



E7T-DBoard Reference Manual

Author: Chris Garry
Date: 23-Oct-2004
Doc Number: sde7t001
Revision: 1.8

Copyright © 2004 Sweeney Design Ltd

Revision History

Document Number: sde7t001

Rev	Change Summary	Date	Author
0.1	Created	14-Feb-2003	Chris Garry
0.2	Updated after document review	19-Feb-2003	Chris Garry
0.3	Updated Mem Map for 32MB SDRAM + general tidy	28-Feb-2003	Chris Garry
0.4	Updated fpgaload command configuration section.	02-Mar-2003	Chris Garry
0.5	Increased size of FPGA Image area in FLASH + corrected serial port names.	16-Mar-2003	Chris Garry
0.6	Added RAM selection CDL option. Removed references to full/lite-builds.	16-Apr-2003	Chris Garry
0.7	Corrected FLASH mem map, updated kit contents	19-Apr-2003	Chris Garry
0.8	Updated to reflect RedBoot FIS support. Added Useful RedBoot Operations sections.	27-Apr-2003	Chris Garry
0.9	Optimised logo images. Updated layout diagram for v2 PCB. Added 'Automating Operations at boot time' section.	30-Apr-2003	Chris Garry
1.0	Corrected use of 'arm-elf-objcopy'. Added details of testing the E7T-DBoard installation.	02-May-2003	Chris Garry
1.1	Added setting of MAC address to RedBoot initialisation sections.	27-May-2003	Chris Garry
1.2	Corrected use of 'fis create' for storing a compressed eCos image in FLASH section.	03-Jun-2003	Chris Garry
1.3	Removed '-m tftp' option from 'load' examples, since it is assumed by default. Added new FLASH memory map to support both types of FLASH.	25-Jul-2003	Chris Garry
1.4	Added 'Using default protocol (TFTP)' line to all tftp load examples.	28-Jul-2003	Chris Garry
1.5	Changed initial use of 'fconfig' to 'fconfig -i'.	19-Aug-2003	Chris Garry
1.6	Updated output for RedBoot examples.	07-Oct-2003	Chris Garry
1.7	Updated RedBoot and eCos CDL options.	10-Oct-2003	Chris Garry
1.8	Document moved from Word to OpenOffice.org	23-Oct-2004	Chris Garry

Contents

1	Introduction.....	5
1.1	About the E7T-DBoard.....	5
1.2	E7T-DBoard Kit Contents.....	5
1.3	E7T-DBoard Architecture.....	6
1.4	E7T-DBoard Layout.....	7
1.4.1	LED Functions.....	7
1.4.2	Power Connector.....	7
1.4.3	JTAG Connector.....	7
1.4.4	Ethernet Connector.....	8
1.4.5	Serial Connector.....	8
1.4.6	Expansion Connectors.....	8
1.4.7	EEPROM Pads.....	8
2	E7T-DBoard Installation.....	9
2.1	Fitting the E7T-DBoard.....	9
2.2	Initialisation and Testing of the E7T-DBoard.....	9
3	Hardware Description.....	10
3.1	FPGA Connections.....	10
3.1.1	Microcontroller Connections.....	10
3.1.2	Other System Connections.....	12
3.1.3	Expansion Bus Connections.....	12
3.1.4	Serial Interface Connections.....	15
3.2	Hardware Options.....	16
3.2.1	Disabling The Serial Interface.....	16
3.2.2	Connecting The Ethernet PHY To The FPGA.....	17
3.2.3	Connecting The SDRAM To The FPGA.....	18
3.2.4	Modifying The Frequency Of The FPGACLK Signal.....	19
3.2.5	Disabling The FPGACLK Signal.....	19
3.2.6	Fitting A Xilinx EEPROM For FPGA Configuration.....	20
4	eCos and RedBoot Support.....	21
4.1	Modifications to eCos for E7T-DBoard.....	21
4.2	Modifications to RedBoot for E7T-DBoard.....	21
4.3	RAM Selection For eCos Applications.....	21
4.3.1	RAM Selection Configuration.....	22
4.4	Accessing the E7T-DBoard's Ethernet PHY.....	22
4.5	Modified System Memory Map.....	23
4.6	FLASH Memory Usage.....	24
4.6.1	FLASH memory map for Evaluator-7Ts with SST39VF400 FLASH.....	24
4.6.2	FLASH memory map for Evaluator-7Ts with AMD Am29LV400 FLASH.....	25
4.6.3	Instructions for writing a RedBoot image to FLASH.....	26
4.7	New RedBoot Command: fpgaload.....	26
4.7.1	'fpgaload' Command Configuration.....	26
4.7.2	'fpgaload' Command Operation.....	27
4.7.2.1	Command Line Operation.....	27
4.8	Useful RedBoot Operations.....	28
4.8.1	Required initialisation after RedBoot installation.....	28
4.8.1.1	Initialisation for RedBoot with FIS support included.....	28
4.8.1.2	Initialisation for RedBoot with FIS and Ethernet support included.....	29
4.8.2	Manipulating FPGA images.....	32
4.8.2.1	Loading an FPGA image over a serial connection.....	32
4.8.2.2	Loading an FPGA image over an ethernet connection.....	32

4.8.2.3 Storing an FPGA Image in FLASH.....	33
4.8.3 Manipulating eCos application images.....	35
4.8.3.1 Loading an eCos application image over a serial connection.....	36
4.8.3.2 Loading an eCos application image over an ethernet connection.....	36
4.8.3.3 Storing an eCos application in FLASH.....	36
4.8.3.4 Storing a compressed eCos application image in FLASH.....	37
4.8.4 Automating Operations at boot time.....	39

1 Introduction

1.1 About the E7T-DBoard

The E7T-DBoard is a daughter board for ARM's Evaluator-7T board. Together the Evaluator-7T and the E7T-DBoard form a powerful development platform for FPGA/ASIC cores and embedded software.

The features of the E7T-DBoard are listed below:

- ◆ Intel LXT972A Ethernet PHY + RJ45 Connector
- ◆ 32MB SDRAM
- ◆ Xilinx Spartan2 FPGA + JTAG Connector
- ◆ RS232 Transceiver + Serial Socket Connector
- ◆ Expansion Bus

1.2 E7T-DBoard Kit Contents

The list below describes the items that are included with each E7T-DBoard kit.

- ◆ 1 × E7T-DBoard
- ◆ 4 × AMP 104655-6 connectors or equivalent (for fitting to the Evaluator-7T board)
- ◆ 1 × Null modem serial cable
- ◆ 1 × E7T-DBoard support CD – Containing:
 - ◆ E7T-DBoard Documentation:
 - Schematic diagrams (PDF format)
 - Reference manual (PDF format – this document)
 - ◆ eCos Support:
 - eCos Repository CVS snapshot (with all required E7T-DBoard updates)
 - .ecm files for various RedBoot and eCos configurations
 - Various pre-built RedBoot images
 - ◆ Example eCos Applications (source and pre-built versions):
 - ◆ StdUart (16x550 compatible UART) FPGA Core:
 - StdUart pre-built FPGA image for E7T-DBoard
 - StdUart FPGA image datasheet (PDF format)
 - StdUart eCos device driver
 - StdUart eCos device driver datasheet (PDF format)
 - ◆ VHDL Files:
 - Blank VHDL entity + architecture file for E7T-DBoard
 - ucf constraints file with fixed pin locations for E7T-DBoard

1.3 E7T-DBoard Architecture

The figure below (Figure 1-1.1) shows the architecture of the E7T-DBoard, including the Evaluator-7T board.

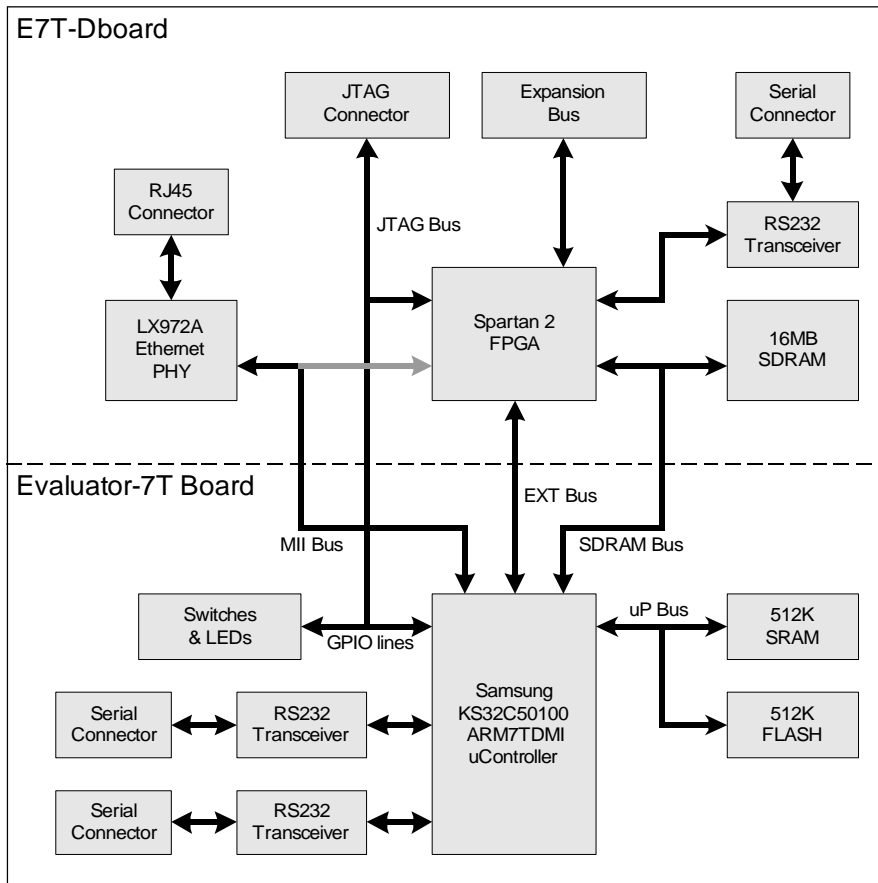


Figure 1-1.1 - E7T-DBoard Architecture

1.4 E7T-DBoard Layout

Figure 1-1.2 below shows the layout of the E7T-DBoard.

