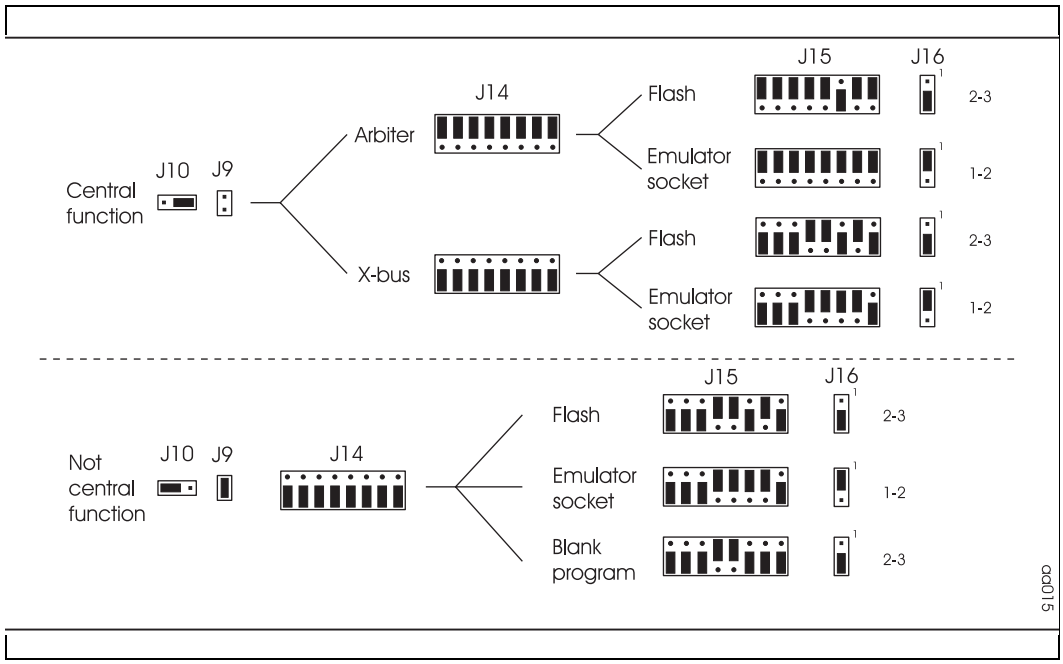


Figure A-3. EBSA-285 Configured as an Add-in Card

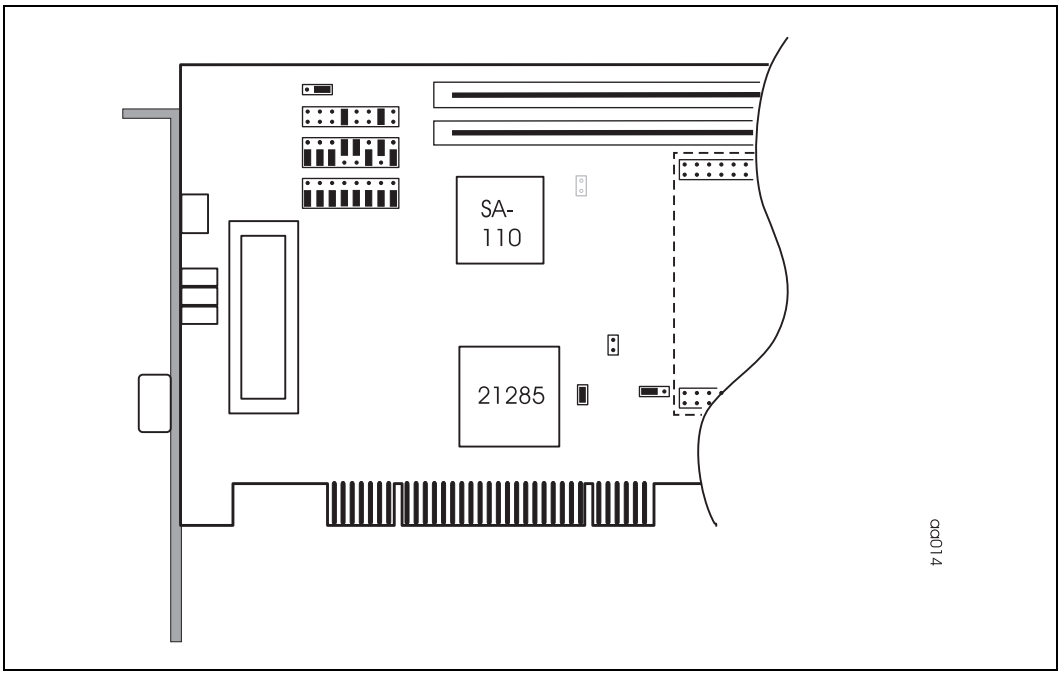
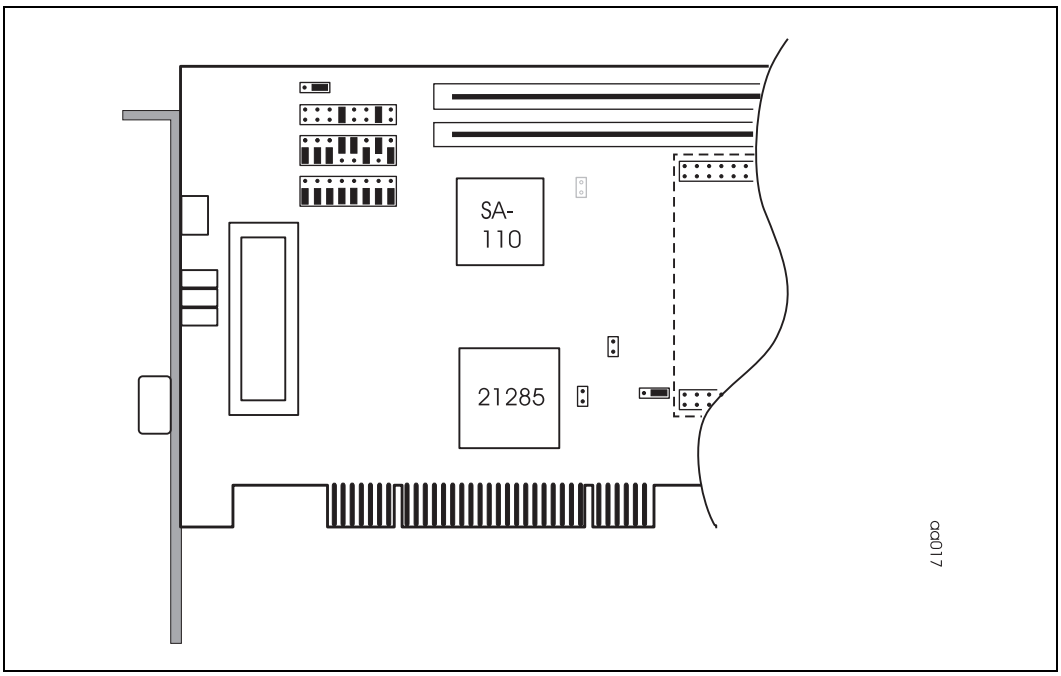


Figure A-4. EBSA-285 Configured as a Host Bridge



A.2 Description of Jumpers and Connectors

While reading this section, it may be useful to refer to the EBSA-285 schematic set. Table A-1 provides a brief description of all jumpers and connectors, and indicates where the relevant circuit detail is to be found.

Table A-1. General Information on EBSA-285 Jumpers and Connectors

Ref	Schematics	Description
J1	Sheet 18	JTAG connector
J2	Sheet 13	Power connector (bench test)
J3	Sheet 4	X-Bus expansion header
J4	Sheet 4	X-Bus expansion header
J5	Sheet 4	X-Bus expansion header
J6	Sheet 4	X-Bus expansion header
J7	Sheets 9 & 13	Onboard PCI clock (Reserved - DO NOT FIT)
J8	Sheet 4	X-Bus expansion header
J9	Sheets 9 & 17	Selection of central function mode
J10	Sheets 9 & 17	Selection of flash reset source
J11	Sheet 12	Secondary SDRAM DIMM socket (for 2-array DIMMs only)
J12	Sheet 11	Primary SDRAM DIMM socket (for 2/4-array DIMMs)
J14	Sheets 9 & 17	Arbiter/X-Bus selection headers
J15	Sheets 9 & 17	21285 mode-configuration
J16	Sheet 17	Flash EPROM selection
J17	Sheet 17	Core clock frequency selection and software flags
J18	Sheet 21	COM0 serial port connector
J19	Sheet 17	Bulkhead hex switch for flash image selection
J20	Sheet 4	Logic analyzer test point for SA-110 bus interface clock (**fbg_mclk**)

