ARM Evaluator-7T Board

User Guide

ARM

ARM DUI 0134A

ARM Evaluator-7T Board User Guide

Copyright © ARM Limited 2000. All rights reserved.

Release information

Change history

Date	Issue	Change
1 August 2000	A	New document

Proprietary notice

ARM, the ARM Powered logo, Thumb, and StrongARM are registered trademarks of ARM Limited.

The ARM logo, AMBA, Angel, ARMulator, EmbeddedICE, ModelGen, Multi-ICE, PrimeCell, ARM7TDMI, ARM7TDMI-S, ARM9TDMI, ARM9E-S, ARM946E-S, ARM966E-S, ETM7, ETM9, TDMI, and STRONG are trademarks of ARM Limited.

All other products or services mentioned herein may be trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM Limited in good faith. However, all warranties implied or expressed, including but not limited to implied warranties or merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Federal Communications Commission Notice

NOTE: This equipment has been tested and found to comply with the limits for a class A digital device, pursuant to part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

CE Declaration of Conformity

This equipment has been tested according to ISE/IEC Guide 22 and EN 45014. It conforms to the following product EMC specifications:

The product herewith complies with the requirements of EMC Directive 89/336/EEC as amended.

Document confidentiality status

This document is Open Access. This means there is no restriction on the distribution of the information.

Product status

The information in this document is Final (information on a developed product).

ARM web address

`http://www.arm.com`

Copyright © ARM Limited 2000. All rights reserved.

Copyright © ARM Limited 2000. All rights reserved.

Contents
ARM Evaluator-7T Board User Guide

Preface

About this document viii
Further reading xi
Feedback xii

Chapter 1 **Introduction**

1.1 About the Evaluator-7T board 1-2
1.2 Evaluator-7T architecture 1-3
1.3 Kit contents 1-4
1.4 System requirements 1-5
1.5 Setting up the Evaluator-7T 1-6
1.6 Precautions 1-8

Chapter 2 **Hardware Description**

2.1 The Samsung KS32C50100 microcontroller 2-2
2.2 Reset circuit 2-4
2.3 Memory 2-5
2.4 Serial ports 2-9
2.5 LEDs 2-11
2.6 Switches 2-13
2.7 JTAG port 2-14
2.8 Power supply 2-15

 Copyright © ARM Limited 2000. All rights reserved.

Chapter 3 **Programmers Reference**
3.1 General memory map 3-2
3.2 Memory usage 3-3
3.3 Microcontroller register usage 3-5
3.4 Accessing LEDs and switches 3-6

Chapter 4 **Bootstrap Loader Reference**
4.1 About the bootstrap loader 4-2
4.2 Basic setup with the BSL 4-3
4.3 BSL commands 4-7
4.4 Modules 4-19
4.5 Preparing a program for download 4-29
4.6 Production test module 4-30

Appendix A **Evaluator-7T Mechanical Outline**
A.1 Mechanical outline A-2

Appendix B **Evaluator-7T Signal Naming**
B.1 Signal naming 4

Index

 Copyright © ARM Limited 2000. All rights reserved.

Preface

This preface introduces the ARM Evaluator-7T board and its reference documentation. It contains the following sections:

- *About this document* on page viii
- *Further reading* on page xi
- *Feedback* on page xii.

 Copyright © ARM Limited 2000. All rights reserved.

About this document

This document describes how to set up and use the Evaluator-7T.

Intended audience

This document has been written for software engineers, hardware engineers, and students to enable you to gain experience with ARM architecture design techniques.

Using this manual

This document is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the ARM Evaluator-7T board. This chapter overviews the architecture of the board and identifies the main components.

Chapter 2 *Hardware Description*

Read this chapter for a description of the onboard hardware.

Chapter 3 *Programmers Reference*

Read this chapter for a description of the memory map and on-board registers.

Chapter 4 *Bootstrap Loader Reference*

Read this chapter for a description of the bootstrap loader.

Appendix A *Evaluator-7T Mechanical Outline*

Refer to this appendix for the mechanical outline of the board.

Appendix B *Evaluator-7T Signal Naming*

Refer to this appendix for a description of the signal naming conventions used on the board schematics.

Copyright © ARM Limited 2000. All rights reserved.

Typographical conventions

The following typographical conventions are used in this manual:

bold — Highlights ARM processor signal names, and interface elements such as menu names. Also used for terms in descriptive lists, where appropriate.

italic — Highlights special terminology, cross-references, and citations.

`typewriter` — Denotes text that can be entered at the keyboard, such as commands, file names and program names, and source code.

`typewriter` (with `type` underlined) — Denotes a permitted abbreviation for a command or option. The underlined text may be entered instead of the full command or option name.

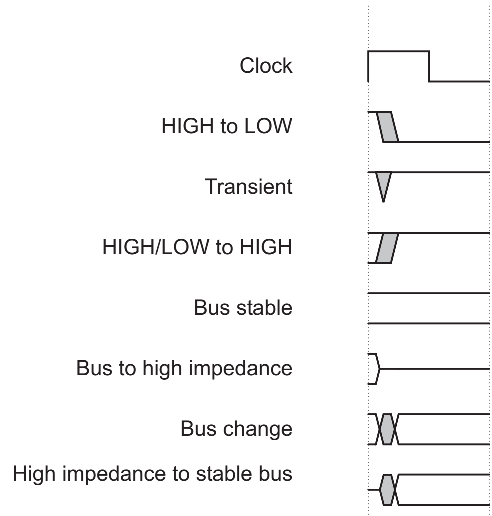
`typewriter italic` — Denotes arguments to commands or functions where the argument is to be replaced by a specific value.

`typewriter bold` — Denotes language keywords when used outside example code.

Copyright © ARM Limited 2000. All rights reserved.

Timing diagram conventions

This manual contains one or more timing diagrams. The following key explains the components used in these diagrams. Any variations are clearly labeled when they occur. Therefore, no additional meaning should be attached unless specifically stated.



Key to timing diagram conventions

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.

Copyright © ARM Limited 2000. All rights reserved.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

This document contains information that is specific to the Evaluator-7T. Refer to the following documents for other relevant information:

- *ARM7TDMI Data Sheet* (ARM DDI 0029)
- *ARM Architecture Reference Manual* (ARM DDI 0100).

Other publications

This section lists relevant documents published by third parties.

- *Samsung KS32C50100 32-BIT RISC Micro Controller Embedded Network Controller User's Manual*.

 Copyright © ARM Limited 2000. All rights reserved.

Feedback

ARM Limited welcomes feedback both on the Evaluator-7T, and on the documentation.

Feedback on the Evaluator-7T

If you have any comments or suggestions about this product, please contact your supplier giving:

- the product name
- a concise explanation of your comments.

Feedback on this document

If you have any comments about this document, please send email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments refer
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Sharing information

An email list server is provided by ARM to enable you to share information with other Evaluator-7T users. To subscribe, send an email to:

`subscribe-evaluator7t@arm.com`

The list server will reply, welcoming you to the Evaluator-7T email group. You can query other Evaluator-7T users by sending email to:

`evaluator7t@arm.com`

To unsubscribe, send an email to:

`unsubscribe-evaluator7t@arm.com`

Copyright © ARM Limited 2000. All rights reserved.

Chapter 1
Introduction

This chapter introduces the ARM Evaluator-7T board. It contains the following sections:

- *About the Evaluator-7T board* on page 1-2
- *Evaluator-7T architecture* on page 1-3
- *Kit contents* on page 1-4
- *System requirements* on page 1-5
- *Precautions* on page 1-8
- *Setting up the Evaluator-7T* on page 1-6.

Copyright © ARM Limited 2000. All rights reserved.

1.1 About the Evaluator-7T board

The ARM Evaluator-7T board is a simple ARM platform that includes a minimal set of core facilities. It is powerful and flexible enough to function as an evaluation platform for ARM technology. The board enables you to:

- download and debug software images
- attach additional input/output devices and peripherals for experimentation.

Figure 1-1 shows the layout of the Evaluator-7T.

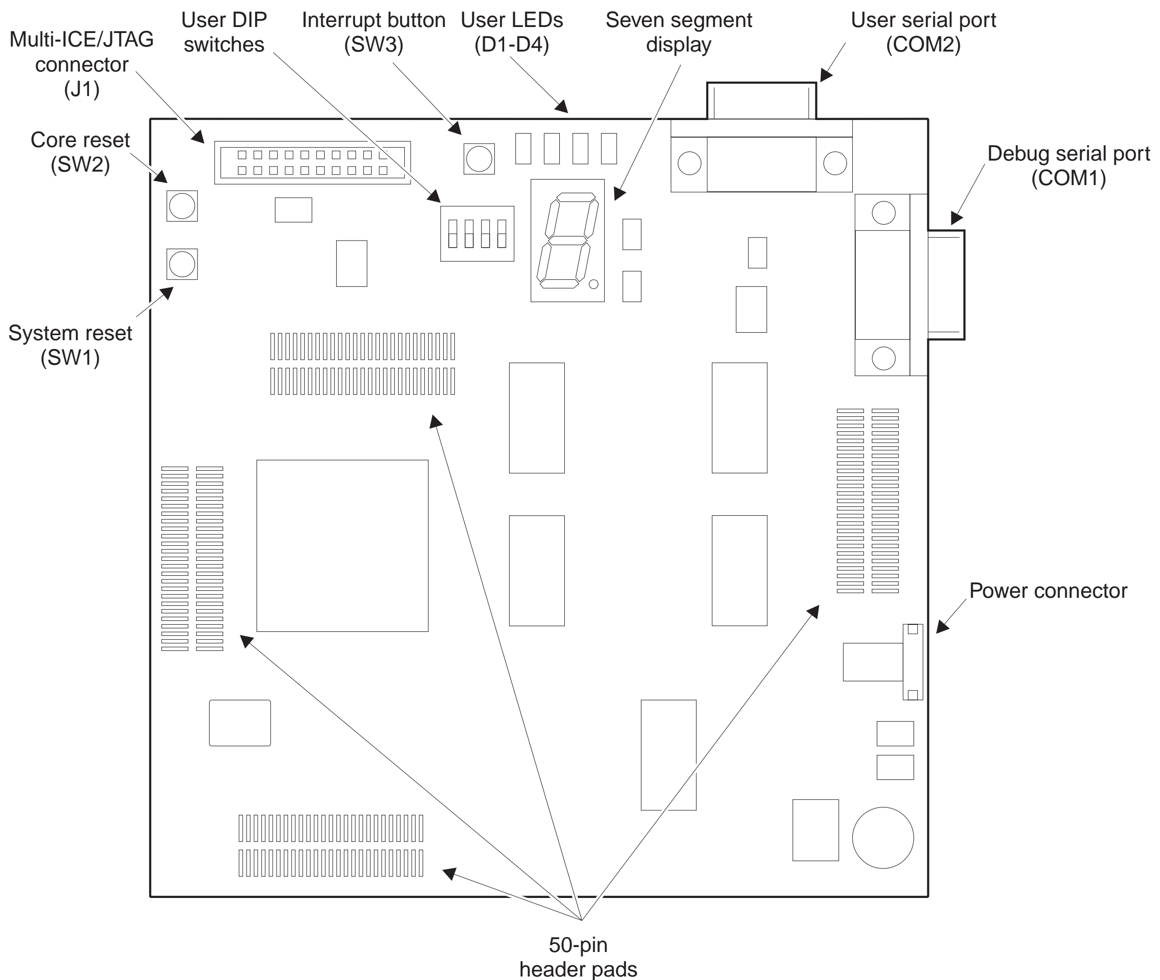


Figure 1-1 Evaluator-7T board layout

 Copyright © ARM Limited 2000. All rights reserved.

1.2 Evaluator-7T architecture

The Evaluator-7T board contains the following major components:

- Samsung KS32C50100 microcontroller
- 512KB flash EPROM
- 512KB SRAM
- two 9-pin D-type RS232 connectors
- reset and interrupt push buttons
- four user-programmable LEDs and a seven-segment LED display
- 4-way user input DIP switch
- Multi-ICE connector
- 10MHz clock (the processor uses this to generate a 50MHz clock)
- 3.3V voltage regulator.

The major components are described in detail in Chapter 2 *Hardware Description*.

Figure 1-2 shows the architecture of the Evaluator-7T.