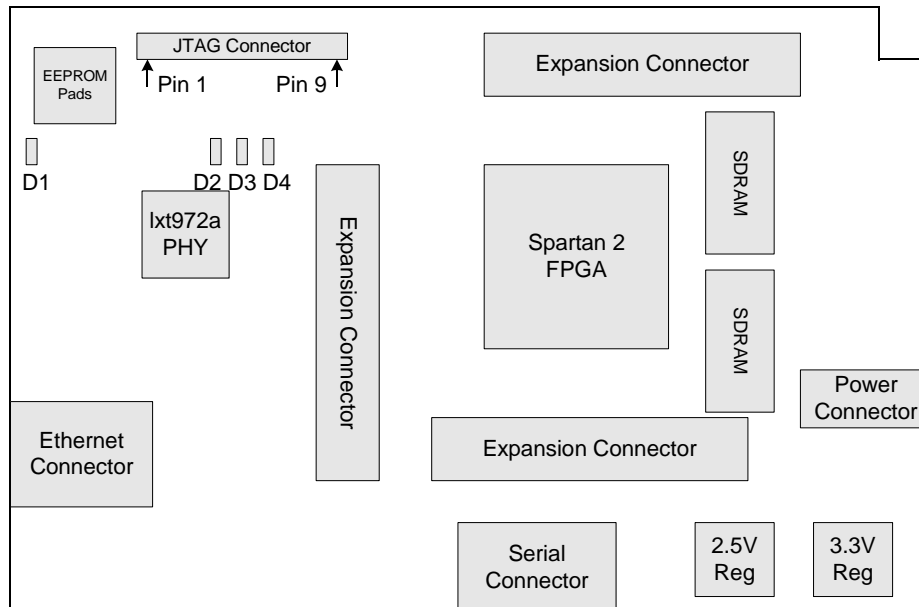


Figure 1-1.2 - E7T-DBoard Layout

1.4.1 LED Functions

The E7T-DBoard has 4 LEDs (D1 – D4), their functions are described here.

- D1: FPGA Done indicator – Lit when FPGA has been successfully configured.
- D2: LED 1 for LXT972A PHY – Configured by software.
- D3: LED 2 for LXT972A PHY – Configured by software.
- D4: LED 3 for LXT972A PHY – Configured by software.

1.4.2 Power Connector

The Power connector is where the Evaluator-7T power supply is connected to when the E7T-DBoard has been installed.

1.4.3 JTAG Connector

The purpose of the JTAG connector is to download the FPGA image directly to the FPGA using a Xilinx download cable. Although the image is not stored in FLASH and will be lost when the board power is removed, this is often the quickest download method when multiple FPGA images need to be tested.

The pinout of the JTAG connector is as follows:

Pin 1	VCC
Pin 2	GND
Pin 3	No Connect
Pin 4	TCK
Pin 5	No Connect
Pin 6	TDO
Pin 7	TDI
Pin 8	No Connect
Pin 9	TMS

1.4.4 Ethernet Connector

The ethernet connector is connected to the ethernet PHY on the E7T-DBoard, which is by default, controlled by the ethernet controller in the microcontroller. When a version of RedBoot with network support is being executed on the Evaluator-7T board, the ethernet port can be used to talk to RedBoot, download images and debug software. Alternatively, the ethernet port may be used by the software application.

The ethernet PHY (and hence connector) may, with some simple hardware changes, be controlled by the FPGA, allowing an ethernet controller to be incorporated in the FPGA.

1.4.5 Serial Connector

The serial connector is connected to the FPGA through an RS232 transceiver, and is only of any use when a UART device has been implemented with the FPGA. This connector is only included to allow a UART to be incorporated in the FPGA.

1.4.6 Expansion Connectors

The three 34 pin expansion connectors (PL2, PL3 and PL4) contain power and FPGA signals. These connectors are included to allow additional functionality to be added. For example, if a USB controller were to be developed, a small daughter board containing a USB transceiver and connector could be constructed and plugged into one or more of these connectors.

1.4.7 EEPROM Pads

The EEPROM pads allow a Xilinx EEPROM (XC18V02) to be fitted to hold FPGA configuration data. When fitted, the FPGA configuration data is downloaded to the EEPROM via the JTAG connector and the FPGA is automatically configured each time the board is powered up. See section 3.2.6 for details of fitting a Xilinx EEPROM.

2 E7T-DBoard Installation

2.1 Fitting the E7T-DBoard

Warning: Electrostatic precautions must be taken before handling either the Evaluator-7T board or the E7T-DBoard.

(1) De-solder and remove the power connector J7 from the Evaluator-7T board.

This is required because the power supply connects to the daughter board instead of the Evaluator-7T, and this power connector shorts Vcc to Gnd when the power supply is not plugged in.

(2) On the Evaluator-7T, remove the solder from the pads and guide holes for connectors J2, J3, J4 and J5.

This is required before the 4 connectors can be fitted. Each connector has 2 guide holes.

(3) Solder the 4 connectors supplied with the E7T-DBoard onto the pads for J2, J3, J4 and J5 on the Evaluator-7T board.

Care needs to be taken to solder the connector on square to the pads or they will not mate correctly when the E7T-DBoard is plugged in. Also note, that the connectors are keyed and the keyed part of the connector should be next to pin 1 as marked on the Evaluator-7T board or the E7T-DBoard will not mate correctly.

(4) Once the 4 connectors have been soldered onto the Evaluator-7T board and carefully checked to ensure there are no short-circuits, the E7T-DBoard should be plugged onto the Evaluator-7T board.

(5) The E7T-DBoard is now fitted. When the power supply is plugged into the power connector on the E7T-DBoard the Evaluator-7T board should operate as before.

2.2 Initialisation and Testing of the E7T-DBoard

Now that the E7T-DBoard has been fitted, further steps are required to initialise and check that the board has been fitted correctly.

(1) Download one of the RedBoot images from the E7T-DBoard support CD to the FLASH by following the instructions in section 4.6.3.

(2) Initialise RedBoot by following the instructions in section 4.8.1.

(3) Run the SDRAM test program from the support CD to check the SDRAM is operating correctly (See section 4.8.3 for details of running applications with RedBoot).

(4) Connect the board to a network and try pinging the board from a PC. Try pinging a PC from RedBoot (use the RedBoot command of the form 'ping -v -h 192.168.11.80')

(5) Load the FPGA with the StdUart FPGA image from the support CD (see section 4.8.2). The FPGA Done LED (D1) should light when the FPGA is configured.

3 Hardware Description

3.1 FPGA Connections

This section details the connections to the FPGA, and is intended as a reference for FPGA designers working with the FPGA.

3.1.1 Microcontroller Connections

The table below details all the FPGA connections to the microcontroller on the Evaluator-7T board. For full details of microcontroller signals, refer to the KS32C50100 datasheet.

Signal Name	FPGA Pin #	Pin Type	Description
Addr0	108	Bi-dir	Address bus bit 0 (lsb)
Addr1	109	Bi-dir	Address bus bit 1
Addr2	110	Bi-dir	Address bus bit 2
Addr3	111	Bi-dir	Address bus bit 3
Addr4	112	Bi-dir	Address bus bit 4
Addr5	113	Bi-dir	Address bus bit 5
Addr6	114	Bi-dir	Address bus bit 6
Addr7	115	Bi-dir	Address bus bit 7
Addr8	119	Bi-dir	Address bus bit 8
Addr9	120	Bi-dir	Address bus bit 9
Addr10	121	Bi-dir	Address bus bit 10
Addr11	122	Bi-dir	Address bus bit 11
Addr12	123	Bi-dir	Address bus bit 12
Addr13	125	Bi-dir	Address bus bit 13
Addr14	126	Bi-dir	Address bus bit 14
Addr15	127	Bi-dir	Address bus bit 15
Addr16	129	Bi-dir	Address bus bit 16
Addr17	132	Bi-dir	Address bus bit 17
Addr18	133	Bi-dir	Address bus bit 18
Addr19	134	Bi-dir	Address bus bit 19
Addr20	135	Bi-dir	Address bus bit 20
Addr21	136	Bi-dir	Address bus bit 21 (msb)
Data0	138	Bi-dir	Data bus bit 0 (lsb)
Data1	139	Bi-dir	Data bus bit 1
Data2	140	Bi-dir	Data bus bit 2
Data3	141	Bi-dir	Data bus bit 3
Data4	142	Bi-dir	Data bus bit 4
Data5	146	Bi-dir	Data bus bit 5
Data6	147	Bi-dir	Data bus bit 6
Data7	148	Bi-dir	Data bus bit 7
Data8	149	Bi-dir	Data bus bit 8
Data9	150	Bi-dir	Data bus bit 9
Data10	151	Bi-dir	Data bus bit 10
Data11	152	Bi-dir	Data bus bit 11
Data12	160	Bi-dir	Data bus bit 12
Data13	161	Bi-dir	Data bus bit 13

