DINK32 Reference Manual

DINKRM
Rev. 13.3, 11/2006

freescale™
semiconductor

How to Reach Us:

Home Page:
www.freescale.com

email:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
(800) 521-6274
480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
(800) 441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: DINKRM
Rev. 13.3, 11/2006

Contents

Paragraph Number	Title	Page Number
	Chapter 1 Introduction	
1.1	Features	1-2
1.2	What's New for DINK Since V13.0	1-3
1.3	DINK32 Design Methodology	1-3
1.4	Memory Map for DINK32 Version 13.2	1-4
1.5	Memory Map for DINK32 Version 13.3	1-5
1.6	eDINK Memory Map	1-6
	Chapter 2 Using DINK	
2.1	Communications Setup	2-1
2.1.1	Cables	2-1
2.1.2	Terminals	2-1
2.1.2.1	Hyperterm on NT	2-2
2.1.2.2	Zterm on Mac	2-2
2.2	Starting DINK	2-3
2.3	Running eDINK on a Simulator	2-3
2.3.1	Running eDINK on e500 Core ISS	2-3
2.3.1.1	sportal_file	2-4
2.3.1.2	sportal_term	2-4
2.3.2	Running eDINK on MPC8540 ISS	2-4
2.3.3	Loading Application Elf Binary to Simulated eDINK	2-4
2.4	DINK Model	2-4
2.5	Interacting with DINK	2-5
2.5.1	Running External Programs	2-5
2.5.2	Assigning Breakpoints	2-5
2.5.3	Executing Code	2-6
2.5.4	Displaying Registers	2-6
2.5.5	Advanced Features	2-6
	Chapter 3 DINK32 Commands	
3.1	Definitions	3-1
3.2	DINK Command List	3-1
3.3	. (Period)	3-4
3.4	About	3-5

Contents

Paragraph Number	Title	Page Number
3.5	Assemble	3-6
3.6	Background	3-8
3.7	Benchmark	3-8
3.8	Bootm	3-11
3.9	Breakpoint	3-12
3.10	Cache	3-14
3.11	CONFIGTEMP	3-15
3.12	CPUINFO	3-17
3.13	CRC	3-18
3.14	DEVDISP	3-18
3.15	DEVMOD	3-19
3.16	DEVTEST	3-21
3.17	DISASSEM	3-21
3.18	Disk	3-23
3.19	DISPTEMP	3-24
3.20	Do	3-25
3.21	Download	3-25
3.22	DownloadFlash	3-27
3.23	Echo	3-28
3.24	Environment	3-28
3.24.1	ENV—BOOT Encoding	3-30
3.24.2	ENV—IO Encoding	3-30
3.24.3	ENV —L2CACHE Encoding	3-31
3.24.4	ENV—L2PRIVATE Encoding	3-32
3.24.5	ENV—L3CACHE Encoding	3-33
3.24.6	ENV—L3PRIVATE Encoding	3-35
3.24.7	ENV—ONTR Encoding	3-35
3.24.8	ENV—PROMPT Encoding	3-35
3.24.9	ENV—TOFFSET Encoding	3-36
3.24.10	ENV—FANPWM Encoding	3-36
3.24.11	ENV—TSHUTDOWN Encoding	3-37
3.25	ETH	3-37
3.26	EX	3-38
3.27	FILESYS	3-40
3.28	Final Assembly Init	3-41
3.29	Flash	3-41
3.30	FOR	3-42
3.31	FUPDATE	3-43
3.32	FW	3-46
3.33	Go	3-46
3.34	GPIC	3-48

Contents

Paragraph Number	Title	Page Number
3.34.1	DINK32 GPIC Initialization Sequence	3-49
3.35	Help	3-51
3.36	History	3-51
3.37	Hardware Monitor	3-52
3.38	I^2C	3-53
3.39	Identify	3-53
3.40	If	3-54
3.41	Log	3-55
3.42	Memory Compare	3-55
3.43	Memory Display	3-56
3.44	Memory Fill	3-57
3.45	Memory Info	3-58
3.46	Memory Modify	3-59
3.47	Memory Move	3-60
3.48	Memory Search	3-61
3.49	Memory Test	3-62
3.50	Menu	3-64
3.51	Northbridge Display	3-65
3.52	Northbridge Modify	3-66
3.53	Net Info	3-68
3.54	Opts	3-69
3.55	PCI	3-69
3.56	PCI Config List	3-70
3.57	PCI Display	3-71
3.58	PCI Map	3-72
3.59	PCI Modify	3-72
3.60	PCI Probe	3-73
3.61	PHY	3-74
3.62	Ping	3-74
3.63	PX	3-75
3.64	REGDISP	3-76
3.65	REGMOD	3-79
3.66	Reset	3-81
3.67	SETBAUD	3-82
3.68	SQUIT	3-83
3.69	Show SPRs	3-83
3.70	STTY	3-84
3.71	Switch CPU	3-85
3.72	SX	3-85
3.73	SYMTAB	3-86
3.74	TAU	3-88

Contents

Paragraph Number	Title	Page Number
3.75	Test	3-89
3.76	Time	3-89
3.77	Transparent	3-90
3.78	Trace	3-91
3.79	Upload	3-91
3.80	Virtual Host File	3-92
3.81	Virtual Memory File	3-93
3.82	VQUIT	3-93
3.83	Virtual Simulator	3-94
3.84	Wait	3-95
3.85	Watch	3-95
3.86	While	3-96
3.87	Shell Escape	3-97

Appendix A
History of DINK32 Changes

A.1	Version 13.3, November 06, 2006	A-1
A.2	Version 13.2, July 15, 2005	A-2
A.3	Version 13.1, April 30, 2003	A-4
A.4	Version 13.0, November 1, 2002	A-5
A.5	Version 12.3, October 25, 2001	A-6
A.6	Version 12.2, February 16, 2001	A-8
A.7	Version 12.1, August 30, 1999	A-8
A.8	Version 12.0, November 30, 1999	A-9
A.9	Version 11.0.2, June 1, 1999	A-9
A.10	Version 11.0.1, May 1, 1999 (Not Released)	A-10
A.11	Version 11.0, March 29, 1999	A-10
A.12	Version 10.7, February 25, 1999	A-10
A.13	Version 10.6, January 25, 1999	A-10
A.14	Version 10.5, November 24, 1998	A-10
A.15	Version 10.4, November 11, 1998	A-11
A.16	Version 10.3, No Date	A-11
A.17	Version 10.2, September 11, 1998	A-11
A.18	Version 10.1, September 10, 1999	A-11
A.19	Version 9.5, August 5, 1998	A-11
A.20	Version 9.4, May 22, 1998	A-11
A.21	Prior to Version 9.4, Approximately October 10, 1997	A-11

Contents

Paragraph Number	Title	Page Number
	Appendix B Example Code	
B.1	agentboot	B-2
B.1.1	Background	B-2
B.1.2	In This Directory	B-2
B.1.3	Assumptions	B-2
B.1.4	Usage	B-2
B.2	Dink_demo	B-3
B.2.1	Building	B-3
B.2.2	Function Addresses	B-3
B.3	Dhrystone - mwnosc	B-3
B.3.1	Building	B-3
B.3.2	Function Addresses	B-3
B.4	l1test and l1teststd	B-3
B.4.1	Building	B-4
B.5	l2sizing	B-4
B.5.1	In this Directory	B-4
B.5.2	Assumptions	B-4
B.5.3	Usage	B-4
B.5.4	To Build	B-4
B.5.5	Notes	B-4
B.6	backsideL2test	B-5
B.6.1	Building	B-5
B.6.2	Function Addresses	B-5
B.7	lab4 (Excimer only)	B-5
B.7.1	Building	B-5
B.7.2	Function Addresses	B-5
B.8	memspeed	B-5
B.8.1	Building	B-6
B.8.2	Function Addresses	B-6
B.9	printtest	B-6
B.9.1	Building	B-6
B.9.2	Function Addresses	B-6
B.10	testfile	B-6
B.10.1	Building	B-6
B.10.2	Function Addresses	B-6
B.11	l3test	B-7
B.11.1	Building	B-7
B.11.2	Function Addresses	B-7
B.12	SPINIT	B-7

Contents

Paragraph Number	Title	Page Number
B.12.1	Building	B-7
B.12.2	Status	B-7
B.13	duart_8245	B-8
B.13.1	Building	B-8
B.13.2	Usage	B-8
B.14	galileo - mvp only	B-8
B.14.1	Building	B-9
B.14.2	Usage	B-9
B.15	galileo_read_write - mvp only	B-9
B.15.1	Building	B-9
B.15.2	Usage	B-9
B.16	linpack	B-9
B.16.1	Building	B-9
B.16.2	Usage	B-10
B.17	watch—Excimer Only	B-10
B.17.1	Building	B-10
B.17.2	Usage	B-10

Appendix C
Building and Extending DINK

C.1	Obtaining DINK32 Source	C-1
C.2	Building DINK32	C-1
C.3	DINK Software Build Process	C-1
C.4	Installation	C-3
C.4.1	Sandpoint	C-3
C.4.1.1	Excimer and Maximer	C-3
C.5	Extending DINK	C-4
C.5.1	Writing Commands	C-4
C.5.2	Linking Commands	C-6

Appendix D
User-Callable DINK Functions

D.1	General Information	D-1
D.2	Methodology and Implementation	D-1
D.3	Setting Up Static Locations	D-3
D.4	Using the Dynamic Functions	D-4
D.5	Error Conditions	D-5
D.6	Alternative Method for Metaware Only	D-6

Contents

Paragraph Number	Title	Page Number
	Appendix E MPC8240 Drivers	
E.1	MPC8240 DMA Memory Controller	E-1
E.1.1	Background	E-1
E.1.2	Overview	E-2
E.1.3	DMA Application Program Interface (API)	E-2
E.1.4	API Example Usage	E-4
E.1.5	DMA Driver Library Internals (DLI)	E-5
E.1.5.1	Common Data Structures and Values	E-5
E.1.6	Function Descriptions of DMA Driver Library Internals	E-8
E.2	MPC8240 I^2C Driver Library	E-12
E.2.1	Background	E-12
E.2.2	Overview	E-12
E.2.3	I2C Application Program Interface (API)	E-13
E.2.3.1	API Functions Description	E-13
E.2.3.2	API Example Usage	E-14
E.2.4	I2C Driver Library Internals (DLI)	E-16
E.2.4.1	Common Data Structures and Values	E-16
E.2.5	MPC8240/MPC8245/MPC8241 I2C Driver Library Internals Function Descriptions	E-17
E.2.5.1	DLI Functions Written but Not Used and Not Tested	E-22
E.2.6	I2C Support Functions	E-23
E.3	MPC8240 I_2O Doorbell Driver	E-23
E.3.1	I_2O Description of Doorbell Communication Between Agent and Host	E-23
E.3.1.1	System Startup and Memory Map Initialization	E-24
E.3.1.2	Interrupt Service Routines: I_2O_ISR_host() and I_2O_ISR_agent()	E-25
E.3.1.3	Enable Doorbell Interrupts	E-25
E.3.1.4	Writing and Reading Doorbell Registers	E-26
E.3.1.4.1	Host Rings an Agent via Agent's Inbound Doorbell	E-26
E.3.1.4.2	Agent Rings a Host via Agent's Outbound Doorbell	E-26
E.3.1.4.3	Descriptions of the I_2O Library Functions	E-28
E.3.2	I2C Driver Library	E-29
E.3.2.1	Background	E-29
E.3.2.2	Overview	E-30
E.3.2.3	I2C Application Program Interface (API)	E-30
E.3.2.3.1	API Functions Description	E-30
E.3.2.3.2	API Example Usage	E-32
E.3.2.4	I2C Driver Library Internals (DLI)	E-33
E.3.2.4.1	Common Data Structures and Values	E-34

Contents

Paragraph Number	Title	Page Number
E.3.2.4.2	MPC8240/MPC8245/MPC8241 I2C Driver Library Internals Function Descriptions	E-35
E.3.2.4.3	I2C Support Functions	E-40
E.4	MPC8240 PIC Interrupt Driver	E-41
E.4.1	General Description	E-41
E.4.2	PIC Specifics	E-41
E.4.2.1	Embedded Utilities Memory Block (EUMB)	E-42
E.4.2.2	PIC Register Summary	E-42
E.4.2.3	PIC Modes	E-43
E.4.3	Drivers/Epic Directory Structure	E-43
E.4.4	PIC Cross-Reference Table Structure	E-43
E.4.5	PIC Sample Routines	E-44
E.4.5.1	Low-Level Routines	E-44
E.4.5.2	High-Level Routines	E-45
E.4.5.2.1	PIC Initialization Routines	E-45
E.4.5.2.2	High-Level Exception Handler	E-45
E.4.5.2.3	Direct/Serial Register Control Routines	E-45
E.4.5.2.4	Global Timer Register Control Routines	E-46
E.4.6	PIC Commands in DINK32	E-46
E.4.7	PIC Unit Startup	E-47
E.4.8	External Interrupt Exception Path in DINK32	E-48
E.4.9	Example Usage on Sandpoint Reference Platform	E-48
E.4.9.1	Sandpoint X2 Reference Platform	E-48
E.4.9.2	Initializing PIC on Sandpoint X2	E-49
E.4.9.3	Running the PIC on Sandpoint X3	E-50
E.4.10	Code and Documentation Updates	E-52

Appendix F
S-Record Format Description

F.1	General Format	F-1
F.2	Specific Formats	F-2
F.3	Examples	F-2
F.4	Summary of Formats	F-3

Appendix G
Networking Support

G.1	Troubleshooting Hints	G-2
G.2	Known Limitations in Telnet Console	G-2

Contents

Paragraph Number	Title	Page Number
	Appendix H Converting Dink32 to Little-Endian	
H.1	General Information	H-1
H.2	Preparation	H-1
H.3	Explanation	H-2
H.3.1	Two Important Considerations	H-2
H.3.2	Copy Algorithm	H-2
H.3.2.1	Little-Endian Swap Instruction Sequence	H-3
H.3.3	DINKLE V7.0 10/8/97 makefile	H-4

Contents

Paragraph Number	Title	Page Number

Chapter 1
Introduction

DINK is an acronym for Demonstrative Interactive Nano Kernel.

DINK32 is a flexible software tool enabling initialization, evaluation, and debugging of the PowerPC™ 32-bit microprocessors (built on Power Architecture™ technology). The introduction of the PowerPC architecture provided an opportunity to create an interactive debugger independent from previous debug monitors. Since this family of microprocessors spans a wide market range, DINK32 has to be extensible and portable, as well as being specific enough to be useful for a wide variety of applications. It is designed to be both a hardware and software debugging tool. DINK32 was written in ANSI C and built with modular routines around a central core. Only a few necessary functions were written in PowerPC assembly. This document describes the DINK32 software, the DINK32 command set, utilities, user program execution, errors and exceptions, and restrictions.

DINK supports numerous compilers, compilation options, and target platforms. Some of these are used often enough to warrant a distinctive name:

DINK	Standard DINK
MDINK	Minimal DINK—fits in one sector of the Excimer/Maximer flash and can load standard DINK32 into the remainder. Only available with Excimer—no source or downloadable code is available.
VDINK	Virtual DINK—a version of DINK which runs under Linux, Unix, or Windows and can emulate both DINK and the 32-bit PowerPC ISA sufficiently to emulate itself
eDINK	DINK for the e500 core processor
SADINK	Standalone DINK. For the UnityX4 MPC8245 or UnityLC X1 MPC8241, standalone DINK runs on the MPMC module outside of the Sandpoint environment (see Freescale Semiconductor Application Note AN2207, *Using the MPMC8245 Card in Standalone Mode*).

However, in this reference manual, all versions of DINK will be referred to as DINK unless a particular variant is specifically referred to.

Starting with release R12.3, source files are not supplied for all functions of DINK32 (but are present for the DINK drivers, demos, and utilities). All chapters dealing with the full source assume that the user has requested a license for these from a Freescale Semiconductor Sales representative or from the Freescale Semiconductor DINK32 website.

There is no requirement to obtain any of these sources in order to use the DINK32 S-records supplied on the DINK32 web site above.

1.1 Features

The DINK32 software package provides:

- Support of the MPC601, MPC603, MPC603e, MPC604, MPC604e, MPC740, MPC750, MPC7400, MPC7410, MPC7440, MPC7445, MPC7447, MPC7447A, MPC7448, MPC7450, MPC7455, MPC7457, MPC8240, MPC8245, MPC8540, MPC8560, MPC8555, MPC8641D, MPC854x and MPC855x processors. Note that support for the older processors such as MPC8240 is only possible with DINK32 version 13.2 or earlier.
- Support of the e500 core Instruction Set Simulator (ISS). See Freescale Semiconductor Application Note AN2336, *Creating edink from DINK32 Code Using the e500 ISS*
- Modification and display of general purpose, floating point, AltiVec™, and special purpose registers
- Assembly and disassembly of PowerPC instructions for modification and display of code
- Modification, display, and movement of system memory
- A simplified breakpoint command that allows setting, display, and removal of breakpoints
- Single-step trace and continued execution from a specified address
- Automatic decompression of compressed s-record files while downloading
- Extensive online help
- Ability to execute user-assembled and/or downloaded software in a controlled environment
- Logging function for generating a transcript of a debugging session
- Register set includes all of the implementation-specific registers
- Modification of memory at byte, half-word, word, and double-word lengths
- Extensive support for the MPC60x, MPC74x, MPC75x, and MPC74x0 simplified or extended mnemonics during assembly and disassembly of PowerPC instructions
- Ability to input immediate values to the assembler as binary, decimal, or hexadecimal
- Command line download functionality that allows the user to select the download port and then send the data
- An assembler and disassembler that understands branch labels and the ability to see and clear the branch table that DINK32 is using while assembling and disassembling PowerPC instructions
- Ability to read and write MPC106 configuration registers (not supported on Excimer and Maximer)
- Support for PCI with new ‘pci-’ commands (not supported in minimal builds, i.e., Excimer and Maximer)
- Support for Excimer and Maximer flash, fl –dsi and –se, will automatically detect flash on Revision 2. Revision 3 of the board, the fl -dsi function, has been expanded to display the memory range for each sector.
- Support for Excimer and Maximer flash, fl -sp, and -su
- Support for the Max chip and AltiVec registers and instructions
- Support for the MPC8245/MPC8241/MPC8240 chips
- Support for the MPC107 memory bridge

- Support for dynamically assigned DINK function addresses and variables for downloaded programs. See Appendix D, “User-Callable DINK Functions.”
- Support for Yellowknife and Sandpoint flash ROMs, fu command
- Support for multiprocessing MVP platform
- Support for remote access via TELNET protocol
- Support for network download via TFTP

1.2 What’s New for DINK Since V13.0

- Support for the MPC7447, MPC7457, MPC7448, MPC8641D, MPC854x and MPC855x processors.
- Enhancements to numerous commands including download and env
- Support of eDINK enhanced
- New and better assembler and disassembler integrated
- Support for Automatic Thermal Monitoring System (ATMS)

See Appendix A, “History of DINK32 Changes,” for more details.

1.3 DINK32 Design Methodology

The modular design of the DINK32 program, its extensive commenting, and its design methodology enable efficient user modification of the code. Thus, DINK32 provides a flexible and powerful framework for users who desire additional functionality.