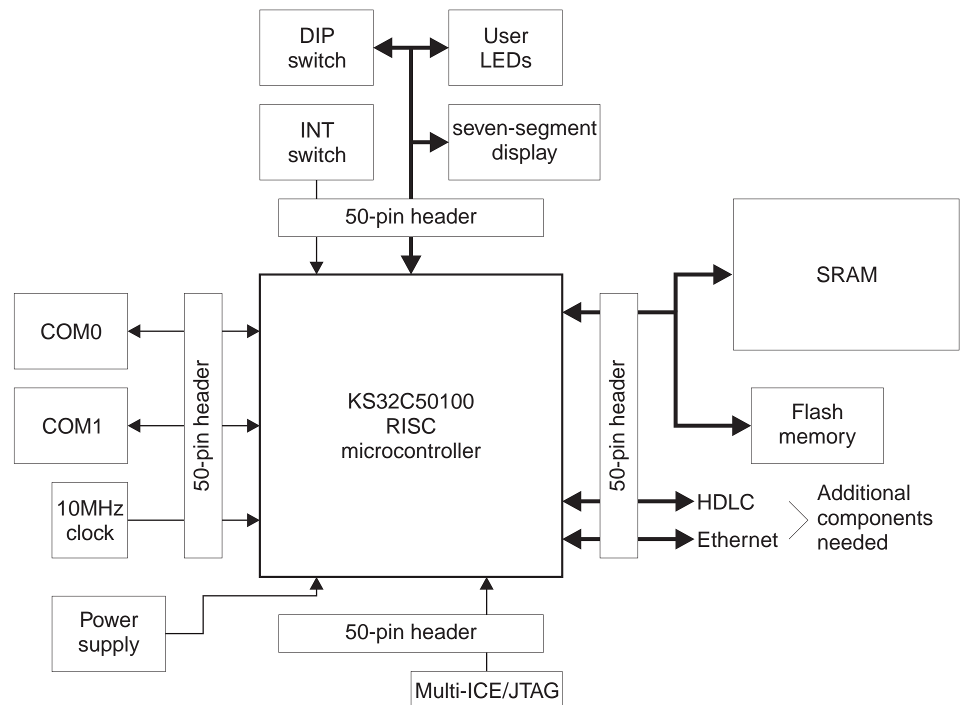


Figure 1-2 Evaluator-7T architecture

Copyright © ARM Limited 2000. All rights reserved.

1.3 Kit contents

This section describes the items supplied as part of the ARM Evaluator-7T.

1.3.1 Hardware

The kit includes the following hardware:

- ARM Evaluator-7T board
- 9-pin straight-through RS232 serial cable
- 9V power adapter.

1.3.2 Software and documentation

The kit includes the following CD-ROMs:

- *Evaluator-7T Tools and Documentation* containing:
  - example code specific to the Evaluator-7T
  - source code and binary image of the firmware
  - documentation, including this manual and the *Samsung KS32C50100 User's Guide*, in PDF format
  - an installer to copy the files onto your hard disk drive and create a menu item.
- *ARM Developer Suite Evaluation Version* containing a fully functional evaluation copy of the *ARM Developer Suite* (ADS) with a 45-day time limit. It runs on Microsoft Windows 95, 98, 2000, and NT 4.0. It includes the following software:
  - C and C++ compilers
  - assembler
  - linker
  - graphical debugger
  - project manager
  - C and C++ libraries
  - example programs.

  Also included on this CD-ROM is the ARM ADS documentation in PDF format.

Copyright © ARM Limited 2000. All rights reserved.

1.4 System requirements

Using the Evaluator-7T with the pre-installed boot monitor requires connection of a computer running a terminal application to the DEBUG serial connector.

To generate and debug code, and to use Angel or Multi-ICE, you will need to connect a computer running suitable development tools. The *ARM Developer Suite Evaluation Version* CD supplied with the Evaluator-7T provides tools for you to use.

Copyright © ARM Limited 2000. All rights reserved.

1.5 Setting up the Evaluator-7T

The ARM Evaluator-7T is a complete target ARM evaluation platform. Apart from the host computer, the kit includes all components required to evaluate a simple ARM system, including a representative software development environment. The ARM Evaluator-7T can be used in the following ways:

- *Using the bootstrap loader*
- *Using the Angel debug monitor*
- *Using Multi-ICE* on page 1-7.

1.5.1 Using the bootstrap loader

The *BootStrap Loader* (BSL) is a component of the resident firmware preloaded into the bottom of the flash memory (see *Flash memory usage* on page 3-4). The BSL is the first program run by the processor when the system is reset or powered on. For guidance on how to set up and use BSL, see *Basic setup with the BSL* on page 4-3.

The bootstrap loader provides the following functionality:

- board configuration commands that enable you to, for example, set the baud rate, and boot modules
- user help
- flash management tools that allow executable modules, such as Angel, to be added or removed from flash
- support for downloading applications to SRAM and executing them.

A complete list of the configuration options is given in Chapter 4 *Bootstrap Loader Reference*.

1.5.2 Using the Angel debug monitor

To use the Angel debug monitor, connect the host computer running an ARM debugger to the DEBUG port on the Evaluator-7T using the straight-through RS-232 cable supplied with the kit.

The Angel debug monitor is preloaded into the flash as a bootstrap loader module. It is executed by default when the board is powered on (unless you press the `Enter` key). You can change this default behavior (see *Modules* on page 4-19).

Note

Angel uses ADP to communicate with the debugger. Some third-party debuggers also support ADP.

 Copyright © ARM Limited 2000. All rights reserved.

Angel re-initializes the board and sets up a communication channel with a debugger on the host PC through the DEBUG port. It is this interaction between the host-based debugger and Angel that allows you to download and debug software. Angel interacts with the software and, in some cases, modifies it, for example, setting software breakpoints.

1.5.3 Using Multi-ICE

Connect the Multi-ICE unit (available separately) as follows:

1. Connect the Multi-ICE unit to the 20-Pin JTAG connector, J1.
2. Connect the Multi-ICE unit to the host computer using the supplied parallel cable.

The ARM Multi-ICE unit is supported by the ARM Developer Suite provided in each kit. It allows you to debug, download, and test software on the Evaluator-7T board. Multi-ICE does not require the use of the Angel debug monitor.

Multi-ICE enables you to monitor software on the Evaluator-7T board.

1.5.4 How Multi-ICE differs from a debug monitor

A debug monitor, such as the Angel debug monitor, is an application that runs on your target hardware in conjunction with the user application. It requires some resources, such as memory and access to exception vectors, to be available.

Multi-ICE requires almost no resources. Rather than being an application on the board, it works by using:

- additional hardware (Embedded ICE logic) that is incorporated into the core
- the Multi-ICE unit to buffer and translate the core signals into a form usable by a host computer

Multi-ICE is designed to allow debugging using JTAG port and to be as non-intrusive as possible:

- the target being debugged needs very little special hardware to support debugging
- in most cases no memory in the system being debugged has to be set aside for debugging, and no special software need be incorporated into the application
- execution of the system being debugged is only halted when a breakpoint or watchpoint unit is triggered, or the user requests that execution is halted.

Copyright © ARM Limited 2000. All rights reserved.

1.6 Precautions

The Evaluator-7T board is intended for use within a laboratory or engineering development environment and is supplied without an enclosure. The absence of an enclosure leaves the board sensitive to electrostatic discharges and allows electromagnetic emissions.

To avoid damaging the Evaluator-7T, you must:

- always wear an earth strap when handling the board
- only hold the board by the edges

Do not use the board near equipment which could be sensitive to electromagnetic emissions (such as medical equipment) or which is a transmitter of electromagnetic emissions.

Copyright © ARM Limited 2000. All rights reserved.

Chapter 2
Hardware Description

This chapter provides hardware and functional description of the Evaluator-7T board. It contains the following sections:

- *The Samsung KS32C50100 microcontroller* on page 2-2
- *LEDs* on page 2-11
- *Memory* on page 2-5
- *Reset circuit* on page 2-4
- *Serial ports* on page 2-9
- *Switches* on page 2-13
- *JTAG port* on page 2-14
- *Power supply* on page 2-15.

Copyright © ARM Limited 2000. All rights reserved.

2.1 The Samsung KS32C50100 microcontroller

The KS32C50100 is a square, 208-Pin *Quad Flat Pack* (QFP), embedded microcontroller manufactured by Samsung Electronics Co., Ltd. It is a *System-on-Chip* (SoC) targeted at the communications market.

The KS32C50100 is an ARM7TDMI-base microcontroller that incorporates a number on-chip functions. These are:

- 8KB unified cache/SRAM
- I^2C serial interface (master only)
- Ethernet controller
- two-channel DMA controller
- memory controller providing 8/16/32-bit external bus support for ROM/SRAM, flash, SDRAM, DRAM, and external input/output
- *High-level Data Link Control* (HDLC) support
- two UARTS
- 18 programmable input/output bit ports
- interrupt controller
- two programmable 32-bit timers.

The KS32C50100 microcontroller is powered by a 3.3V switching regulator and driven with a single 10MHz clock generator.

The microcontroller pins are connected to four sets of 50-pin connector pads (J2, J3, J4, and J5). For more information about the microcontroller, consult the *Samsung KS32C50100 32-BIT RISC Micro Controller Embedded Network Controller User's Manual*.

Figure 2-1 on page 2-3 shows the block diagram of the KS32C50100.

Copyright © ARM Limited 2000. All rights reserved.

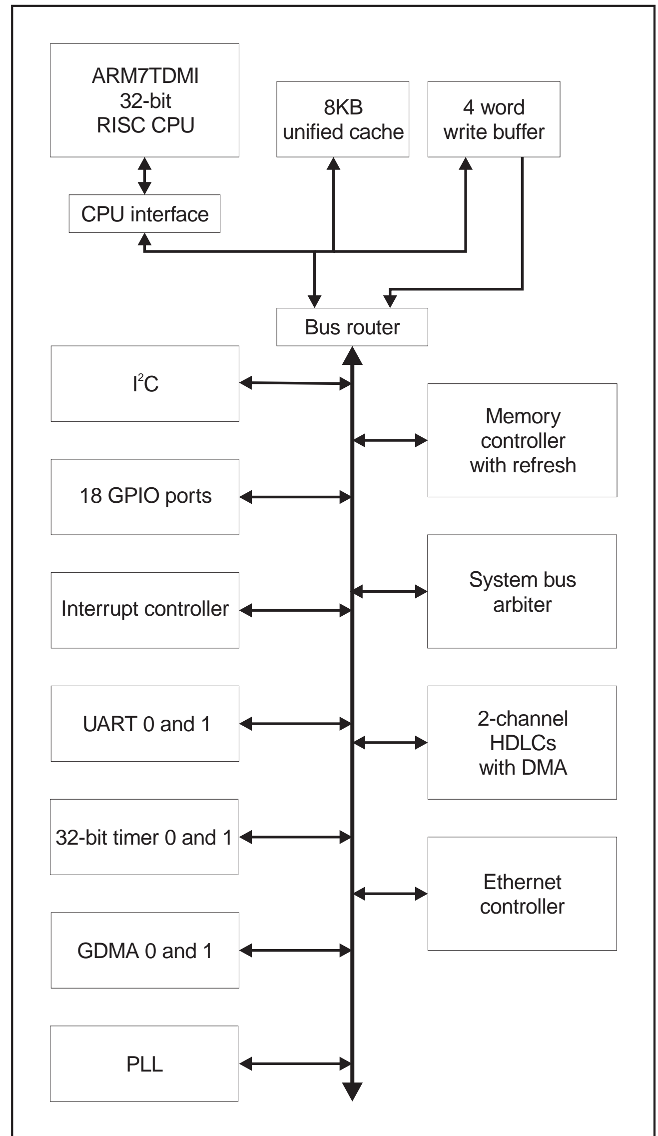


Figure 2-1 KS32C50100 block diagram

Copyright © ARM Limited 2000. All rights reserved.

2.2 Reset circuit

The architecture of the reset circuit on the Evaluator-7T board is shown in Figure 2-2.

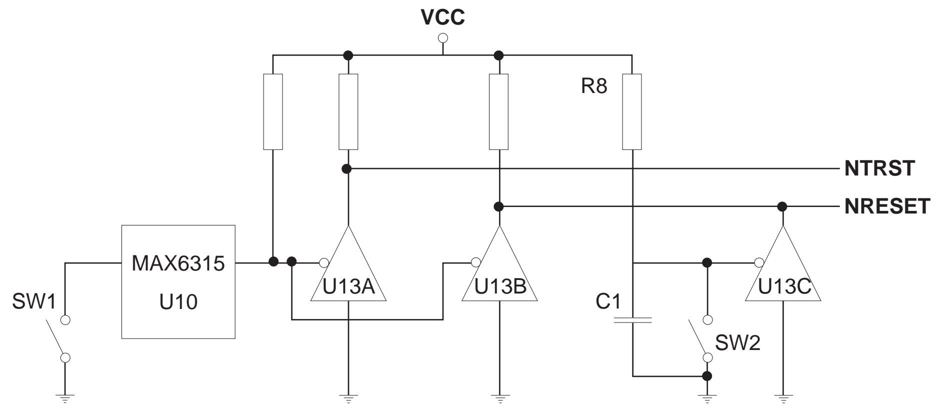


Figure 2-2 Reset circuit

The circuit provides two reset switches and a Maxim MAX6315 reset controller. The circuit controls two reset signals:

NRESET This signal resets the ARM7TDMI processor core.