Data14	162	Bi-dir	Data bus bit 14
Data15	163	Bi-dir	Data bus bit 15
Data16	202	Bi-dir	Data bus bit 16
Data17	203	Bi-dir	Data bus bit 17
Data18	204	Bi-dir	Data bus bit 18
Data19	205	Bi-dir	Data bus bit 19
Data20	206	Bi-dir	Data bus bit 20
Data21	3	Bi-dir	Data bus bit 21
Data22	4	Bi-dir	Data bus bit 22
Data23	5	Bi-dir	Data bus bit 23
Data24	6	Bi-dir	Data bus bit 24
Data25	7	Bi-dir	Data bus bit 25
Data26	8	Bi-dir	Data bus bit 26
Data27	9	Bi-dir	Data bus bit 27
Data28	10	Bi-dir	Data bus bit 28
Data29	14	Bi-dir	Data bus bit 29
Data30	15	Bi-dir	Data bus bit 30
Data31	16	Bi-dir	Data bus bit 31 (msb)
nRCS3	48	Input	Not ROM/SRAM/FLASH Chip Select 3
nRCS4	49	Input	Not ROM/SRAM/FLASH Chip Select 4
nRCS5	57	Input	Not ROM/SRAM/FLASH Chip Select 5
nECS0	37	Input	Not External I/O Chip Select 0
nECS1	41	Input	Not External I/O Chip Select 1
nECS2	42	Input	Not External I/O Chip Select 2
nECS3	43	Input	Not External I/O Chip Select 3
nWBE0	58	Bi-dir	Not Write Byte Enable 0
nWBE1	59	Bi-dir	Not Write Byte Enable 1
nWBE2	60	Bi-dir	Not Write Byte Enable 2
nWBE3	100	Bi-dir	Not Write Byte Enable 3
nOE	45	Input	Not Output Enable
nEWAIT	44	Output	Not External Wait Signal
SDCLKIN	80	Input	System Clock/SDRAM Clock In (GCK0 Input)
SDCLKOUT	81	Output	System Clock/SDRAM Clock Out
CLKOE	46	Output	System Clock/SDRAM Clock Enable
EXTMACK	102	Input	External Bus Master Acknowledge
EXTMREQ	101	Output	External Bus Master Request
PIO8/XINTREQ0	17	Bi-dir	Programmable I/O Port 8/Ext Interrupt Request 0
PIO9/ XINTREQ1	18	Bi-dir	Programmable I/O Port 9/Ext Interrupt Request 1
PIO10/XINTREQ2	20	Bi-dir	Programmable I/O Port 10/Ext Interrupt Request 2
PIO11/XINTREQ3	21	Bi-dir	Programmable I/O Port 11/Ext Interrupt Request 3
PIO12/nXDREQ0	22	Bi-dir	Programmable I/O Port 12/Not Ext DMA Request 0
PIO13/nXDREQ1	23	Bi-dir	Programmable I/O Port 13/Not Ext DMA Request 0
PIO14/nXDACK0	24	Bi-dir	Programmable I/O Port 14/Not Ext DMA Ack 0
PIO15/nXDACK1	27	Bi-dir	Programmable I/O Port 15/Not Ext DMA Ack 1

3.1.2 Other System Connections

Signal Name	FPGA Pin #	Pin Type	Description
nRESET	47	Input	Not Reset Signal (From Evaluator-7T Board)
FPGACLK (GCK1)	77	Input	40MHz (modifiable) Generic Clock Input

The FPGACLK signal is a generic clock generated on the E7T-DBoard, for general use by FPGA cores. This clock is generated with a 20 MHz crystal and an ICS501 clock generator IC. By default this clock is 40 MHz, but this may be changed.

See section 3.2.4 for details of how to modify the frequency of FPGACLK and section 3.2.5 for details of how to disable FPGACLK to allow a clock to be fed into this input from the expansion connector, PL2.

3.1.3 Expansion Bus Connections

The Expansion Bus consists of 3 × 0.1” 34 way connectors (PL2, PL3 and PL4) containing FPGA Input/Output signals and power. The three tables below list the connections of each of the expansion bus connectors. Where the signal name has another name in parenthesis after it, this indicates that pin has an alternative function that can be used with some hardware modifications. The entries in this table are for an E7T-DBoard that has not been modified.

Signal Name	FPGA Pin #	Pin Type	Expansion Con. Pin #	Description
IO0 (nSDCS)	61	Bi-dir	PL2: 1	FPGA IO 0 [1]
IO1 (nSDRAS)	62	Bi-dir	PL2: 2	FPGA IO 1 [1]
IO2 (nSDCAS)	63	Bi-dir	PL2: 3	FPGA IO 2 [1]
IO3 (SDCKE)	67	Bi-dir	PL2: 4	FPGA IO 3 [1]
IO4 (nSDWE)	68	Bi-dir	PL2: 5	FPGA IO 4 [1]
TXDATA (IO5)	69	Output	PL2: 6	Serial Transmit Data [2]
Gnd	-	Power	PL2: 7	Ground
3V3	-	Power	PL2: 8	3.3 V Supply
nRTS (IO6)	70	Output	PL2: 9	Not Serial Request To Send [2]
nCTS (IO7)	71	Input	PL2: 10	Not Serial Clear To Send [2]
RXDATA (IO8)	73	Input	PL2: 11	Serial Receive Data [2]
nDTR (IO9)	74	Output	PL2: 12	Not Serial Data Terminal Ready [2]
nRI (IO10)	75	Input	PL2: 13	Not Serial Ring Indicator [2]
nDSR (IO11)	82	Input	PL2: 14	Not Serial Data Set Ready [2]
Gnd	-	Power	PL2: 15	Ground
3V3	-	Power	PL2: 16	3.3 V Supply
FPGACLK (GCK1)	77	Input	PL2: 17	Generic Clock Input [3]
nDCD (IO12)	83	Input	PL2: 18	Not Serial Data Carrier Detect [2]
IO13	84	Bi-dir	PL2: 19	FPGA IO 13
IO14	86	Bi-dir	PL2: 20	FPGA IO 14
IO15	87	Bi-dir	PL2: 21	FPGA IO 15
IO16	88	Bi-dir	PL2: 22	FPGA IO 16
Gnd	-	Power	PL2: 23	Ground
3V3	-	Power	PL2: 24	3.3 V Supply
IO17	89	Bi-dir	PL2: 25	FPGA IO 17
IO18	90	Bi-dir	PL2: 26	FPGA IO 18
IO19	94	Bi-dir	PL2: 27	FPGA IO 19

IO20	95	Bi-dir	PL2: 28	FPGA IO 20
IO21	96	Bi-dir	PL2: 29	FPGA IO 21
IO22	97	Bi-dir	PL2: 30	FPGA IO 22
Gnd	-	Power	PL2: 31	Ground
3V3	-	Power	PL2: 32	3.3 V Supply
IO23	98	Bi-dir	PL2: 33	FPGA IO 23
IO24	99	Bi-dir	PL2: 34	FPGA IO 24

[1] See section 3.2.3 Connecting The SDRAM To The FPGA

[2] See section 3.2.1 Disabling The Serial Interface

[3] See section 3.2.5 Disabling The FPGACLK Signal

Signal Name	FPGA Pin #	Pin Type	Expansion Con. Pin #	Description
IO25	164	Bi-dir	PL3: 1	FPGA IO 25
IO26	165	Bi-dir	PL3: 2	FPGA IO 26
IO27	166	Bi-dir	PL3: 3	FPGA IO 27
IO28	167	Bi-dir	PL3: 4	FPGA IO 28
IO29	168	Bi-dir	PL3: 5	FPGA IO 29
IO30	172	Bi-dir	PL3: 6	FPGA IO 30
Gnd	-	Power	PL3: 7	Ground
3V3	-	Power	PL3: 8	3.3 V Supply
IO31	173	Bi-dir	PL3: 9	FPGA IO 31
IO32	174	Bi-dir	PL3: 10	FPGA IO 32
IO33	175	Bi-dir	PL3: 11	FPGA IO 33
IO34	176	Bi-dir	PL3: 12	FPGA IO 34
IO35	178	Bi-dir	PL3: 13	FPGA IO 35
IO36	179	Bi-dir	PL3: 14	FPGA IO 36
Gnd	-	Power	PL3: 15	Ground
3V3	-	Power	PL3: 16	3.3 V Supply
IO37	180	Bi-dir	PL3: 17	FPGA IO 37
IO38	181	Bi-dir	PL3: 18	FPGA IO 38
GCK2	182	Bi-dir	PL3: 19	FPGA GCK2 Input (or generic IO)
IO39	187	Bi-dir	PL3: 20	FPGA IO 39
IO40 (CRS)	188	Bi-dir	PL3: 21	FPGA IO 41 [4]
IO41 (RXDV)	189	Bi-dir	PL3: 22	FPGA IO 42 [4]
Gnd	-	Power	PL3: 23	Ground
3V3	-	Power	PL3: 24	3.3 V Supply
IO42 (RXD0)	191	Bi-dir	PL3: 25	FPGA IO 42 [4]
IO43 (RXD1)	192	Bi-dir	PL3: 26	FPGA IO 43 [4]
IO44 (RXD2)	193	Bi-dir	PL3: 27	FPGA IO 44 [4]
IO45 (RXD3)	194	Bi-dir	PL3: 28	FPGA IO 45 [4]
IO46 (RXER)	195	Bi-dir	PL3: 29	FPGA IO 46 [4]
IO47 (RXCLK)	199	Bi-dir	PL3: 30	FPGA IO 47 [4]
Gnd	-	Power	PL3: 31	Ground
3V3	-	Power	PL3: 32	3.3 V Supply
IO48 (COL)	200	Bi-dir	PL3: 33	FPGA IO 48 [4]
IO49 (TXD0)	201	Bi-dir	PL3: 34	FPGA IO 49 [4]