A.2.1 X-Bus Expansion Headers

Figure A-5 shows the pinout of the X-Bus expansion headers J3, J4, J5, J6 and J8. Signals are listed in Table A-2 to Table A-5.

Figure A-5. X-Bus Headers Pinout

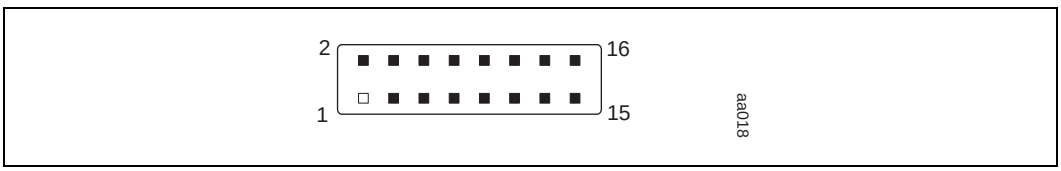


Table A-2. X-Bus Connector J3

Pin	Signal	Function
2	xbus_xcs1_l	Chip select for daughter card device
4	fbg_xcs2_l	Chip select for daughter card device
6	xbus_xcs0_irq_l	Interrupt request from daughter card device
8	nc	
10	nc	
12	nc	
14	flash_rst	For daughter-board chips that require a high level reset signal
16	flash_rst_l	For daughter-board chips that require a low level reset signal

Table A-3. X-Bus Connectors J4/J5

Pin	Signal	Function
2	xbuf_buf_d0/d8	Buffered X-Bus data
4	xbuf_buf_d1/d9	“
6	xbuf_buf_d2/d10	“
8	xbuf_buf_d3/d11	“
10	xbuf_buf_d4/d12	“
12	xbuf_buf_d5/d13	Buffered X-Bus data
14	xbuf_buf_d6/d14	“
16	xbuf_buf_d7/d15	“

Table A-4. X-Bus Connector J6

Pin	Signal	Function
2	fbg_xd_wren_l	X-Bus data write enable (from 21285 [Footbridge])
4	fbg_xior_l	X-Bus read strobe
6	fbg_xiow_l	X-Bus write strobe
8	vdd	
10	buf_a2	Buffered X-Bus address
12	buf_a3	“
14	buf_a4	“
16	buf_a5	“

Table A-5. X-Bus Connector J8

Pin	Signal	Function
2	buf_a6	Buffered X-Bus address
4	buf_a7	“
6	buf_a8	“
8	buf_a9	“
10	buf_a10	“
12	buf_be0	Buffered X-Bus enable signal
14	buf_be1	Buffered X-Bus enable signal
16	buf_a11	Buffered X-Bus address

A.2.2 Configuration Jumpers

A.2.2.1. CPU Core Clock Frequency Selection

The header links that determine the SA-110 core clock frequency are shown in Figure A-6. Selection of specific frequencies is explained in Figure A-7.