The DINK32 software package can be executed on microprocessor boards that include the following devices and minimum memory configuration:

- MPC60x, MPC74x, MPC75x, MPC824x, MPC74xx, MPC8641D, MPC854x, MPC855x microprocessors.
- Industry-standard ‘PC16x50’ serial port (such as the National Semiconductor PC87308 DUART (Yellowknife and Sandpoint Reference Design) or PC16552 DUART (Excimer and Maximer Minimal Evaluation Board)).
- 512 Kbyte EPROM or Flash for DINK32 version 13.1.1 or earlier. 2MBytes of EPROM or Flash for DINK32 version 13.3.
- 512 Kbyte RAM for DINK32 version 13.2 or earlier. 256MBytes of RAM for DINK32 version 13.3.

eDINK software package can be executed on simulators or microprocessor boards that include the following devices and minimum memory configuration:

- MPC8540 Instruction Set Simulator
- e500 core Instruction Set Simulator

Also, on microprocessor boards such as the ‘Mars:Elysium’ MPC8540 reference board that include the following devices:

- Flash on LCS0

- At least 64 MB DDR SDRAM
- Internal DUART

Note that support for Mars and Elysium MPC8540 boards as well as the Instruction Set Simulator has stopped at DINK32 version 13.2.

1.4 Memory Map for DINK32 Version 13.2

The memory model for DINK32 is shown in Table 1-1. The exception vectors and exception code are located within address offsets 0x0000–0x2100. The DINK32 code through 0x7FFFF is copied from the EPROM to RAM so that the data structures can be modified at run time. For example, the data structures for the chip registers need to be modified when the ‘register modify’ command is executed.

The EPROM must be located at address 0xFFF00000 because this is the beginning of the exception address space at system reset. The RAM must be located at address 0x00000000, because this is the low-memory exception address space, where the DINK32 code will be copied. Available user memory space begins at address 0x100000 and ends at the RAM’s upper boundary; address space below 0x100000 is reserved for DINK32.

DINK32 sets the stack pointer, GPR1, to one of the addresses shown for the indicated processor and to maintain isolation between user’s and DINK’s stack.

Table 1-1. DINK Address Map

Address	Description	Notes
0xFFFF_FFFF	End of ROM space	
0xFFF7_FFFF	End of DINK32 code	
0xFFF0_0100	Start of DINK vectors (reset vector)	
0xFFF0_0000	Start of DINK image	
0xFFEF_FFFF	End of replicated flash images	1
0xFF80_0000	Start of replicated flash images	1
0xFF7F_0000	End of secondary flash images	
0xFF00_0000	Start of secondary flash images	
0xFEFF_FFFF	End of PCI aperture	
0x8000_0000	Start of PCI aperture	
0x7FFF_FFFF	End of user space	
0x0010_0000	Start of user space	2
0x000F_FFFF	Bottom of CPU0 DINK stack	3
0x000E_0000	Top of CPU0 DINK stack	
0x000D_FFFF	Bottom of CPU1 DINK stack	
0x000C_0000	Top of CPU1 DINK stack	
0x000B_FFFF	Bottom of CPU1 user stack	

Table 1-1. DINK Address Map (continued)

Address	Description	Notes
0x000A_0000	Top of CPU1 user stack	
0x0009_FFFF	Bottom of CPU0 user stack	
0x0008_0000	Top of CPU0 user stack	
0x0007_FFFF	Top of '.data' section End of Excimer/Maximer ENV storage	4
0x0007_F000	Start of Excimer/Maximer ENV storage	4
0x0004_0000	Bottom of '.data' section (read-write data)	
0x0003_03FF	Top of '.rodata' section	
0x0002_FD00	Bottom of '.rodata' section (read-only data)	
0x0002_FFFF	End of '.text' section	
0x0000_3000	Start of '.text' section (code)	
0x0000_2FFF	End of exception table	
0x0000_0000	Start of exception table	

1 Due to incomplete address decoding, other images of the DINK program will appear at various addresses.

2 This is the new recommended user download address.

3 Note that stacks grow down, so 'top' and 'bottom' have opposite senses.

4 For Excimer/Maximer, part of the code space is used for NVRAM emulation; Excimer/Maximer builds of DINK are typically smaller and can accommodate this (no networking support, etc.)

1.5 Memory Map for DINK32 Version 13.3

The memory model for DINK32 version 13.3 varies with the platform that the image is compiled for (with the exception of HPC2 (Taiga) DINK32 image which is identical to 13.2 memory map seeTable 1-1).

The DINK32 and user code space in flash and RAM is as follows. The exception vectors and exception code are located within address offsets 0x0000–0x2100. The DINK32 code through 0xFFFFF (exception table and text sections) is copied from EPROM address of 0xFFE00000 to RAM so that the data structures can be modified at run time Data sections that reside at flash address of 0xFFF00000 to the end of the flash area are copied to low address of 0x100000. This allows bigger space (2MBytes) for DINK32 code to grow.

With version 13.3 the EPROM must be located at address 0xFFE00000. The beginning of the exception tables is located at 0xFFF00000. The RAM must be located at address 0x00000000, because this is the low-memory exception address space, where the DINK32 code will be copied. Available user memory space begins at address 0x200000 and ends at the RAM's upper boundary; address space below 0x200000 is reserved for DINK32.

DINK32 sets the stack pointer, GPR1, to one of the addresses shown for the indicated processor and to maintain isolation between user's and DINK's stack.

On DINK32 version 13.3 each platform has slightly different memory maps. For the memory maps of the DINK32 version 13.3 supported platforms see main.c.

1.6 eDINK Memory Map

Unlike processors that implement the classic PowerPC architecture, the e500 is based on the embedded PowerPC architecture (Book E). These processors use TLBs rather than BAT registers. For a detailed description of TLB entries and how to program them, refer to the *MPC8540 PowerQUICC™ III Integrated Host Processor Reference Manual* or the *MPC8560 PowerQUICC™ III Integrated Host Processor Reference Manual*.

At reset, a 4K memory space is predefined starting at 0xFFFF_F000 with supervisor read, write, and execute permission. This memory area is caching-inhibited. This startup code (in 'startup.S') sets up the processor, the TLBs (to enable more than 4K), the local access windows (LAWs) to access internal peripherals, and then corresponding embedded device setup.

Note that the CCSRBAR is relocated from the default 0xFF70_0000 to 0xFC00_0000.

Once initialization is complete and the system has been set up, the memory map shown in Table 1-2 is available.

Table 1-2. eDINK Memory Map

Start Address	End Address	Size (MB)	TLB	LAW	BR/OR	Description
FF00_0000	FFFF_FFFF	16	0	0	0	Flash array #1
FE00_0000	FEFF_FFFF	16			1	Flash array #2
FD10_0000	FDFF_FFFF	15			—	Unused
FD00_0000	FD0F_FFFF	1			2	8K NVRAM
FC10_0000	FCFF_FFFF	15		—	—	Unused
FC00_0000	FC0F_FFFF	1		—	—	CCSRBAR
F000_0000	FBFF_FFFF	192		—	—	Unused
E000_0000	EFFF_FFFF	256	3	—	—	Unused; test
C000_0000	DFFF_FFFF	—	—	—	—	Unused
A000_0000	BFFF_FFFF	512	6 & 7	3	—	RapidIO I/O
9000_0000	9FFF_FFFF	256	5	2	—	PCI IO space
8000_0000	8FFF_FFFF	256	4		—	PCI memory space
2000_0000	7FFF_FFFF	—	—	—	—	Unused
0000_0000	1FFF_FFFF	512	1 & 2	1	—	DDR SDRAM

Note that some address spaces require more than one TLB to be covered. For details, see Freescale Semiconductor Application Note AN2336, *Creating edink from DINK32 Code Using the e500 ISS*.

Chapter 2
Using DINK

DINK works with a wide variety of platforms. The only essential features needed to communicate with it are a serial cable connected to a terminal, and a terminal or a terminal emulator program on a computer.

This chapter covers the setup and use of DINK.

2.1 Communications Setup

The following sections describe the hardware and settings necessary to communicate with DINK.

2.1.1 Cables

The cable between the target system (Sandpoint, Excimer/Maximer, MVP, Arcadia, Elysium) and the host processor must be a “null-modem” cable—that is, a cable designed for connecting two computers together.

For the Macintosh, the following cable may be created:

Mac Pins	PC Pins
1	6
2	1,7,8
3	2
4	5
5	3
6	NC
7	NC
8	5

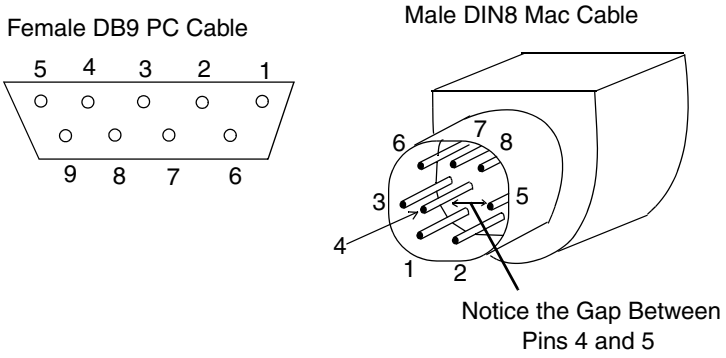

Figure 2-1. Serial Cable Wiring

2.1.2 Terminals

There are numerous terminal emulator programs available, in all cases the following settings should be specified:

- Baud rate—9,600
- Bits—8
- Parity—N
- Stop bits—1
- Flow control—None
- Terminal—VT100

Note that 9600 baud is slow but can be easily changed to default to other values, up to 115,200 in most cases.

2.1.2.1 Hyperterm on NT

In addition to the above, set these properties:

- Function—terminal keys checked
- Emulation—ANSI
- Backscroll buffer lines—500

Use this ASCII setup:

- ASCII sending
  - — Send line ends with line feeds—unchecked
  - — Echo typed character locally—unchecked
- Line delay—0
- Character delay—0
- ASCII receiving
  - — Wrap lines that exceed terminal width—checked
  - — All others unchecked

Use these settings:

- Baud —9600 (default)
- Data bits—8
- Parity—none
- Stop bits—1
- Flow control—none

2.1.2.2 Zterm on Mac

Under Settings, go to Connection and set only the following:

- Service name: SENDIT
- Data rate: 9600
- Data bits: 8
- Parity: none
- Stop bits: 1

Under Settings, go to Terminal and set only the following:

- Don't drop DTR on exit
- PC ANSI-BBS

Under Settings, go to Text Packing and set only the following:

- Delay between chars: 0
- Delay between lines: 1

Under Settings, go to Modem preferences and make sure there is nothing set in this window. All other settings should be the default.

To use Z-Term connect Excimer and power it on. Z-Term should automatically detect it and display the bootup output on the screen.

2.2 Starting DINK

Once the terminal program is running, turn on the power to the Sandpoint, Excimer/Maximer, MVP, or other platform. After a short pause, the DINK prompt should appear:

```
I/O system initialized...
Memory Enabled: [ 128MB at CL=2 ]
Caches Enabled: [ L1-ICache L1-DCache L2Cache(256K) ]
Register Inits: [ 32 GPRs, 32 FPRs, 255 SPRs, 32 VECs ]


                ##  ##           ##
                ##  ##           ##
                ##               ##
             #######  ##  #######   ##    ##
            ##    ##  ##  ##    ##  ##   ##
            ##    ##  ##  ##    ##  ######
            ##    ##  ##  ##    ##  ##   ##
             ######   ##  ##    ##  ##    ##


   Version : 13.2, GCC Build
  Released : April 30, 2003, Built on April 30, 2003 11:28:00
Written by : Freescale Semiconductor Inc., PowerPC Apps, Austin, Texas
    System : Sandpoint X3 with Valis MPC7455
 Processor : MPC7450 V2.3 @ 1250 MHz, Memory @ 133 MHz
    Memory : 128MB at 3/1/1/1

Copyright 1993-2005, by Freescale Inc.


DINK32[MPC7447A] {1} >>
```

2.3 Running eDINK on a Simulator

Besides hardware, eDINK runs on an e500 Core Instruction Set Simulator (ISS) or MPC8540 ISS.

2.3.1 Running eDINK on e500 Core ISS

1. Setup the following variables:
   a) export ISS_BIN_PATH=$MODEL_PATH/bin
   b) export LD_LIBRARY_PATH=$MODEL_PATH/simPortal/sim/lib:$LD_LIBRARY_PATH
2. cd to directory containing the eDINK executables
3. Run ISS
4. source sportal_file or source sportal_term

5. go

2.3.1.1 sportal_file

The sportal_file contains the following lines. The lines set up the sportal to read and write to/from files instead of a simulator terminal. Writing and reading to files is used for regression testing or running in batch mode. Read the ISS user's manual for details.

```
sportal open
sportal sim_stdin src/my_stdin
sportal sim_stdout my_stdout
sportal sim_stderr my_stderr
ld obj/edink.src
```

2.3.1.2 sportal_term

The sportal_term contains the following lines. The lines set up the sportal to read and write to a simulator terminal.

```
sportal open
sportal sim_stdall term
ld obj/edink.src
```

2.3.2 Running eDINK on MPC8540 ISS

1. Read the MPC8540 ISS for environment setup and other requirements
2. Load the eDINK image using the simulator's load command
3. Run the eDINK image using the simulator's run command

NOTE: The MPC8540 ISS doesn't support the sportal. However, the simulator must be run with the -s option to enable a serial terminal.

2.3.3 Loading Application Elf Binary to Simulated eDINK

The best way to load an application to eDINK is through the simulator. The same commands used to load eDINK into the simulator can be used to load an application elf binary. It is important that the application be linked at 0x1000000, because there is no option in the simulator for an offset load.

2.4 DINK Model

There are two sets of registers maintained in DINK32: one for DINK and one for the user. While the user is running DINK32, the DINK register set is active in the hardware. The user register set is maintained in a table. Register commands access this user table. When the user does a go or trace command, the DINK hardware registers are stored in the DINK table and the user registers are loaded from the table into the hardware. Then control is transferred to the user program. Whenever control is returned to DINK, via an error, a step, or user code completion, the registers are again swapped.

2.5 Interacting with DINK

When DINK has successfully initialized the target platform, it then starts up the DINK shell which can be used to execute many built-in tasks (modify/display memory, assemble programs) or to execute downloaded or flash-resident commands.

While using the DINK shell, terminal emulators that support VT102-sequences can be used to map the up, down, left, and right arrows. Up and down arrows recall previously executed commands from the history buffer (a list of each command entered). Right and left arrows, as well as the backspace and DEL keys can be used to move the cursor and edit the line.

Otherwise (if not an editing or VT102 sequence), DINK stores the characters into a buffer until a newline character is seen (0x0A—generally ENTER on the keyboard). Once a line is complete, the DINK shell looks for the presence of a semicolon (;), indicating that more than one command is present. If found, DINK separates the line into one or more commands and executes them separately. For example, the command ‘rd r; md 1000’ will display the register file and memory starting at location 0x1000.

For each command portion, DINK examines the line and alters it according to these rules:

- The character ‘$’ followed by an existing environment variable name is replaced with the value of the environment variable. For example, assuming ‘env RX=r3-r7,r15-r31’ has been defined, ‘rd $RX’ will show the indicated registers.
- The characters ‘@R’ followed by a number from 0 to 31 are replaced with the contents of the indicated GPR. For example, ‘md @R1’ shows memory at the current stack pointer.
- The quote characters (‘, “) prevent DINK from examining/altering the value between the starting quote and the ending quote. For example, “env RD=@R1” sets the variable RD to “@R1”, literally.

Once the command has been edited as stated previously, it is passed to the various programs for further interpretation. Entering ‘help’ shows a list of commands, and ‘help’ plus the command name (such as he rd) shows details on command options.

2.5.1 Running External Programs

In order to run user programs, the user must compile and link a program to start at 0x100000 or higher. Various PowerPC compilers are available to convert user programs to executable files.

Once the file is ready, it needs to be transferred to DINK’s memory space. DINK supports S-record or binary file (including ELF), and such files can be downloaded either through a serial port or by ethernet. Once in memory, the loaded code can be set into action with either a go or trace command. The demo directory and Appendix B, “Example Code,” have examples of compiling programs for this purpose.

2.5.2 Assigning Breakpoints

A breakpoint halts execution of user code at arbitrary locations, allowing inspection of the processor and memory state at a particular point. Breakpoints are set with the ‘bp’ command along with one or more desired stop-point addresses. When ‘go’ or ‘tr’ are used to run code, execution will be halted and DINK will restart when any breakpoint addresses are encountered.

Executing ‘go’ or ‘tr’ on top of a breakpoint deletes the breakpoint; this allows ease in stepping over a breakpoint, but caution is required for breakpoints in loops.

2.5.3 Executing Code

Code can begin executing with the go command. Code will execute until either an error occurs (such as an exception) or until the program exits with a return to DINK. Exceptions are reported by DINK, which regains control. DINK sets up a stack for the user and places the DINK return address in the link register. Thus, the user can return to DINK via this return address.

2.5.4 Displaying Registers

Registers are displayed by the register display, rd command. The modify register, rm command is used to modify registers. These commands do not affect the actual hardware registers at the time the command is entered. Rather they affect the user register set maintained by DINK. The go and trace commands actually copy the registers to the hardware before the user code begins executing.

2.5.5 Advanced Features

DINK contains an assembler command, ‘as’, and a disassembler command, ‘ds’.

Chapter 3
DINK32 Commands

This chapter describes the DINK32 user commands. The full command mnemonic is listed in the upper left-hand corner and the short command (abbreviation) is listed next in smaller type. All commands listed (except fw -e) are available to DINK32; those commands available to MDINK32 are marked as MDINK32 compatible.

Commands appear in boldface throughout this chapter.

NOTE

All addresses entered must be in hexadecimal but not preceded by '0x.'

Leading zeros will be added as needed.

3.1 Definitions

MDINK32 compatible	This command is also available in MDINK32. Where commands are different between MDINK32 and DINK32, the DINK32 format will be shown first.
Plus	Usually implies that the command form includes '+.' This allows the command to continue to the next stopping place appropriate for its functionality.
Range	Indicates a two-address form, and usually signifies an inclusive area of code or memory that will be operated on by the command.
Entire family	Refers to a family of registers. The general purpose registers are a family of thirty-two 32-bit registers, numbered 0 to 31. The floating-point registers are a family of thirty-two 64-bit registers, numbered 0 to 31. The AltiVec registers are a family of thirty-two 128-bit registers, numbered 0 to 31.The special purpose registers are not classified as a family due to their architectural design.
x	Typing 'x' will exit a command if DINK32 is in an interactive mode when a particular command form is used.

3.2 DINK Command List

The DINK command list is shown in Table 3-1.