NTRST This signal resets the tap controller and EmbeddedICE logic. This resets the internal functionality used by any debugger or other device connected to the JTAG port.

The MAX6315 provides switch debouncing for the system reset switch and also provides a power-on reset delay. The resistor and capacitor (R8 and C1) extend the assertion of the **NRESET** to guarantee reliable core reset.

The reset switches are used as follows:

SW1 Press the system reset switch, SW1, to reset the entire board and the assert **NRESET** and **NTRST** simultaneously.

SW2 Press the core reset switch, SW2, to reset the microcontroller, but not the TAP controller, by asserting only **NRESET**.

Pressing SW2 enables you to stop and take control of the ARM7TDMI processor before its first instruction fetch from address `0x0` without resetting other components on the board.

 Copyright © ARM Limited 2000. All rights reserved.

2.3 Memory

The Evaluator-7T provides two areas of memory:

- flash memory, in which the *BootStrap Loader* (BSL), Angel, and other non-volatile programs are stored
- SRAM for general program and data storage.

2.3.1 Flash

The Evaluator-7T includes 512KB of flash memory. When the Evaluator-7T is shipped, this contains the BSL and debug monitor. The remaining space is available for your own programs (see *Flash memory usage* on page 3-4). The flash is implemented as a single 16-bit device and is mapped to memory bank 0 (**NRCS0**).

On reset, the KS32C50100 default settings cause memory bank 0, the flash ROM, to be mapped at address `0x0` with a data bus width of 16-bits and the maximum number of wait states per memory access.

2.3.2 SRAM

Two 64K x 32 arrays of SRAM are connected to the microcontroller. The two arrays provide a total of 512KB. Figure 2-3 on page 2-6 shows one memory array.

The first SRAM array consists of the devices U2 and U5, and is mapped to bank 1. The second SRAM array consists of U3 and U6 and is mapped to bank 2. U5 and U6 connect to the lower 16 bits of the microcontroller 32-bit data bus. U2 and U3 connect to the upper 16 bits. The *Upper Byte* (UB) and *Lower Byte* (LB) select pins of each part are driven by an AND gate combination of the **NOE** and corresponding **NWBEx** outputs from the microcontroller. The **WE** pin of each SRAM part is driven by the AND gate combination of the two **NWBEx** signals that apply to the part.

Note

The microcontroller incorporates an internal address bus shifter that determines the number of bits to shift the external address bus. This is determined by the data bus width value set in the **EXTDBWTH** configuration register. For more details refer to the *Samsung KS32C50100 32-BIT RISC Micro Controller Embedded Network Controller User's Manual*.

Copyright © ARM Limited 2000. All rights reserved.

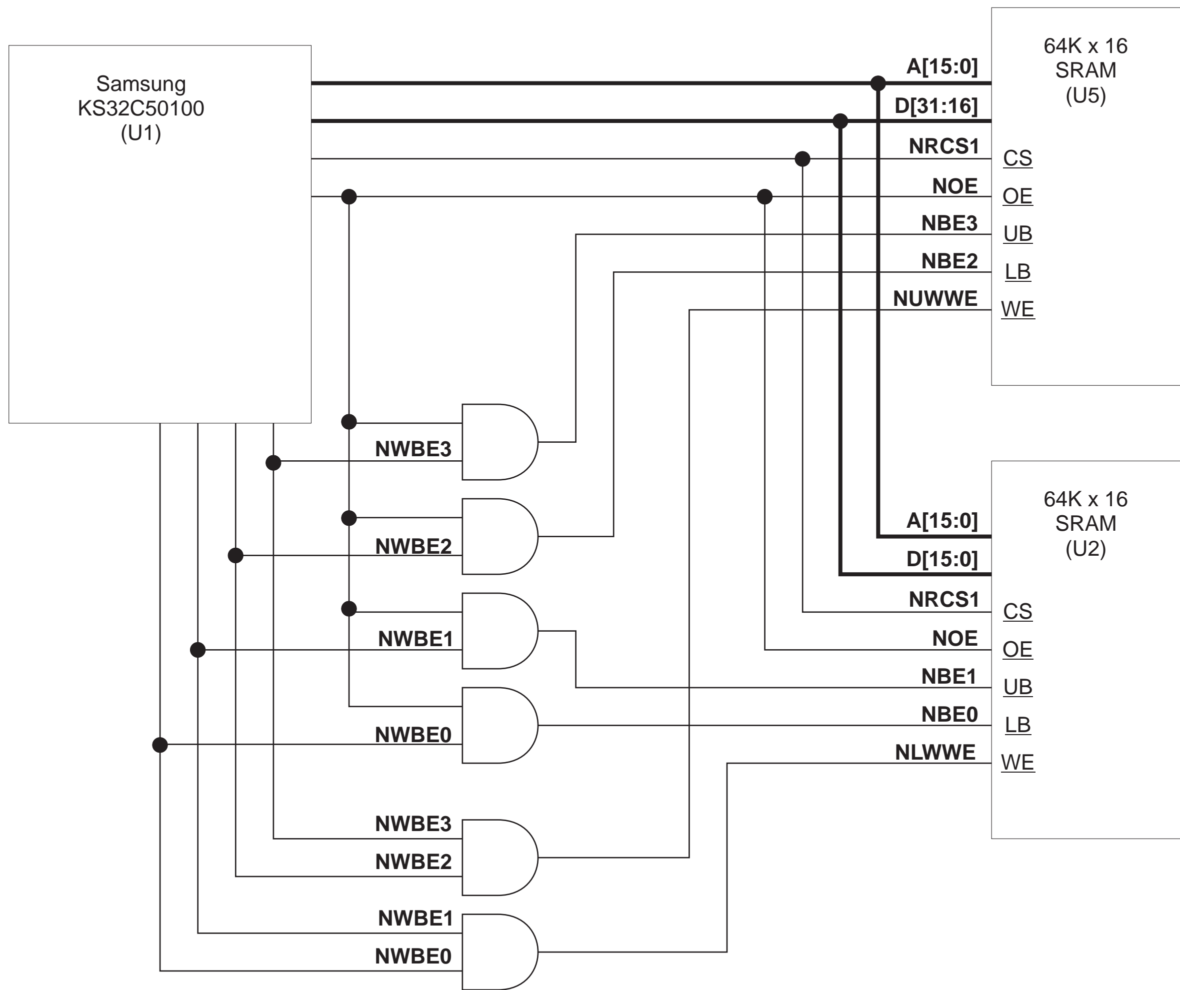


Figure 2-3 SRAM memory array

Copyright © ARM Limited 2000. All rights reserved.

Figure 2-4 and Figure 2-5 show the read and write cycle timing diagrams for both external SRAM arrays.

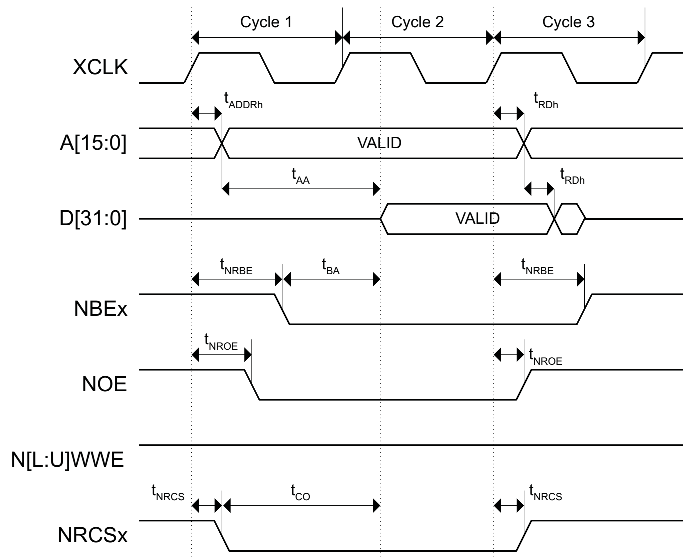


Figure 2-4 SRAM read cycle timing

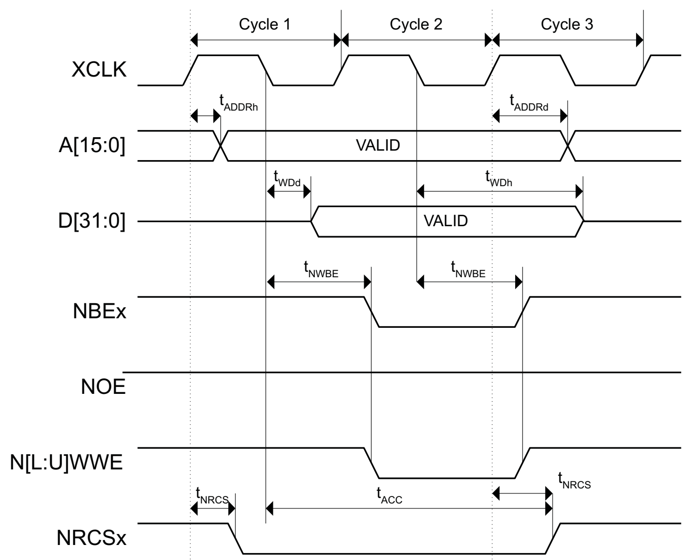


Figure 2-5 SRAM write cycle timing

 Copyright © ARM Limited 2000. All rights reserved.

The timing parameters for SRAM read and write accesses are listed in Table 2-1.

Table 2-1 SRAM/ROM access timing

Parameter	Description	Min	Max
t_{ADDRh}	Address hold time	8.5	-
t_{ADDRd}	Address delay time	7.08	17.5
t_{NROE}	ROM and SRAM output enable	5.7	13.6
t_{NWBE}	ROM and SRAM write byte enable delay	7.2	19.1
t_{NRCS}	ROM and SRAM chip select delay	5.2	12.4
t_{RDh}	Read data hold	3	-
t_{BA}	Byte access time	16.2	28.1
t_{AA}	Address access time	28.5	-
t_{CO}	Chip select to output time	25.2	32.4

Copyright © ARM Limited 2000. All rights reserved.

2.4 Serial ports

The Evaluator-7T provides two RS232 serial ports:

DEBUG — This uses COM1 as a console port. It is used by the debug monitor or bootstrap program running on the board. COM1 is connected to UART1 of the microcontroller.

USER — This uses COM0 as a general purpose port for program use. COM0 is connected to UART0 of the microcontroller.

The pinout of the two serial connectors is shown in Figure 2-6.

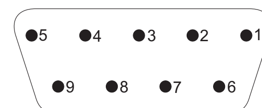


Figure 2-6 Pinout of the RS232 serial port connectors (P1 and P2)

Table 2-2 shows the signal assignment for the two serial connectors.

Table 2-2 Pinout of the RS-232 serial port connectors (P1 and P2)

Pin	Signal	Board use
1	**DCD**	NC
2	**RXD**	Connected
3	**TXD**	Connected
4	**DTR**	Connected
5	**GND**	Connected
6	**DSR**	Connected
7	**RTS**	NC
8	**CTS**	NC
9	**RI**	NC

 Copyright © ARM Limited 2000. All rights reserved.