Figure A-6. J17 Pinout Showing Default Jumper Configuration

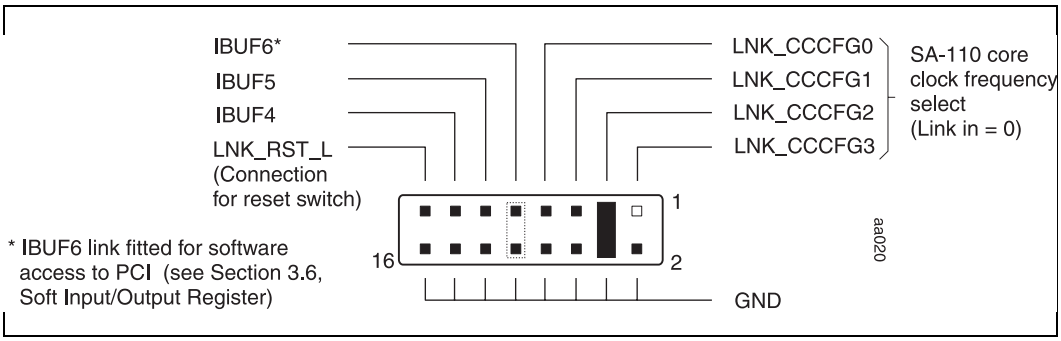
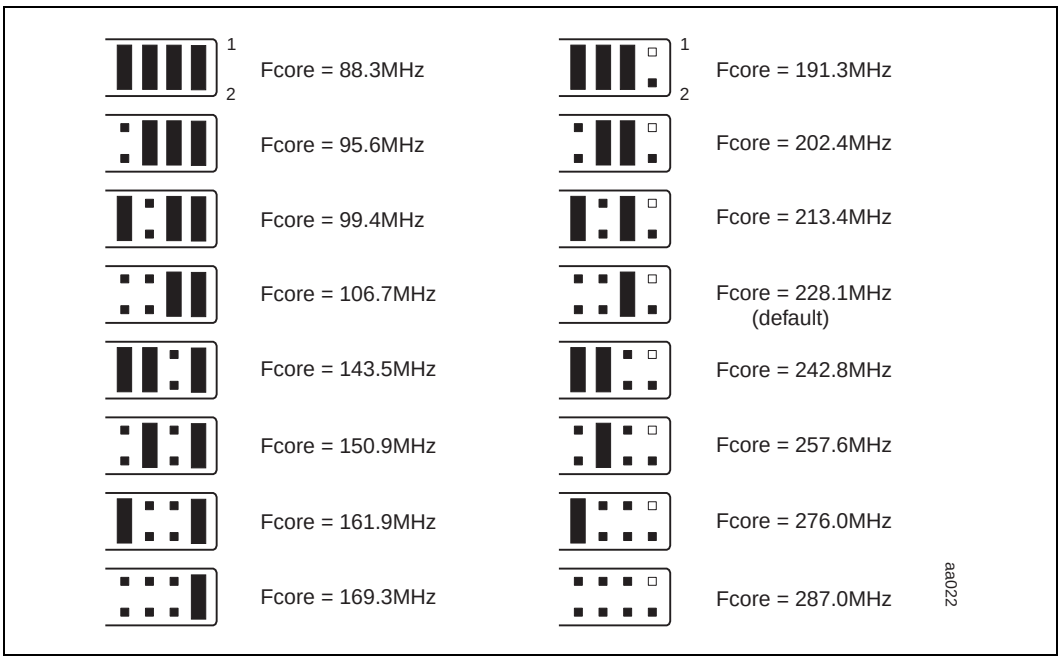


Figure A-7. J17 Core Clock Selection Jumpers



Note: SA-110 microprocessors operate at any one of 16 core clock frequencies between 88.3 MHz and 287 MHz, with the upper limit determined by the speed grade of the CPU fitted. The EBSA-285 fits a 233 MHz part, thereby limiting the maximum frequency of the core clock to 228.1 MHz. The card is delivered with this frequency selected.

A.2.2.2. Arbiter/X-Bus Selection

Selection of the 21285's internal arbiter or the X-Bus is made by jumpering pins 1-3 or 2-3 on J15. Jumpers on J14, and pins 17-24 on J15, reroute the signal paths for the selected configuration. In X-Bus mode, chip selects, write-enable and read and write strobes are directed to the X-Bus, and the X-Bus interrupt request is connected to the 21285. In arbiter mode, PCI arbitration request and grant paths are made between the PCI bus and the 21285.

Jumper settings and their effect are given in Table A-6.

Note: The aforementioned links operate as a block; all must be set to arbiter mode or all to X-Bus mode.