Table 3-1. DINK Command List

Command	Section
.	Section 3.3, ". (Period)," on page 3-4
about	Section 3.4, "About," on page 3-5
assemble	Section 3.5, "Assemble," on page 3-6
background	Section 3.6, "Background," on page 3-8
benchmark	Section 3.7, "Benchmark," on page 3-8
bootm	Section 3.8, "Bootm," on page 3-11
breakpoint	Section 3.9, "Breakpoint," on page 3-12
cache	Section 3.10, "Cache," on page 3-14
config temp	Section 3.11, "CONFIGTEMP," on page 3-15
cpuinfo	Section 3.12, "CPUINFO," on page 3-17
crc	Section 3.13, "CRC," on page 3-18
device display	Section 3.14, "DEVDISP," on page 3-18
device modify	Section 3.15, "DEVMOD," on page 3-19
device test	Section 3.16, "DEVTEST," on page 3-21
disassemble	Section 3.17, "DISASSEM," on page 3-21
disk	Section 3.18, "Disk," on page 3-23
display temp	Section 3.19, "DISPTEMP," on page 3-24
do	Section 3.20, "Do," on page 3-25
download	Section 3.21, "Download," on page 3-25
downloadflash	Section 3.22, "DownloadFlash," on page 3-27
echo	Section 3.23, "Echo," on page 3-28
environment	Section 3.24, "Environment," on page 3-28
ethernet	Section 3.25, "ETH," on page 3-37
expr	Section 3.26, "EX," on page 3-38
filesys	Section 3.27, "FILESYS," on page 3-40
final setup	Section 3.28, "Final Assembly Init," on page 3-41
flash	Section 3.29, "Flash," on page 3-41
flash update	Section 3.31, "FUPDATE," on page 3-43
flash write	Section 3.32, "FW," on page 3-46
for	Section 3.30, "FOR," on page 3-42
go	Section 3.33, "Go," on page 3-46
gpic	Section 3.34, "GPIC," on page 3-48
help	Section 3.35, "Help," on page 3-51

Table 3-1. DINK Command List (continued)

Command	Section
history	Section 3.36, “History,” on page 3-51
hwm	Section 3.37, “Hardware Monitor,” on page 3-52
I^2C	Section 3.38, “I^2C,” on page 3-53
identify	Section 3.39, “Identify,” on page 3-53
if	Section 3.40, “If,” on page 3-54
log	Section 3.41, “Log,” on page 3-55
memory compare	Section 3.42, “Memory Compare,” on page 3-55
memory display	Section 3.43, “Memory Display,” on page 3-56
memory fill	Section 3.44, “Memory Fill,” on page 3-57
memory info	Section 3.45, “Memory Info,” on page 3-58
memory modify	Section 3.46, “Memory Modify,” on page 3-59
memory move	Section 3.47, “Memory Move,” on page 3-60
memory search	Section 3.48, “Memory Search,” on page 3-61
memory test	Section 3.49, “Memory Test,” on page 3-62
menu	Section 3.50, “Menu,” on page 3-64
northbridge display	Section 3.51, “Northbridge Display,” on page 3-65
northbridge modify	Section 3.52, “Northbridge Modify,” on page 3-66
net initialize	Section 3.53, “Net Info,” on page 3-68
opts	Section 3.54, “Opts,” on page 3-69
pci	Section 3.55, “PCI,” on page 3-69
PCI bus probe	Section 3.60, “PCI Probe,” on page 3-73
PCI config list	Section 3.56, “PCI Config List,” on page 3-70
PCI register display	Section 3.57, “PCI Display,” on page 3-71
PCI map	Section 3.58, “PCI Map,” on page 3-72
PCI register modify	Section 3.59, “PCI Modify,” on page 3-72
phy	Section 3.61, “PHY,” on page 3-74
ping	Section 3.62, “Ping,” on page 3-74
px	Section 3.63, “PX,” on page 3-75
register display	Section 3.64, “REGDISP,” on page 3-76
register modify	Section 3.65, “REGMOD,” on page 3-79
reset	Section 3.66, “Reset,” on page 3-81
set baud rate	Section 3.67, “SETBAUD,” on page 3-82
squit	Section 3.68, “SQUIT,” on page 3-83

Table 3-1. DINK Command List (continued)

Command	Section
switch cpu	Section 3.71, “Switch CPU,” on page 3-85
show SPRs	Section 3.69, “Show SPRs,” on page 3-83
stty	Section 3.70, “STTY,” on page 3-84
sx	Section 3.72, “SX,” on page 3-85
symbol table	Section 3.73, “SYMTAB,” on page 3-86
tau	Section 3.74, “TAU,” on page 3-88
test	Section 3.75, “Test,” on page 3-89
time	Section 3.76, “Time,” on page 3-89
transparent mode	Section 3.77, “Transparent,” on page 3-90
trace	Section 3.78, “Trace,” on page 3-91
upload	Section 3.79, “Upload,” on page 3-91
virtual host file	Section 3.80, “Virtual Host File,” on page 3-92
virtual memory	Section 3.81, “Virtual Memory File,” on page 3-93
virtual quit	Section 3.82, “VQUIT,” on page 3-93
virtual sim	Section 3.83, “Virtual Simulator,” on page 3-94
wait	Section 3.84, “Wait,” on page 3-95
watch	Section 3.85, “Watch,” on page 3-95
while	Section 3.86, “While,” on page 3-96
!	Section 3.87, “Shell Escape,” on page 3-97

3.3 . (Period)

Command	Repeat last command
Abbreviation	.
Syntax	.
Builds	All
Boards	All
Processors	All

Typing a period will repeat the last command entered. Unlike other commands, a ‘.’ at the start of a line is immediately handled and does not require carriage return/enter. In addition, ‘.’ does not change the last command (on the history stack). Therefore, when tracing a program, you can use ‘.’ to step to the next location.

Arguments:

None.

Examples:

```
DINK32 [MPC750] {1} >> trace 102100
0x00102104 stw r13, 0xfff8(r01)
DINK32 [MPC750] {2} >> trace +
0x00102108 add r03, r00, r01
DINK32 [MPC750] {2} >> .
0x0010210c mfspr r04, s0274
```

3.4 About

Command	about
Abbreviation	ab
Syntax	ab
Builds	All
Boards	All
Processors	All

The “about” command displays information on the current implementation of the DINK32 monitor.

Arguments:

None.

Examples:

```
DINK32 [MPC603r] {3} >>ab

                ##  ##            ##
                ##  ##            ##
                ##                ##
             #######  ##  #######   ##    ##
            ##    ##  ##  ##    ##  ##   ##
            ##    ##  ##  ##    ##  ######
            ##    ##  ##  ##    ##  ##   ##
             ######   ##  ##    ##  ##    ##


   Version : 13.2, Metaware Build
  Released : April 30, 2003 Built on Apr 30 2003 11:28:00
Written by : Freescale Semiconductor’s RISC Applications Group, Austin, TX
    System : Sandpoint X3 with Valis MPC7455
 Processor : MPC7450 V2.3 @ 1250 MHz, Memory @ 133 MHz
    Memory : 128MB at 3/1/1/1

Copyright 1993-2005, by Freescale Semiconductor Inc. 1993
```

Note that the actual banner varies depending on the platform, processor, speed, and other selections made.

3.5 Assemble

Command	assemble
Abbreviation	as
Syntax	as addr ['+' | ['-' addr2]
Builds	All except MDINK
Boards	All
Processors	All

The "as" command assembles user-supplied source code and stores the opcodes at the supplied addresses.

The assembler for the DINK32 system will disassemble the contents of memory at the current location and wait for user instructions. A single line of source may be supplied and assembled into a valid opcode, with the results stored at that memory location. A location can be left unmodified by typing <return> to pass over it.

Branch labels are recognized by the assembler as a word followed by a colon (:) at the address currently being displayed by the assembler. The assembler tracks the current branch labels and automatically calculates the address to be entered into future instructions. The *symtab/st* instruction is available for manipulating the branch table in DINK32. Branch labels within PowerPC assembly instructions will not be recognized by the assembler if the branch label has not yet been entered into the table. The user may display the branch table list with the st instruction.

The DINK32 assembler ignores any comments preceded by a '#' and any ".org" and ".dc" commands. The assembler does recognize the .word and .long operations. The assembler also accepts simplified mnemonics; in general, immediate values, including condition register bit offsets, are assumed to be decimal unless preceded by 0b (binary), 0x (hexadecimal) or 0d (decimal). Floating-point and general-purpose registers are recognized just like previous versions of DINK32 where the register number may be preceded by an "r" (general-purpose), an "f" (floating-point), or a "v" (vector—AltiVec) but these preceding letters are not required. Simplified branch mnemonics involving the condition registers may have the condition register number preceded by "cr" but is not necessary. The assembler always expects a "cr" field for compare and branch instructions where, according to the architecture, cr0 is implied if a "cr" field is not given. DINK32 does implement the implied cr0 functionality of the simplified mnemonics.

Simple mnemonics that use the form "4*cr<n>+<cc>" must enter this form with no spaces. n = condition reg, cc = condition.

Comments can be specified with the '//' or the '/*... */' constructs. The rest of the line is ignored following a comment specifier.

Arguments:

addr	Address to begin assembly
addr2	Address to stop assembly

NOTE: 'as' continues to assemble even if '+' is not specified; enter 'ESC' or 'x' to stop assembly.

Examples:

```
DINK32[MPC7455] {1} >>as 100000+
  00100000  FFFFFFFF             fnmadd.  f31, f31, f31, f31    rotlw r4,r6,r8
  00100004  FFFFFFFF             fnmadd.  f31, f31, f31, f31    lfd f0,1000(r1)
  00100008  FFFFFFFF fnmadd.  f31, f31, f31, f31    rlwnm 0,13,23,1,0xa
  0010000C  BFFFFFFF             stmw  r31, -1(r31)     loop:
  0010000C  BFFFFFFF   loop:     stmw  r31, -1(r31)     ori r26,r2,0xfff
  00100010  FFFFFFFF             fnmadd.  f31, f31, f31, f31    lfd f0,3000(r1)
  00100014  FFFFFFFF             fnmadd.  f31, f31, f31, f31    cmpw cr3,r26,r0
  00100018  FFFFFFFF             fnmadd.  f31, f31, f31, f31    bne loop
  0010001C  FFFFFFFF             fnmadd.  f31, f31, f31, f31    x

DINK32[MPC7455] {2} >>as 110000+
  00110000  FFFFFFFF             fnmadd.  f31, f31, f31, f31    .word 12345678
  00110004  FFFFBFFF             fnmadd.  f31, f31, f31, f23    .word 0x34
  00110008  FFFFFFFF             fnmadd.  f31, f31, f31, f31    .long 0xabcde08
  0011000C  BFFFFFFF             stmw  r31, -1(r31)     loop1: subf. 1,2,3
  00110010  FFFFFFFF             fnmadd.  f31, f31, f31, f31    bnelr
  00110014  FFFFFFFF             fnmadd.  f31, f31, f31, f31    b loop1
  00110018  FFFFFEFF             fnmadd.  f31, f31, f27, f31    x

DINK32[MPC7455] {7} >>st
Current list of DINK branch labels:
         KEYBOARD:    0x0
          getchar:    0x111b0
          putchar:    0x11188
        TBaseInit:    0xd000
   TBaseReadLower:    0xd018
   TBaseReadUpper:    0xd024
     CacheInhibit:    0xce94
    InvEnL1Dcache:    0xc9d0
      DisL1Dcache:    0xcd9c
    InvEnL1Icache:    0xc9c4
      DisL1Icache:    0xcdc4
        dink_loop:    0x367ac
      dink_printf:    0x2ff10

Current list of USER branch labels:
             loop:    0x10000c
            loop1:    0x11000c
DINK32[MPC7455] {8} >>
DINK32[MPC7455] {10} >>as 120000
  00120000  FFFFFFFF             fnmadd.  f31, f31, f31, f31    vmuleuh v3,v4,v5
  00120004  FFFFBFFF             fnmadd.  f31, f31, f31, f23    vmaxsw v4,v5,v6
00120008  FFFFFFFF fnmadd.  f31, f31, f31, f31    vspltisw v6,0x2000
  0012000C  BFFFFFFF             stmw  r31, -1(r31)     x
```

3.6 Background

Command	background
Abbreviation	bg
Syntax	{<address>	‘rsweep’	‘wsweep’ l’blink’	‘halt’}
Builds	DINK
Boards	MVP
Processors	MPC745X

The “bg” command causes the idling processor on the MVP platform to begin execution of a dedicated task, or halts it.

Arguments:

address	Address of code for idling CPU to execute
‘rsweep’	Pre-defined task which reads 16MB starting at 0x0000_0000
‘wsweep’	Pre-defined task which writes 16MB starting at 0x0100_0000
‘blink’	Pre-defined task which blinks the MVP LEDs
‘halt’	Halts pre-defined task, or any user-task which monitors the global variable ‘halt_sema’

If no argument is entered, the current run status is shown.

Examples:

```
DINK32 [MPC7455 #0] {2} >> bg blink
  CPU1 background task = 0x0007E440
DINK32 [MPC7455 #0] {3} >> bg
  CPU0: DINK 13.0
  CPU1: running task @ 0x0007E440
DINK32 [MPC7455 #0] {4} >> bg halt
  CPU0: DINK 13.0
  CPU1: halting
DINK32 [MPC7455 #0] {5} >> bg
  CPU0: DINK 13.0
  CPU1: idle
```

3.7 Benchmark

Command	benchmark
Abbreviation	bm
Syntax	bn address
Builds	DINK
Boards	All
Processors	All

The “benchmark” command will display how many timebase ticks it takes to execute the user code at ‘address’. This command will take multiple passes at the benchmark with different processor

configurations. It will report the timebase ticks on each configuration. If the timebase register is reported to be ‘0x00000000 00000000’, check to see if it is followed by a ‘.’. If it is, then this means that the benchmark actually took less time than it took to measure how long it took to get to the beginning of the benchmark (which is used as a base run and subtracted out of the full run before being displayed on screen).

The result of the benchmark can be converted to seconds by the following formula. These results are time-base clicks, which, for the MPC7400 (*MPC7400 RISC Microprocessor User’s Manual,* p. 2-8), are one-fourth the bus rate, hence (benchmark tics * 4) * 1/50000000 at 100 Mhz for the MPC7400. Each processor has its own specification, which is shown in the appropriate reference manual (RM) or user’s manual (UM).

Thus, for the MPC7400, the time base increments once for every four bus cycles. The formula is: ((benchmark tics * 4) * 40 nanoseconds) for a 100-Mhz clock. Thus for this test, the results of the last configuration, with the Icache ON Dcache ON, we got 0x1b26e18c time base tics, which is 455,532,940 decimal time tics. Applying our formula, (455,532,940 * 4) * 0.00000004 is 72.88 seconds. This is reasonable for this test, because it gives a series of displays each separated by a wait loop.

This command requires that the user code (to be benchmarked) be terminated with an illegal opcode!

More discussion on “Bus Speed” for the Sandpoint. There are three clock speeds that determine the system performance:

- PCI Bus Speed
- System (Memory/CPU) Bus Speed
- Processor Core Speed

We usually refer to these as, say, 400/100/33 indicating a 400 MHz CPU with a 100 MHz system bus and 33 MHz PCI. The PCI bus should be running at 33 MHz; no other bus speed is supported on SP X2. If the board is misconfigured, it might be in 66 MHz—make sure the PCI66MHZ LED is OFF. It is unlikely the system would run at all if that were so.

The CPU bus and core speeds can be set to various multiples of the PCI clock, as shown in the configuration sheet in the bound manual shipped with the SPX2 system. DINK reports these when it boots up.

Note that the decrementer and timebase operate at 1/4th the “bus speed”. Typically, this is 100 MHz, so it ticks at a 25-MHz or 40-ns rate. The CPU has no concept of the PCI clock speed; it could be 1Hz. It has no concept of PCI, for that matter. Only the MPC107 produces the relationship between PCI clock and system bus clock.

Arguments:

address Address containing code to benchmark.

Examples:

Download a program, such as “demo.src”:

```
DINK32 [MPC603ev] >> dl -k -o 70000
  Download from Keyboard Port
  132 lines received.
  Download complete.
```

Locate a function to stop on in the file “xref.txt”, in this case use “001003e4 PackageInfo” and set it to illegal instruction, 0x0.

```
DINK32 [MPC603ev] >> mm 1002b8
     0x001002b8 : 0x7c0802a6 : ?  0
```

Try it out first and ensure illegal instruction is set at the correct place.

```
DINK32 [MPC603ev] >> go 100000

                                P R E S E N T I N G
                                      Freescale Semiconductor
                    E X C I M E R   R E F E R E N C E   P L A T F O R M
                         A   M I N I M A L   Freescale Semiconductor   S Y S T E M
...etc....

A Program exception 0x00000700 has occurred.
Current Instruction Pointer: 0x001002b8  WORD        0x00000000
```

Now, run the benchmark command, note that it runs it with Icache/Dcache in all possible combinations of off and on.

```
DINK32 [MPC603ev] >> bm 100000
Icache OFF Dcache OFF:
                                P R E S E N T I N G
                                      Freescale Semiconductor
...etc...

 Benchmark 0x00100000  ---> 0x001002b8: 0x00000000 407ffb0e
Icache  ON Dcache OFF:
                                           P R E S E N T I N G
                                      Freescale Semiconductor
...etc...
 Benchmark 0x00100000  ---> 0x001002b8: 0x00000000 1b26deb7
Icache OFF Dcache  ON:
                                P R E S E N T I N G
                                      Freescale Semiconductor
...etc...
 Benchmark 0x00100000  ---> 0x001002b8: 0x00000000 407ffcec
Icache  ON Dcache  ON:
                                P R E S E N T I N G
                                      Freescale Semiconductor
...etc...
 Benchmark 0x00100000  ---> 0x000902b8: 0x00000000 1b26e18c
DINK32 [MPC603ev] >>
```

3.8 Bootm

Command	bootm
Abbreviation	bo
Syntax	bootm <-a addr> [-b string -l addr -n -r addr -t n -v]
Builds	DINK
Boards	All. Tested on: <HPC2, HPCN>
Processors	All. Tested on: 7448, 8641D

This command allows the execution of the OS or user program that is in the uImage format (see NOTES below for how to create). The bootm command will boot a uImage which can be Linux or a user program built from a .bin file. The only required input is the address of the image. uImage kernel/program must be uncompressed. Recommended usage is to place the kernel at 0x01000000 and the ramdisk at 0x02000000. This can be done differently but read NOTES on guidelines.

Arguments:

-a addr	hex address that uImage kernel/program is located
-b string	boot arguments to pass to Linux
-l addr	location of boot info structure and the boot args passed to Linux
-n	no verify, disables verification of images
-r addr	ramdisk hex address that uImage ramdisk is located
-t n	time to wait in seconds before booting (useful with autoboot, see examples below for more)
-v	verbose output, same as enabling verbose through environment variable VERBOSE

Examples and notes:

IMAGE BUILDING:

To build a uImage kernel, if it is not done automatically through kernel compile, use the mkimage command as follows:

```
mkimage -A ppc -O linux -T kernel -C none -a 01000000-e 01000000 -n Linux-2.6.15 -d vmlinux.bin
uImage
```

To build a uImage ramdisk run a command as below. Note that the ramdisk can be compressed since Linux will do the decompression.

```
mkimage -A ppc -O linux -T ramdisk -C gzip -a 02000000-e 02000000 -n 'Linux Ramdisk Image' -d
ramdisk.img.gzip uRamdisk
```

To build a uImage user program:

```
mkimage -A ppc -O u-boot -T standalone -C none -a 01000000-e 01000000 -n Hello_World -d
userprog.bin uImage-userprog-uc
```

AUTOBOOT:

To autoboot, set the environment variable BOOT equal to the arguments desired to be passed to the bootm command. Any options passed to the bootm command can be put in the BOOT environment variable. Be sure to set the timer, otherwise you cannot get to DINK prompt. For example,

```
env BOOT="-a ff800000 -t 5"          #set BOOT environment variable
env –s                               #save environment variables
```

Then reset the board and it will autoboot based on the parameters setup.