Figure 2-7 shows the serial transceivers used to convert the 3.3V logic level of the microcontroller to the RS232 line levels required at the DB-9 serial port connectors. Conversion is performed by U4 for COM1 and U12 for COM0

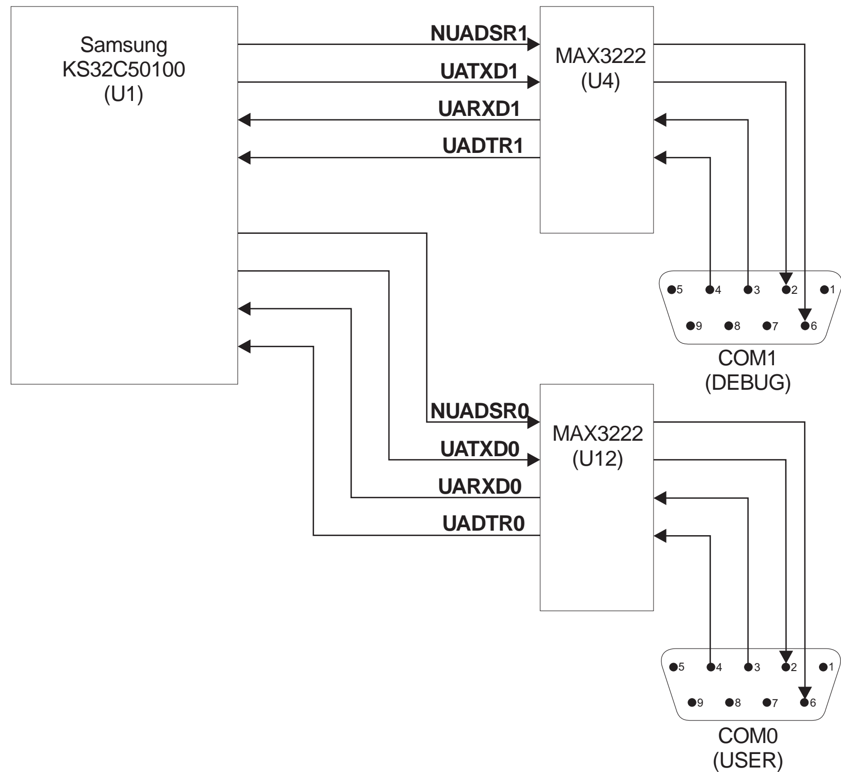


Figure 2-7 Serial interface architecture

Copyright © ARM Limited 2000. All rights reserved.

2.5 LEDs

There are two LED indicator circuits on the ARM Evaluator-7T:

- four surface-mounted LEDs
- a seven-segment LED display.

2.5.1 Surface-mounted LEDs

The four user-programmable LEDs, D1 to D4, are connected to a 74HC125 tristate buffer. The inputs to the buffer are driven by **PIO[7:4]** from the microcontroller. The LEDs control architecture is shown in Figure 2-8.

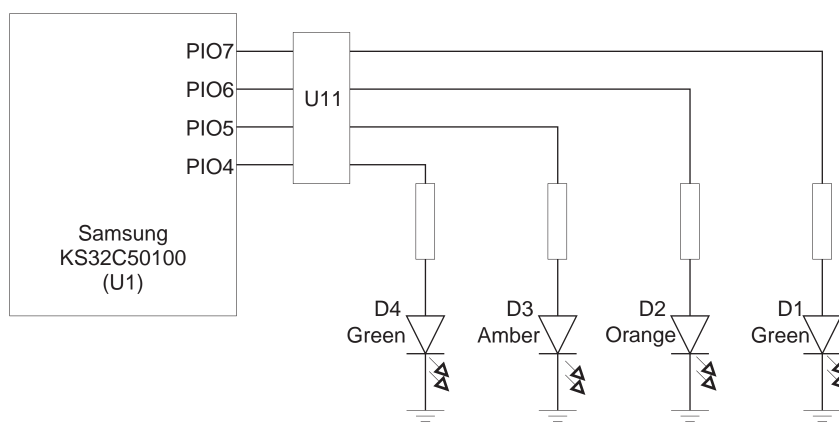


Figure 2-8 Architecture of the surface mount LEDs

2.5.2 Seven-segment display

The seven segments are controlled by **PIO[16:10]** from the microcontroller and two 74HC125 tristate buffers. The display also contains a decimal point LED. This is used as a power ON indicator and is connected to the 3.3V power plane.

Figure 2-9 on page 2-12 shows the assignment of the display segments to the PIO pins of the microcontroller.

 Copyright © ARM Limited 2000. All rights reserved.

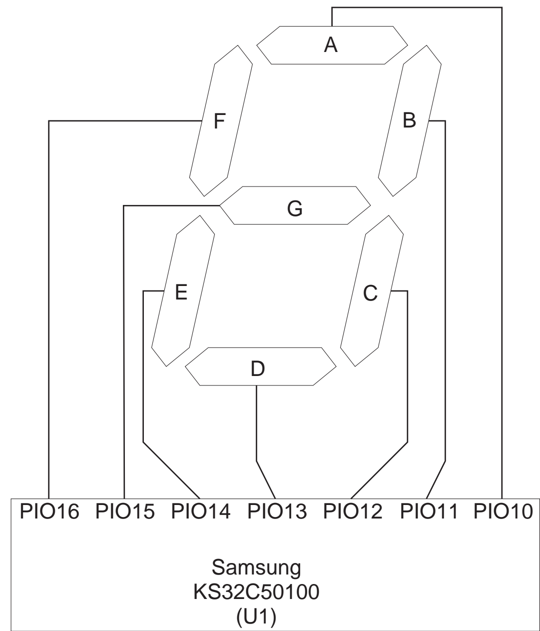


Figure 2-9 PIO to segment assignment

Copyright © ARM Limited 2000. All rights reserved.

2.6 Switches

The Evaluator-7T provides a 4-way DIP switch, a user interrupt switch, and two reset switches.

2.6.1 DIP switch

The four switches within the DIP are independent and are connected to **PIO[3:0]**. Select the ON position to pull the corresponding PIO input HIGH. Select the OFF position to pull the corresponding PIO input LOW. Figure 2-10 shows the circuit for the DIP switch.

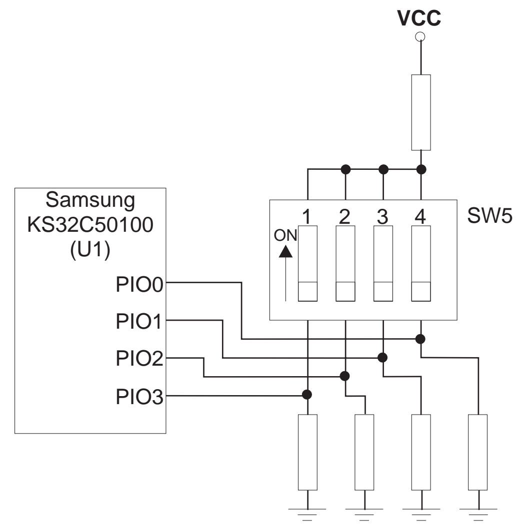


Figure 2-10 Schematic of DIP switch

2.6.2 User interrupt switch

The user interrupt switch is a momentary switch SW3. When pressed and released it results in a pulse on the **XINREQ0/P8** input of the microcontroller.

 Copyright © ARM Limited 2000. All rights reserved.

2.7 JTAG port

The 20-pin connector (J1) is connected to the JTAG interface of the microcontroller. The pinout is compatible with the ARM Multi-ICE interface unit. A pinout of the JTAG connector is shown in Figure 2-11.

Signal	Pin	Pin	Signal
Vcc	1	2	**Vcc**
NTRST	3	4	**GND**
TDI	5	6	**GND**
TMS	7	8	**GND**
TCK	9	10	**GND**
RTCK	11	12	**GND**
TDO	13	14	**GND**
NRESET	15	16	**GND**
NC	17	18	**GND**
NC	19	20	**GND**

Figure 2-11 Pinout of JTAG connector (J1)

Copyright © ARM Limited 2000. All rights reserved.

2.8 Power supply

The Evaluator-7T is powered through an external unregulated 9V DC power supply unit. This is plugged into the jack connector J7. It supplies an input to the on-board switch-mode regulator that supplies the 3.3V power to components on the board. Diode D12 is used to protect against reverse polarity on the power input.

Pin 3 on the jack socket is connected to the VCC (3.3V) power plane and shorts to ground when the power plug is removed. This discharges the bulk capacitance in the board power plane.

Copyright © ARM Limited 2000. All rights reserved.

Copyright © ARM Limited 2000. All rights reserved.

Chapter 3
Programmers Reference

This chapter describes the memory map and registers. It contains the following sections:

- *General memory map* on page 3-2
- *Memory usage* on page 3-3
- *Microcontroller register usage* on page 3-5
- *Accessing LEDs and switches* on page 3-6.

 Copyright © ARM Limited 2000. All rights reserved.

3.1 General memory map

The Evaluator-7T uses both flash and SRAM memory devices:

- the flash contains the *BootStrap Loader* (BSL), Angel debug monitor, and production test code
- you can us the SRAM for read-write data and for code.

On power-up, the microcontroller only has access to the flash memory. The BSL code modifies registers in the system memory controller to allow access to the installed memory.

3.1.1 Memory map at system reset

Refer to *Samsung KS32C50100 32-BIT RISC Micro Controller Embedded Network Controller User's Manual* for details on the system memory map at reset.

3.1.2 Memory map after remap

After reset the BSL code begins running from address `0x0`, and then reconfigures the memory map very early in its execution. After the BSL reconfigures the memory map, it is structured as shown in Table 3-1.

Table 3-1 Memory map after remap

Address range	Size	Description
`0x00000000` to `0x0003FFFF`	256KB	32 bit SRAM bank, using ROMCON1
`0x00040000` to `0x0007FFFF`	256KB	32 bit SRAM bank, using ROMCON2
`0x01800000` to `0x0187FFFF`	512KB	16 bit flash bank, using ROMCON0
`0x03FE0000` to `0x03FE1FFF`	8KB	32 bit internal SRAM
`0x03FF0000` to `0x03FFFFFF`	64KB	Microcontroller register space

Note

The BSL does not enable the cache. When the caches are enabled, you cannot use the 32-bit internal SRAM.

 Copyright © ARM Limited 2000. All rights reserved.

3.2 Memory usage

Memory usage changes slightly depending on whether BSL or Angel is running.

3.2.1 SRAM usage under the BSL

Table 3-2 shows the SRAM usage under BSL.

Table 3-2 SRAM usage under BSL

Address range	Description
`0x00000000` to `0x0000003F`	Exception vector table and address constants
`0x00000040` to `0x00000FFF`	Unused
`0x00001000` to `0x00007FFF`	Read-write data space for BSL
`0x00008000` to `0x00077FFF`	Available as download area for user code and data
`0x00078000` to `0x0007FFFF`	System and user stacks

3.2.2 SRAM usage under Angel

Table 3-3 shows the SRAM usage under Angel.

Table 3-3 SRAM usage under Angel

Address range	Description
`0x00000000` to `0x0000003F`	Exception vector table and address constants
`0x00000040` to `0x000000FF`	Unused
`0x00000100` to `0x00007FFF`	Read-write data and privileged mode stacks
`0x00008000` to `0x00073FFF`	Available as download area for user code and data
`0x00074000` to `0x0007FFFF`	Angel code execution region

 Copyright © ARM Limited 2000. All rights reserved.

3.2.3 Flash memory usage

Table 3-4 shows the flash memory usage.

Table 3-4 Flash memory usage

ADDRESS RANGE	DESCRIPTION
0x01800000 to 0x01806FFF	Bootstrap loader
0x01807000 to 0x01807FFF	Production test
0x01808000 to 0x0180FFFF	Reserved
0x01810000 to 0x0181FFFF	Angel
0x01820000 to 0x0187FFFF	Available for your programs and data

Copyright © ARM Limited 2000. All rights reserved.

3.3 Microcontroller register usage

Table 3-5 lists the registers used by the system software.

Caution

Exercise caution before modifying any of the registers to prevent improper functioning.

For details on how they are used by the system software, refer to `\Source\afs11\uHAL\Boards\EVALUATOR7T` and `\Source\afs11\angel\Evaluator7t`.

Table 3-5 Microcontroller register usage

System manager group	Input/output ports	Interrupt controller	UART
SYSCFG	IOPMOD	INTMOD	ULCON1
EXTDBWTH	IOPCON	INTPND	UCON1
ROMCON0	IOPDATA	INTMSK	USTAT1
ROMCON1	-	-	UTXBUF1
ROMCON2	-	-	URXBUF1
-	-	-	UBRDIV1

Copyright © ARM Limited 2000. All rights reserved.

3.4 Accessing LEDs and switches

Refer to Chapter 2 *Hardware Description* for details on how the LEDs and switches are connected to the microcontroller. You are recommended to use a read-modify-write strategy when writing to system registers.

Note

The example code excerpts shown in this section are taken from `\Source\prod_test\prodtest.c` and `segdisp.h`. For other examples see `\Source\afs11\uHAL\Boards\EVALUATOR7T`, `\Source\examples\DIPS`, and `\Source\examples\Switch`.

3.4.1 Simple LEDs

Use the input/output ports PIO[7:4] to control the four simple LEDs as follows:

- SET bits [7:4] in the register IOPMOD to configure ports as outputs.
- SET bits [7:4] in the register IOPDATA to light LEDs.
- CLEAR bits [7:4] in the register IOPDATA to turn LEDs OFF.

Example 3-1 shows an example code segment used to control the simple LEDS. Programs that are downloaded under Angel or the BSL can assume the LEDs are available and ready for use.