Figure A-8. J14/J15 Pinout

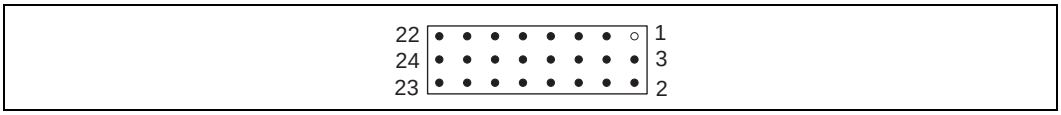


Table A-6. Arbiter/X-Bus Selection Jumpers

Ref	Link	Direction	Signal/function
J15	2-3	Input	fbg_ma7 (Select X-Bus [ma7=1])
J14	2-3	Input	pci_gnt_l (grant of 21285 bus master request)
"	5-6	Output	pci_req_l (21285 bus master request to PCI bus)
"	8-9	Input	xbus_xcs0_irq_l (XCS0 configured as X-Bus interrupt input)
"	11-12	Output	xbus_xcs1_l (X-Bus chip select)
"	14-15	Output	fbg_xcs2_l (soft I/O enable)
"	17-18	Output	fbg_xd_wren_l (X-Bus data buffer direction control)
"	20-21	Output	fbg_xior_l (X-Bus Read strobe)
"	23-24	Output	fbg_xior_l (X-Bus Write strobe)
J15	1-3	Input	fbg_ma7 (Select the 21285's internal PCI arbiter [ma7=0])
J14	1-3	Output	pci_gnt0_l (bus grant 0 to PCI bus [conn. A17])
"	4-6	I/O	fbg_pci_req_l (self-grant to fbg_pci_gnt_l on 21285)
"	7-9	Input	pci_req_l (bus master request 0 from PCI bus [conn. B18])
"	10-12	Input	pci_req1_l (bus master request 1 from PCI bus [conn. B10])
"	13-15	Input	pci_req2_l (bus master request 2 from PCI bus [conn. A19])
"	16-18	Output	pci_gnt1_l (bus grant 1 to PCI bus [conn. A14])
"	19-21	Output	pci_idsel_gnt2_l (bus grant 2 to PCI bus [conn. A26])
"	22-24	Output	pci_gnt3_l (bus grant 3 to PCI bus [conn. B11])
J15	17-18	PRSNT 1	Power requirement detection when add-in board (tied to 0V)
"	20-21	PRSNT 2	"
"	23-24	IDSEL	Connected to PCI fingers
J15	16-18	PCI-REQ3	When arbiter selected
"	19-21	PCI-GNT3	When arbiter selected
"	22-24		Connects GNT2 to IDSEL on PCI fingers

A.2.2.3. Flash/EPROM Selection

J15 is also used to configure the functionality as outlined in Table A-7.

Table A-7. Flash/EPROM Socket Selection

Ref	Link	Option
J15	4-6	Normal operation of flash ROM (ma6 = 1)
“	5-6	Blank ROM mode (ma6 = 0)
“	7-9	8-bit Emulator socket (ma4 = 1)
“	8-9	32-bit flash ROM (ma4 = 0)
“	10-12	Intel reserved test modes (ma3 = 1)
“	11-12	“ (ma3 = 0)
“	13-15	“ (ma2 = 1)
“	14-15	“ (ma2 = 0)

J16 is used to route the chip select to the emulator socket or flash block, as shown in Table A-8.

Table A-8. Flash/EPROM Socket Selection (J16)

Ref	Link	Option
J16	1-2	Emulator socket
“	2-3	Flash

Table A-9 shows the links that should be jumpered to enable the different modes.

Table A-9. Jumper Combinations for ROM Selection

For this mode	Jumper these links	and these links
32-bit flash ROM	J15, 4-6, and 8-9,	J16, 2-3
Blank flash ROM	J15, 5-6, and 8-9,	J16, 2-3
8-bit emulator	J15, 4-6, and 7-9,	J16, 1-2

A.2.2.4. Selection of the 21285 as the Central Function

Table A-10 shows the configuration of the two jumpers that effect the selection of Central Function. J9 controls the CFN signal to the 21285. J10 is used to route the reset signal correctly for the selected mode.

Table A-10. Selection of Central Function

Header	Jumper	Action
J9	In	lnk_pci_cfn is negated at power up. 21285 is NOT Central Function.
J9	Out	lnk_pci_cfn is asserted. 21285 is Central Function.
J10	1-2	pci_rst_l supplies reset (reset from PCI).
J10	2-3	mst_rst_l supplies reset (reset from power up, switch[a], JTAG connector, or watchdog timer).

a. Via a jumper or reset switch attached to J17. See Figure A-6.