IMAGE PLACEMENT GUIDELINES:

The following are guidelines on placing the kernel/user program and ramdisk in memory:

- Must be outside final execution of kernel, 0 to size_of (uncompressed kernel)
- Must be outside of where DINK runs, typically 0-0x00140000
- Must be outside of DINK stack

There are other cases but these are the major ones; watch the WARNING and ERROR messages for more information on the conflicts.

EXPLANATION OF KERNEL RELOCATION:

The load address in the uImage header specifies where the image is supposed to be moved. Typically this is set to 0, but since DINK runs in this space the load address is ignored if there is any conflict. Linux will relocate itself down to address 0 as part of its initialization.

NOTE

Booting Linux and user programs was tested on Taiga with MPC7448 and ArgoNavis with MPC8641D.

3.9 Breakpoint

Command	breakpoint
Abbreviation	bp, bkpt
Syntax	bp [address] | [-d val] | [-c] [-l]
Builds	DINK
Boards	All
Processors	All

The “bp” command allows the user to manage breakpoints. Breakpoints allow the user to run an application program and stop execution when code reaches the specified address. This command will set or delete only one breakpoint at a time, and must be repeated for each breakpoint.

Setting a breakpoint will not remove a breakpoint from an address if a breakpoint already exists there. Deleting a breakpoint from an invalid index has no effect. Breakpoints can be set or deleted one at a time and all are displayed during a breakpoint list. A maximum of 20 breakpoints can be set in the system.

Arguments:

-d val Delete the breakpoint matching the supplied val. On previous versions of DINK, val could be a value from 0..31 and this is still supported, and represents the order in the list of the breakpoint to remove.
On DINK 13.0, values > 31 are assumed to be addresses of the breakpoint to remove, and thus the order in the list is not important.

-l Lock the breakpoints. Normally, DINK removed the breakpoint after trapping on it; however, with this option the breakpoint will remain set.

-c Delete all breakpoints.

addressSet a breakpoint at the supplied address.

If no argument is supplied, the current breakpoints are listed.

Examples:

```
DINK32 [MPC8245] >> bkpt 260100
Breakpoint set at 0x00260100

DINK32 [MPC8245] >> bp
  BREAKPOINT  ADDRESS   LOCKED  COUNT
  1.          00260100  No          0

DINK32 [MPC8245] >> go 260000
Breakpoint Encountered:
0x00260100 cmp 0, 0, r03, r04

DINK32 [MPC8245] >> bp
  BREAKPOINT  ADDRESS   LOCKED  COUNT
    (none)
DINK32 [MPC8245] >> bp -l 260100
Breakpoint set at 0x00260100

DINK32 [MPC8245] >> go 260000
Breakpoint Encountered:
0x00260100 cmp 0, 0, r03, r04

DINK32 [MPC8245] >> bp
  BREAKPOINT  ADDRESS   LOCKED  COUNT
  1.          00260100  Yes         0

DINK32 [MPC8245] >> bp -d 260100
Breakpoint @0x00260100 deleted

DINK32 [MPC8245] >> bp
  BREAKPOINT  ADDRESS   LOCKED  COUNT
    (none)
```

3.10 Cache

Command	cache
Abbreviation	ca
Syntax	[-fx] [{[on|1] | {off|0]}] [L1] [L1I] [L1D] [L2] [L3]
Builds	DINK
Boards	All
Processors	All

The “cache” command allows direct manipulation of the cache settings used by DINK (while running DINK). In addition, these values can be exported to the user settings for manipulating values used while running user code.

Arguments:

-f Flush the indicated cache(s).

-x Export any changes made to user register settings (takes effect on ‘go’ or ‘tr’ commands).

on Sets a flag for enabling specified cache(s).

1 Same as ‘on’.

off Sets a flag for disabling specified cache(s).

0 Same as ‘off’.

L1 Enable or disable L1 I and D cache.

L1I Enable or disable L1 Icache only.

L1D Enable or disable L1 Dcache only.

L2 Enable or disable L2 cache.

L3 Enable or disable L3 cache.

If no argument is supplied, the current cache status is listed.

Examples:

```
DINK32 [MPC755] {1} >> ca
Caches Enabled: [ L1-ICache L1-DCache L2-Cache(512K) ]
DINK32 [MPC755] {2} >> ca off L2
Caches Enabled: [ L1-ICache L1-DCache ]
DINK32 [MPC755] {3} >> ca off L1
Caches Enabled: [ none ]
DINK32 [MPC755] {3} >> ca -f
DINK32 [MPC755] {4} >>
```

3.11 CONFIGTEMP

Command	configtemp
Abbreviation	ct
Syntax	ct [-ade] [[-o therm] [-t therm] [-s hyst] [-x xlim] [-l loclim] [-c rate] [-r alert] [-g config]]
Builds	DINK
Boards	HPC II (Taiga)
Processors	MPC7447A, MPC7448

This command allows the user to fully configure the ADT7461 temperature monitor from the command line. It can also disable/enable the device, which in turn disables/enables the entire thermal monitoring system. Typing “ct” with no arguments will display the current register settings of the ADT7461, which are also displayed after every “ct” command.

NOTE

The flags -a, -d, and -e can only be used separately from all other flags/options and must be the first flags in the command line. For example:

ct -a (CORRECT)

ct -d (CORRECT

ct -e (CORRECT)

ct -a -d [or any other flag] (WRONG)

Arguments:

-a	Auto configure Thermal Sensor to power-on default settings.
-d	Disable the ATMS and put Thermal Sensor in standby mode.
-e	Enable the ATMS and take Thermal Sensor out of standby mode.
-o therm	Set the Local THERM Limit.
-t therm	Set the External THERM Limit.
-s hyst	Set the THERM Hysteresis Limit.
-x xlim	Set the External Temperature High Limit.
-l loclim	Set the Local Temperature High Limit.
-c rate	Set the Conversion Rate of the Thermal Sensor.
-r alert	Set the Consecutive ALERT count of the Thermal Sensor.
-g config	Modify Configuration Register bits in the Thermal Sensor.

The “ct” command with no parameters will display the current register settings of the ADT7461.

therm	THERM limit value given in hex from 0x00-0xFF. Default (0x55) (85degC).
hyst	THERM Hysteresis value given in hex from 0x00-0xFF. Default (0x0A) (10degC).

xlim External Temperature High Limit value given in hex from 0x00-0xFF. Default (0x55) (85degC).

loclim Local Temperature High Limit value given in hex from 0x00-0xFF. Default (0x55) (85degC).

rate Conversion rate value given in hex. Default (0x08) (16c/s).

c/s stands for conversions per second

0x00 -> 0.0625 c/s0x06 -> 4.0 c/s

0x01 -> 0.125 c/s0x07 -> 8.0 c/s

0x02 -> 0.25 c/s0x08 -> 16.0 c/s

0x03 -> 0.5 c/s0x09 -> 32.0 c/s

0x04 -> 1.0 c/s0x0A -> 64.0 c/s

0x05 -> 2.0 c/s

alert Determines how many out-of-limit measurements must occur before an ALERT is generated.

0x00 -> 10x03 -> 3

0x01 -> 20x07 -> 4

config 8-bit hex value from 0x00-0xFF. Default (0x00).

See ADT7461 data sheet for an in-depth description of these bits.

BitNameFunction

7MASK10=ALERT Enabled, 1=ALERT Masked

6RUN/STOP0=Run, 1=Standby

5ALERT/THERM20=ALERT, 1=THERM2

4/3Reserved

2Temp Range0=(0-127)degC, 1=Extended Range

1/0Reserved

Examples:

The following command will change the external temperature high limit register in the ADT7461 to 0x20 (32degC).

```
DINK32 [MPC7448] >> ct -x 0x20
```

The following command will change the conversion rate register in the ADT7461 to 0x03 (3).

```
DINK32 [MPC7448] >> ct -c 0x03
```

While not shown here, after each “ct” command the register values of the ADT7461 will be displayed in table form.

3.12 CPUINFO

Command	cpuinfo
Abbreviation	ci
Syntax	ci [pvr]
Builds	All
Boards	All
Processors	All

The “ci” command displays the internal database of CPU-specific features known and used by DINK. Unless a PVR value is supplied, the current CPU is described.

Arguments:

pvr 32-bit PVR value of the processor to display.

Examples:

```
DINK32[MPC7455] {1} >>ci
  Processor Name   : MPC7455 ("Apollo")
  Processor Class  : G4+
  Exception handler: PPC7455
  L1I+D Cache Size :   32K
  L2 Cache Size    :  256K
  L3 External Cache: 2048K
  TLBs             : 128
  BATs             : 8 IBAT + 8 DBAT  (extendable: Yes)
  Floating Point   : Yes
  AltiVec          : Yes (Classic)
  PCI Embedded     : No
  RapidIO Ports    : 0
  Ethernet Ports   : 0
DINK32[MPC7455] {5} >>ci 0x80200000
  Processor Name   : MPC8540 ("Draco")
  Processor Class  : G5
  Exception handler: PPCDRACO
  L1I+D Cache Size :   32K
  L2 Cache Size    :  256K
  TLBs             : 64
  BATs             : 0 IBAT + 0 DBAT  (extendable: No)
  Floating Point   : No
  AltiVec          : Yes (Book E)
  PCI Embedded     : Yes
  RapidIO Ports    : 1
  Ethernet Ports   : 3
```

3.13 CRC

Command	crc
Abbreviation	crc
Syntax	crc [-d dev] [-l] addr1-addr2
Builds	All
Boards	Excimer, MVP, Sandpoint, Yellowknife
Processors	All

The “crc” command calculates the 16-bit cyclic-redundancy-code (CRC) of the specified memory range, using the CCITT-16 algorithm. All memory between the specified addresses is accessed using byte accesses and the resulting CRC is displayed.

Arguments:

-d dev The name of the device to use, instead of system memory. See the “devdisp” command for more details.

-l Loop; the block of memory is repeatedly checked and the results displayed. This may be useful for monitoring a block of memory which may be modified, as the CRC value will change if any bits in the specified range of memory are altered.

addr1 The starting address to use.

addr2 The ending address.

Examples:

```
DINK32 [MPC603ev] >> crc 1000-2000
  CRC[00001000...00002000] = 6E48
```

3.14 DEVDISP

Command	devdisp
Abbreviation	dd
Syntax	dd [device [-bl-hl-w] [-rs] [-a devaddr] addr1[‘-’ addr2]
Builds	DINK
Boards	All
Processors	All

The “devdisp” command displays the contents of memory or device registers in a manner similar to that of the memory display command. While it displays memory as normal, it also handles the one or two levels of indirection necessary to access memory or registers within IO devices.

Arguments:

device The name of the device. If not entered, a list of known devices is shown.

-r Displays data byte-reversed.

-s Summarizes data; if the current line of data is the same as the previous line, only "..." is printed.

-b, -h, -w Sets size of device accesses. If not specified, the default size is bytes for devices (as shown in the device listing).

-a addr Sets the subaddress for a device. Used for I^2C EEPROM access, which is typically 0x50-0x57.

addr1 The starting address to display.

addr2 The optional ending address.

The "dd" command with no parameters will display a list of all the known devices.

Examples:

```
DINK32 [MPC750] {1} >> dd
Device    Start     End       Sizes  Device Addr
========  ========  ========  =====  ===========
mem       00000000  FFFFFFFF  [BHW]
nvram     00000000  00000FFF  [B]
I2C       00000000  0000007F  [B]      50
rtc       00000080  000000FF  [B]
apc       00000040  00000048  [B]
DINK32 [MPC750] {2} >> dd nvram 40-4F
0x0040 0D 0A 1A 00 00 08 09 0A 0D 1A 41 61 7F FF 80 00  .. ...... .Aa
dd> x
DINK32 [MPC750] {3} >> dd i2c -a 57 0-ff
...
```

3.15 DEVMOD

Command	devmod
Abbreviation	dm
Syntax	dm [device [-bl-hl-w] [-n] [-a devaddr] addr1['-' addr2]
Builds	DINK
Boards	All
Processors	All

The "devmod" command displays and modifies the contents of memory or device registers in a manner similar to that of the memory modify command. While it displays memory as normal, it also handles the one or two levels of indirection necessary to access memory or registers within IO devices. It also handles modifying these values.

Arguments:

device The name of the device. If not entered, a list of known devices is shown.

-r Displays data byte-reversed.

-n Does not read and display any data, only "--" is shown. Used to write to write-only registers that should not be read.

-b, -h, -w	Sets size of device accesses. If not specified, the default size is bytes for devices (as shown in the device listing).
-a addr	Sets the subaddress for a device. Used for I^2C EEPROM access, which is typically 0x50-0x57.
addr1	The starting address to display.
addr2	The optional ending address.

The “dm” command with no parameters will display a list of all the known devices.

The “dm” command will display the contents of a particular location in the requested format, then wait for user commands, which may be any of the following:

value	Write the hexadecimal value to the current location. The value entered may be truncated, if necessary, to the memory size.
‘abcd’	Write the character literal value(s) “abcd” to the current location. The leading single quote is required.
return	Go to the next location using the current selected direction (defaults to forward)
v	Set the direction to forward. Upon return, the current location advances by +1 byte, halfword or word.
^	Set the direction to reverse. Upon return, the current location advances by -1 byte, halfword or word.
=	Set the direction to 0. “dm” will keep examining and modifying the same location until ‘v’ or ‘^’ is entered.
?	Help
ESC	Stop
x	Stop

The direction commands are “sticky”—once set, “dm” will continue in that direction until a new direction is entered. The VT100 cursor keys may be used to move up and down.

Examples:

```
DINK32 [MPC750] >> dm nvram 40
      0x0040 : 14 ? <enter>      -- skip
      0x0041 : 3E ? 47            -- new value
      0x0042 : 27 ? ^             -- go back
      0x0041 : 47 ? 48            -- right value
      0x0040 : 14 ? v             -- go forward
      0x0041 : 48 ? =<enter>
      0x0041 : 48 ? <enter>
      0x0041 : 48 ? <enter>
      0x0041 : 4A ? <enter>       -- erratic bit?
```

3.16 DEVTEST

Command	devtest
Abbreviation	dev
Syntax	dev epic [command] dev [+] [-r] i2c <addr> <-n> [<timeout>] dev [+] -w i2c <addr> <-n> <str> [<timeout>] dev [+] DMA [<type>] <src> <dest> [<chn>] [<n>] dev i2o <mode> [<bit>]
Builds	DINK
Boards	All
Processors	All

The “devtest” command accesses several MPC824x/MPC107-specific embedded utility devices. “dev” is a front-end to several device test/control routines, each with different syntax and capabilities.

Arguments:

For details on specific sub-device controls and arguments, enter the command: “he dev [epic | i2c | i2o | dma]”.

Examples:

Perform a given I/O test on the MPC8245/MPC8241/MPC8240.

```
DINK32 [MPC8240] >> devtest -r i2c

0x40:  FE FE FE FE 47 4A 4E 4F FE FE FE FE 47 4A 4E 4F   ....GJMN....GJMN
```

3.17 DISASSEM

Command	disassem
Abbreviation	ds
Syntax	ds start_address [+ | {- end_address}]
Builds	DINK
Boards	All
Processors	All

The “disassem” command displays the contents of memory at the given address(es). The contents are shown in hexadecimal opcode format as well as in PowerPC assembly instruction format.

Arguments:

addr	Address to begin disassembly.
addr2	Address to stop disassembly

If the “plus” argument is used, the command will display multiple lines of disassembled code beginning at the starting address. DINK will pause after 22 lines, and continue when return is pressed, or cancel if the user types “x”.

If the “range” form is used, the command will continue reading and disassembling for each inclusive address in the range specified.

Branch labels entered during an assemble session are displayed during disassembly. In order for branch labels to be calculated correctly, branch labels must be entered before instructions refer to that label.

Simplified mnemonics are disassembled both as a simplified mnemonic and as a comment in the full assembly form.

Examples:

DIND32 examples:

```
DINK32[MPC7455] {3} >>ds 100000
  00100000  5CC4403E             rotlw r4,r6,r8 //rlwnm r4, r6, 8, 0, 31
DINK32[MPC7455] {4} >>ds 100000-100018
  00100000  5CC4403E             rotlw r4,r6,r8 //rlwnm r4, r6, 8, 0, 31
  00100004  C80103E8             lfd  f0, 1000(r1)
  00100008  5DA0B854             rlwnm r0, r13, 23, 1, 10
  0010000C  63420FFF   loop:     ori  r26, r2, 4095 //(0xfff)
  00100010  C8010BB8             lfd  f0, 3000(r1)
  00100014  7D9A0000             cmp   cr3, 0, r26, r0
  00100018  4082FFF4             bne  0,loop  //bc  4, 2, 1048588  //(0x10000c)
(loop)
DINK32[MPC7455] {5} >>ds 100000-100004
  00100000  5CC4403E             rotlw r4,r6,r8 //rlwnm r4, r6, 8, 0, 31
  00100004  C80103E8             lfd  f0, 1000(r1)
DINK32[MPC7455] {6} >>ds 110000+
  00110000  00BC614E             .word  00bc614e
  00110004  00000034             .word  00000034
  00110008  ABCDEF08             lha  r30, -4344(r13)
  0011000C  7C221851   loop1:    subf.  r1, r2, r3
  00110010  4C820020             bnelr  0       //bclr 4, 2
  00110014  4BFFFFF8             b 1114124  //(0x11000c) (loop1)
  00110018  FFFFFEFF             fnmadd.  f31, f31, f27, f31
  0011001C  FFFFFFFF             fnmadd.  f31, f31, f31, f31
00110020  FFFF7FFF            fnmadd.  f31, f31, f31, f15
  00110024  FFFFFFFF             fnmadd.  f31, f31, f31, f31
  00110028  BFFFFFFF             stmw  r31, -1(r31)
  0011002C  FBFFFFEF             .word  fbfffef
  00110030  FFFFFFFF             fnmadd.  f31, f31, f31, f31
  00110034  FFFFFF9F             fctiwz.  f31, f31
  00110038  FFFFFEFF             fnmadd.  f31, f31, f27, f31
  0011003C  EFFFEFFF             fnmadds.  f31, f31, f31, f29
  00110040  FFFFFFFF             fnmadd.  f31, f31, f31, f31
  00110044  DFFFFFFF             stfdu  f31, -1(r31)
  00110048  FFFBFFFF             fnmadd.  f31, f27, f31, f31
  0011004C  FFFFFFFF             fnmadd.  f31, f31, f31, f31
  00110050  FFFFFFFF             fnmadd.  f31, f31, f31, f31
  00110054  7FFFBFFF             .word  7fffbfff
DINK32[MPC7455] {7} >>st
Current list of DINK branch labels:
         KEYBOARD:    0x0
```

```
          getchar:    0x111b0
          putchar:    0x11188
        TBaseInit:    0xd000
   TBaseReadLower:    0xd018
   TBaseReadUpper:    0xd024
     CacheInhibit:    0xce94
InvEnL1Dcache:    0xc9d0
      DisL1Dcache:    0xcd9c
    InvEnL1Icache:    0xc9c4
      DisL1Icache:    0xcdc4
        dink_loop:    0x367ac
      dink_printf:    0x2ff10

Current list of USER branch labels:
             loop:    0x10000c
            loop1:    0x11000c
DINK32[MPC7455] {8} >>
DINK32[MPC7455] {11} >>ds 120000-12000c
  00120000  10642A48             vmuleuh v3, v4, v5
  00120004  10853182             vmaxsw v4, v5, v6
  00120008  10C0038C             vspltisw v6, 0 //(0x0)
  0012000C  BFFFFFFF             stmw  r31, -1(r31)
DINK32[MPC7455] {12} >>
```

3.18 Disk

Command	disk
Abbreviation	di
Syntax	di [-iqx][-d n][-r lba][-w lba][-a adr][-l len][-t lba]
Builds	DINK
Boards	
Processors	

The “di” command allows control of the IDE interface present in the Winbond 53C586, the VIA 82C686, or an external IDE controller in a PCI slot. It performs basic IDE initialization and track-level access (that is, the disk is not used in any particular filesystem format). Instead, LBA (logical block number) addressing is used, where LBA is a number from 0 to a large value, each LBA representing 512 bytes of data.