Example 3-1 Simple LED control

```
#define ALL_LEDS                    0xF0

void SetLEDs( unsigned val )
{

   *(volatile unsigned *)IOPDATA &= ~ALL_LEDS;
   *(volatile unsigned *)IOPDATA |= val << 4;
}
```

 Copyright © ARM Limited 2000. All rights reserved.

3.4.2 Seven segment LED Display

Use the input/output ports P[16:10] to control the seven segment display as follows:

- SET bits [16:10] in register IOPMOD to configure ports as outputs.
- SET bits [16:10] in register IOPDATA to light segments.
- CLEAR bits[16:10] in the register IOPDATA to turn segments OFF.

Example 3-2 shows a code fragment that controls the seven-segment LED display. Programs that are downloaded under Angel or the BSL can assume the seven-segment display is available and ready for use.

Example 3-2 Seven segment display and DIP switch reading

```
/* The bits taken up by the display in IODATA register */
#define    SEG_MASK   (0x1fc00)

/* define segments in terms of IO lines */
#define    SEG_A   (1 << 10)
#define    SEG_B   (1 << 11)
 [ … ]

#define    DISP_0   (SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F)
#define    DISP_1   (SEG_B | SEG_C)
[ … ]

const unsigned numeric_display[] = { DISP_0, DISP_1, DISP_2, DISP_3, DISP_4, DISP_5, DISP_6,
                           DISP_7, DISP_8, DISP_9, DISP_A, DISP_B, DISP_C, DISP_D, DISP_E,
                           DISP_F };

unsigned poll_dipSwitch( void )
{
  unsigned   ioData, Switch;

   Switch = SWITCH_MASK & *(volatile unsigned *)IOPDATA;
   SetLEDs( Switch );
   ioData = numeric_display[Switch];
   *(volatile unsigned *)IOPDATA &= ~SEG_MASK;
   *(volatile unsigned *)IOPDATA |= ioData;
   return( Switch );
}
```


Copyright © ARM Limited 2000. All rights reserved.

3.4.3 DIP switch

Use input/output ports P[3:0] to read the DIP switch SW5 as follows:

- CLEAR bits [3:0] of register IOPMOD to configure ports as inputs.
- Read the current setting of the switches from the register IOPDATA:

  1 = switch set to ON

  0 = switch set to OFF.

The DIP switch can be read immediately after system reset, because the power-on default for IOPCON is zero. See Example 3-2 on page 3-7 for an example of how to read the DIP switch.

3.4.4 User interrupt switch

The input/output port P8 is connected to SW3. You can use this to as an interrupt input INT0. To enable this operation:

- SET bit 3 of register IOPCON.
- CLEAR bit 0 of register INTMSK so that an interrupt can be triggered by pressing the switch.

Example 3-3 shows how the SW3-generated interrupts are enabled and cleared.

You can freely acquire interrupts under the BSL, because it does not use interrupts. Programs running under Angel need to carefully chain in a new interrupt handler, because Angel makes use of serial IRQs on the serial port. Refer to the *ADS Developer Guide*.

Example 3-3 User interrupt control

```
#define EnableInterrupt( n )    ( *(volatile unsigned *)INTMSK &= ~(1 << n) )
#define DisableInterrupt( n )   ( *(volatile unsigned *)INTMSK |= (1 << n) )

/* Interrupt controller defines, SW3 is tied to external INT0 */
#define      INT_GLOBAL      (21)
#define      INT_SW3_MASK    (1)
#define      INT_SW3_NUM     (0)

/* IO controller defines for SW3 */
#define      IO_ENABLE_INT0         (1 << 4)
#define      IO_ACTIVE_HIGH_INT0    (1 << 3)
#define      IO_RISING_EDGE_INT0    (1)
```


Copyright © ARM Limited 2000. All rights reserved.

```
unsigned cmain( void )
{

[ … ]

/* disable interrupts, but pending bit will still be set by an active
   interrupt */
   EnableInterrupt( INT_SW3_NUM );
   DisableInterrupt( INT_GLOBAL );

  *(volatile unsigned *)IOPCON = IO_ENABLE_INT0 | IO_ACTIVE_HIGH_INT0 | IO_RISING_EDGE_INT0;

[ … ]

while ( 0 == ( (1 << INT_SW3_NUM ) & *(volatile unsigned *)INTPND) )
     {
     ;
      }   /* wait untill we sense the switch */

     *(volatile unsigned *)INTPND |= INT_SW3_MASK;      /* clear interrupt */

[ … ]

}
```


Copyright © ARM Limited 2000. All rights reserved.

Copyright © ARM Limited 2000. All rights reserved.

Chapter 4
Bootstrap Loader Reference

This chapter describes the use of the Evaluator-7T bootstrap loader. It contains the following sections:

- *About the bootstrap loader* on page 4-2
- *Basic setup with the BSL* on page 4-3
- *BSL command-line editor* on page 4-7
- *Modules* on page 4-19
- *Preparing a program for download* on page 4-29.

 Copyright © ARM Limited 2000. All rights reserved.

4.1 About the bootstrap loader

The *BootStrap Loader* (BSL) is located in the bottom of flash memory (see *Flash memory usage* on page 3-4). The BSL is the first code to be executed by the KS32C50100 microcontroller when it powers up or resets. The BSL code has the following main functions:

- connecting to the host using a standard serial port and terminal application
- providing facilities to configure the board
- providing user help
- managing the images in flash as a set of executable modules
- allowing you to download applications to SRAM and execute them.

Copyright © ARM Limited 2000. All rights reserved.

4.2 Basic setup with the BSL

This section describes how to set up the Evaluator-7T and communicate with the BSL. The subsections that describe the steps are as follows:

- *Connecting the Evaluator-7T* on page 4-4
- *Communicating with a Unix host* on page 4-4 or
- *Communicating with a PC host* on page 4-5
- *Resetting the Evaluator-7T* on page 4-5
- *Solving communications problems* on page 4-6.

Figure 4-1 shows the Evaluator-7T setup.

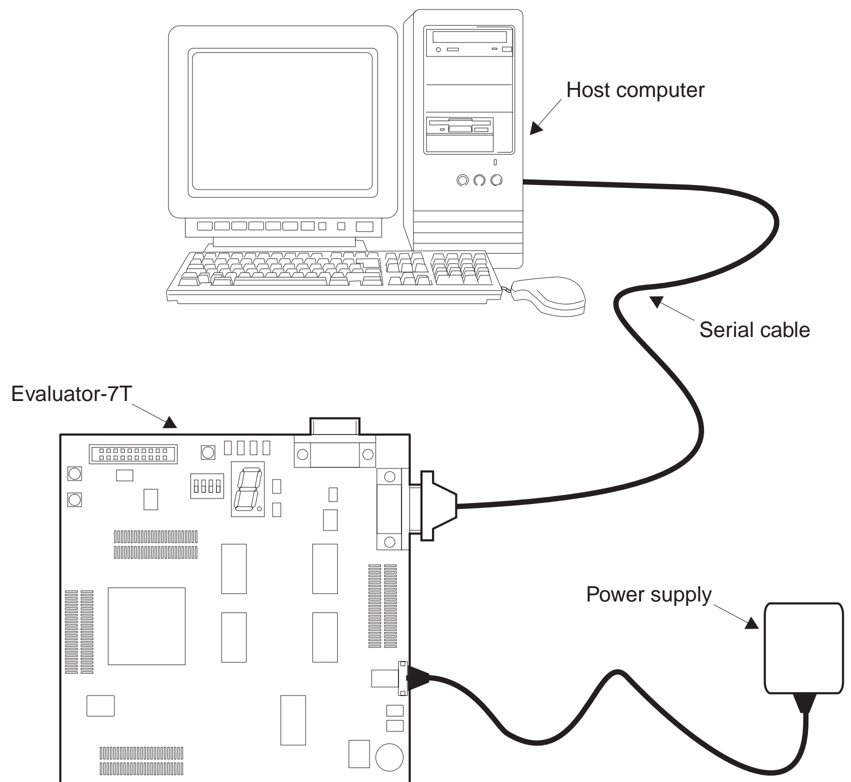


Figure 4-1 Bootstrap loader setup configuration

Copyright © ARM Limited 2000. All rights reserved.

4.2.1 Connecting the Evaluator-7T

Set up the Evaluator-7T as follows:

1. Connect the serial cable between the Evaluator-7T board and the host computer. Make a note of the serial port on the computer that you use.
2. Connect the power adapter to the power connector on the Evaluator-7T board.
3. Connect the power adapter to an AC power socket. The dot on the seven-segment display lights up as a power indicator.

4.2.2 Communicating with a Unix host

To communicate with the BSL you need to run a simple terminal application on the Unix host. In this example `tip` is used.

To start `tip` enter:

```
tip –<baud-rate> <device name>
```

Where `baud-rate` is one of the baud rates listed in Table 4-1 and `device name` is the name of the device associated with the serial port attached to the board (usually `/dev/ttya` or `/dev/ttyb`). For example:

```
tip -38400 /dev/ttya
```

Table 4-1 Supported BSL serial line settings

Baud rate	Data bits	Parity	Stop bits	Flow control
9600	8	None	1	None
19200	8	None	1	None
38400	8	None	1	None
57600	8	None	1	None
115200	8	None	1	None

Copyright © ARM Limited 2000. All rights reserved.

4.2.3 Communicating with a PC host

You can use Windows HyperTerminal to communicate with the BSL. Start and configure HyperTerminal as follows:

1. To start the HyperTerminal program, select **Start**, **Programs**, **Accessories**, and then **HyperTerminal.** The HyperTerminal **Connection Description** dialog is displayed.
2. Enter a name for this setup in the dialog box (for example *ArmEval*) and click on **OK**. The **Connect To** dialog is displayed.
3. Select the COM port you have connected the Evaluator-7T to from the **Connect using** menu and click on **OK**. The **COMx Properties** dialog is displayed.
4. In **COMx Properties** dialog, select a baud rate (refer to the of supported baud rates shown in Table 4-1 on page 4-4). If you are not using a VT100 emulator, connect initially at 9600 baud. (The board cannot detect the baud rate if you are not using a VT100 emulator.) You can configure the board later to use a higher baud rate.
5. Select **None** from the **Flow Control** menu and click on **OK**. HyperTerminal is now prepared for output from the board.

4.2.4 Resetting the Evaluator-7T

1. Press the SYS RESET button (SW1) on the Evaluator-7T. A banner similar to the following is displayed in the HyperTerm window:

```
ARM Evaluator7T Boot Monitor Release 1.00
Press ENTER within 2 seconds to stop autoboot
```

Note

If a banner is not displayed, refer to *Solving communications problems* on page 4-6.

2. Press `Enter` *within 2 seconds* to prevent the board from autobooting any other modules that may be stored in flash. The prompt `Boot:` is displayed and the LEDs D3 and D4 are lit.
3. Type `boot` at the `Boot:` prompt. The following response is displayed:

```
Scanning ROM for modules ...
  Found module 'BootStrapLoader' at 018057c8
  Found module 'ProductionTest' at 018072c0
  Found module 'Angel' at 0181a818
Boot:
```


Copyright © ARM Limited 2000. All rights reserved.

4.2.5 Solving communications problems

If the banner, described in step 6 above, is not displayed, check the following:

1. Check that you are using one of the supported baud rates shown in Table 4-1 on page 4-4.
2. Check that you are using a VT100 emulator and not another type.
3. Switch to 9600 baud. If the board cannot detect the baud rate you are using it defaults to 9600 baud.
4. Regardless of the baud rate, always configure your terminal emulator for 8 bits data, No parity, 1 stop bit. Ensure that you disable any flow control on your terminal emulator (Xon/Xoff or hardware handshaking). If you cannot disable hardware flow control then tie some or all of CTS, DSR, and CD lines on your serial port HIGH.
5. Check that you are using the correct serial cable. The cable requires three connections, signal ground, Rx, and Tx. Rx, and Tx must not be crossed over (that is, it must be a straight-through cable).

Note

The BSL stores environment variables that are used to configure the board. One of these environmental variables is used to set the baud rate (see *setenv* on page 4-9).

Copyright © ARM Limited 2000. All rights reserved.

4.3 BSL commands

This section describes the BSL command-line editor and the available BSL commands:

- *BSL command-line editor*
- *Basic commands* on page 4-8
- *Flash and module management* on page 4-12
- *Downloading and executing an application* on page 4-15.

4.3.1 BSL command-line editor

The BSL provides a command-line editor that allows you to type in and modify commands. These editing facilities are built into the BSL Read Line Software Interface so that any other module that uses this interface can use the editing facilities. The command-line editor keys are shown in Table 4-2.