[4] See section 3.2.2 Connecting The Ethernet PHY To The FPGA

Signal Name	FPGA Pin #	Pin Type	Expansion Con. Pin #	Description
GCK3	185	Bi-dir	PL4: 1	FPGA GCK2 Input (or generic IO)
IO50 (TXD1)	154	Bi-dir	PL4: 2	FPGA IO 50 [4]
IO51 (TXD2)	29	Bi-dir	PL4: 3	FPGA IO 51 [4]
IO52 (TXD3)	30	Bi-dir	PL4: 4	FPGA IO 52 [4]
IO53 (TXEN)	31	Bi-dir	PL4: 5	FPGA IO 53 [4]
IO54 (TXCLK)	33	Bi-dir	PL4: 6	FPGA IO 54 [4]
Gnd	-	Power	PL4: 7	Ground
3V3	-	Power	PL4: 8	3.3 V Supply
IO55 (TXER)	34	Bi-dir	PL4: 9	FPGA IO 55 [4]
IO56 (MDIO)	35	Bi-dir	PL4: 10	FPGA IO 56 [4]
IO57 (MDC)	36	Bi-dir	PL4: 11	FPGA IO 57 [4]
IO58 / nINIT	107	Bi-dir	PL4: 12	FPGA IO 58 [5]
IO59 / DIN	153	Bi-dir	PL4: 13	FPGA IO 59 [5]
No Connect	-		PL4: 14	
Gnd	-	Power	PL4: 15	Ground
3V3	-	Power	PL4: 16	3.3 V Supply
SCL	-	Bi-dir	PL4: 17	Microcontroller I ² C Serial Clock [6]
SDA	-	Bi-dir	PL4: 18	Microcontroller I ² C Serial Data [6]
No Connect	-		PL4: 19	
No Connect	-		PL4: 20	
No Connect	-		PL4: 21	
No Connect	-		PL4: 22	
Gnd	-	Power	PL4: 23	Ground
3V3	-	Power	PL4: 24	3.3 V Supply
2V5	-	Power	PL4: 25	2.5 V Supply
No Connect	-		PL4: 26	
2V5	-	Power	PL4: 27	2.5 V Supply
No Connect	-		PL4: 28	
2V5	-	Power	PL4: 29	2.5 V Supply
No Connect	-		PL4: 30	
Gnd	-	Power	PL4: 31	Ground
3V3	-	Power	PL4: 32	3.3 V Supply
RAWDC	-	Power	PL4: 33	Raw DC from power supply
RAWDC	-	Power	PL4: 34	Raw DC from power supply

[4] See section 3.2.2 Connecting The Ethernet PHY To The FPGA

[5] This IO signal should not be used when a Xilinx EEPROM has been fitted.

[6] See the KS32C50100 datasheet for signal details.

3.1.4 Serial Interface Connections

Eight of the FPGA pins are connected, via an RS232 transceiver, to a 9-pin serial connector to form a serial interface. This serial interface is configured as a DTE device (it is wired the same as the serial connector on a PC). This means that a null modem cable is required to connect this connector to a PC.

Signal Name	FPGA Pin #	Pin Type	Description	Serial Connector Pin #
TXDATA (IO5)	69	Output	Transmit Data	3
nRTS (IO6)	70	Output	Not Request To Send	7
nCTS (IO7)	71	Input	Not Clear To Send	8
RXDATA (IO8)	73	Input	Receive Data	2
nDTR (IO9)	74	Output	Not Data Terminal Ready	4
nRI (IO10)	75	Input	Not Ring Indicator	9
nDSR (IO11)	82	Input	Not Data Set Ready	6
nDCD (IO12)	83	Input	Not Data Carrier Detect	1

See section 3.2.1 for details of how to disable the serial interface and make the 8 signals, IO5 to IO12, available as generic IO signals in the expansion connector.

3.2 Hardware Options

The E7T-DBoard was designed to create as flexible a development board as possible. To this end, the functionality of the E7T-DBoard may be modified by removing certain components and/or adding components to reserved component pads. The possible hardware modifications are now listed:

- ◆ Disable The Serial Interface
- ◆ Connect The Ethernet PHY To The FPGA
- ◆ Connect The SDRAM To The FPGA
- ◆ Modify The Frequency Of The FPGACLK Signal
- ◆ Disable The FPGACLK Signal
- ◆ Fit A Xilinx EEPROM For FPGA Configuration

The following sub-sections detail each possible modification.

3.2.1 Disabling The Serial Interface

The Serial Interface FPGA pins are connected to both the RS232 transceiver and the expansion connector PL2 by default. If these connections (IO5 to IO12) are required to be used as generic IO signals on the expansion connector, and not for the serial interface, then some hardware modifications are required to disconnect them from the RS232 transceiver.

Modifications required to disable the serial interface:

1. Remove resistor R29. (This disables the RS232 transceiver, IC3)
2. Remove resistor networks RN8 and RN9. (This disconnects IO5 to IO12 from the RS232 transceiver).

Component details for this modification:

Ref	Part	Package	Description	Manufacturer & Code
R29	10K resistor	0603	Ni barrier chip resistor	Any
RN8, RN9	0R by 4 resistor pack	SMT	4 individual resistors	Phillips ARC241 series

The serial connector connections, IO5 to IO12, may now be used as generic IOs.

3.2.2 Connecting The Ethernet PHY To The FPGA

By default, the ethernet PHY (IC9) has the 18 connections that form its MII (Media Independent Interface) bus connected to the ethernet controller in the microcontroller on the Evaluator-7T board. This means that the ethernet controller on the Evaluator-7T board controls the ethernet port on the E7T-DBoard and the FPGA connections IO40 to IO57 (connected to expansion connectors PL3 and PL4) are free for use as generic IOs.

However, it is also possible for the FPGA to be connected to the ethernet PHY (using FPGA connections IO40 to IO57), allowing an ethernet controller to be developed in the FPGA, but the ethernet controller in the microcontroller will be disabled in this configuration.

Modifications required to connect the Ethernet PHY to the FPGA:

1. Remove resistor networks RN1 and RN2. (This disconnects the MDIO, MDC, TXEN, TXER and TXD[3:0] signals from the microcontroller on the Evaluator-7T board.
2. Fit resistor networks RN6, RN10, RN11, RN12 and RN13. (This connects the 18 MII connections on the ethernet PHY to the FPGA.

Component details for this modification:

Ref	Part	Pack	Description	Manufacturer & Code
RN1, RN2, RN6, RN10, RN11, RN12, RN13	0R by 4 resistor pack	SMT	4 individual resistors	Phillips ARC241 series

Once the hardware modifications have made to connect the FPGA to the ethernet PHY, the 18 FPGA connections used to form the MII bus are as listed in the table below. For full details of each of these signals refer to the LXT972A datasheet.

Signal Name	FPGA Pin #	Pin Type	Description
CRS (IO40)	188	Input	Carrier Sense
RXDV (IO41)	189	Input	Receive Data Valid
RXD0 (IO42)	191	Input	Receive Data Bit 0 (lsb)
RXD1 (IO43)	192	Input	Receive Data Bit 1
RXD2 (IO44)	193	Input	Receive Data Bit 2
RXD3 (IO45)	194	Input	Receive Data Bit 3 (msb)
RXER (IO46)	195	Input	Receive Error
RXCLK (IO47)	199	Input	Receive Clock
COL (IO48)	200	Input	Collision Detected
TXD0 (IO49)	201	Output	Transmit Data Bit 0 (lsb)
TXD1 (IO50)	154	Output	Transmit Data Bit 1
TXD2 (IO51)	29	Output	Transmit Data Bit 2
TXD3 (IO52)	30	Output	Transmit Data Bit 3 (msb)
TXEN (IO53)	31	Output	Transmit Enable
TXCLK (IO54)	33	Input	Transmit Clock
TXER (IO55)	34	Output	Transmit Error
MDIO (IO56)	35	Bi-dir	Management Data Input/Output
MDC (IO57)	36	Output	Management Data Clock

The connections IO40 to IO57 may no longer be used on the expansion connectors PL3 and PL4 as generic IO.

3.2.3 Connecting The SDRAM To The FPGA

The E7T-DBoard is fitted with 32MB of SDRAM (2 × 128Mbit SDRAM ICs), the modifications described here allow the FPGA to fully control this SDRAM.

The FPGA has many connections to the SDRAM already listed in the microcontroller Connections list (section 3.1.1). These are Addr[13:0], Data[31:0], nWBE[3:0] and SDCLKOUT. These connections are always connected and must be controlled accordingly.

There are another five SDRAM connections, which are not connected to the SDRAM by default.

Modifications required to connect the SDRAM to the FPGA:

1. Fit resistor network RN7. (This connects the FPGA to SDRAM signals nSDCS, nSDRAS, nSDCAS and SDCKE)
2. Fit resistor R47. (This connects the FPGA to SDRAM signal nSDWE)

Component details for this modification:

Ref	Part	Pack	Description	Manufacturer & Code
R47	0R resistor	0603	Ni barrier chip resistor	Any
RN7	0R by 4 resistor pack	SMT	4 individual resistors	Phillips ARC241 series

Once the hardware modifications have been made to connect these five signals to the SDRAM, the FPGA is able to fully control the SDRAM. The five signals are listed in the table below.

Signal Name	FPGA Pin #	Pin Type	Description
nSDCS (IO0)	61	Output	Not SDRAM Chip Select
nSDRAS (IO1)	62	Output	Not SDRAM Row Address Strobe
nSDCAS (IO2)	63	Output	Not SDRAM Column Address Strobe
SDCKE (IO3)	67	Output	SDRAM Clock Enable
nSDWE (IO4)	68	Output	Not SDRAM Write Enable

These signals may only be driven when the microcontroller is not driving them; this is achieved by the FPGA taking control of the bus using the EXTMREQ and EXTMACK signals.