The DI command, and other commands which use DI, use the following association between drives and external hardware:

Table 3-2. IDE Drive Letter Association

Drive	Controller	Master/Slave
0 “A:”	Primary	Master
1 “B:”	Primary	Slave
2 “C:”	Secondary	Master
3 “D:”	Secondary	Slave

NOTE: DI does not handle drives larger than 8GB; or rather it does, but anything over 8GB is unusable.

Arguments:

-i	Initialize the current disk, or disk "A:" if not specified. Initialization refers only to preparation to use the disk; the disk is not altered.
-q	Report information on the disk.
-d n	Select drive 0/1/2/3.
-x	Toggle fast block access (default is off). Faster if multiple blocks are read/written at once, but some disks are less reliable.
-t LBA	Do a read-write test of the indicated logical block address.
-r LBA	Read 1MB from the indicated LBA to memory starting at 0x10_0000.
-w LBA	Write 1MB to the indicated LBA from memory starting at 0x10_0000.
-a addr	Specify a different address for -r or -w (rather than 0x10_0000).
-l len	Specify a different length for -r or -w (rather than 1MB).

Examples:

```
DINK32 [MPC750] >> di -i
  A: 6400 MB, Maxtor 12345.
  D:  no drive detected.
  C:  no drive detected.
  D:  no drive detected.
```

3.19 DISPTEMP

Command	disptemp
Abbreviation	dt
Syntax	dt
Builds	DINK
Boards	HPC II (Taiga)
Processors	MPC7447A, MPC7448

This command will display the temperature read by the thermal monitor in degrees Celsius.

Examples:

```
DINK32 [MPC750] >> dt

        Register values (hex): 0x01: 30
                               0x10: 00
        Local Temperature (degrees C): 29
        Remote Temperature (degrees C):48.00

DINK32 [MPC750] >>
```

3.20 Do

Command	do
Abbreviation	do
Syntax	dl {-k} loop_count command [command ...]
Builds	All
Boards	All
Processors	All

The “do” command executes a DINK command a specified number of times.

Arguments:

-k Check the keyboard for keypresses between each iteration of a command. If not specified, the loop cannot be interrupted.

Examples:

```
DINK32[MPC7448] {17} >>do -k 5 echo hi
hi
hi
hi
hi
hi
```

3.21 Download

Command	download
Abbreviation	dl
Syntax	dl {-kl-hl} [-q][-fx]]-v][-b][-o offset] Sandpoint, MVP dl {-nw} Sandpoint, MVP
Builds	All
Boards	All
Processors	All

The “download” command captures data from S-record files taken from either the keyboard or host serial ports. The received data is stored in the memory locations specified by the input file.

NOTE

To cancel a download at the terminal, type in 'S9' and ENTER/return to terminate the S-record downloads. For binary downloads, wait for the timeout.

Arguments:

-nw Use network download via TFTP protocol.

-k Read S-record files from the keyboard. Use a smart terminal program to send files to DINK.

-h Read S-record files from the host port. The host must be ready to send S-record files upon reception of a newline. Use “tm” to set up the host first.

-q Select quiet mode, no indication of download progress is supplied.

-fx Enables XON/XOFF (software) flow control for downloading at higher speeds. Not usually needed.

-v Verifies a previous download, printing an error message for each difference found.

-o offset Adds a hexadecimal offset to the address of each of the S-Record lines, to relocate code.

-b Download using 8-bit binary data in lieu of S-records. The srec2bin program in METDINK2/demos/utilities/srec2bin can translate S-record files to contiguous binary streams. Binary downloads do not have address information, so the “-o offset” option is almost always needed.
Once a binary download is started, the user has thirty seconds to begin sending 8-bit binary values; if no data is seen, the download is cancelled.
After the initial thirty second start period has elapsed, a 3 second idle timeout is in effect.

-nw -b binary file (S-record is the default file format)

-nw -f absolute path and filename (required)

-nw -o offset adds a hex offset to the addresses in the S-Record file or overwrite the default load address for binary download.

The default download baud rate is 9600. To speed up the download rate, the baud rate may be changed; see Section 3.67, “SETBAUD.”

Examples:

```
DINK32[MPC7455] {17} >>dl -nw -b -fdemo.src
Filename = demo.src
File format =  Plain binary
Default Offset = 0x01000000
Successfully read 10159 bytes via TFTP at 1212303 bytes per second.

DINK32 [MPC750] >> dl -k
 147 lines received.
...
DINK32 [MPC750] >>
```

Use the following to do a binary upgrade of DINK.

```
DINK32 [MPC750]  >> sb -k 115200
DINK32 [MPC750]  >> dl -k -b -o 100000
  Binary protocol
  Offset:     0xffc00000
  147387 bytes transferred.
DINK32 [MPC750]  >> fu -l 100000 fff00000 7ff00
  ...
```

Information on S-Records can be found in this reference manual.

3.22 DownloadFlash

Command	downloadflash	
Abbreviation	dlf	
Syntax	dlf [-e] [-o offset]	Excimer/Maximer
Builds	All	
Boards	All	
Processors	All	

The “download” command captures data from S-record files and programs the data directly to flash. This is intended for the Excimer platform (which has minimal SRAM)

NOTE

To cancel a download at the terminal, type in 'S9' and ENTER/return to terminate the S-record downloads. For binary downloads, wait for the timeout.

Arguments:

-e Forces all of flash memory to be erased before the load.

-o addr Specifies the offset address, default is 0xfff00000.

The default download baud rate is 9600. To speed up the download rate, the baud rate may be changed; see Section 3.67, “SETBAUD.”

Examples:

Use the following example when upgrading DINK on Excimer with an S-record from our Freescale Semiconductor DINK32 website:

```
MDINK32_603e >> dl -fl -o ffc00000
Offset:     0xffc00000
Writing new data to flash.
Line: 50
```

The complete sequence for upgrading DINK on Excimer would be:

```
MDINK32_603e >> fw -e
Reboot the Excimer board
MDINK32_603e >> sb -k 57600
MDINK32_603e >> dl -fl -o ffc00000


MDINK32_603e >>
```

Information on S-Records can be found in this reference manual.

3.23 Echo

Command	echo
Abbreviation	ec
Syntax	echo args
Builds	All
Boards	All
Processors	All

The “echo” command simply prints all arguments on the terminal. It is used for scripts and printing the value of environment variables.

Arguments:

arg prints ‘arg’

Examples:

```
DINK32 [MPC7450] >> echo hello world
hello world
```

3.24 Environment

Command	env
Abbreviation	env
Syntax	env [-c][-d][-s][var[=value]]
Builds	All
Boards	Excimer, MVP, Sandpoint, Yellowknife
Processors	All

The ‘env’ command manages the environment variables, which is simply a list of names and corresponding definitions. DINK uses environment variables for the following purposes:

- System initialization
- Mode selections
- Command aliases

Environment variables are stored in non-volatile memory, the location and type of memory used varies with each system, but is typically battery-backed SRAM or flash.

- For YK/SP, NVRAM is used and preserved, and 4K bytes are available.
- For MVP, NVRAM is used and preserved, and 256 bytes are available.
- For Excimer and Maximer, the uppermost 1K of SRAM is used. The environment will be lost if the command “env -s” is not issued to write the environment to flash.

Arguments:

If one argument is given, the environment variable named is displayed, defined, or deleted, depending on the presence of the “-d” option and/or the “=” character in the name.

name Display the value of “name”, if any.

name=val Set environment variable “name” to the value “val”. Quotes are often required for variable definitions if special characters will be needed.

-c Initialize and erase all environment variables.

-d name Erases the named environment variable.

-s Write the Excimer/Maximer environment variables in SRAM to the only non-volatile storage available, flash. This command is for Excimer/Maximer only. The environment is automatically preset from flash if present; no command is needed.

If no argument is given, all current settings are displayed.

Table 3-3 lists the special environment names known to DINK, and their interpretations.

Table 3-3. Environment Names

Environment Name	When Used	Definition	Typical Values
AUTO	Startup	Command to be executed when DINK starts up	AUTO=vmem vm
BOOT	Startup	Hex value specifies boot address, otherwise bootm command (see below).	BOOT=0xFF000000,15,foo BOOT=LINUX
CL	Startup	Memory initialization code MAY use this value to override the value in the SPD EEPROMs. Specifying an overly-aggressive value may result in a system that will not startup.	CL=2 CL=3
DO	Always	Dink Global Options. Use the ‘op’ command to manage.	DO=2
DOTCOM	With ‘.’ command	Command to be executed when ‘.’ command is used	DOTCOM=”rd r3-r7”
IO	Startup	IO encoder values (see below)	IO=”COM1:57600” IO=”XIO:9600” IO=”PMC:38400@100”
L2CACHE	Startup	Hex value specifies L2CR value, L2CR encoder values (see below).	L2CACHE=0 L2CACHE=0x4D012000 L2CACHE=”2M pb2 /2.5”
L2ERRINTEN L2ERRDIS L2ERRCTL	Startup	L2 ECC control values; values are transferred directly to the respectively named SPRs.	Varies
L2PRIVATE	Startup	Hex value specifies L2PM value or encoded values (see below)	L2PRIVATE=”1M
L3CACHE	Startup	Hex value specifies L3CR value or encoded values (see below)	L3CACHE=”0x9F024000”
L3PRIVATE	Startup	Hex value specifies L3PM value or encoded values (see below)	L3PRIVATE=”1M
LINES	Always	Number of lines to display for “help”, “md”, “dd” and other multi-line print commands.	LINES=5 LINES=24 LINES=0 (0 means infinite)

Table 3-3. Environment Names (continued)

Environment Name	When Used	Definition	Typical Values
MALLOC	Startup	Allocates space for user programs to allocate memory. Arguments are starting address, comma, and size.	MALLOC=0x1000000,0x1000000
ONTR	After go/tr	Command to execute after GO/TR	ONTR=”rd r3-r7”
PROMPT	Always	Encoder values for prompt (see below)	PROMPT=”$p $d >> “
RDLAST	Always	Remembers last argument used with ‘rd’ command; used when ‘rd’ used without arguments.	RDLAST=”r0-r8”
RDMODE	Always	Value of ‘q’ inhibits long display, producing only hex values. Value of ‘e’ replaces long display with field name list.	RDMODE=e
TAUCAL	Always	When ‘tau’ command used	TAUCAL=0x80000000
VERBOSE	Always	If defined, commands may produce additional messages	VERBOSE=1
TOFFSET	Startup	Sets the thermal DC offset of the ADT7461 (see below)	TOFFSET=0, -4
TDISABLE	Startup	Disables the Automatic Thermal Monitoring System (ATMS)	TDISABLE=1,0
FANPWM	Startup	Allows control of the CPU fan speed (see below)	FANPWM=0xFFFF, 0x5555
TSHUTDOWN	ATMS interrupts	Sets the critical temperature shutdown temperature for the ATMS (see below)	TSHUTDOWN=100, 110
other	Always	Unrecognized command names that are found in the environment variable list may be used as command aliases	CMD=”rd r3; tr +”

If any key is pressed on startup (recommendation is Backspace), then settings listed as used on “Startup” are ignored. If invalid settings are entered, particularly L2/L3 cache settings, DINK may not be able to start. Pressing a key will cause them to be ignored, so the settings can be adjusted.

3.24.1 ENV—BOOT Encoding

The “BOOT” variable, if set, contains information that DINK can use to automatically start a user program using the “bootm” command. The value of this argument is passed to the “bootm” command as-is; see the bootm command for format and description.

3.24.2 ENV—IO Encoding

The “IO” variable, if set, is used to set information that DINK can use to automatically adjust or redirect IO processes. There are three major arguments.

The IO value has the following format:

device [‘:’ <baud> [‘@’ <busspeed>]][“USE=n”]

where:

device is the name of the device driver to use, one of ‘COM1’, ‘PMC’, or ‘XIO’. COM1 is the standard serial driver on the Sandpoint/Yellowknife/MVP/Excimer boards. PMC is the serial driver resident on MPMC8245 cards (internal MPC8245 UART). XIO

is the mini-VGA driver resident in DINK (Cirrus CLGD543x and S3 only, or true-VGA-compatible).

baud Set the baud rate to any serial compatible rate (see “setbaud”).

busspeed Set the system bus speed to the supplied value in MHz. DINK uses the serial port to measure the system bus speed, so when using the PMC driver, which is bus-speed-dependant, the system bus speed must be specified.

USE=nn Use the VGA card found on slot #nn, even if it does not appear to be a video card (for old cards without CLASS codes).

NOTE

The VGA video driver does not implement x86 emulation, so only VGA cards that power up in useful modes, or those with custom drivers in the “xio.c” file, can be used.

Examples:

```
DINK32 [MPC8240] env IO="COM1:57600"
DINK32 [MPC8245] env IO="PMC:9600@100"
DINK32 [MPC8245] env IO="VGA:"
DINK32 [MPC8245] env IO="VGA:USE=16"
```

3.24.3 ENV —L2CACHE Encoding

The “L2CACHE” environment variable specifies a value to be used in setting the L2CR register. The value may be a hexadecimal value, which is copied to the L2CR as-is.

Alternately, it may be a string of terms separated by a space or comma. The terms allowed depend on the CPU, as shown in Table 3-4.

Table 3-4. Term Strings

L2CACHE Term	MPC750, MPC755 MPC7400, MPC7410	MPC745x	Description
par	Y	Y	Parity support
256K	Y	Y	L2 cache size
512K	Y		
768K			
1M	Y		
2M	Y		Only 7400/7410 support 2 MB L2

Table 3-4. Term Strings (continued)

L2CACHE Term	MPC750, MPC755 MPC7400, MPC7410	MPC745x	Description
/1	Y		L2 clock divider
/1.5	Y		
/2	Y		
/2.5	Y		
/3	Y		
/3.5	Y		
/4	Y		
late	Y		L2 SRAM type
pb	Y		
pb2	Y		
pb3	Y		
flow	Y		
do	Y	Y	L2 restrictions
io	Y	Y	
wt	Y		
zz	Y		L2 sleep enable
rep		Y	L2 replacement algorithm

In addition, the values “0” and “OFF” are allowed, to prevent any other initialization attempts (see “identify”).

Examples:

```
DINK32 [MPC7450] env L2CACHE="0x80000000"
DINK32 [MPC7450] env L2CACHE="par"

DINK32 [MPC7400] env L2CACHE="0x3D014000"
DINK32 [MPC7400] env L2CACHE="2M /2.5 pb2 do"
```

3.24.4 ENV—L2PRIVATE Encoding

The “L2PRIVATE” environment variable specifies a value to be used in setting the L2PM (private memory) register. The value may be a hexadecimal value, which is copied to the L2PM as-is.

Examples:

The following will enable the L2 as 1MB of private memory, at address range 0x0D00_0000 - 0x0D0F_FFFF. The L2 interface will be set up as: L2 cache disabled, parity enabled, L2CLK = /3, PB2 SRAM.

```
DINK32 [MPC7410] env L2PRIVATE="0x0D000007"
```

```
DINK32 [MPC7410] env L2CACHE="0x4D010000"
```

3.24.5 ENV—L3CACHE Encoding

The “L32CACHE” environment variable specifies a value to be used in setting the L3CR register. The value may be a hexadecimal value, which is copied to the L3CR as-is.

Alternately, it may be a string of terms separated by a space or comma. The terms allowed depend on the CPU, as shown in Table 3-5.

Table 3-5. L3 Cache Descriptions

L3CACHE Term	Description
par	Parity support
1M	L3 cache size
2M	
/2	L3 clock divider
/2.5	
/3	
/3.5	
/4	
/5	
msug2	L3 SRAM type
late	
ddr	
pb2	
cksp:2	Clock sample points
cksp:3	
cksp:4	
cksp:5	
psp:0	L3 sample points
psp:1	
psp:2	
psp:3	
psp:4	
psp:5	
do	L2 options
io	
dmen	
cya	
dm1M	
dm2M	
rep	L2 replacement algorithm

In addition, the values “0” and “OFF” are allowed, to prevent any other initialization attempts (see “identify”).

Examples:

```
DINK32 [MPC7450] env L3CACHE="0x9F032000"
DINK32 [MPC7450] env L3CACHE="2M /3 msug2 cksp:3 psp:3 rep"
```

3.24.6 ENV—L3PRIVATE Encoding

The “L3PRIVATE” environment variable specifies a value to be used in setting the L3PM (private memory) register. The value may be a hexadecimal value, which is copied to the L3PM as-is.

Examples:

The following will enable the L3 as 1MB of private memory, at address range 0x0D00_0000 - 0x0D0F_FFFF. The L3 interface will be set up as: L3 cache disabled, parity enabled, L3CLK = /5, L3CKSP = 4 clocks, DDR SRAM, private memory enabled for 1MB.

```
DINK32 [MPC7450] env L3PRIVATE="0x00D00000"
DINK32 [MPC7450] env L3CACHE="0x4F820004"
```

3.24.7 ENV—ONTR Encoding

The ONTR environment command can be used to cause a command to be automatically executed after every GO or TR command. It is often set to a register display command to examine the results of registers after each TR command. For example,

```
DINK32 [MPC7450] env ONTR="rd r3-r10"
DINK32 [MPC7450] tr +
  R03=00000000            R04=00000000
  R05=00000000            R06=00000000
  R07=00000000            R08=00000000
  R09=00000000            R10=00000000
```

will display the contents of registers R3-R10 after each command.

3.24.8 ENV—PROMPT Encoding

The PROMPT environment command can be used to dynamically set the prompt shown after every command. The PROMPT environment variable is loaded and printed according to the terms found in the string, using the following rules:

Table 3-6. String Expansions

String	Expansion
$d	DINK name, either DINK32 or MDINK32
$p	Informal processor name, for example, MPC7400 is “MAX”
$P	Formal processor name, for example, MPC7400.
$T	Current time, for example, "12:34:56PM"
$H	Host/Agent selection.
$#	CPU Number (0..n).

Table 3-6. String Expansions (continued)

String	Expansion
$t	TAU temperature, for example, "26" if 26 deg. C or "26u" if not yet calibrated.
$!	History index.
$_	Carriage return and Linefeed.

All other characters are copied as-is.

Examples:

```
DINK32 [MPC750] env PROMPT="$d $p#$#@$T ? "
DINK32 Arthur#0@09:33:18 PM ? env -d PROMPT
```

3.24.9 ENV—TOFFSET Encoding

The TOFFSET environment variable is used to apply a DC offset to the Offset register in the ADT7461. If this variable is not set, then a default offset value will be used depending on the processor type. The value entered is a decimal value representing the degrees (in Celsius) to apply to the offset. The minimum is -128 and the maximum is +127.