The legal combinations for J9 and J10 are given in Table A-11.

Table A-11. Jumper Settings for Selection of Central Function

Mode	J9	J10
Host Bridge	Out	2-3
Add-in card	In	1-2

A.2.2.5. Reserved Mode

When not plugged into a PCI backplane, the EBSA-285 must supply the PCI clock it would otherwise have picked up from the PCI slot. When fitted, J7 connects an onboard 24 MHz oscillator to the PCI clock line. This is required in manufacture when the board is powered up in isolation.

A.2.2.6. Boot Image Selection

Boot image selector J19 on the bulkhead provides a hexadecimal code that is read from the X-Bus at power up. Boot image selection is covered in Chapter 7.

A.2.2.7. SA-110 Clock Probe Connection

Two pin header J20 provides a logic analyzer test point for monitoring fbg_mclk.

A.2.2.8. Test Points

Table A-12 provides a description of test points provided for monitoring the EBSA-285.

Table A-12. Description of Test Points

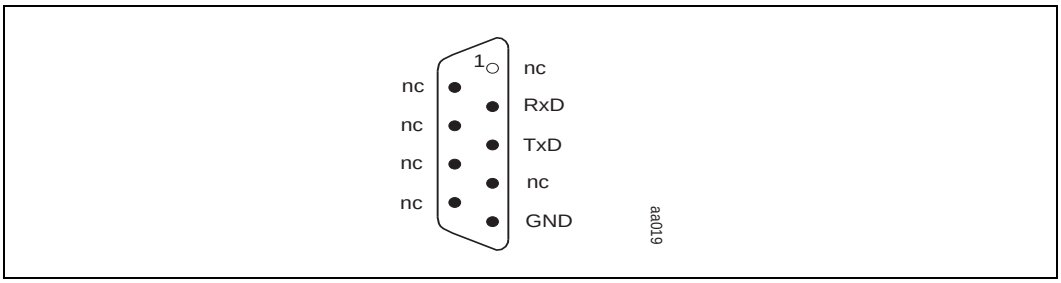
Ref	Schematics	Description
TP1	Sheet 15	+3.3V level
TP2	Sheet 15	+2V level
TP3	Sheet 4	cpu_write (this is the nR/W signal from the SA-110 or 21285)
TP4	Sheet 4	21285 clock signal fbg_fclk_0
TP5	Sheet 4	SDRAM clock signal fbg_sdclk0
TP6	Sheet 4	cpu_nrsto (the nRESET_OUT signal from the SA-110)

A.3 Connectors

A.3.1 Serial Port Connector

COM0 RS232 port is connected to a 9-way male D-type connector on the bulkhead. Pinout is shown in Figure A-9.

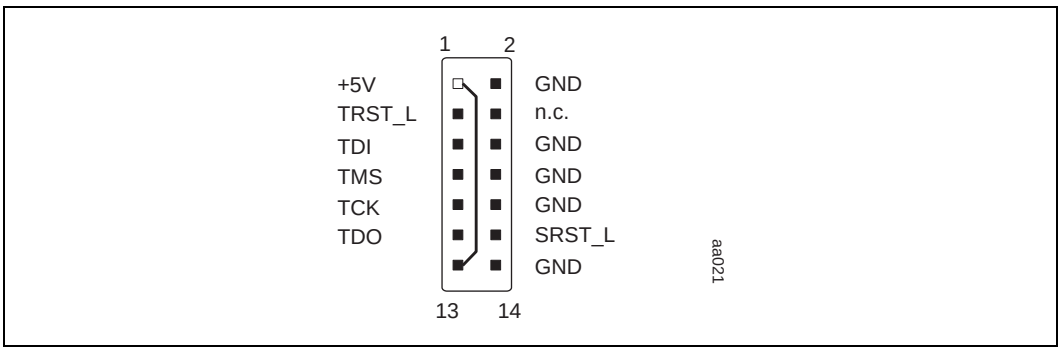
Figure A-9. Serial Port Connector Detail



A.3.2 JTAG Connector

The pinout of the JTAG connector is shown in Figure A-10.