Table 4-2 Command-line editor keys

Key	Function
`Backspace`	Delete the character before the cursor.
`Delete`	Same function as backspace.
`CTRL-A`	Move the cursor to the start of the current line.
`CTRL-B`	Move the cursor back one character.
`CTRL-D`	Forward delete. Delete the character under the cursor. If entered on an empty line `CTRL-D` is treated as End Of File
`CTRL-E`	Move the cursor to the end of the current line.
`CTRL-F`	Move the cursor forward one character.
`CTRL-R`	Redraw the current line.
`CTRL-U`	Erase the current line.

Copyright © ARM Limited 2000. All rights reserved.

4.3.2 Basic commands

This section describes the basic commands:

- *boot*
- *help*
- *setenv* on page 4-9
- *unsetenv* on page 4-11
- *printenv* on page 4-11.

The commands are not case sensitive.

boot

Usage `boot`

Use the `boot` command to scan the flash ROM for bootable modules:

```
Boot: boot
Scanning ROM for modules ...
  Found module 'BootStrapLoader' at 018057c8
  Found module 'ProductionTest' at 018072c0
  Found module 'Angel' at 0181a818
```

help

Usage `help <command>`

Enter `help` with no arguments to return a list of commands supported by the BSL. The `help` command goes through each module in flash ROM and lists all the commands supported by each module. For example:

```
Boot: help
Module is BootStrapLoader v1.0 Apr 27 2000 10:33:58
Help is available on:
Help        Modules        ROMModules       UnPlug       PlugIn
Kill        SetEnv         UnSetEnv         PrintEnv     DownLoad
Go          GoS            Boot             PC           FlashWrite
FlashLoad   FlashErase

Module is ProductionTest  v1.0 Apr 27 2000 10:49:47

Module is Angel           1.31.1 (20 Mar 2000)
```

 Copyright © ARM Limited 2000. All rights reserved.

To get help on a specific command, enter `help <Command>`. This displays a brief one-line help on the command. For example:

```
Boot: help help
Usage: Help [<command>]
Help gives help on the command, if none specified, gives a list of
commands.
```

You can also specify a module name instead of the `<command>`. This lists all the commands supported by that module. For example:

```
Boot: help bootstraploader
Module is BootStrapLoader v1.0 Apr 27 2000 10:33:58

Help is available on:
Help        Modules     ROMModules     UnPlug     PlugIn
Kill        SetEnv      UnSetEnv       PrintEnv   DownLoad
Go          GoS         Boot           PC         FlashWrite
FlashLoad   FlashErase
```

This only gives help about the BSL module.

setenv

Usage `setenv <variable-name> <value>`

Use `SetEnv` to set an environment variable in flash. You can program any variable name into flash. However, certain variable names are recognized by different modules in the system to provide for configuration options. These are listed in Table 4-3 on page 4-10.

If you are writing your own module you are likely to assign your own variable names to have specific meaning for your module, for example:

```
Boot: setenv baud 38400
```

This tells the BSL to use a baud rate of 38400, but does not take effect until the board is reset.

You can omit the `<value>` part to set a BOOLEAN type variable that is assumed to be TRUE if the variable exists or FALSE if it does not. For example:

```
Boot: setenv baud 38400
Boot: setenv noautobaud
```

The command `setenv noautobaud` tells the BSL not to do automatic baud rate detection on startup. Used in conjunction with the `setenv baud 38400` command, a fixed baud rate of 38400 is set on the board.

 Copyright © ARM Limited 2000. All rights reserved.

If you just use the `setenv baud 38400` command, then auto baud rate detection overrides the configured baud rate. The configured baud rate is applied only if the board cannot determine the baud rate you are using. Enter these commands if you had difficulty getting started with the board and had to revert to 9600 baud.

Caution

Do not set the baud rate to a baud rate higher than your terminal can support. If you do, you might not be able to regain control of the board.

Table 4-3 Environment variables used by the basic BSL

Variable	Value	Effect
`noautoboot`		If this variable is set then the BSL bypasses the normal autoboot sequence and goes straight to the `Boot:` prompt. You can use this to prevent the BSL from automatically starting another module stored in flash.
`boot`	`<boot-module>`	This variable is used to specify the name of a module to boot at startup. If this variable is set, BSL boots that module. Otherwise the BSL boots the last module in the module list that has the `AutoBoot` bit set.
`noautobaud`		Set this variable to force the BSL to bypass the normal baud rate detection and default to the configured baud rate, or to 9600 baud if no baud rate is configured.
`baud`	`<baud-rate>`	Use this to configure the baud rate for the board to one of 9600, 19200, 38400, 57600, or 115200. The BSL first performs automatic baud rate detection (subject to the setting of the `noautobaud` variable) and only uses this value if the baud rate could not be determined or if the `noautobaud` variable is set.

Copyright © ARM Limited 2000. All rights reserved.

unsetenv

Usage `unsetenv <variable-name>`

Use `UnSetEnv` to remove an environment variable previously created with `setenv`.

For example:

```
Boot: unsetenv noautobaud
Boot: unsetenv baud
```

printenv

Usage `printenv`

Use `PrintEnv` to list the variables currently stored in the environment area in the flash. For example:

```
Boot: printenv
Variable        Value
========        =====
noautobaud
baud            38400
```


Copyright © ARM Limited 2000. All rights reserved.

4.3.3 Flash and module management

The flash memory stores a number of executable modules. The flash shipped with the evaluation board contains three modules. These are:

- BSL module
- Production test module (see *Production test module* on page 4-30)
- Angel.

By default, Angel is automatically run unless the BSL is interrupted by pressing `Enter` within 2 seconds after startup.

The flash and module management commands are as follows:

- *modules*
- *rommodules* on page 4-13
- *modulename* on page 4-13
- *unplug* on page 4-13
- *plugin* on page 4-14
- *kill* on page 4-14.

modules

Usage `modules`

Use `modules` to display a list of all initialized modules. For example:

```
Boot: modules
Header   Base     Limit    Data
018057c8 01800000 018059e7 00000000 BootStrapLoader v1.0 Apr 27 2000
018072c0 01807000 01807308 00000000 ProductionTest  v1.0 Apr 272 000
0181a818 01810000 0181a860 00000000 Angel           1.31.1 (20 Mar 2000)
```

where:

`Header` Is the address of the module header within the module.

`Base` Is the first address of the module in flash.

`Limit` Is the last address (+1) of the module in flash.

`Data` Is the address of the modules data (0 => none).

Note

The displayed information for a specific board may be slightly different.

Copyright © ARM Limited 2000. All rights reserved.

rommodules

Usage `rommodules`

Enter `rommodules` to display a list of all modules in flash (as opposed to `modules` which lists only those modules that have been initialized). For each module, `rommodules` prints the `Header`, `Base` and `Limit` information, as for `modules`, but does not print the `Data` information. This is because an uninitialized module cannot have any data.

This command displays a list of all modules available in flash with the version number, date, and base address in flash of each module. For example:

```
Boot: rommodules
Header   Base     Limit
018057c8 01800000 018059e7 BootStrapLoader v1.0 Apr 27 2000 10:33:58
018072c0 01807000 01807308 ProductionTest  v1.0 Apr 27 2000 10:49:47
0181a818 01810000 0181a860 Angel           1.31.1 (20 Mar 2000)
```

modulename

Usage `modulename`

Enter the name of a module to run that module. For example:

```
Boot: bootstraploader
ARM Evaluator7T Boot Monitor PreRelease 1.00
Press ENTER within 2 seconds to stop autoboot
Boot:
```

This reruns the BSL module, which has the effect of rebooting the board.

unplug

Usage `unplug <module name>`

Enter `unplug` to:

- prevent BSL initializing the specified module when the board is next booted
- kill an active module (which has been initialized).

The `unplug` command is useful if you have a module that is causing the board to crash when it is booted. In this situation:

1. Boot the board.
2. Press `<ENTER>` to interrupt the boot.
3. Enter `unplug <modulename>`.

Copyright © ARM Limited 2000. All rights reserved.

4. Reboot the board.

Repeat this process to isolate the problem module, and then use the `plugin` command to reinstate the modules that you know to be problem free.

Note

Do not `unplug` the BSL itself. However, you can still recover by booting the board and pressing `<ENTER>`. The BSL initializes itself allowing you to regain control by using the `plugin` command.

---

plugin

Usage `plugin <module name>`

Enter the `plugin` command to reinstate a module that has been unplugged with the `unplug` command. The `plugin` command marks the module so that the BSL finds it next time the board is booted. The `plugin` command also initializes the module.

You can use the `plugin` command to initialize a module which failed to initialize at boot time.

kill

Usage `kill <module name>`

Use the `kill` command to halt a module by calling its finalization code. Unlike the `unplug` command it does not mark the module as unplugged so the module is initialized the next time the board is booted.

Use `kill` to remove a module temporarily, or use `unplug` to remove it permanently.

 Copyright © ARM Limited 2000. All rights reserved.

4.3.4 Downloading and executing an application

This section describes commands used to download and execute images on the Evaluator-7T. These commands are as follows:

- *download*
- *go*
- *gos* on page 4-17
- *pc* on page 4-17
- *flashwrite* on page 4-17
- *flashload* on page 4-17
- *flasherase* on page 4-18.

download

Usage download [<address>]

Use the download command to download an image (for example an application) into RAM. The image must be converted to uuencoded format before it is downloaded. If no address is specified the image is downloaded at the address 0x8000, otherwise it is downloaded at the address specified.

To download an image:

1. Convert the image to uuencoded format, see *Preparing a program for download* on page 4-29.
2. Enter the download command at the Boot: prompt on the terminal connected to the board.
3. Transmit the uuencoded file down the serial line using the transmit file option on your terminal:
   a. If you are using HyperTerminal on a PC, select the **Send Text** file option from the **Transfer** menu, and enter the name of the uuencoded file you want to download in the dialog box.
   b. If you are using tip on a UNIX system, enter the command < ~> > followed by the name of the uuencoded file you wish to download. (You might need to press <RETURN> before entering < ~> >.)

If the BSL detects any errors during downloading, it prints a message similar to:

```
Error: 00000001 errors encountered during download.
```

 Copyright © ARM Limited 2000. All rights reserved.

If this occurs, try downloading again. If you are using a high baud rate (57600 or 115200), try using a lower baud rate.

Note

If after having entered `download` you want to exit the download function without downloading an image, type `CTRL-D`.

go

Usage `go [<program arguments>]`

Use the `go` command to start User mode execution of a program previously downloaded using the `download` command. The starting address of the program is set to the address at which the program was downloaded. Arguments to the program can be specified after the `go` command.

For example, if you build and download the following program:

```
--- echo.c ---
#include <stdio.h>
int main(int argc, char **argv)
{
int i;
for (i = 0; i < argc; i++)
puts(argv[i]);
return 0;
}
```

and then run it with the command:

```
Boot: go 1 2 3 4
```

you get the following output:

```
1
2
3
4
Program terminated with return code 00000000
```

For details on how to prepare programs for download, refer to *Preparing a program for download* on page 4-29.

 Copyright © ARM Limited 2000. All rights reserved.

gos

Usage gos [<program arguments>]

User gos command to execute a program in Supervisor (SVC) mode instead of in User mode.

pc

Usage pc <address>

Use the pc command to set the value of the stored *Program Counter* (PC). This command is used to set the address before entering a go or gos command. The go and gos commands read the stored pc into the ARM pc register (r15). If executed without any argument the pc command prints the current value of the stored pc.

flashwrite

Usage flashwrite <address><source><length>

Use the flashwrite command to write the area of memory specified by *source* and length to the flash, starting at the address specified by address. The address is the mapped address of the flash memory on the board. To convert a flash offset to an address, add 0x01800000, the base address of the flash in the memory map.

Caution

You must not write to the bottom 64KB of the flash memory (from 0x01800000 to 0x0180FFFF). This area of flash is reserved for the BSL module and production test module.

flashload

Usage flashload <address>

Use flashload to perform a download command and then write the result of the download into flash at the specified address.

Caution

As with flashwrite, do not attempt to load anything into the lower 64KB of flash memory.

 Copyright © ARM Limited 2000. All rights reserved.

flasherase

Usage `flashErase <address length>`

Use `flasherase` to erase the section of flash specified by address and length by overwriting it with `0xFF`.

Caution

As with `flashwrite` and `flashload` do not attempt to erase flash in the lower 64KB region of the flash.

 Copyright © ARM Limited 2000. All rights reserved.

4.4 Modules

The flash on the Evaluator-7T is provided to allow multiple independent programs to be stored and easily managed by the BSL. A single independent program is described as a *module*.