The connections IO0 to IO4 may no longer be used on the expansion connector PL2 as generic IO.

3.2.4 Modifying The Frequency Of The FPGACLK Signal

The frequency of the generic FPGA clock, FPGACLK, is dependent on which of the 0R resistors R43, R44, R45 and R46 are fitted. By default, **none** of these resistors are fitted.

R43	R44	R45	R46	FPGACLK Frequency
Not Fitted	Not Fitted	Fitted	Fitted	80 MHz (4 × XTAL)
Not Fitted	Not Fitted	Fitted	Not Fitted	106.25 MHz (5.3125 × XTAL)
Not Fitted	Fitted	Fitted	Not Fitted	100 MHz (5 × XTAL)
Not Fitted	Not Fitted	Not Fitted	Fitted	125 MHz (6.25 × XTAL)
Not Fitted	Not Fitted	Not Fitted	Not Fitted	40 MHz (2 × XTAL) - Default
Not Fitted	Fitted	Not Fitted	Not Fitted	62.5 MHz (3.125 × XTAL)
Fitted	Not Fitted	Not Fitted	Fitted	120 MHz (6 × XTAL)
Fitted	Not Fitted	Not Fitted	Not Fitted	60 MHz (3 × XTAL)
Fitted	Fitted	Not Fitted	Not Fitted	160 MHz (8 × XTAL)

Note, if a clock frequency not listed in the above table is required, the crystal X2 may be replaced with a different value. Refer to the ICS501 datasheet for details of crystal values that may be used. Alternatively, another clock may be supplied at the GCK2 or GCK3 inputs via the expansion connectors PL3 or PL4.

Component details for this modification:

Ref	Part	Pack	Description	Manufacturer & Code
R43, R44, R45, R46	0R resistor	0603	Ni barrier chip resistor	Any

3.2.5 Disabling The FPGACLK Signal

The circuitry generating the FPGACLK signal may be disconnected from the FPGA and expansion connector PL2. This allows the GCK1 input to the FPGA to be driven by the expansion connector. Alternatively, this connection can then be used as another generic Input/Output.

Modifications required to disable the FPGACLK signal:

1. Remove resistor R40

Component details for this modification:

Ref	Part	Pack	Description	Manufacturer & Code
R40	33R resistor	0603	Ni barrier chip resistor	Any

Once this modification has been made, the FPGACLK signal will not be connected to the FPGA or the expansion connector.

3.2.6 Fitting A Xilinx EEPROM For FPGA Configuration

The E7T-DBoard can be fitted with a Xilinx EEPROM to hold an FPGA configuration file. Once the EEPROM has been fitted, an FPGA configuration file can be written to it using the JTAG port, and then the configuration file will be used to configure the FPGA each time the board is powered-up.

Note: It is not possible to fit a Xilinx EEPROM on Rev 1.0 versions of the E7T-DBoard.

Fitting a Xilinx EEPROM means that the Evaluator-7T FLASH does not need to store an FPGA configuration file and the RedBoot firmware does not need to have the 'fpgaload' command included; this all frees up space in the FLASH.

Modifications required to fit a Xilinx EEPROM:

1. Fit the Xilinx EEPROM IC2.
2. Remove resistor R39. (This removes the connection between the JTAG TDI and TDO signals on the Xilinx EEPROM)
3. Fit resistor R31. (This sets the FPGA's Mode[0:2] pins to '000')

Component details for this modification:

Ref	Part	Pack	Description	Manufacturer & Code
R31, R39	0R resistor	0603	Ni barrier chip resistor	Any
IC2	XC18V02	VQ44	Serial EEPROM	Xilinx

Once the modifications have been made, the E7T-DBoard JTAG chain will be as in Figure 3-3.1 below.

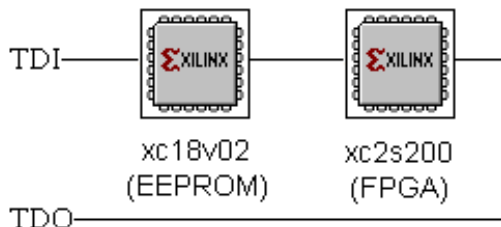


Figure 3-3.1 - JTAG Chain With The Xilinx EEPROM Fitted

Once this modification has been made, the 'fpgaload' command in RedBoot must not be used.

4 eCos and RedBoot Support

Software support for the Evaluator-7T with the E7T-DBoard is supplied for eCos (the embedded configurable operating system) and RedBoot (Red Hat Embedded Debug and Bootstrap firmware). Both of these are open source, royalty-free and were originally developed by Red Hat. This document assumes a working knowledge of both eCos and RedBoot, and no detailed description is included. For details of eCos and RedBoot, including links to the relevant web sites, see: <http://www.sweeneydesign.co.uk/swtools.htm>.

Sweeney Design supplies support for eCos and RedBoot by supplying the updates required to convert the standard Evaluator-7T eCos port to work with the Evaluator-7T and E7T-DBoard combination. An updated version of the eCos code repository, v1.0 beta 1, is included with the E7T-DBoard support CD. Alternatively, downloads can be found at: <http://www.sweeneydesign.co.uk/downloads.htm>, that allow the latest eCos source repository from CVS to be updated to work with the Evaluator-7T and E7T-DBoard combination.

The E7T-DBoard support CD also contains various pre-built versions of the RedBoot bootstrap firmware ready to download to the FLASH memory of an Evaluator-7T board with an E7T-DBoard fitted.

4.1 Modifications to eCos for E7T-DBoard

These are the changes (optional via the CDL) that the updates make to eCos.

- ◆ eCos can use either the SDRAM on the E7T-DBoard or the Evaluator-7T's SRAM, selectable via a configuration option
- ◆ eCos allows the microcontroller to interface to the FPGA so that devices may be developed in the FPGA.

4.2 Modifications to RedBoot for E7T-DBoard

These are the changes (optional via the CDL) that the updates make to RedBoot.

- ◆ RedBoot enables the SDRAM on the E7T-DBoard for application use, although Redboot itself still runs from the Evaluator-7T's SRAM itself.
- ◆ RedBoot has a new command, 'fpgaload', added to allow it to configure the FPGA with an image stored in memory (RAM or FLASH).

4.3 RAM Selection For eCos Applications

eCos applications running from RAM (CDL option `CYG_HAL_STARTUP = RAM`), may run in either the Evaluator-7T's 512KB of SRAM, or in the E7T-DBoard's 32MB of SDRAM. Which is used by the application is selected at the eCos library build-time, using a new CDL configuration option.

When the application is running from FLASH (CDL option `CYG_HAL_STARTUP = ROM`), the Evaluator-7T's SRAM is always used for all the applications RAM needs. RedBoot is an example of an eCos application that runs from ROM.

Technical note: The fixed_vectors section of the eCos image is always located in the same place in the Evaluator7T's SRAM, no matter which RAM is selected for use. This is done so that RedBoot (including the GDB stubs) will work correctly with all eCos applications, no matter which RAM they are running from.

4.3.1 RAM Selection Configuration

The new RAM Selection configuration option may be set up with the graphical configuration tool (or by editing the .ecc configuration file for developers using the command line configuration tool). This options controlling E7T-Board support and RAM selection are:

cdl_component	CYG_HAL_SUPPORT_E7TDBOARD
display	Include support for E7T-DBoard
type	bool
default value	0
description	Selecting this option includes support for the E7T-DBoard.

cdl_option	CYG_HAL_RAM_SELECT
display	RAM selection
type	data
default value	SDRAM [SDRAM or SRAM]
description	This option specifies which RAM to use. Choices are the SDRAM (32MB) on the E7T-DBoard or the SRAM (512KB) on the Evaluator-7T board. The fixed_vectors section is placed in the SRAM regardless of this option. When support for E7T-DBoard is not enabled or ROM Startup type is selected this option has no effect.

4.4 Accessing the E7T-DBoard's Ethernet PHY

Full support for accessing the LXT972A ethernet PHY on the E7T-DBoard is available with the standard release of eCos and RedBoot. This is found in the *Samsung KS32C5000 ethernet driver* package that may be added to your eCos or RedBoot configuration.

Note: The MII address of the ethernet PHY on the E7T-DBoard is 0x00.

4.5 Modified System Memory Map

This is the system memory map as set-up by the modified eCos files.

0x03FFFFFF	64KB Microcontroller Register Space	Control Register SYSCFG
0x03FF0000		
0x03FE1FFF	8KB Microcontroller Internal SRAM (*)	SYSCFG
0x03FE0000		
0x03A03FFF	16KB Evaluator-7T FPGA Processor Interface	REFEXTCON
0x03A00000		
0x038FFFFFF	32MB E7T-DBoard 32-bit SDRAM	DRAMCON0
0x01900000		
0x0187FFFF	512KB Evaluator-7T 16-bit FLASH	ROMCON0
0x01800000		
0x0007FFFF	512KB Evaluator-7T 32-bit SRAM	ROMCON1 & ROMCON2
0x00000000		

(*) Not available when the microcontroller cache is enabled.

Adding 0x04000000 to any address will access the same location, but with the cache disabled.

This is particularly useful for microcontroller registers and the FPGA μ P interface.

4.6 FLASH Memory Usage

4.6.1 FLASH memory map for Evaluator-7Ts with SST39VF400 FLASH

This is the suggested FLASH memory map for Evaluator-7Ts where SST39VF400 FLASH is fitted. This is mostly the same as used by the standard Evaluator-7T port of eCos, except more space has been allocated for RedBoot (64KB is not invariably enough) and an area has been reserved for RedBoot configuration and the FLASH Image System (FIS) directory data.