Figure A-10. JTAG Connector J1 Pinout



A.3.3 Power Connector

J2 is the standard PC/AT 12-pin male connector for power.

A.4 Cables for External Connection

A.4.1 Serial Port

COM0 is wired as a DTE (Data Terminal Equipment) port with TxD, RxD and GND only. Use a null-modem cable to connect a terminal or host system to this port. Wiring detail for suitable standard null-modem cables is given in Table A-13 and Table A-14, but note that none of the handshake signals are required.

Table A-13. Null-Modem Cable

9-way Connector A Pin	25-way Connector B Pin	9-way Connector B Pin	Description
5	7	5	GND - GND
3	3	2	TxD - RxD
7[a]	5	8	RTS-CTS
6,1[1]	20	4	DSR, DCD - DTR
2	2	3	RxD - TxD
8[1]	4	7	CTS - RTS
4[1]	6,8	6,1	DTR - DSR, DCD

a. Not required on EBSA-285

Table A-14. Sun Null-Modem Cable

9-way Connector A Pin	25-way Connector B Pin	Description
2	2	RxD - TxD
3	3	TxD - RxD
5	7	GND - GND
7-8	-	RTS - CTS
-	4-5	RTS - CTS
4-6-1	-	DTR-DSR-DCD
-	20-6-8	DTR-DSR-DCD

A.4.2 JTAG Port

The wiring of the JTAG port is given in Table A-15. The JTAG port operates at 5 V TTL levels.

Table A-15. JTAG Cable

Pin	Type	Description
1	-	Pulled up to 5 V by a 33R resistor
2	-	GND
3	Input	TRST_L
4	-	Not connected
5	Input	TDI
6	-	GND
7	Input	TMS
8	-	GND
9	Input	TCK
10	-	GND
11	Output	TDO
12	Input	SRST_L - when 21285 is CFN, asserting this signal resets the board
13	-	Connected to pin 1
14	-	GND

A.5 Upgrading the SDRAM DIMMs

The EBSA-285 requires 168-pin 64-bit unbuffered SDRAM DIMMs. It can support a maximum of four, 32-bit arrays. 64-bit DIMMs typically provide either two or four 32-bit arrays. If a four-array DIMM is used, it must be fitted in J12, and J11 must be left empty. If two-array DIMMs are used, either one or two may be fitted but J12 must be populated first. Table A-16 lists DIMMs that are known to work with the EBSA-285.

Table A-16. DIMMs For Use With The EBSA–285

Manufacturer	Part Number	Size	Arrays	Mode
Samsung	KMM366S203BTN-G2	16Mbyte	2x8Mbyte	1
Samsung	KMM366S403BTN-G2	32Mbyte	4x8Mbyte	1
IBM	13N1649NCC-10T	8Mbyte	2x4Mbyte	1
IBM	13N2649JCC-10T	16Mbyte	2x8Mbyte	1
IBM	13N4649JCC-10T	32Mbyte	4x8Mbyte	1
IBM	13N4649CCC-10T	32Mbyte	2x16Mbyte	1
Hitachi	526C464EN-10IN/C	32Mbyte	4x8Mbyte	1
Hitachi	526C264EN-10IN/C	16Mbyte	2x8Mbyte	1
Micron	MT8LSDT264AG-66CL2	16Mbyte	2x8Mbyte	1
Micron	MT16LSDT464AG-66CL2	32Mbyte	4x8Mbyte	1
*Kingston	KTC-2428/32	32Mbyte	4x8Mbyte	1
*Kingston	KTC-2428/64	64Mbyte	2x32Mbyte	4
*Kingston	KTC-2428/128	128Mbyte	4x32Mbyte	4

* Limited test time as DIMMs had to be returned.

The Design Database B

If you received this Reference Manual as part of the SA-110/21285 Evaluation Board Kit (21A85-01) or as part of the SA-110/21285 Design Kit (QR-21A85-11) you will also have received machine-readable media, either on CDROM or 3.5 inch floppy disk, containing hardware and software design databases for the EBSA-285.