Examples:

The following will set the DC offset to +34degC and -114degC respectively.

```
DINK32 [MPC7448] env TOFFSET=34
DINK32 [MPC7448] env TOFFSET=-114
```

3.24.10 ENV—FANPWM Encoding

Then FANPWM evironment variable controls the CPU fan speed by modifying the PWM registers in the TICK register set. The value must be a 16-bit hex value. Table 3-7 gives a list of valid hex values with the approximate fan speeds. If the ATMS is enabled, this environment variable will be used to set the initial fan speed when no thermal limits have been exceeded. If the ATMS is disabled, the fan speed reverts back to this environment variable setting. If the ATMS is enabled but this environment variable is not defined, then the fan speed is set to a default of 50%.

Table 3-7. PWM Register Values with Approximate Fan Speeds

Hex Value	Speed %
0x0000	0% (off)
0x0001	6%
0x0101	13%
0x8420	19%
0x1111	25%
0x2492	31%
0x4545	38%
0x5545	44%

Table 3-7. PWM Register Values with Approximate Fan Speeds (continued)

Hex Value	Speed %
0x5555	50%
0xD555	56%
0xD5D5	63%
0xDDD5	69%
0xDDDD	75%
0xFDDD	81%
0xFDFD	88%
0xFFFD	94%
0xFFFF	100% (on)

Examples:

The following will set the fan speed to 25% and 88% respectively.

```
DINK32 [MPC7448] env FANPWM=0x1111
DINK32 [MPC7448] env FANPWM=0xFDFD
```

3.24.11 ENV—TSHUTDOWN Encoding

The TSHUTDOWN environment variable is used by the ATMS as the critical temperature shut down limit. If the ATMS is enabled, upon reaching this limit, the system will power off. If this environment variable is not defined, then the default critical temperature is set to a default of 100°C. The value entered is a decimal value representing the degrees (in Celsius) that the critical temperature will be set to. The minimum is 0 and the maximum is +255.

Examples:

The following will set the critical temperature limit to +95°C and 120°C respectively.

```
DINK32 [MPC7448] env TSHUTDOWN=95
DINK32 [MPC7448] env TSHUTDOWN=120
```

3.25 ETH

Command	ethernet
Abbreviation	eth
Syntax	eth [-b addr][-s size][-l]
Builds	DINK
Boards	All
Processors	All

The “eth” command implements a simple ethernet traffic capture/report facility. As of Revision 13.2 of this document, only Marvell-based boards support the promiscuous setting necessary for this to be of interest.

Arguments:

-b addr	The starting address of the capture buffer. If not specified, the default is 0x100000.
-s size	The size of the capture buffer. If not specified, the default is 0x800000.
-l	Summarize the capture buffer.
-ll	Report all details of the capture buffer.

If no arguments are specified, the ethernet port (the one selected by DINK with the ‘ni’ command) is placed into promiscuous mode, and all packets captured are stored into memory. If memory fills, the pointer wraps around so only the last packets are kept.

Examples:

```
DINK32[MPC7447A] {1} >>eth
Entering Promiscuous mode:  ESC to exit, any key to report.
.............................................               (space pressed)
================================================
     RxPackets:     6844     RxErrors:     1234
     IPPackets:     1234   ARPPackets:     1234
    UDPPackets:       34   TCPPackets:     1234
   ICMPPackets:      633
------------------------------------------------
  Packet Store: 01000000-09000000
   Last Packet: 01083448
------------------------------------------------
  Receive Rate: 63.3 packets/s
================================================
.............................................               (ESC pressed)

DINK32[MPC7447A] {2} >>eth -l
PACKET  TYPE  DATA
======  ====  ====
ETH IP: ...
IPV4:   ....
```

3.26 EX

Command	expression
Abbreviation	ex
Syntax	ex expression
Builds	DINK
Boards	All
Processors	All

The “ex” command implements a simple expression evaluator.

Arguments:

expr The expression to evaluate

An expression consists of:

values Decimal, hexadecimal or binary values. Values follow usual DINK standards of hexadecimal by default, a leading ‘&’ for decimal, and a leading ‘%’ or ‘0b’ for binary.

intermingled with one or more operators:

‘+’ Addition
‘-’ Subtraction
‘*’ Multiplication
‘/’ Division
‘%’ Modulus
‘|’ Logical OR
‘&’ Logical AND
‘^’ Logical XOR
‘~’ Logical NOT
‘<<‘ Shift left
‘>>’ Shift right (unsigned)
‘==’ Equality
‘!=’ Inequality
‘<‘ Less than
‘<=’ Less than or equal
‘>’ Greater than
‘>=’ Greater than or equal

Note that no precedence is implemented; therefore, “1+2*3” returns “9”, rather than “7” as precedence would imply.

Examples:

```
VDINK32[MPC750] {1} >>ex 1+2*3
  =0x00000009
  =9 (unsigned)
  =9 (signed)
VDINK32[MPC750] {4} >>ex 4<<2
  =0x00000010
  =16 (unsigned)
  =16 (signed)
VDINK32[MPC750] {13} >>ex 1+2*3<<4|5==&149
  =0x00000001
  =1 (unsigned)
  =1 (signed)
```

3.27 FILESYS

Command	filesys
Abbreviation	fs
Syntax	-{r|x|w|i|l|l|d|a|A}[-n notes][-a addr] [filename] [addr[-addr]]
Builds	DINK
Boards	SANDPOINT, MVP
Processors	All

The “fs” command implements a miniature file system for use with the “ide” and “bootm” commands. The format is strictly primitive, being in essence a track-based directory catalog.

Arguments:

-r file	Read a filename into memory.
-w file	Write a filename to disk.
-d file	Delete the filename from the disk.
-x file	Execute a file (read it and boot from it).
-a addr	Set the execution address of a file (set with -w, used with -x).
-A file	Alter the execution address of file (requires -a).
-n note	Set the notes field (descriptive text).
-i n	Format a disk with space for <n> files.
-l file	List the file information. If no file, list the whole disk catalog. Add a second -l for more details.
addr	Starting address of memory to write
-addr	Ending address of memory to write

In all cases, ‘file’ may be 1-32 characters, and may be preceded by [A-D] and ‘:’ to select a different IDE disk drive.

Examples:

```
DINK32 [MPC7450] {1} >> fs -i 10
  Are you sure ? y
  pfs: root created for 10 files.
DINK32 [MPC7450] {2} >> fs -w loader 100000-1ffff0
DINK32 [MPC7450] {3} >> fs -l loader 100000-1ffff0
__FILENAME_____________________  __SIZE__ NOTES
loader                            ffff0
```

3.28 Final Assembly Init

Command	fi
Abbreviation	fi
Syntax	fi y
Builds	All
Boards	All
Processors	All

The “fi” command is used to initialize a board environment to that shipped from the factory. It may be used to restore a system when loading a new software environment, etc.

Arguments:

y extra argument required for safety.

Examples:

```
DINK32 [MPC7450] {1} >> fi y
<etc.>
```

3.29 Flash

Command	flash
Abbreviation	fl
Syntax	env [-dsi][-e][-sp][-su][-se][-n no]
Builds	DINK, MDINK(partial)
Boards	Excimer, Maximer
Processors	All

The “flash” command will perform a variety of flash memory operations on the Excimer and Maximer only.

NOTE: “flash” works only for Excimer/Maximer; for Yellowknife, Sandpoint and MVP see the “fu” command.

NOTE: All sector operations require the connection of a 12V power supply. See AMD Bulletin, NVD Flash, Sector Protection, available on the www.amd.com web site.

Arguments:

-dsi Display sector information

-e Erase all of flash

-sp Protect sector specified with “-n” (not supported in MDINK)

-su Unprotect specified with “-n” (not supported in MDINK)

-se Erase the specified with “-n” (not supported in MDINK)

-n value A sector number from 0-18. Required for some arguments.

Examples:

```
DINK32 [MPC603e] >> fl -se -n 6
Erasing sector 6

DINK32 [MPC603e] >> fl -dsi
Display Sector Information 0.7  Excimer Rev 2 and prior
Description      value
Manufacturer ID is 0x1, Device ID is 0x225b
Sector SA0   UNPROTECTED
Sector SA1   UNPROTECTED
Sector SA2   UNPROTECTED
Sector SA3   UNPROTECTED
Sector SA4   UNPROTECTED
Sector SA5   UNPROTECTED
Sector SA6   UNPROTECTED
Sector SA7   UNPROTECTED
Sector SA8   UNPROTECTED
Sector SA9   UNPROTECTED
Sector SA10  UNPROTECTED
Sector SA11  UNPROTECTED
Sector SA12  UNPROTECTED
Sector SA13  UNPROTECTED
Sector SA14  UNPROTECTED
Sector SA15  UNPROTECTED
Sector SA16  UNPROTECTED
Sector SA17  UNPROTECTED
Sector SA18  UNPROTECTED
```

3.30 FOR

Command	for
Abbreviation	for
Syntax	if [-k] ‘{‘ test ‘}’ command [command ...]
Builds	All
Boards	All
Processors	All

The “for” command executes a DINK command as long as the specified expression is non-zero.

Arguments:

-k Check the keyboard for keypresses between each iteration of a command. If not specified, the loop cannot be interrupted.

Examples:

```
DINK32[MPC7448] {17} >> for {i=0;$i<0x10000;i=$i+0x1000;} { md $i }
  00000000      00000000 00000000 00000000 00000000         ................
  00001000      00000000 00000000 00000000 00000000         ................
  00002000      00000000 00000000 00000000 00000000         ................
  00003000      00000000 00000000 00000000 00000000         ................
```

```
00004000      00000000 00000000 00000000 00000000          ................
00005000      00000000 00000000 00000000 00000000          ................
...
```

3.31 FUPDATE

Command	fupdate
Abbreviation	fu
Syntax	fu {-ll-h} [-qsxkeno] src_addr dest_addr length
Builds	DINK
Boards	Motherboards
Processors	All

The “fupdate” command writes blocks of memory to various flash devices on the motherboard platforms (Sandpoint, MVP, Freeserve, HPC1-3, etc.). “fupdate” handles both the PCI-based boot flash, as well as the local boot and auxiliary flash device. F, and local-bus ROMs on PMC cards. See the **fl** command for Excimer and Maximer.

Arguments:

-l	Program a local bus flash. NOTE: On PPMC cards the PROGMODE switch must be enabled.
-h	Program a host flash on the PCI bus (YK/SP systems).
-e	Erase flash; do not program.
-n	Do not check manufacturer ID's.
-o	Overwrite flash without erasing.
-x	Suppress progress display (quieter).
-q	Query: just report the flash detected and exit.
-s	Sector erasing. If specified, 'fu' attempts to use sector erasing when erasing is needed, as opposed to erasing the entire flash. This is faster and allows multiple images to be easily managed; however, as of Revision 13.2 of this document, it does not always work. It does not cause data loss, so it can still be retried without '-s'if it fails.
src_addr	Address of data to copy to flash
dest_addr	Address of data to store flash data; typically FFF00000 for PCI ROM and FF000000 for PMC ROM (when in PROGMODE).
length	Length of data to copy (in hex!)

Examples:

1. Store a program in the PCI-based ROM of a Sandpoint or Yellowknife (for example, a DINK upgrade).

```
DINK32 [MPC7410] >> dl -k -o 100000
Download from Keyboard Port
Offset Srecords by 0x00100000
      ...
Download Complete.

DINK32 [MPC7410] >> fu -h 100000 fff00000 7FF00
YK/SP PCI Flash Programmer
Are you sure? Y
Check flash type: AMD Am29F040
Erasing flash   : OK
Program flash   : OK
Verifying flash : OK
DINK32 [MPC7410] >>
```

2. Use the following example to copy DINK32 into a local-bus flash on a MPMC. Be sure that PROGMODE is set and that booting is from the PCI flash.

```
DINK32 [MPC7410] >> fu -l FFF00000 FF700000 7FF00
PPMC Local Flash Programmer
Are you sure? Y
Check flash type: AMD Am29LV800BB
Erasing flash   : OK
Program flash   : OK
Verifying flash : OK
DINK32 [MPC7410] >>
```

3. Add a Linux loader to the local flash.

```
DINK32 [MPC7410] >> dl -k -o 100000
Download from Keyboard Port
Offset Srecords by 0x00100000
      ...
Download Complete.

DINK32 [MPC7410] >> fu -l -o 100000 FF600000 7FF00
PPMC Local Flash Programmer
Are you sure? Y
Check flash type: AMD Am29LV800BB
No Erase flash  : OK
Program flash   : OK
Verifying flash : OK
DINK32 [MPC7410] >> env BOOT=”0xffe00000”
```

4. Sandpoint: Add a linux kernel/boot, built to use an NFS root file system, to the local flash and execute it.

```
(1)  Build the MVL kernel with whatever patches and settings as needed:
         make xconfig
         make dep
         make zImage

(2)  Convert the kernel plus bootwrapper to SRecords as follows:

        cd arch/ppc/boot/images
        zsrec -s 800000 zImage.sandpoint > vm.srec

(3)  Download the kernel to the Sandpoint board:

     [on the target]:
        sb -k 38400
        [switch monitor baud rate to 38400]
        dl -k

     [ on the host ]
        ascii-xfr -svn -l 10 vm.srec > /dev/ttyS0
     [on the target]:
        sb -k 9600
        [switch monitor baud rate to 9600]

     (NOTE that the kernel+wrapper gets downloaded to 0x800000)

(4)  Burn 1 MB of data from 0x800000 into the flash
     at 0xff000000 after erasing the flash contents:

        fu -l -o 800000 ff000000 100000

(5)  Test the kernel:

       go ff000000

     Kernel should get uncompressed and should run
```

```
(6)  Set up for automatic boot:

       [reset the board]

       env -c   [if necessary—clears out the environment variables]
       env LINUX="0xff000000,15,MVLinux 2.1 "
       env BOOT=LINUX
```

3.32 FW

Command	fw
Abbreviation	fw
Syntax	fw -e [-o flash_addr]
Builds	MDINK
Boards	Excimer
Processors	All

The “fw” command copies the contents of the entire 512K of RAM to flash memory, starting at flash address 0xFFF00000.

Arguments:

-e Erase the flash. Always required.

-o addr Specify the specific address to copy from RAM to ROM address.

Examples:

```
MDINK32_603e >> fw -e
Chip erase set.
Erasing entire flash memory...
Entering verify erase loop ...
Flash erased!!!
Done erasing flash memory.
Copying 512K ram to flash address fff00000...
```

3.33 Go

Command	go
Abbreviation	go
Syntax	go [‘+’	address] [args]
Builds	All
Boards	All
Processors	All

The “go” command allows the user to execute user code starting at specified address, until one of the following events occurs:

- A breakpoint is encountered.

- A BLR instruction returns control to DINK.
- An exception is caused.

NOTE

If a breakpoint is encountered, the user must clear the breakpoint in order for execution to continue.

In order to return from user code to DINK, the stack pointer (the value in R1 initially) must be preserved.

Arguments:

+	Run code at the address in the SRR0 (Machine Status Save/Restore) register (bits 0-29). This is useful for continuing where a breakpoint had previously stopped execution.
address	Specify the address to start executing code.
args	Arbitrary command line arguments passed to application code using the DINK transfer table “dink_transfer_table.args” (thus, the contents of memory are pointed to by [GPR21 + 72]).

Examples:

```
DINK32 [MPC750] >> ds 1181dc-1181f8
0x001181dc 0x3c600000 addis r03, r00, 0x0000
0x001181e0 0x60631234 ori r03, r03, 0x1234
0x001181e4 0x3c800000 addis r04, r00, 0x0000
0x001181e8 0x60845678 ori r04, r04, 0x5678
0x001181ec 0x7c632214 add r03, r03, r04
0x001181f0 0x38841234 addi r04, r04, 0x1234
0x001181f4 0x7c032000 cmp 0, 0, r03, r04
0x001181f8 0x4182ffe4 bc 0x0c, 0x02, 0xffe4

DINK32 [MPC750] >> bkpt 1181f4
breakpoint set at 0x001181f4

DINK32 [MPC750] >> go 1181dc
Breakpoint Encountered:
0x001181f4 cmp 0, 0, r03, r04

DINK32 [MPC750] >> go +
Breakpoint Encountered:
0x001181f4 cmp 0, 0, r03, r04

DINK32 [MPC750] >> go fe010000 root=/dev/fsx readonly

BootX bootloader V1.0.66
cmdline='root=/dev/fsx readonly'
...
```

3.34 GPIC

Command	gpic
Abbreviation	gpic
Syntax	gpic [command]
Builds	DINK
Boards	MVP
Processors	All

The “gpic” command is used on the MPV reference platform only. The gpic commands are used to initialize the Galileo Discovery programmable interrupt controller (GPIC). The initialization process sets the system up in a default configuration for the MVP platform. This default configuration initializes the discovery multi-purpose pins (MPP), general purpose port (GPP), and interrupt controller. The interrupt scheme on MVP utilizes the MPP on Discovery to map several of the GPP pins to serve as the external interrupt source inputs and interprocessor interrupts. The interrupt sources are the Super IO, PCI 0 Slots 3-0, PCI 1 Slots 3-0, and CPU 0 and CPU 1 cross-processor interrupt inputs.

The cross-processor interrupt scheme uses four GPP pins. GPP[5] is used as the CPU 0 cross-processor output signal and GPP[30] is the corresponding CPU 1 cross-processor input. GPP[31] is used as the CPU 1 cross-processor output signal and GPP[4] is the corresponding CPU 0 cross-processor input. Software can write to the GPP Value Register to assert and de-assert GPP[5] for CPU 0 or GPP[31] for CPU 1 to signal a cross-processor interrupt to the other CPU. In DINK32, CPU 1 is only setup to be interrupted by an assertion of the GPP[30] interrupt input, or a Timer 2 or 3 interrupt. An interrupt to CPU 1 is signalled via the Discovery INT1_ interrupt output. All other interrupts detected by Discovery are routed to CPU 0 via the Discovery CPU_INT_ output.