A module consists of two major components:

- a binary executable image of the program
- a `ModuleHeader` data structure that describes the image.

The BSL uses the `ModuleHeader` data structure in each module to manage the flash. This descriptive data structure is not required to be the first item in the module.

4.4.1 The module header data structure

The module header structure must take the following form:

```
typedef struct ModuleHeader ModuleHeader;
struct ModuleHeader {
    unsigned magic;
    unsigned flags:16;
    unsigned major:8;
    unsigned minor:8;
    unsigned checksum;
    ARMWord *ro_base;
    ARMWord *ro_limit;
    ARMWord *rw_base;
    ARMWord *zi_base;
    ARMWord *zi_limit;
    ModuleHeader *self;
    StartCode start;/* Optional - may be 0 */
    InitCode init;/* Optional - may be 0 */
    FinalCode final;/* Optional - may be 0 */
    ServiceCode service;/* Optional - may be 0 */
    TitleString title;
    HelpString help;
    CmdTable *cmdtbl;/* Optional - may be 0 */
    SWIBase swi_base;/* Optional - may be 0 */
    SWICode swi_handler;/* Optional - may be 0 */
};
```

 Copyright © ARM Limited 2000. All rights reserved.

4.4.2 Module header field descriptions

Table 4-4 lists the fields used in the module header.

Table 4-4 Module header fields

Offset	Name	Description
0x00	magic	Magic word (value = 0x4D484944) used to identify this as a module.
0x04	flags	16-bit flags field. The individual flags are described below.
0x06	major	Major version number. Currently this has the value 1.
0x07	minor	Minor version number. Currently this has the value 1.
0x08	checksum	An EOR checksum of the entire module used to validate the module.
0x0C	ro_base	The linked read-only base of the module =Image$$RO$$Base
0x10	ro_limit	The linked read-only limit of the module =Image$$RO$$Limit
0x14	rw_base	The linked read-write base of the module =Image$$RW$$Base
0x18	zi_base	The linked Zero Init base of the module =Image$$ZI$$Base
0x1C	zi_limit	The linked Zero Init limit of the module =Image$$ZI$$Limit
0x20	self	A pointer to the linked address of the module header.
0x24	start	The linked address of the start code called to boot a module.
0x28	init	The linked address of the init code called to initialize a module.
0x2C	final	The linked address of the final code called to kill a module.
0x30	service	The linked address of the service call entry of a module.
0x34	title	The linked address of the title string of the module.
0x38	help	The linked address of the help string of the module.

Copyright © ARM Limited 2000. All rights reserved.

Table 4-4 Module header fields (continued)

Offset	Name	Description
`0x3C`	cmdtbl	A pointer to the command table for this module.
`0x40`	swi_base	The base address of the 64 entry SWI chunk handled by this module.
`0x44`	swi_handler	A pointer to the SWI handler for this module.

The module header fields are as follows:

- *magic*
- *flags* on page 4-22
- *major, minor* on page 4-22
- *checksum* on page 4-22
- *ro_base, ro_limit, rw_base, zi_base, zi_limit* on page 4-22
- *start* on page 4-24
- *init* on page 4-24
- *final* on page 4-25
- *service* on page 4-25
- *title* on page 4-25
- *help* on page 4-26
- *cmdtbl* on page 4-26
- *swi_base* on page 4-27
- *swi_handler* on page 4-27.

magic

This word identifies a module header. You must set this word to the value MODULE_MAGIC which has the following definition:

```
#define MODULE_MAGIC 0x4d484944; /* 'MHID' */
```

You can use the word MAGIC to identify the big-endian or little-endian setting for a module. Therefore, the above definition must always be used. Because they do not correctly identify the byte ordering of the module, do not use definitions such as the following:

```
#define MODULE_MAGIC'MHID'
```

or

```
#define MODULE_MAGIC *(unsigned *)"MHID"
```


Copyright © ARM Limited 2000. All rights reserved.

When the BSL is booted, it searches the ROM(s) for the MODULE_MAGIC word. Each occurrence of the MODULE_MAGIC word is identified as a module, provided that the module checksum succeeds.

flags

The following flags are currently defined:

```
#define UNPLUGGED_FLAG 0x0001
#define AUTOSTART_FLAG 0x0002
```

Set all other bits in this field to zero.

The UNPLUGGED_FLAG is used to identify which modules have been unplugged (removed from the list of modules). Unplugged modules will be entered into the module list. However, none of their entries are ever called.

The AUTOSTART_FLAG is used to identify a single module that is automatically booted on startup in the absence of a boot module. Usually, the AUTOSTART_FLAG is set for only one module. In the case that more than one module has the AUTOSTART_FLAG set, the BSL boots the last such module found in a flash.

major, minor

These fields identify the major and minor version numbers of the module header. You can use these to allow future extension of the module header. The current version number is:

```
#define MAJOR_VERSION 1
#define MINOR_VERSION 1
```

checksum

This field is used to validate the module. The checksum is calculated over the range `<real_base>` to `<real_limit>`. The checksum value is set so that the checksum over this range is equal to 0. It is calculated as the *End of Range* (EOR) of each word in the range `<real_base>` to `<real_limit>`. The checksum is included in the range `<real_base>` to `<real_limit>` so there is no need to perform a final EOR of checksum to generate a zero result.

ro_base, ro_limit, rw_base, zi_base, zi_limit

These fields identify the extents of the module ROM and RAM regions. You should set these fields to the linked address of the regions.

 Copyright © ARM Limited 2000. All rights reserved.

The following list shows how these fields can be defined in an assembly language file using an ARM assembler:

```
IMPORT|Image$$RO$$Base|
IMPORT|Image$$RO$$Limit|
IMPORT|Image$$RW$$Base|
IMPORT|Image$$ZI$$Base|
IMPORT|Image$$ZI$$Limit|
ModuleHeaderDCDMODULE_MAGIC
    ...
    DCD|Image$$RO$$Base|
    DCD|Image$$RO$$Limit|
    DCD|Image$$RW$$Base|
    DCD|Image$$ZI$$Base|
    DCD|Image$$ZI$$Limit|
    DCDModuleHeader
    ...
```

The following list shows how these fields can be defined in a C file:

```
extern ARMWord Image$$RO$$Base[];
extern ARMWord Image$$RO$$Limit[];
extern ARMWord Image$$RW$$Base[];
extern ARMWord Image$$ZI$$Base[];
extern ARMWord Image$$ZI$$Limit[];
ModuleHeader module_header = {
    ...
    Image$$RO$$Base,
    Image$$RO$$Limit,
    Image$$RW$$Base,
    Image$$ZI$$Base,
    Image$$ZI$$Limit,
    &module_header,
    ...
};
```

The actual base and limit of the module are calculated as follows (where `<module_address>` is the address where the BSL located the MODULE_MAGIC word):

```
<real_RO_base> = ro_base + ( <module_address> - self)
<real_RO_limit> = ro_limit + ( <module_address> - self)
    + (zi_base - rw_base)
```

 Copyright © ARM Limited 2000. All rights reserved.

The actual base and limit of the module RAM region(s) are calculated as follows (where <static_base> is the static base address of the modules instantiation):

```
<real_RW_base> = <static_base>
<real_RW_limit> = <static_base> + (zi_limit - rw_base)
```

If a module is statically linked, it does not support position-independent data or multiple instantiation, then <static_base> == rw_base. Otherwise <static_base> is the static base address of the data for the current instantiation which is held in R9.

The self entry is also used to calculate the real entry points of the various entry points of the module as follows.

```
<real_start>= start + ( <module_address> - self)
<real_init>= init + ( <module_address> - self)
<real_final> = final + ( <module_address> - self)
<real_service> = service + ( <module_address> - self)
<real_title>= title + ( <module_address> - self)
<real_help> = help + ( <module_address> - self)
<real_cmdtbl>= cmdtbl + ( <module_address> - self)
<real_swi_handler>= swi_handler + ( <module_address> - self)
```

start

The start entry point is called to execute a module. A module does not have to have a start entry. In this case the start entry must be 0. The start code is only ever called after the module has been instantiated by a call to its `init` entry. A C language definition of this entry point is as follows:

```
typedef void (*StartCode)(char *cmd);
```

Where on entry:

R0 = Pointer to static data

R9 = ModuleHandle returned by `init`

init

The init entry point is called to instantiate a module. The value returned is used to identify the module instantiation in subsequent calls to other entries. The value returned is passed in R9 to other entry points. Usually the init entry allocates memory for its static data and initializes the static data. It then returns a pointer to its static data in R0. If the init entry is 0, the init entry is not called. A C language definition of this entry point is as follows:

```
typedef struct ModuleInfo *ModuleHandle;
typedef ModuleHandle (*InitCode)(void);
```

 Copyright © ARM Limited 2000. All rights reserved.

Where on exit:

R0 = Pointer to static data.

R9 = ModuleHandle used in subsequent calls to other entry points.

final

The final entry point is called to finalize a module. The module frees any resources allocated by it. Usually a module frees its static data which is pointed to by R0 on entry. The final entry might be 0 in which case the final entry is not called. A C code definition of this entry point is as follows:

```
typedef void (*FinalCode)(void);
```

Where on entry:

R9 = ModuleHandle return by `init`

service

The service entry is called to alert a module of various conditions. The module can use this to intercept certain conditions. For example, a debugger module intercepts a GO or DOWNLOAD service call. To intercept a service call the module should return a service continuation routine pointer in R0. This is called when the system is unthreaded, just before it returns to the original caller. If you do not want the module to intercept the call it must return 0 in R0. If a module does not support any service, the value 0 must be used. The C code definitions for the service entry point as well as a service continuation routine are shown below:

```
typedef void (*ServiceCont)(void);
typedef ServiceCont (*ServiceCode)(int service);
```

Where on entry:

R0 = Service number

R9 = ModuleHandle returned by `init`

And on exit:

R0 = Address of service continuation routine.

title

The title entry points to the title string for the module. This must be a 0 terminated string of 16 characters or less. A C code definition of this is:

```
typedef char *TitleString;
```


Copyright © ARM Limited 2000. All rights reserved.

help

The help entry points to the help string for the module. A C code definition of this is:

```
typedef char *HelpString;
```

The actual help string must use the following format:

```
<Module Name> V.VV (DD MMM YYYY) <Comment>
```

cmdtbl

The command table points to an array of command descriptions:

```
typedef struct CmdTable CmdTable;
typedef void (*CommandCode)(char *cmd);
struct CmdTable {
    char *command;
    CommandCode code;
    unsigned flags;
    char *syntax;
    char *help;
};
```

The `cmdtbl` array is terminated by an entry with a command field of 0.

The module can be relocatable and the real address of the command, code, syntax, and help entries are calculated using the following real address calculations:

```
<real_command> = command + ( <module_address> - self)
<real_code> = code + ( <module_address> - self)
<real_syntax> = syntax + ( <module_address> - self)
<real_help> = help + ( <module_address> - self)
```

The `cmdtbl` field is optional and can be 0 if no commands are supported.

The cmtdt data structure filed descriptions are as follows:

`command` This points to the name of the command. The command name must be less than 16 characters for the help command to work correctly. The command can be in any mixture of upper and lower case. The *Command Line Interface* (CLI) performs a case insensitive match on commands.

`code` This entry is called when a command matching the command field is entered. For a command to unthread the system, it should return the address of a continuation routine in R0. This can be used, for example, by a debugger when a `go` command is executed. The debugger unwinds the

Copyright © ARM Limited 2000. All rights reserved.

SVC stack before continuing execution of the debugger by returning the address of a continuation routine in R0. The C code definition of both the continuation and command entry-points are shown below:

```
typedef void (*CommandCont)(void);
```

```
typedef CommandCont (*CommandCode)(char *cmd);
```

Where on entry:

R0 = command tail.

And on exit:

R0 = address of continuation routine.

`flags` The flags field contains flags for the command. At the moment no flags are defined, so all bits in this word must be zero.

`syntax` This points to a string to give a syntax error message. The syntax message must be of the form

Usage: `<command> <arguments>`

`help` This points to help on the command.

swi_base

The `swi_base` entry gives a SWI chunk base for the module. A chunk is 64 entries so bits 0 to 5 must be zero in the `swi_base`. When a SWI occurs, which is in a modules `swi_chunk` range, the `swi_handler` entry is called. A value of 0 is used if no SWIs are supported. The C code definition of this entry is:

```
typedef unsigned SWIBase;
```

swi_handler

The swi_handler entry is called when a SWI is executed in the modules SWI chunk range. If you want the handler to intercept the SWI and not return to the caller, return the address of a continuation routine in R0. The system then unwinds the SVC stack and calls the continuation routine. A value of 0 can be used if no SWIs are supported. The C code definition of both the continuation and command entry points are shown below:

```
typedef void (*SWICont)(void);
typedef SWICont (*SWICode)(unsigned swino, SWIRegs *regs);
```

Where on entry:

R0 = the SWI number modulo 64.