This appendix gives an overview of the material supplied on the first release of the disks. The disks themselves include other relevant documentation. Later releases of the disks may include additional information and you should check the README files and release notes for details.

B.1 Hardware Material

The hardware design database includes:

- EBSA-285 schematics (PostScript and PDF formats)
- EBSA-285 board assembly drawings (PostScript and PDF formats)
- EBSA-285 board mechanical drawings (PostScript and PDF formats)
- EBSA-285 board layer plots (PostScript and PDF formats)
- EBSA-285 netlist (ASCII text format)
- EBSA-285 parts list (ASCII text format)
- EBSA-285 design database for VIEWlogic CAE systems (tar file)
- EBSA-285 reference manual (PostScript and PDF versions of this document)

B.2 Software Material

There are two sets of software material for the EBSA-285. The first is the firmware tree for the EBSA-285 onboard software and Flash Management Utility. The second is the uHAL software portability library.

All of the software requires an ARM** 2.1 (or later) SDT to build. It can be built either using ARM's V2.1 Project Manager or using GNU make. Key images are supplied pre-built.

The EBSA-285 firmware tree consists of these components:

- Flash Management Utility (FMU) - Sources for building 8-, 16- and 32-bit mode flash utilities that can either be run from DOS or loaded and run via the ARM remote debugger.
- Angel Remote Debug Stub - Incremental sources to ARM's official released Angel source tree. These sources will migrate into an ARM release and will be superseded by such a release. The EBSA-285 Angel sources include the Primary Boot Loader (PBL). A variant of Angel is built with the PBL code embedded in it.
- Onboard Self-tests - Sources for the onboard self-test. This code performs a number of self tests (for example, memory and flash tests).
- Standard Include Files - Including assembler macros for functions such as memory initialization for all of the above code.

The following tested, pre-built Angel images are included:

- Angel (‘PBL’ variant) - This variant of Angel is built for flash block 0 and contains the PBL. If flash block 0 is selected as the boot image or if a corrupt or non-existent image is chosen as the boot image then the PBL will default to running this version of Angel. This Angel variant enables clock switching and the Echoic.
- Angel (‘Block 2’ variant) - This variant of Angel is built for flash block 2 and can be started by the PBL if image 2 is selected. It turns on clock switching and Echoic.
- Onboard self-test - This image runs from flash block 1.

The following tested, pre-built FMU images are included:

- **fmu.exe** - This is a DOS executable image which can be used to program images into the EBSA-285’s flash ROM when the EBSA-285 is plugged into a PCI slot in a PC
- **fmu.axf** - This version of the FMU can be loaded and run via the ARM remote debugger

uHAL is a portability library that builds for a number of StrongARM**-based boards. uHAL 0.5 has been ported to EBSA-285 and will build images that can either be loaded and run using Angel or that can be blown into flash and run directly from there. The uHAL source tree includes a number of example images:

- Benchmarking code - There are several benchmarks, for example, a bubble-sort.
- “hello world” - There are several very simple programs, including one that prints “hello world” and another that shows the system timer running.
- uC/OS - uHAL contains a port of uC/OS as an example of an OS ported to run against the uHAL API.

uHAL 0.5 has its own documentation. This includes a description of the API and a FAQ (Frequently Asked Questions) which includes board-specific information.

Index

J

L

M

P

R

S

T

U

V

X

Support, Products, and Documentation

If you need technical support, a *Product Catalog*, or help deciding which documentation best meets your needs, visit the Intel World Wide Web Internet site:

http://www.intel.com

Copies of documents that have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling **1-800-332-2717** or by visiting Intel's website for developers at:

http://developer.intel.com

You can also contact the Intel Massachusetts Information Line or the Intel Massachusetts Customer Technology Center. Please use the following information lines for support:

For documentation and general information:	
Intel Massachusetts Information Line	
United States:	1–800–332–2717
Outside United States:	1–303-675-2148
Electronic mail address:	techdoc@intel.com

For technical support:	
Intel Massachusetts Customer Technology Center	
Phone (U.S. and international):	1–978–568–7474
Fax:	1–978–568–6698
Electronic mail address:	techsup@intel.com