Arguments:

```
Usage: gpic [init|eie|eid|...]
    gpic ex     - "gpic" command example uses
    gpic        - Display various GPIC status info
    gpic init   - Initialize the GPIC unit to default mode
    gpic eie    - External interrupt enable
    gpic eid    - External interrupt disable
    gpic cpu0   - Assert cross-processor interrupt to CPU0
    gpic cpu1   - Assert cross-processor interrupt to CPU1
    gpic sl     - GPIC Sleep Demo
    gpic tmcnt [timer(0-7)]
                - Print Timer count value for all or specific timer
    gpic tmcnt timer[0-7] count
                - Write given count value to given timer
    gpic tmcon [timer(0-7)]
                - Print Timer Config Info for all or specific timer
    gpic tmcon timer(0-7) enable(0|1) [mode(0|1) trigger(0|1)]
                - Configure Timer Counters
                - Enable: enable (1) disable (0) counting
                - Mode: timer (1) counter (0)
                - Trigger: external (1) none (0)
                - Mode and Trigger parameters optional
                  (left unchanged)
                - If Trigger is given, Mode must also be given
```

```
    gpic tmmsk [timer(0-7)]
                - Print Timer Int Mask Info for all or specific timer
    gpic tmmsk timer[0-7] mask[0|1]
                - Enble/disable Timer interrupt
                - Mask: enable (1) disable (0)
    gpic mask|m code[0-4] mask[0-31]
    gpic unmask|um code[0-4] mask[0-31]
                - Mask/Unmask an interrupt
                - Code: CPU Int Mask High Reg  = 0
                -       CPU Int Mask Low Reg   = 1
                -       PCI1 Int Mask High Reg = 2
                -       PCI1 Int Mask Low Reg  = 3
                -       GPP Int Mask Reg       = 4
                - Mask/Unmask bit, must be one bit location 0-31
```

Examples:

```
"gpic" Command Example Uses from the DINK32>> prompt:
        gpic              - Display various GPIC status info
                          - Set 'env VERBOSE=1' for more
                            verbose printout
        gpic init         - Initialize the GPIC unit to default mode
        gpic tmcnt        - Display All Timer/Counter Count Values
        gpic tmcnt 1      - Display Timer/Counter 1 Count Value
        gpic tmcnt 1 FFFFFF
                          - Set Timer/Counter 1 Count
                            Value to 0xFFFFFF
        gpic tmen 2       - Enable counting on Timer/Counter 2
                            Same as 'gpic tmcon 2 1'
        gpic tmdis 2      - Inhibit counting on Timer/Counter 2
                            Same as 'gpic tmcon 2 0'
        gpic tmcon        - Display All Timer/Counter Config Info
        gpic tmcon 1      - Display Timer/Counter 1 Config Info
        gpic tmcon 1 1    - Enable Counting on Timer/Counter 1
        gpic tmcon 1 0    - Disable Counting on Timer/Counter 1
        gpic tmcon 1 1 1  - Set Timer/Counter 1 as a Timer
                            Enable Counting on Timer 1
        gpic tmcon 1 1 0  - Set Timer/Counter 1 as a Counter
                            Enable Counting on Counter 1
        gpic tmmsk        - Display All Timer Mask Values
        gpic tmmsk 1 1    - Unmask Timer 1 Interrupt
        gpic tmmsk 1 0    - Mask Timer 1 Interrupt
        gpic tmen 2       - Enable counting on Timer 2
        gpic tmdis 2      - Inhibit counting on Timer 2
        gpic mask 1 8     - Mask interrupts from Timer0_1 in the Main
                            Interrupt Mask Low register
        gpic mask 4 23    - Mask the GPP[23] interrupt in the
                            GPPInterrupt Mask register
        gpic unmask 0 24  - Unmask interrupts from GPP7_0 in the Main
                            Interrupt Mask High register
```

3.34.1 DINK32 GPIC Initialization Sequence

```
DINK32 [MPC7450 #0] >>gpic init
GPIC: Initialize GPIC
GPIC: In gpicInit()

GPIC: Disable External Interrupts
```

```
GPIC: Mask all external interrupt sources
GPIC: Clear all possible GPP interrupts

GPIC: Setup MPPs
GPIC:   MPP[22] = GPP[22] = Super I_O Interrupt
GPIC:   MPP[9:6] = GPP[9:6] = PCI 0 Slot 3-0 Interrupts
GPIC:   MPP[13:10] = GPP[13:10] = PCI 1 Slot 3-0 Interrupts
GPIC:   MPP[30] = GPP[30] = CPU1 Cross-Processor Interrupt Input
GPIC:   MPP[4] = GPP[4] = CPU0 Cross-Processor Interrupt Input
GPIC:   MPP[31] = GPP[31] = CPU1 Cross-Processor Interrupt Output
GPIC:   MPP[5] = GPP[5] = CPU0 Cross-Processor Interrupt Output

GPIC: Setup GPPs
GPIC:   GPP[22] = Super I_O = Input, Active High, Int enabled
GPIC:   GPP[9:6] = PCI0 Slot 3-0 = Input, Active Low, Int enabled
GPIC:   GPP[13:10] = PCI1 Slot 3-0 = Input, Active Low, Int enabled
GPIC:   GPP[30] = CPU1 CPI = Input, Active High, Int enabled
GPIC:   GPP[4] = CPU0 CPI = Input, Active High, Int enabled
GPIC:   GPP[31] = CPU1 CPI = Output, Active High
GPIC:   GPP[5] = CPU0 CPI = Output, Active High

GPIC: Setup Timer0 to interrupt CPU0 for sleep demo

GPIC: Setup Timer1 to interrupt CPU0
GPIC:   User must enable/disable with ‘gpic tmcon 1 [1|0]’
GPIC:   Timer Count = 0x0FFFFFFF

GPIC: Setup Timer2 to interrupt CPU1 for sleep demo

GPIC: Setup Timer3 to interrupt CPU1
GPIC:   User must enable/disable with ‘gpic tmcon 3 [1|0]’
GPIC:   Timer Count = 0x0FFFFFFF

GPIC: Unmask interrupts to CPUINT* (CPU0)
GPIC:   Unmask GPP[23:16]
GPIC:   Unmask GPP[15:8]
GPIC:   Unmask GPP[7:0]
GPIC:   Unmask Timer0

GPIC: Unmask interrupts to INT1* (CPU1)
GPIC:   Unmask GPP[31:24]
GPIC:   Unmask Timer2

GPIC: Enable External Interrupts
DINK32 [MPC7450 #0] >>
```

3.35 Help

Command	help
Abbreviation	he
Syntax	help [name]
Builds	All
Boards	All
Processors	All

The “help” command provides information on the available commands in DINK (which vary according to board and processor type). ‘help’ followed by the command name and arguments provides additional detailed information.

Arguments:

name	Name of the command for which additional help is needed
‘cache’	Information on setting L2/L3 cache variables
‘who’	Information on DINK team

If no “name” is supplied, a menu of all available commands is shown. See “menu” for details. Help is also available by adding the argument ‘?’ to any command.

Examples:

```
DINK32 [MPC8240] >> help rtc
RTC COMMAND
===========
Mnemonic: rtc
Syntax: rtc [-s][-w]
Description: This command sets or displays the Real-Time Clock
Flags:  -s   Set the clock
        -w   Watch the clock (until a key is pressed)

If no option is entered, the date and time is printed.
```

3.36 History

Command	history
Abbreviation	hi
Syntax	hi
Builds	DINK
Boards	All
Processors	All

The “history” command manages the history table and the buffer containing the most recently executed commands. The only function of the history command is to display the current contents of the history table.

The indices reported by the history table can be used to re-enter a recent command.

Arguments:

None.

Examples:

```
DINK32 [MPC7450] >> hi
DINK32 [MPC7450] >> rd r3
gpr03: 0x12345678        gpr04: 0xabcdef01
DINK32 [MPC7450] >> hi
    1  rd r3
DINK32 [MPC7450] >> hi
    2  rd r3
    1  hi
DINK32 [MPC7450] >> go 90100
Return to DINK via blr
DINK32 [MPC7450] >> hi
    4  rd r3
    3  hi
    2  hi
    1  go 90100
```

3.37 Hardware Monitor

Command	hardware monitor
Abbreviation	hwm
Syntax	hwm [-iw] object
Builds	All
Boards	All
Processors	All

The “hwm” command interfaces with industry-standard, hardware-monitoring devices, reporting voltages, fan speed, temperature, etc, for those platforms in which the command is present.

Arguments:

-i Initialize the HWM system using per-board data.

-w Watch: repeatedly report on the given ‘object’.

If no ‘object’ is specified, all objects are reported. Since ‘objects’ vary by board, use this means to discover them.

Examples:

```
DINK32 [MPC7447A] >> hwm cpu1
         CPU1: 46 degC
DINK32 [MPC7447A] >> hwm -w cpu1
         CPU1: 46 degC
         CPU1: 47 degC
         CPU1: 46 degC
...
```

3.38 I²C

Command	i2c
Abbreviation	i2c
Syntax	i2c [-p] [i2c_addr]
Builds	All
Boards	All
Processors	All

The “i2c” command performs a few common I^2C functions: displaying a block of memory (typically for EEPROM devices) and probing the I^2C bus.

Arguments:

-p Check all I^2C addresses, and report address responding with non-zero values with the first four bytes.

addr Dump memory from 0x00-0xFF for the given I^2C address.

Examples:

```
DINK32[MPC7448] {17} >> i2c -p
  46
  51
  52
  57
DINK32[MPC7448] {18} >>
```

3.39 Identify

Command	identify
Abbreviation	id
Syntax	id [-i]
Builds	DINK
Boards	Motherboards
Processors	All

The “id” command manages the 256-byte ‘identity’ serial EEPROM present on MPMC cards for Sandpoint/MVP systems. The ‘identify’ EEPROM contains the following information:

- Board serial number 6 characters
- Board revision 1 character
- Board assembly date Set when “-i” option used
- External cache string 30 characters
- Standalone bus speed 1 character
- Standalone environment 100 characters (not managed by “id”)

The “id” command alone displays the current information. Note that upon startup, the identity EEPROM is checked for a valid entry for an L2 and/or L3 initialization string. If found, that value is used to initialize the corresponding cache, favoring the identify EEPROM values over any found in the Sandpoint environment variables (since MPMC cards can be moved from Sandpoint to Sandpoint, facing different environments, while the identity EEPROM moves with the card).

The standalone environment variables are only used by MPMC8245 cards configured for stand-alone operation. Refer to the corresponding application note for details.

Arguments:

-i Causes DINK to initialize the identity EEPROM with values supplied by the user, as well as the current date and time (see “rtc”).

If no argument is supplied, the current settings as shown.

Examples:

```
DINK32 [MPC7450] >> id -i
Serial No: V743
Revision: A
L2Init: 0x8000000
L3Init: 0x9F032000
SA Bus Speed:
  writing I2C@99
DINK32 [MPC7450] >>
```

3.40 If

Command	if
Abbreviation	if
Syntax	if ‘{‘ test ‘}’ command [command ...]
Builds	All
Boards	All
Processors	All

The “if” command executes a DINK command if the specified expression is non-zero.

Arguments:

(none)

Examples:

```
DINK32[MPC7448] {17} >> ex i=1; if { $i > 0; echo $i; md 0-100 }
1
  00000000      00000000 00000000 00000000 00000000         ................
  00000010      00000000 00000000 00000000 00000000         ................
  00000020      00000000 00000000 00000000 00000000         ................
  00000030      00000000 00000000 00000000 00000000         ................
  00000040      00000000 00000000 00000000 00000000         ................
  00000050      00000000 00000000 00000000 00000000         ................
  ...
```

3.41 Log

Command	log
Abbreviation	log
Syntax	log
Builds	DINK
Boards	All
Processors	All

The “log” command toggles whether DINK messages and character echoes that are sent to the “keyboard”/COM1 port will also be sent to the “host”/COM2 port.

Arguments:

None.

Examples:

```
DINK32 [MPC750] >> log
  Logging enabled: IO is copied to the host port.
DINK32 [MPC750] >> log
  Logging disabled.
```

3.42 Memory Compare

Command	memcompare
Abbreviation	mc
Syntax	mc [-q] <addr1> <addr2> <addr3>
Builds	All
Boards	All
Processors	All

The “memcompare” command compares two blocks of memory, a source and a target, using 32-bit accesses, reporting each difference found. The report stops at each page, allowing the command to be cancelled in the event of a large number of differences found.

Arguments:

-q	If specified, only the count of differences is shown, not each one.
addr1	First source address to compare
addr2	Last source address to compare
addr3	First target address to compare

The last target end address is not specified; comparison stops when the source address range has been checked.

Examples:

```
DINK32 [MPC750] >> mc 100000 100200 200000
  00100100 = ABCDEF01  <>  00200100 = ABCDEFFF
  1 difference(s) found.
DINK32 [MPC750] >> mc 400000 500200 200000
  64551 difference(s) found.
DINK32 [MPC750] >>
```

3.43 Memory Display

Command	memdisp
Abbreviation	md
Syntax	md [-b][-h][-w] [-r] start ['+' l end]
Builds	All
Boards	All
Processors	All

The "memdisp" command displays data stored in the specified memory locations. The display will always be aligned on a 16-byte boundary in which the address given will be included. By default, "md" displays up to nine lines of data and then pauses, waiting for <return> before continuing (unless the end address has been reached). This number can be changed with the ENV LINES value.

The start address is normalized to the previous quad-word boundary. Likewise, the ending address is normalized to the next quad-word boundary. For example, if the start address was 0x00100104, then the first memory address to be displayed would be 0x00100100. If the end address was 0x00100104, then the last memory location to be displayed would be 0x0010010C.

Arguments:

-b	Display data as bytes
-h	Display data as halfwords
-w	Display data as words (default)
-r	Display data as byte-reversed data
start	The starting address to display
+	Display from the starting address to the end of memory
end	Display from the starting address to the ending address

Examples:

```
DINK32 [MPC7400] >> md 160100-160130
0x00160100 00000041 00000042 00000043 00000044
0x00160110 00000045 00000046 00000047 00000048
0x00160120 00000000 00000000 00000000 00000000
0x00160130 00000000 00000000 00000000 00000000
DINK32 [MPC7400] >> md 160260+
0x00160260 00000000 00000000 00000000 00000000
0x00160270 00000000 00000000 00000000 00000000
0x00160280 00000000 00000000 00000000 24002400
```

```
...
DINK32 [MPC7400] >> md -w 160260+
0x00160260 0000 0000 0000 0000 0000 0000 0000 0000
0x00160270 0000 0000 0000 0000 0000 0000 0000 0000
0x00160280 0000 0000 0000 0000 0000 0000 2400 2400
DINK32 [MPC7400] >> md -r 160260+
0x00160260 00000000 00000000 00000000 00000000
0x00160270 00000000 00000000 00000000 00000000
0x00160280 00000000 00000000 00000000 00240024
```

3.44 Memory Fill

Command	memfill
Abbreviation	mf
Syntax	mf <addr1> <addr2> <value>
Builds	All
Boards	All
Processors	All

The “memfill” command writes the fill value, a 32-bit hex value, to every memory location between the two addresses specified.

Arguments:

addr1	First address to fill
addr2	Last address to fill (inclusive)
value	Value to fill memory with

Examples:

```
DINK32 [MPC750] >> mf 100000 100200 89898989
DINK32 [MPC750] >> memfill 100100 100200 89898989
DINK32 [MPC750] >> memfill 100140 10015c 00000000
DINK32 [MPC750] >> memdisp 100120-100160
0x00100120 89898989 89898989 89898989 89898989
0x00100130 89898989 89898989 89898989 89898989
0x00100140 00000000 00000000 00000000 00000000
0x00100150 00000000 00000000 00000000 00000000
0x00100160 89898989 89898989 89898989 89898989

DINK32 [MPC750] >> mf 100144 100144 44444444
DINK32 [MPC750] >> md 100120-100160
0x00100120 89898989 89898989 89898989 89898989
0x00100130 89898989 89898989 89898989 89898989
0x00100140 00000000 44444444 00000000 00000000
0x00100150 00000000 00000000 00000000 00000000
0x00100160 89898989 89898989 89898989 89898989
```

3.45 Memory Info

Command	meminfo
Abbreviation	mi
Syntax	mi [-b] [-c] [-r] [-s] [-n no] [-w no]
Builds	DINK
Boards	All
Processors	All

The “meminfo” command displays or controls various memory controller settings, including the following:

- memory bank settings
- DIMM/SODIMM SPD EEPROM data
- comparison between SPD data and memory controller settings.

“meminfo” can also be used to correct memory controller settings based on the I^2C SPD information, or to initialize the I^2C SPD information for non-SODIMM-based MPMC cards.

Arguments:

-b Causes the ‘-w’ command to select single-bank SPD EEPROM data. The default is for two-bank data. Single-bank data is required for recent non-SODIMM MPMC cards.

-s Causes the I^2C SPD EEPROM information to be decoded and displayed

-n no Select DIMM SPD number. The default is ‘1’ on Yellowknife/Sandpoint and ‘0’ on MVP.

-w no If specified, default SPD data is written to the SPD EEPROM. “no” may be the values 64 or 128, and causes 64-MB or 128-MB SPD data to be written, based upon a 64-MB SODIMM. This option is recommended for non-SODIMM-based MPMC cards and must be executed once to optimize memory settings.

-c Causes “meminfo” to compare the I^2C SPD information to the memory controller settings and recommend changes. If ‘-c’ is entered a second time, the changes recommended will be automatically made.

-r Causes the memory controller to reinitialize the external SDRAM (required if certain parameters need to be updated).

If no option is specified, the current memory controller settings are displayed.

Examples:

```
DINK32 [MPC755] >> mi
Memory settings:
  ROM Speed: 30 ns (2 clocks)
  SDRAM Bank 0:  Enabled
    Range: [00000000 -> 01ffffff] 32 MBytes
    Speed: 3/1/1/1
  SDRAM Bank 1:  Disabled
```

```
    Range: [02000000 -> 03ffffff] 32 MBytes
    Speed: 3/1/1/1
  SDRAM Bank 2:  Disabled
  SDRAM Bank 3:  Disabled
  SDRAM Bank 4:  Disabled
  SDRAM Bank 5:  Disabled
  SDRAM Bank 6:  Disabled
  SDRAM Bank 7:  Disabled
DINK32 [MPC755] >> mi -s
  Addr  Data  Definition         Value
  ====  ====  ================   =========
   02    02   DIMM/SODIMM Type   SDRAM
  ...
DINK32 [MPC755] >> mi -c
I2C: 14 Rows (12+2)
10X: 14 Rows x N (12+2) 64/128Mb.
  OK
...
```

3.46 Memory Modify

Command	memmod
Abbreviation	mm
Syntax	mm [-b] [-h] [-w] [-r] start_address ['+' | end_address]
Builds	DINK
Boards	All
Processors	All

The "memmod" command displays and modifies the contents of memory and allows the user to change the value stored there. Memory is considered to be a contiguous set of 32-bit integers.

Arguments:

-b	Display data as bytes
-h	Display data as halfwords
-w	Display data as words (default)
-r	Display data as byte-reversed data
start	The starting address to display
+	Display from the starting address to the end of memory
end	Display from the starting address to the ending address

The "mm" command will display the contents of a particular location in the requested format, then wait for user commands, which may be any of the following:

value	Write the hexadecimal value to the current location. The value entered may be truncated, if necessary, to the memory size.
'abcd'	Write the character literal value abcd to the current location.
return	Go to the next location using the current selected direction (defaults to forward)

v	Set the direction to forward. Upon return, the current location advances by +1 byte, halfword, or word.
^	Set the direction to reverse. Upon return, the current location advances by -1 byte, halfword, or word.
=	Set the direction to 0. “dm” will keep examining and modifying the same location until ‘v’ or ‘^’ is entered.
?	Help
ESC	Stop
x	Stop

The direction commands are “sticky” -- once set, “mm” will continue in that direction until a new direction is entered. The VT100 cursor keys may be used to move up and down.

Examples:

```
DINK32 [MPC8245] >> memod 160100
0x00160100 : 0x89898989 : ? 44444444

DINK32 [MPC8245] >> mm -b 160100
0x00160100 : 0x44444444 : ? 66

DINK32 [MPC8245] >> memod -h 160100
0x00160100 : 0x66444444 : ? 3333

DINK32 [MPC8245] >> memod -w 60100
0x00160100 : 0x33334444 : ? 22222222

DINK32 [MPC8245] >> mm 160110-160118
0x00160110 : 0x89898989 : ? 11111111
0x00160114 : 0x89898989 : ? 22222222
0x00160118 : 0x89898989 : ? 33333333

DINK32 [MPC8245] >> memod 160200+
0x00160200 : 0x89898989 : ? 12341234
0x00160204 : 0x00000000 : ? 12341234
0x00160208 : 0x00000000 : ? x
```

3.47 Memory Move

Command	memmove
Abbreviation	mv
Syntax	mv [-g] <addr1> <addr2> <addr3>
Builds	All
Boards	All
Processors	All

The “memmove” command copies data from a block of memory, bounded inclusively by the first two addresses, to a block of memory starting at the third address. The result of this command will be two

identical blocks of memory. If the third address falls between the first two addresses, an error message is returned and memory will not be modified.