Address		FLASH Sector
0x0187FFFF	4KB	127
0x0187F000	FIS Directory data	
0x0187EFFF	4KB	126
0x0187E000	RedBoot Configuration data	
0x0187DFFF		125
	248KB Free Space	
0x01840000		64
0x0183FFFF		63
	128KB RedBoot	
0x01820000		32
0x0181FFFF		31
	64KB Angel	
0x01810000		16
0x0180FFFF		15
	64KB Bootstrap Loader, Production Test + Reserved	
0x01800000		0

Notes:

1. The Bootstrap Loader, Production Test + Reserved area should not be overwritten.
2. The Angel program from ARM has been left to preserve the Evaluator-7T's compatibility with ARM's tools.
3. Although 128KB has been reserved for RedBoot, this may be reduced if a cut down version of RedBoot is used.
4. The 4KB of FIS directory data is automatically created when RedBoot's 'fis init -f' command is run (this command should be only run once).
5. The 4KB reserved for configuration data is used automatically when RedBoot updates configuration data in FLASH (after 'alias' or 'fconfig' commands are ran).
6. The 248KB of free space is free to hold FPGA images or application code using RedBoot's FLASH Image System (FIS).

4.6.2 FLASH memory map for Evaluator-7Ts with AMD Am29LV400 FLASH

This is the suggested FLASH memory map for Evaluator-7Ts where Am29LV400 FLASH is fitted. This is mostly the same as used by the standard Evaluator-7T port of eCos, except more space has been allocated for RedBoot (64KB is not invariably enough) and an area has been reserved for RedBoot configuration and the FLASH Image System (FIS) directory data.

Address		FLASH Sector
0x0187FFFF	32KB	7
0x01878000	RedBoot Configuration data	
0x01877FFF	32KB	
0x01870000	FIS Directory data	
0x0186FFFF		6
	192KB Free Space	
0x01840000		5
0x0183FFFF		3
	128KB RedBoot	
0x01820000		2
0x0181FFFF		1
	64KB Angel	
0x01810000		
0x0180FFFF		0
	64KB Bootstrap Loader, Production Test + Reserved	
0x01800000		

Notes:

1. The Bootstrap Loader, Production Test + Reserved area should not be overwritten.
2. The Angel program from ARM has been left to preserve the Evaluator-7T's compatibility with ARM's tools.
3. Although 128KB has been reserved for RedBoot, this may be reduced if a cut down version of RedBoot is used.
4. The 4KB of FIS directory data is automatically created when RedBoot's 'fis init -f' command is run (this command should be only run once).
5. The 4KB reserved for configuration data is used automatically when RedBoot updates configuration data in FLASH (after 'alias' or 'fconfig' commands are ran).
6. The 248KB of free space is free to hold FPGA images or application code using RedBoot's FLASH Image System (FIS).
7. Because of the Am29LV400 FLASH's larger sector size, the FIS and configuration data area is larger (1 sector minimum) than with the SST39VF400 FLASH and less free space is available.

4.6.3 Instructions for writing a RedBoot image to FLASH

These instructions detail how to write a RedBoot image of up to 128KB to FLASH, beginning at address 0x01820000. Note: The RedBoot image itself is written to FLASH using ARM's bootstrap loader, it is recommended that all other data to be written to FLASH, is done using RedBoot via its FIS commands.

- ◆ Connect a terminal emulator to the COM1 serial connector on the Evaluator-7T board.
- ◆ Reset the board and press enter on the emulator when the ARM Bootstrap Loader is waiting for 2 seconds.
- ◆ Type: `flasherase 01820000 20000` on the terminal emulator.
- ◆ Type: `Download 10000` on the terminal emulator.
- ◆ Use the send file option to send the RedBoot image file (called redboot.UU or similar) to the board.
- ◆ Type: `flashwrite 01820000 10000 20000` on the terminal emulator.
- ◆ Reset the board and the new RedBoot should boot.

4.7 New RedBoot Command: fpgaload

This section details the use of the new RedBoot command, 'fpgaload'.

4.7.1 'fpgaload' Command Configuration

The 'fpgaload' command is added to RedBoot by including the CYGPKG_REDBOOT_FPGALOAD_COMMAND (fpgaload command for RedBoot) package in your RedBoot configuration.

Once included, the 'fpgaload' command package has a single option that may be set up with the graphical configuration tool (or by editing the .ecc configuration file for developers using the command line configuration tool). This options is:

cdl_package	CYGPKG_REDBOOT_FPGALOAD_COMMAND
display	fpgaload command for RedBoot
description	This package adds the 'fpgaload' command to RedBoot. This command is for the E7T target and is used to configure the FPGA on the E7T-DBoard.

cdl_component	CYGBLD_BUILD_REDBOOT_WITH_FPGALOAD
display	Include fpgaload command
type	Boolean
default value	1
description	This option specifies whether or not the fpgaload command is included in RedBoot or not. If set to 0, the command is not included.

Note: The 'fpgaload' command requires the CYGPKG_COMPRESS_ZLIB (Zlib compress and decompress) package to be added to the configuration before it can be build.

4.7.2 'fpgalload' Command Operation

Any use of the 'fpgalload' command requires that switches 1, 2 and 3 on the Evaluator-7T board are in their OFF positions. The FPGA configuration operation will fail if they are not.

4.7.2.1 Command Line Operation

Once RedBoot is running, the 'fpgalload' command can be accessed in the same way as any of RedBoot's other commands are accessed.

The format of the 'fpgalload' command is:

```
fpgalload [-f] [-b <image_location>] [-n <fis_image_name>]
```

Where:

- f** Force FPGA configuration, even if the FPGA is already configured.
- b** Address of the FPGA image to be used.
- n** Name of FPGA image stored using RedBoot's FIS.

Command Line Examples:

```
fpgalload -b 0x02020000
```

Configure FPGA with the image at address 0x02020000 (SDRAM). Do not configure the FPGA if it is already configured.

```
fpgalload -b 0x01860000 -f
```

Configure FPGA with the image at address 0x01860000 (FLASH). Configure the FPGA whether or not it has already been configured.

```
fpgalload -n stduart.fpga
```

Configure FPGA with the image named 'stduart.fpga', stored using RedBoot's FIS. Do not configure the FPGA if it is already configured. Note: RedBoot's FIS does not require file extensions, but the use is recommended so that human users may identify file types of images in the FIS.

```
fpgalload -n stduart.fpga -f
```

Configure FPGA with the image named 'stduart.fpga', stored using RedBoot's FIS. Configure the FPGA whether or not it has already been configured.

Note, in the examples attempting to configure using an FPGA image in SDRAM, an FPGA image would need to be downloaded to the SDRAM first. This may be achieved using RedBoot's 'load' command. Instructions for manipulating FPGA images are given in section 4.8.2.

4.8 Useful RedBoot Operations

This section describes useful operations that may be performed using RedBoot. This is not intended to be a replacement for the RedBoot user manual, which should be referenced for full details of RedBoot's commands.

4.8.1 Required initialisation after RedBoot installation

After a RedBoot image with FIS support has been written to the FLASH, an initialisation process is required. This is not required for RedBoot images without FIS support.

4.8.1.1 Initialisation for RedBoot with FIS support included

This section describes the procedure to correctly initialise a newly installed RedBoot image with FIS support, but not ethernet support included. Connect a terminal emulator to serial port com1 on the Evaluator-7T board, with the emulator set to 38400 baud, 8-bit data, no parity, 1 stop bit.

When power is applied to the board, the following will be displayed on the terminal emulator:

```
ARM Evaluator7T Boot Strap Loader Release 1.01
Press ENTER within 2 seconds to stop autoboot
+FLASH configuration checksum error or invalid key

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version E7T-DBoard-built 21:04:28, Oct 6 2003

Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x00080000, [0x0000e008-0x00070000] available
      0x01900000-0x03900000, [0x01900000-0x03900000] available
FLASH: 0x01800000 - 0x01880000, 128 blocks of 0x00001000 bytes each.
RedBoot>
```

The board is now ready to be initialised. The FIS should be initialised by typing 'fis init -f' at the 'RedBoot>' prompt. The result should be as shown below:

```
RedBoot> fis init -f
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x01840000-
0x0187e000: .....
.....
... Erase from 0x0187f000-0x0187f000:
... Erase from 0x01880000-0x01880000:
... Erase from 0x0187f000-0x01880000: .
... Program from 0x0007e000-0x0007f000 at 0x0187f000: .
RedBoot>
```

Warning: All data in FLASH after address 0x01840000 will be lost. The initialisation is now complete and the FIS is ready for use.

Typing the RedBoot command 'fis list' now gives a list of all the files stored in the FIS (just the default files for now):

```
RedBoot> fis list
Name                FLASH addr  Mem addr    Length      Entry point
(reserved)          0x01800000 0x01800000 0x00020000 0x00000000
RedBoot[post]       0x01820000 0x01820000 0x00020000 0x00000000
RedBoot config      0x0187E000 0x0187E000 0x00001000 0x00000000
FIS directory       0x0187F000 0x0187F000 0x00001000 0x00000000
RedBoot>
```

4.8.1.2 Initialisation for RedBoot with FIS and Ethernet support included

This section describes the procedure to correctly initialise a newly installed RedBoot image with both FIS and ethernet support included. Connect a terminal emulator to serial port com1 on the Evaluator-7T board, with the emulator set to 38400 baud, 8-bit data, no parity, 1 stop bit.

Power up the board and wait for the 'RedBoot>' prompt. Note, If the board is not connected to an ethernet network with a BOOTP server, then this could take about 30 seconds.