Copyright © ARM Limited 2000. All rights reserved.

R1 = a pointer to register R0 through R12 on the stack. These registers can be modified by the SWI handler.

And on exit:

R0 = Address of continuation routine.

Copyright © ARM Limited 2000. All rights reserved.

4.5 Preparing a program for download

To prepare a program for download:

1. Compile or assemble the source code.
2. Link the resulting object files to create a standalone binary image.
3. Convert the binary image into a `uuencoded` format.

   The Evaluator-7T tools and documentation CD, which are supplied with the board, installs a `uuencode` application onto your host PC system. Use this application to convert a linked application image from DOS command line. For example:

   ```
   uuencode dhry.bin dhry.uue
   ```

This `uuencoded` image can be downloaded and debugged using a terminal application to the BSL on the Evaluator-7T board for execution.

Note

A standalone binary application in this case is defined as a program that is self-initializing and requires only services that are available directly from the Evaluator-7T hardware or the BSL. This definition includes any BSL module image.

Copyright © ARM Limited 2000. All rights reserved.

4.6 Production test module

The production test module can be invoked from the bootstrap loader in the usual way modules are invoked, by entering in its name at the boot prompt:

```
Boot: productiontest
```

The production test module operates as follows:

- It first lights all LEDs, D1-D4 and the seven segment display
- When SW3 is pressed, the test module reads the value from the DIP switch and displays it on the seven segment display.
- Setting the DIP switch to `0xF` and pressing SW3 causes the test module to exit this part of the test and start a test of the main user SRAM. (This corrupts any data in the SRAM.)
- If an error is found, LED D2 stays lit while the module waits for SW3 to be pressed. If no error is detected, the program returns to the boot prompt.

The production test module is automatically started if connect a loopback connection between COM0 and COM1, and then reset the board.

Copyright © ARM Limited 2000. All rights reserved.

Appendix A
Evaluator-7T Mechanical Outline

This appendix contains the mechanical outline of the Evaluator-7T. It contains the following section:

- *Mechanical outline* on page A-2.

Copyright © ARM Limited 2000. All rights reserved.

A.1 Mechanical outline

Figure A-1 shows the mechanical outline of the Evaluator-7T.

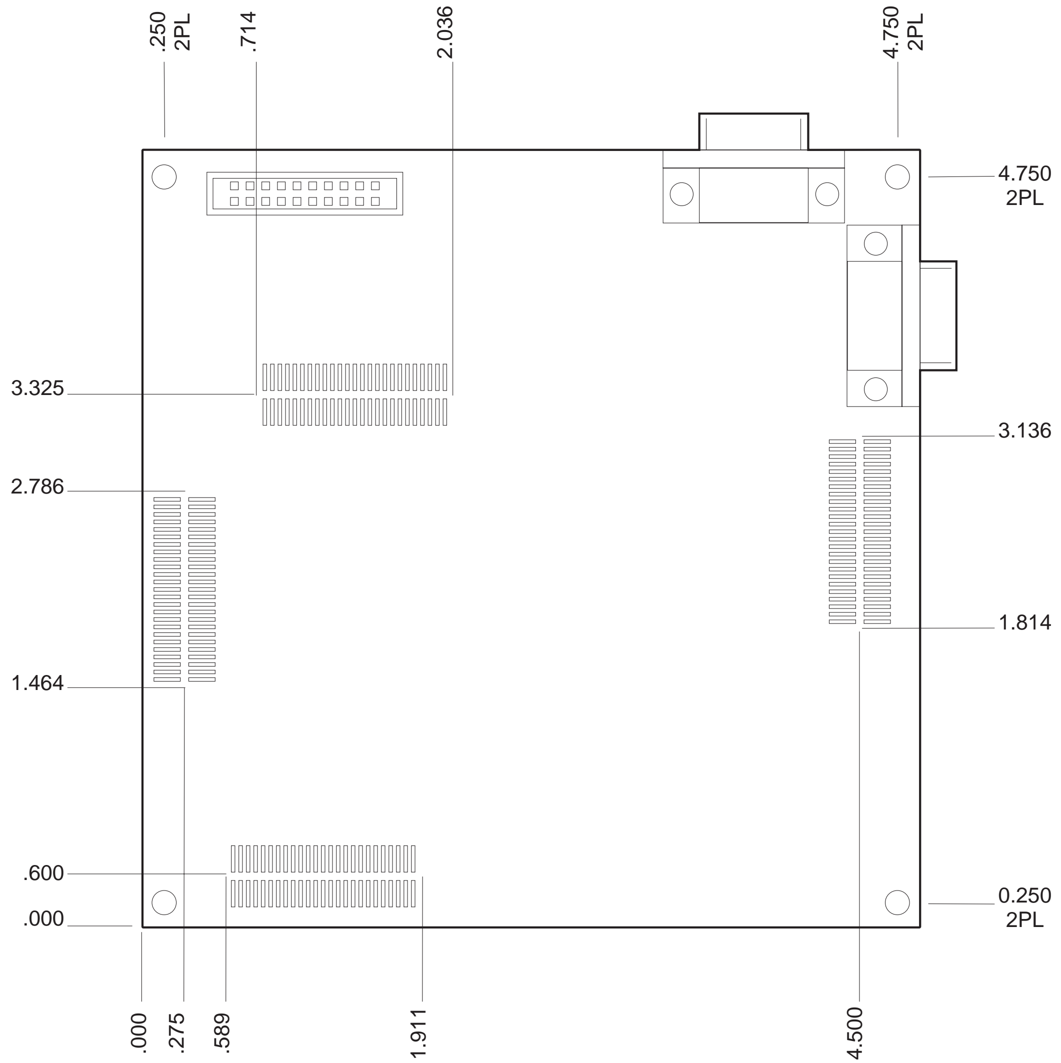


Figure A-1 Evaluator-7T mechanical outline

Copyright © ARM Limited 2000. All rights reserved.

Appendix B
Evaluator-7T Signal Naming

This appendix describes the signal naming used for the Evaluator-7T. It contains the following section:

- *Signal naming* on page B-4.

Copyright © ARM Limited 2000. All rights reserved.

B.1 Signal naming

The ARM Evaluator-7T schematics and this manual use different signal names to the *Samsung KS32C50100 32-BIT RISC Micro Controller Embedded Network Controller User's Manual*. In general the Samsung document uses **n<SIGNAL_NAME>** and this document uses **N<SIGNAL_NAME>**. For example **nWBE0** becomes **NWBE0**.

Other naming differences are shown in Table B-1.

Table B-1 Signal name differences

Description	This document	Samsung
Address bus	**A[21:0]**	**ADDR[21:0]**
External data bus	**D[31:0]**	**XDATA[31:0]**
General purpose input/output lines	**PIO[17:0]**	**P[17:0]**

Copyright © ARM Limited 2000. All rights reserved.

Index

The items in this index are listed in alphabetical order, with symbols and numerics appearing at the end. The references given are to page numbers.

A

Address access time 2-8
Address delay time 2-8
Address hold time 2-8
Alphanumeric display 2-11
Angel
  using 1-6
Angel debug monitor 3-2
Architecture
  overview 1-3
  reset system 2-4
ARM Multi-ICE unit 1-7
AUTOSTART_FLAG 4-22

B

Basic commands 4-8
Basic setup with BSL 4-3
Binary executable image 4-19
Board configuration commands 1-6
Board layout 1-2
Boot command 4-8
Bootstrap loader 2-5, 3-2, 4-1
  address 3-4
  communicating with 4-4
  functionality 1-6
  functions 4-2
  using 1-6
BSL command line editor 4-7
BSL commands 4-7
Byte access time 2-8

C

CE Declaration of Conformity ii
Chip Select delay 2-8
Command line editor 4-7
Command line editor keys 4-7
Command line interface 4-26
Communications problems 4-6
Compiling the source code 4-29
COMs ports 1-2
COM0 2-9
COM1 2-9
Connectors
  header 1-2
  Power 1-2
  serial 1-2
Contents, product package 1-4
Converting flash offset to an address 4-17
Core reset switch 1-2, 2-4

D

Debug monitor, Angel 1-6
DEBUG port 2-9
DIP switch 1-3, 2-13, 3-8
Download command 4-15
Downloading applications to SRAM 1-6

Copyright © 2000 ARM Limited. All rights reserved.
Confidential Draft

E

Erasing a section of flash 4-18
EXTDBWTH configuration register 2-5

F

Federal Communications Commission ii
Feedback, product xii
Flags field 4-27
flash 2-2
Flash bootROM 2-5
Flash management 4-12
Flash management tools 1-6
Flash memory 3-4
Flash memory usage 3-4
Flasherase command 4-18
Flashload command 4-17
Flashwrite command 4-17

G

Go command 4-16
Gos command 4-17

H

Help command 4-8
High-level data link control 2-2
Host system requirements 1-5

I

Interrupt button 1-2
Interrupt controller 2-2
Interrupt switch 2-13
Itended audience viii
I2C serial interface 2-2

J

JTAG connector 1-7
JTAG interface 2-14

K

Kill command 4-14
Kit contents 1-4

L

LED access 3-6
LED-PIO pots assignments 2-11
LEDs 2-11
Lower Byte, SRAM 2-5

M

Main components 1-2
Manual audience viii
Memory
  flash 2-5
  SRAM 2-5
Memory map 3-2
Memory map after remap 3-2
Memory usage 3-3
Microcontroller block diagram 2-2
Microcontroller pin connections 2-2
Microcontroller power 2-2
Microcontroller register usage 3-5
Microcontroller,overview 2-2
Module header field descriptions 4-20
Module header fields 4-20
Module header structure 4-19
Module management 4-12
module relocation 4-26
ModuleHeader data structure 4-19
Modulename command 4-13
Modules 4-19
Modules command 4-12
MODULE_MAGIC 4-21
Mulit-ICE
  Using 1-7

N

NRESET signal 2-4
NTRST signal 2-4

O

Other ARM publications xi
Output enable 2-8
Overview of Evaluator-7T 1-2

P

Pc command 4-17
Plugin command 4-14
Ports
  RS232 2-9
Power connector 1-2
Power indicator 2-11
Power supply 2-15
Precautions 1-8
PrintEnv command 4-11
Problems, solving 4-6
Product feedback xii
Product package contents 1-4
Program counter 4-17
Program download 4-29
Program modules 4-19
Programmable 32-bit timers 2-2

R

Read data hold 2-8
Related publications xi
Remap 3-2
Reset cicuit, description 2-4
Reverse polarity protection 2-15
rommodules command 4-13

S

Samsung microcontroller 2-2
SDRAM 2-2
Serial cable requirements 4-6
Serial connector pinout 2-9
Serial interface
  circuit 2-10
Serial ports 1-2, 2-9

Copyright © 2000 ARM Limited. All rights reserved.
Confidential Draft

SetEnv command 4-9
Setting up 1-6
Setting up the Evaluator-7T 4-4
Seven-segment display 1-2, 2-11, 3-7
Simple LED control 3-6
Simple LEDs 3-6
SRAM 2-5, 3-2
  timing parameters 2-8
  usage 3-3
  write timing 2-7
SRAM read timing 2-7
SRAM usage
  Angel 3-3
  BSL 3-3
Surface mounted LEDs 2-11
Switch
  core reset 2-4
  DIP 3-8
  system reset 2-4
  user interrupt 2-13, 3-8
Switches
  core reset 1-2
  DIP 2-13
  reset 1-2
Switch-mode regulator 2-15
SW1 2-4
SW2 2-4
System requirements, host 1-5
System reset
  debounce 2-4
  switch 2-4
System reset switch 1-2
System reset, memory map 3-2

T

Timing
  SRAM read 2-7
Timing parameters, SRAM 2-8
Tming diagram conventions x
Typographical conventions ix

U

Unplug command 4-13
UNPLUGGED_FLAG 4-22
UnSetEnv command 4-11
Upper Byte, SRAM 2-5
User DIP switches 1-2
User help 1-6
User interrupt switch 3-8
User LEDs 1-2
USER port 2-9
Using Angel 1-6
Using Multi-ICE 1-7

V

VT100 emulator 4-6

W

Write byte enable 2-8
Write Byte Enable, SRAM 2-5
Write timing, SRAM 2-7

Copyright © 2000 ARM Limited. All rights reserved.
Confidential Draft

Copyright © 2000 ARM Limited. All rights reserved.
Confidential Draft