Once the 'RedBoot>' prompt is there, the board is ready to be initialised. Firstly the FIS should be initialised by typing 'fis init -f' at the 'RedBoot>' prompt. The result should be as shown below.

```
RedBoot> fis init -f
About to initialize [format] FLASH image system - continue (y/n)? y
*** Initialize FLASH Image System
... Erase from 0x01840000-
0x0187e000: .....
.....
... Erase from 0x0187f000-0x0187f000:
... Erase from 0x01880000-0x01880000:
... Erase from 0x0187f000-0x01880000: .
... Program from 0x0007e000-0x0007f000 at 0x0187f000: .
RedBoot>
```

Warning: All data in FLASH after address 0x01840000 will be lost. The FIS is now initialised.

Next, the network configuration options for the board should to be set up. This is done by typing 'fconfig -i' at the 'RedBoot>' prompt. An example of setting up these options for a board not using a BOOTP server is shown below:

Note: The '-i' option is only required when the configuration database needs to be reset to its default state. This is needed the first time RedBoot is installed on the target, or when updating to a newer RedBoot with different configuration keys."

```

RedBoot> fconfig -i
Initialize non-volatile configuration - continue (y/n)? y
Run script at boot: false
Use BOOTP for network configuration: false
Gateway IP address: 192.168.1.1
Local IP address: 192.168.1.50
Local IP address mask: 255.255.255.0
Default server IP address: 192.168.1.80
DNS server IP address: 194.168.8.100
Network hardware address [MAC] for eth0:0xDE:0xAD:0xBE:0xEF:0x00:0x01
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0x0187e000-0x0187f000: .
... Program from 0x00071000-0x00072000 at 0x0187e000: .
RedBoot>

```

The IP addresses should be set up to suit your own network.

If you do want to use a BOOTP server then the fconfig set-up will look like the example below:

```

RedBoot> fconfig -i
Initialize non-volatile configuration - continue (y/n)? y
Run script at boot: false
Use BOOTP for network configuration: true
Default server IP address: 192.168.1.80
DNS server IP address: 194.168.8.100
Network hardware address [MAC] for eth0:0xDE:0xAD:0xBE:0xEF:0x00:0x01
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0x0187e000-0x0187f000: .
... Program from 0x00071000-0x00072000 at 0x0187e000: .
RedBoot>

```

If a BOOTP server is used, then only IP addresses that need to be specified are the Default server IP address and the DNS server IP address.

The initialisation is now complete. Typing the RedBoot command 'fis list' now gives a list of all the files stored in the FIS (just the default files for now):

```

RedBoot> fis list
Name                FLASH addr  Mem addr    Length      Entry point
(reserved)          0x01800000 0x01800000 0x00020000 0x00000000
RedBoot[post]       0x01820000 0x01820000 0x00020000 0x00000000
RedBoot config      0x0187E000 0x0187E000 0x00001000 0x00000000
FIS directory       0x0187F000 0x0187F000 0x00001000 0x00000000
RedBoot>

```

When the Evaluator-7T is reset, the output seen on the terminal emulator should be similar to this:

```
Press ENTER within 2 seconds to stop autoboot
+Ethernet eth0: MAC address de:ad:be:ef:00:01
IP: 192.168.1.50/255.255.255.0, Gateway: 192.168.1.1
Default server: 192.168.1.80, DNS server IP: 194.168.8.100

RedBoot(tm) bootstrap and debug environment [ROM]
Non-certified release, version E7T-DBoard-built 21:04:28, Oct 6 2003

Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x00080000, [0x00017c10-0x00070000] available
      0x01900000-0x03900000, [0x01900000-0x03900000] available
FLASH: 0x01800000 - 0x01880000, 128 blocks of 0x00001000 bytes each.
RedBoot>
```

4.8.2 Manipulating FPGA images

For the purposes of this section, an FPGA image is a .bit file generated by the Xilinx FPGA tools and used to configure the FPGA on the E7T-DBoard. Also referred to an FPGA image is a .bit file that has been compressed with gzip – a .bit.gz file.

The ‘fpgaload’ command handles both kinds of file automatically, so there is no real point in distinguishing between them – however, the zipped versions are substantially smaller and therefore preferable for storing in the limited FLASH memory, or downloading over a slow serial connection.

4.8.2.1 Loading an FPGA image over a serial connection

In this example an FPGA image, fpgastduart.bit.gz, on the host PC is going to be used to configure the FPGA on the E7T-DBoard.

Firstly the ‘load’ command is issued as below, then the file, fpgastduart.bit.gz, is sent using the ‘send file with Xmodem protocol function’ of the terminal emulator. Note: Ymodem protocol is also supported by RedBoot.

```
RedBoot> load -r -m xmodem -b 0x01900000
Raw file loaded 0x01900000-0x0190919c, assumed entry at 0x01900000
xyzModem - Cksum mode, 1307(SOH)/0(STX)/0(CAN) packets, 2 retries
RedBoot>
```

At this point the file has been transferred to the SDRAM (0x01900000).

Next, the ‘fpgaload’ command is issued as below to configure the FPGA with the image at address 0x01900000.

```
RedBoot> fpgaload -b 0x01900000
XC2S200 FPGA detected
Compressed FPGA image found at address 0x1900000
  Design name: fpgastduart.ncd
  Build date: 2003/03/16
Loading FPGA with image
FPGA programming completed OK
RedBoot>
```

The FPGA has now been configured and the FPGA done indicator LED on the E7T-DBoard should now be lit.

4.8.2.2 Loading an FPGA image over an ethernet connection

To load an image over an ethernet connection, a TFTP server containing the FPGA image must be present on the network. TFTP server programs are widely available free of charge on the Internet.

In this example the Default Server IP Address as set-up by the ‘fconfig’ command (or specified by the BOOTP server) is assumed to be the TFTP server. If this is not the case, then the ‘load’ command below will need to have an extra argument ‘-h ip_address’ added, to specify the TFTP server’s IP address.

Firstly the 'load' command is used to get the file fpgastduart.bit.gz from the TFTP server (if no '-m' option is specified '-m tftp' is assumed):

```
RedBoot> load -r -b 0x01900000 fpgastduart.bit.gz
Using default protocol (TFTP)
Raw file loaded 0x01900000-0x0190919c, assumed entry at 0x01900000
RedBoot>
```

At this point the file has been transferred to the SDRAM (0x01900000).

Next, the 'fpgaload' command is issued as below to configure the FPGA with the image at address 0x01900000.

```
RedBoot> fpgaload -b 0x01900000
XC2S200 FPGA detected
Compressed FPGA image found at address 0x1900000
  Design name: fpgastduart.ncd
  Build date: 2003/03/16
Loading FPGA with image
FPGA programming completed OK
RedBoot>
```

The FPGA has now been configured and the FPGA done indicator LED on the E7T-DBoard should now be lit.

4.8.2.3 Storing an FPGA Image in FLASH

This section details how to store an FPGA image in the Evaluator-7T's FLASH using RedBoot's FIS (FLASH Image System). Once an FPGA image has been stored in FLASH it can be used at any time to configure the FPGA.

Firstly, the file needs to be transferred to the board RAM as before. The example below uses an ethernet transfer, but a serial transfer could be used instead:

```
RedBoot> load -r -b 0x01900000 fpgastduart.bit.gz
Using default protocol (TFTP)
Raw file loaded 0x01900000-0x0190919c, assumed entry at 0x01900000
RedBoot>
```

Now the file is in RAM, it may be transferred to FLASH using the 'fis create' command as shown below. The length of the file ('-l' option) was deduced from 'load' command's output above. The filename specified, stduart.fpga, is completely arbitrary, but must be no longer than 16 characters long. The .fpga extension is recommended so that human users can tell it is an FPGA image stored in the FLASH.

```
RedBoot> fis create -b 0x01900000 -l 0x919c stduart.fpga
** WARNING: RAM address: 0x01900000 may be invalid
   valid range is 0x00000000-0x00080000
... Erase from 0x01840000-0x0184a000: .....
... Program from 0x01900000-0x0190919c at 0x01840000: .....
... Erase from 0x0187f000-0x01880000: .
... Program from 0x0007e000-0x0007f000 at 0x0187f000: .
RedBoot>
```

The file is now stored in the FLASH. The 'fis list' command can be used to verify this:

```
RedBoot> fis list
Name                FLASH addr  Mem addr   Length    Entry point
(reserved)          0x01800000 0x01800000 0x00020000 0x00000000
RedBoot[post]      0x01820000 0x01820000 0x00020000 0x00000000
stduart.fpga       0x01840000 0x01900000 0x0000A000 0x01900000
RedBoot config     0x0187E000 0x0187E000 0x00001000 0x00000000
FIS directory      0x0187F000 0x0187F000 0x00001000 0x00000000
RedBoot>
```

Now that the file is stored in the FLASH, it can be used to configure the FPGA at any time using the 'fpgaload' command with the '-n fis_image_name' option:

```
RedBoot> fpgaload -n stduart.fpga
XC2S200 FPGA detected
Using fis image 'stduart.fpga' found at address 0x1840000
Compressed FPGA image found at address 0x1840000
  Design name: fpgastduart.ncd
  Build date: 2003/03/16
Loading FPGA with image
FPGA programming completed OK
RedBoot>
```

The FPGA has now been configured and the FPGA done indicator LED on the E7T-DBoard should now be lit.

4.8.3 Manipulating eCos application images

During normal development, eCos applications are downloaded to the board and executed using the GDB debugger on the host PC communicating with the GDB stubs included in RedBoot. RedBoot can manipulate eCos applications without the GDB debugger being used though.

The typical command to compile an eCos application (hangman.c in this example) is as shown below:

```
$ arm-elf-gcc -Wall -g -O2 -o hangman.elf -D__ECOS  
-I/ecos-build/ecos-default_install/include hangman.c  
-L/ecos-build/ecos-default_install/lib -Ttarget.ld -nostdlib
```

This produces a file in ELF (Executable and Linking Format), hangman.elf, which is ready to be used with the GDB debugger. The .elf file extension is optional, but recommended.

hangman.elf (1091K)

This file contains a lot of debug information for the debugger making the file much larger than it needs to be. This debug information is of no use when the application is not being ran with the GDB debugger, and should be removed with the strip command, as shown below:

```
$ arm-elf-strip hangman.elf
```

This produces a new ELF file of the same name, but containing no debug information. As a result the file is much smaller.

hangman.elf (129K)

This ELF file, with the debug information stripped out is what we shall refer to as an eCos Application image.

This section also refers to compressed eCos application images, these are not just ELF files that have been compressed with the gzip command, the process is slightly more complicated. Firstly, the stripped ELF format file (hangman.elf in this example), needs to be converted to a plain binary format. This is done with the objcopy command, as below:

```
$ arm-elf-objcopy -O binary hangman.elf hangman.bin
```

This produces a binary format file. The .bin file extension is optional, but recommended.

hangman.bin (94K)

Next this file is compressed using the gzip command, as below:

```
$ gzip -9 hangman.bin
```

This produces a compressed version of the binary format file.

hangman.bin.gz (46K)

This file is what is referred to as a compressed eCos application throughout this section.

4.8.3.1 Loading an eCos application image over a serial connection

In this example an eCos application image, hangman.elf, is going to be downloaded to the board via a serial connection and executed.

Firstly the 'load' command is issued as below, then the file, hangman.elf, is sent using the 'send file with Xmodem protocol function' of the terminal emulator. Note: RedBoot also supports Ymodem protocol.

Once the file has been transferred to the board, the 'go' command is issued. The eCos application will now execute on the target board.

```
RedBoot> load -m xmodem
Entry point: 0x01900040, address range: 0x01900000-0x0191791c
xyzModem - CRC mode, 1033(SOH)/0(STX)/0(CAN) packets, 2 retries
RedBoot> go
```

4.8.3.2 Loading an eCos application image over an ethernet connection

In this example an eCos application image, hangman.elf, is going to be downloaded to the board via an ethernet connection and executed.

To load an image over an ethernet connection, a TFTP server containing the eCos application image must be present on the network. TFTP server programs are widely available free of charge on the Internet.

In this example the Default Server IP Address as set-up by the 'fconfig' command (or specified by the BOOTP server) is assumed to be the TFTP server. If this is not the case, then the 'load' command below will need to have an extra argument '-h ip_address' added, to specify the TFTP server's IP address.

Firstly the 'load' command is used to get the file hangman.elf from the TFTP server:

Once the file has been transferred to the board, the 'go' command is issued. The eCos application will now execute on the target board (if no '-m' option is specified '-m tftp' is assumed).

```
RedBoot> load hangman.elf
Using default protocol (TFTP)
Entry point: 0x01900040, address range: 0x01900000-0x0191791c
RedBoot> go
```

4.8.3.3 Storing an eCos application in FLASH

This section details how to store an eCos application image in the Evaluator-7T's FLASH using RedBoot's FIS (FLASH Image System). Once an application image has been stored in FLASH it can be transferred to RAM and executed at any time.

Firstly, the eCos application image needs to be transferred from the host PC to the board's RAM as before. The example below uses an ethernet transfer, but a serial transfer could be used instead:

```
RedBoot> load hangman.elf
Using default protocol (TFTP)
Entry point: 0x01900040, address range: 0x01900000-0x0191791c
RedBoot>
```

Now the file is in RAM, it may be transferred to FLASH using the 'fis create' command as shown below. The length of the file ('-l' option) was deduced from 'load' command's output above. The filename specified, hangman.elf, is completely arbitrary, but must be no longer than 16 characters long. The .elf extension is recommended so that human users can tell it is an application image stored in the FLASH and issue the appropriate commands to load and execute it.

```
RedBoot> fis create -b 0x01900000 -l 0x1791c hangman.elf
** WARNING: RAM address: 0x01900000 may be invalid
    valid range is 0x00000000-0x00080000
... Erase from 0x0184a000-0x01862000: .....
... Program from 0x01900000-0x0191791c at
0x0184a000: .....
... Erase from 0x0187f000-0x01880000: .
... Program from 0x0007e000-0x0007f000 at 0x0187f000: .
RedBoot>
```

The file is now stored in the FLASH. The 'fis list' command can be used to verify this:

```
RedBoot> fis list
Name                FLASH addr  Mem addr    Length      Entry point
(reserved)          0x01800000 0x01800000 0x00020000 0x00000000
RedBoot[post]      0x01820000 0x01820000 0x00020000 0x00000000
stduart.fpga       0x01840000 0x01900000 0x0000A000 0x01900000
hangman.elf        0x0184A000 0x01900000 0x00018000 0x01900040
RedBoot config     0x0187E000 0x0187E000 0x00001000 0x00000000
FIS directory      0x0187F000 0x0187F000 0x00001000 0x00000000
RedBoot>
```

Now the file is stored in the FLASH, it can be transferred to RAM and executed at any time. To transfer the file to RAM the 'fis load' command is used as below. Then the 'go' command is issued and the application will execute on the target board.

```
RedBoot> fis load hangman.elf
RedBoot> go
```

4.8.3.4 Storing a compressed eCos application image in FLASH

This section details how to store a compressed eCos application image (.bin.gz file) in the Evaluator-7T's FLASH using RedBoot's FIS (FLASH Image System). Storing the compressed version saves space in the FLASH, and once the compressed application image has been stored in FLASH it can be loaded to RAM and executed at any time.

Firstly, the compressed eCos application image needs to be transferred to the board RAM. The example below uses an ethernet transfer, but a serial transfer could be used instead. Note that extra options '-r' and '-b ram_address' are required:

```
RedBoot> load -r -b 0x01900000 hangman.bin.gz
Using default protocol (TFTP)
Raw file loaded 0x01900000-0x0190b2f0, assumed entry at 0x01900000
RedBoot>
```

Now the file is in RAM, it may be transferred to FLASH using the 'fis create' command as shown below. The length of the file ('-l' option) was deduced from 'load' command's output above.

The filename specified, hangman.gz, is completely arbitrary, but must be no longer than 16 characters long. The .gz extension is recommended so that human users can tell it is a compressed application image stored in the FLASH and issue the appropriate commands to load and execute it.

Notice that because we are not storing an ELF format file, we need to supply 'fis create' with an address in RAM where the 'fis load' command will load the image ('-r' option) and the address in RAM where the 'go' command jumps to in order to execute the application ('-e' option).

These two new values can be obtained by using the objdump utility on the ELF file used to create the .bin.gz file:

```
$ arm-elf-objdump hangman.elf -h
```

The output from this command will be a table of hex sizes and addresses. The '-r' option of the 'fis create' command is the address in the VMA column of the .rom_vectors row, and the '-e' option is the address in the VMA column of the .text row.

```
RedBoot> fis create -b 0x01900000 -l 0xb2f0 -e 0x01900040
-r 0x01900000 hangman.gz
** WARNING: RAM address: 0x01900000 may be invalid
   valid range is 0x00000000-0x00080000
... Erase from 0x01862000-0x0186e000: .....
... Program from 0x01900000-0x0190b2f0 at 0x01862000: .....
... Erase from 0x0187f000-0x01880000: .
... Program from 0x0007e000-0x0007f000 at 0x0187f000: .
RedBoot>
```

The file is now stored in the FLASH. The 'fis list' command can be used to verify this:

```
RedBoot> fis list
Name                FLASH addr  Mem addr    Length      Entry point
(reserved)          0x01800000 0x01800000 0x00020000 0x00000000
RedBoot[post]      0x01820000 0x01820000 0x00020000 0x00000000
stduart.fpga        0x01840000 0x01900000 0x0000A000 0x01900000
hangman.elf         0x0184A000 0x01900000 0x00018000 0x01900040
hangman.gz          0x01862000 0x01900000 0x0000C000 0x01900040
RedBoot config     0x0187E000 0x0187E000 0x00001000 0x00000000
FIS directory       0x0187F000 0x0187F000 0x00001000 0x00000000
RedBoot>
```

Now the file is stored in the FLASH, it can be transferred to RAM and executed at any time. To transfer the file to RAM the 'fis load' command is used with the '-d' (decompress) option as shown below. Then the 'go' command is issued and the application will execute on the target board.

```
RedBoot> fis load -d hangman.gz
Image loaded from 0x01900000-0x0191791c
RedBoot> go
```

4.8.4 Automating Operations at boot time

By using RedBoot's boot script feature, it is possible to perform various complex operations at boot time. The boot script function is accessed through the 'fconfig' command. The example below shows a simple script that configures the FPGA with an image stored in the FLASH memory each time the board boots.

```
RedBoot> fconfig
Run script at boot: true
Boot script:
Enter script, terminate with empty line
>> fpgaload -n stduart.fpga
>>
Boot script timeout (1000ms resolution): 1
Use BOOTP for network configuration: false.
Update RedBoot non-volatile configuration - continue (y/n)? y
... Erase from 0x0187e000-0x0187f000: .
... Program from 0x00071000-0x00072000 at 0x0187e000: .
RedBoot>
```

Note the use of the '.' at the 'Use BOOTP for network configuration:' prompt. This tells RedBoot that we do not want to change any more configuration options, and shortens the 'fconfig' operation.

Other, more complex, boot scripts are possible. For example, the following script will get an FPGA image from a TFTP server and configure the FPGA with it. Then, it will get the hangman application from the TFTP server and execute it.

```
>> load -r -b 0x01900000 fpgastduart.bit.gz
>> fpgaload -b 0x01900000
>> load hangman.elf
>> go
```