Arguments:

-g If specified, the move command will move the upper 8 bits of a 64-bit data source to each 8 bits of the destination. The source address is incremented by 64, and the target is incremented by 8. This option allows loading code or data from the 8-bit flash in the 64-bit flash space of an MPC8245/MPC8241/MPC8240.

addr1 First source address to read

addr2 Last source address to read

addr3 First target address to write

Examples:

```
DINK32 [MPC755] >> memfill 160100 160110 ffffffff
DINK32 [MPC755] >> memdisp 160100-160150
0x00160100 ffffffff ffffffff ffffffff ffffffff
0x00160110 ffffffff 00000000 00000000 00000000
0x00160120 00000000 00000000 00000000 00000000
0x00160130 00000000 00000000 00000000 00000000
0x00160140 00000000 00000000 00000000 00000000
0x00160150 00000000 00000000 00000000 00000000

DINK32 [MPC755] >> memove 160100 160110 160140
DINK32 [MPC755] >> memdisp 160100-160150
0x00160100 ffffffff ffffffff ffffffff ffffffff
0x00160110 ffffffff 00000000 00000000 00000000
0x00160120 00000000 00000000 00000000 00000000
0x00160130 00000000 00000000 00000000 00000000
0x00160140 ffffffff ffffffff ffffffff ffffffff
0x00160150 ffffffff 00000000 00000000 00000000
```

3.48 Memory Search

Command	memsrch
Abbreviation	ms
Syntax	ms [-n] [-s] [-m mask] <addr1> <addr2> <value>
Builds	All
Boards	All
Processors	All

The “memsrch” command searches all memory between the first and second addresses for the third, the 32-bit search value. If the second address is less than the first address, an error message is returned and no search is performed.

Matching addresses are printed.

Arguments:

-n	Causes non-matching lines to be printed.
-s	Causes ‘value’ to be treated as a literal string to search for.
-m mask	Both ‘value’ and values read from memory are AND’ed with mask first. This allows partial (non-32bit) comparisons.
addr1	First address to search
addr2	Last address to search (inclusive)
value	Value to search memory for (hexadecimal value or string if -s used).

Examples:

```
DINK32 [MPC7400] >> md 160100-160120
0x00160100  10ff7f00 00ffff00 ff2023ff ff0402ff   ....... #.....
0x00160110  00ffff00 00ffff00 ff5008ff 474A474A   .........PGAGA
0x00160120  00efef00 00ffff00 ff0100ff ff0030ff   ..............

DINK32 [MPC7400] >> ms 160100 160120 ff5008ff
        0x00160118
1 occurrances found.
DINK32 [MPC7400] >> ms -n 160100 160120 0
        0x001060100
...
9 occurrances found.
DINK32 [MPC7400] >> ms -s 160100 160120 GAGA
        0x0016011C
1 occurrances found.
```

3.49 Memory Test

Command	memtest
Abbreviation	mt
Syntax	mt [-d dev] [-bl-hl-w] [xvatqN] [-m max] [-l loop] addr1-addr2
Builds	All
Boards	All
Processors	All

The “memtest” command performs various memory tests on local memory or device registers.

Arguments:

-d dev	Test the indicated device instead of memory. Use the "dm" command to get a list of devices. NOTE: testing non-volatile I^2C EEPROM devices can destroy valuable information as well as reduce the life expectancy of those devices.
-b, -h, -w	Sets size of memory/device tests. If not specified, the default size is bytes for devices (as shown in the device listing). Memory can be tested in any size, while devices

may be limited to bytes. If not specified, the default size is word for memory and bytes for devices.

-l loop — Specify the number of times the memory test should perform all tests. If not specified, each test is performed once. If ‘0’ is specified, the test(s) are run forever.

-m max — If specified, tests exit when ‘max’ errors have been detected. Otherwise, tests continue (regardless of the number of errors) until all are complete.

-x — If specified, halt all tests on the first error. This is useful for extended passes to trap on any error.

-q — Perform only quick tests.

-a — Perform all non-infinite tests.

-v — Report current test address every 1K.

-N — Minimize IO and IO checking for minimal PCI/IO bus traffic.

-n list — Perform only test specified by “list”, where list is one or more of the following characters:

- 0 walking 0’s test (non-destructive, slow)
- 1 walking 1’s test (non-destructive, slow)
- A address=data test (destructive, slow)
- H bus hammer test (destructive, infinite)
- Q quick pattern test (non-destructive)
- R random pattern test (non-destructive)
- S write sensitivity test (destructive, slow)

-t — Show elapsed time (only on systems with a real-time clock).

addr1 — Starting address of memory to test.

addr2 — Ending address of memory to test.

Addresses must be aligned to the size of the access (as specified by the -b/-h/-w option). Tests labelled “non-destructive” modify memory but restore the contents to the former condition, unless an error occurs.

NOTE

Be careful not to test memory regions used by DINK, whether the test is destructive or not.

Examples:

1. Quick test of memory.

```
DINK32 [MPC750] >> mt -q 100000-1fffffc
  PASS 1:
    Quick Test.............................................PASS
Completed tests: No errors.
```

2. Test memory forever until an error is found.

```
DINK32 [MPC750] >> mt -b -a -l 0 -x 100000-1fffffc
```

```
  PASS 1:
    Quick Test.............................................PASS
    Random Pattern Test....................................PASS
    Walking 1's Test.......................................PASS
    Walking 0's Test.......................................PASS
    Address March Test.....................................PASS
    Write Sensitivity Test.................................PASS
  PASS 2:
...
```

3. Test memory using the write sensitivity test and report the elapsed time.

```
DINK32 [MPC750] >> mt -t -n S -x 100000-1fffffc
  PASS 1:
    Write Sensitivity Test.................................PASS
Completed tests: No errors.
Elapsed time: 0:00:16
DINK32 [MPC750] >>
```

3.50 Menu

Command	menu
Abbreviation	me
Syntax	menu
Builds	All
Boards	All
Processors	All

The "menu" command lists all of the commands that are available in the current implementation of DINK32 (varies by processor and platform detected).

Arguments:

None.

Examples:

```
VDINK32[MPC7450] {1} >>me
            Virtual emulation DINK COMMAND LIST
  Command             Mnemonic    Command             Mnemonic
  =======             ========    =======             ========
  About...              AB        Assemble code         AS
  Breakpoint ops        BP        Cache Control         CA
  CRC Memory            CRC       Device Display        DD
  Device Modify         DM        Disassemble           DS
  Download code         DL        Echo text             EC
  Environ. Settings     ENV       Run at address        GO
  Command help          HE        Command history       HI
  Log session           LO        Memory Compare        MC
  Memory Display        MD        Modify Memory         MM
  Memory Fill           MF        Memory Info           MI
  Memory Move           MV        Memory Search         MS
  Memory Test           MT        Menu of commands      ME
  NorthBridge Display   ND        NorthBridge Modify    NM
```

```
  Register Display      RD        Register Modify       RM
  Real-time clock       RTC       Set baud rate         SB
  Show SPRs             SX        Symbol Table          ST
  Transparent Mode      TM        Trace code            TR
  Virtual Host File     VHF       Virtual Memory        VM
  Virtual Quit          VQ        Virtual Simulator     VS
  Unix shell            !         Repeat Last Command   .

Note that lists of valid commands vary by platform, processor, and target.
```

3.51 Northbridge Display

Command	northbridge display
Abbreviation	nd
Syntax	nd [-vebhw] [nb_reg | nb_reg '.' field_name]
Builds	DINK
Boards	All
Processors	All

The "ND" command is used to display the contents of registers in the Northbridge (NB) device (a general term for a CPU-to-PCI interface).

Arguments:

-d	Dump all target registers
-e	Display using fields.
-v	Display using verbose (bit expansion) format. Used when verbose mode is overridden by ENV RDOPT.
-b	Displays Northbridge (NB) registers as a byte.
-h	Displays NB registers as a halfword.
-w	Displays NB registers as a word.
nbreg	Name or PCI index of register to display.
'.' field	If appended to register name, limits display to the following register field name.

Examples:

```
DINK32[MPC8245] {7} >>nd stat
Status                                                                  0x06
------------------------------------------------------------------------------
STAT : 0x00A0
0000000010100000
+.......|....... :     PERR : PERR Detected
 +......|....... :     SERR : SERR Asserted
  +.....|....... :  RcvMAbt : Received Master Abort
   +....|....... :  RcvTAbt : Received Target Abort
    +...|....... :  SigTAbt : Signalled Target Abort
     ++.|....... : DEVSELtm : DEVSEL timing
       +|....... :  DPARDet : Data Parity Error
        +....... : FastB2Bc : Fast Back-to-Back Capable
```

```
         +...... : -------- :
          +..... :  66MHZ_c : 66MHz Capable
           +++++ : -------- :
DINK32[MPC8245] {11} >>nd -f stat
  PERR=0         SERR=0         RcvMAbt=0      RcvTAbt=0      SigTAbt=0
  DEVSELtm=00    DPARDet=0      FastB2Bc=1     66MHZ_c=1
DINK32[MPC8245] {8} >>nd stat.perr
  PERR=0
DINK32[MPC8245] {13} >>nd -b 42
  42     00          (unknown)
DINK32[MPC8245] {13} >>nd -d
00: 1057 0002 0006 0080 0013 0600 0000 0000
10: 0000 0000 0000 0000 0000 0000 0000 0000
20: 0000 0000 0000 0000 0000 0000 0000 0000
30: 0000 0000 0000 0000 0000 0000 0000 0000
40: 0000 0000 0000 0000 0000 0000 0000 0000
50: 0000 0000 0000 0000 0000 0000 0000 0000
60: 0000 0000 0000 0000 0000 0000 0000 0000
70: 0000 CD00 0000 0000 0000 0000 0000 0000
80: 0000 0000 0000 0000 0000 0000 0000 0000
90: 0000 0000 0000 0000 0000 0000 0000 0000
A0: 0000 0000 0000 0000 0010 FF00 060C 000C
B0: 0000 0000 0000 0000 0000 0004 0000 0000
C0: 0000 0100 0000 0000 0000 0000 0000 0000
D0: 0000 0000 0000 0000 0000 0000 0000 0000
E0: 0042 0FFF 0000 0000 0020 0000 0000 0000
F0: 0000 FF02 0003 0000 0000 0000 0000 0010
```

3.52 Northbridge Modify

Command	northbridge modify
Abbreviation	nm
Syntax	nm [-vebhw] [nb_reg | nb_reg '.' field_name] ['=' value]
Builds	DINK
Boards	All
Processors	All

The "NM" command is used to display and modify the contents of registers in the Northbridge (NB) device (a general term for a CPU-to-PCI interface).

Arguments:

-e	Display and modify using fields
-v	Display using verbose (bit expansion) format. Used when verbose mode is overridden by ENV RDOPT.
-b	Display and modify NB registers as a byte
-h	Display and modify NB registers as a halfword
-w	Displays and modify NB registers as a word
nbreg	Name or PCI index of register to display and modify

‘.’ field If appended to register name, limits display to the following register field name

‘=’ value If ‘=’ is appended to a register or register field, the supplied value is used to set the register / register field, without further prompting from the user.

Examples:

```
DINK32[MPC8245] {7} >>nm stat
Status                                                                    0x06
------------------------------------------------------------------------------
STAT : 0x00A0
0000000010100000
+.......|....... :     PERR : PERR Detected
 +......|....... :     SERR : SERR Asserted
  +.....|....... :  RcvMAbt : Received Master Abort
   +....|....... :  RcvTAbt : Received Target Abort
    +...|....... :  SigTAbt : Signalled Target Abort
     ++.|....... : DEVSELtm : DEVSEL timing
       +|....... :  DPARDet : Data Parity Error
        +....... : FastB2Bc : Fast Back-to-Back Capable
         +...... : -------- :
          +.....  :  66MHZ_c : 66MHz Capable
           +++++ : -------- :
  New value ?
VDINK32[MPC8245] {4} >>nm stat.perr
  PERR=0
  New field value [1bit] %1
VDINK32[MPC8245] {5} >>nd stat
Status                                                                    0x06
------------------------------------------------------------------------------
STAT : 0x80A0
1000000010100000
+.......|....... :     PERR : PERR Detected
 +......|....... :     SERR : SERR Asserted
  +.....|....... :  RcvMAbt : Received Master Abort
   +....|....... :  RcvTAbt : Received Target Abort
    +...|....... :  SigTAbt : Signalled Target Abort
     ++.|....... : DEVSELtm : DEVSEL timing
       +|....... :  DPARDet : Data Parity Error
        +....... : FastB2Bc : Fast Back-to-Back Capable
         +...... : -------- :
          +.....  :  66MHZ_c : 66MHz Capable
           +++++ : -------- :
VDINK32[MPC8245] {9} >>nm stat.perr=0
VDINK32[MPC8245] {10} >>nd stat
Status                                                                    0x06
------------------------------------------------------------------------------
STAT : 0x00A0
0000000010100000
+.......|....... :     PERR : PERR Detected
 +......|....... :     SERR : SERR Asserted
  +.....|....... :  RcvMAbt : Received Master Abort
   +....|....... :  RcvTAbt : Received Target Abort
    +...|....... :  SigTAbt : Signalled Target Abort
     ++.|....... : DEVSELtm : DEVSEL timing
       +|....... :  DPARDet : Data Parity Error
        +....... : FastB2Bc : Fast Back-to-Back Capable
         +...... : -------- :
```

```
          +..... :  66MHZ_c : 66MHz Capable
           +++++ : -------- :
```

3.53 Net Info

Command	net info
Abbreviation	ni
Syntax	ni [-i] [-m] [-p] [-r] [-t]
Builds	DINK
Boards	Sandpoint, MVP
Processors	All

This command can be used to show and modify the network parameters stored in NVRAM.

Arguments:

-i	Initialize network device and TCP/IP parameters
-m	Show MAC address of embedded Ethernet controller (MVP only)
-p	Modify TCP/IP network parameters, select TFTP server IP address
-r	Resets network information
-t	Set TELNET session timeout (platform will be reset if timeout)

NOTE

No NIC initialization is allowed if connected via TELNET console. Use ni -p to change TFTP server IP address in TELNET console.

At the time of Revision 13.2 of this document, only Realtek 8129/8139 or equivalent clones are supported by DINK driver.

Examples:

```
DINK32[MPC7455] {8} >>ni
  NETWORK INFORMATION
    PCI CARD
      Type 8139/10EC on slot 16

    SETTINGS
      SERVER(TFTP) :  10. 18.15.27
      GATEWAY      :  10. 18.177.254
      NETMASK      : 255.255.252.  0
      DHCP         :   0.  0.  0.  0
      CLIENT(DINK) :  10. 18.177.201

    DHCP: Disabled
    SESSION TIMEOUT: Never
```

3.54 Opts

Command	opts
Abbreviation	op
Syntax	opts
Builds	DINK
Boards	Sandpoint, MVP
Processors	All

This command can be used to show and/or modify the DINK global options parameter. The options are stored in NVRAM as an environment variable.

Arguments:

None.

Examples:

```
DINK32[MPC7455] {8} >>op
DINK Global Options                                                    SPR #0000
------------------------------------------------------------------------------
DO : 0x00000000
00000000000000000000000000000000
++++++++++++++++++++++++++.... : -------- :
                         +.... :   SHELLX : Show shell expansions
                          +... :    GOMSG : Show debug messages for go/tr
                           +.. :  NOAGENT : Disable PrPMC Agent Enable
                            +. :   NOPMAP : Disable PCI Mapping
                             + :    MINIM : Minimal Initialization
  New value ? 2
DINK32[MPC7455] {5} >>env
  DO=2
```

3.55 PCI

Command	PCI info
Abbreviation	pci
Syntax	pci
Builds	All
Boards	All
Processors	All

This command can be used to show all PCI status.

Arguments:

None.

Examples:

```
DINK32[MPC7455] {8} >>pci
  PCI Bus #0
  ==========
           Speed: 66 MHz
    MV64460 Mode: 66 MHz PCI-X
        Protocol: PCI-X
           Width: 64 bits
    Force PCI/33: Off
          Slot 1: Occupied
          Slot 2: Empty
```

3.56 PCI Config List

Command	pciconfig
Abbreviation	pcf
Syntax	pcf [-a] [-n fno] dev_num
Builds	DINK
Boards	All
Processors	All

The “pciconf” command displays the common PCI configuration registers and 16 additional device-specific registers of a PCI device.

Arguments:

-a Show all registers, not just the 0x00-0x3F range

-n fno Show configuration space for function number “fno” on a multifunction device; default is 0.

dev_numPCI device to access, 0 or 10..31.

Examples:

```
DINK32 [MPC755] >> ppr
devNo             PCI ADR.                  DEVICE ID         VENDOR ID
=====             ========                  ========= =========
11                0x80005800                0x0565            0x10ad

DINK32 [MPC755] >> pcf 11
ADDR.             VALUE                     DESCRIPTION
=====             =====                     ===========
0x00              0x10ad                    Vendor ID
0x02              0x0565                    Device ID
0x04              0x0007                    PCI command
0x06              0x0200                    PCI status
0x08              0x04                      Revision ID
0x09              0x00                      Standard Programming Interface
0x0a              0x01                      Subclass code
0x0b              0x06                      Class code
0x0c              0x00                      Cache line size
0x0d              0x00                      Latency timer
0x0e              0x80                      Header type
```

```
0x0f            0x00                    BIST control
0x10            0x00000000              Base Address Register 0
0x14            0x00000000              Base Address Register 1
0x18            0x00000000              Base Address Register 2
0x1c            0x00000000              Base Address Register 3
0x20            0x00000000              Base Address Register 4
0x24            0x00000000              Base Address Register 5
0x28            0x00000000              Cardbus CIS Pointer
0x2c            0x0000                  Subsystem Vendor ID
0x2e            0x0000                  Subsystem ID
0x30            0x00000000              Expansion ROM Base Address
0x3c            0x00 Interrupt line
0x3d            0x00 Interrupt pin
0x3e            0x00 MIN_GNT
```

3.57 PCI Display

Command	pcidisp
Abbreviation	pd
Syntax	pd [-bhw] [-f fno] dev_num reg_addr
Builds	DINK
Boards	All
Processors	All

The “pcidisp” command causes the contents of the specified PCI device register to be displayed.

The PCI devices accessible are 0 and 10 to 31.

Arguments:

-b	Display and modify NB registers as a byte.
-h	Display and modify NB registers as a halfword.
-w	Display and modify NB registers as a word.
-f fno	Show configuration space for function number “fno” on a multifunction device; default is 0.
dev_num	PCI device to access, 0 or 10..31
reg_addr	Register address (hex index of standard PCI configuration registers)

Examples:

```
DINK32 [MPC603e] >> pcidisp 11 10
  0x10 0x12345678    Base Address Register 0
DINK32 [MPC603e] >> pd 0 0
  0x00 0x1057        Vendor ID
DINK32 [MPC603e] >> pd -b 11 40
  0x40 0x17          (unknown)
```

3.58 PCI Map

Command	pci map
Abbreviation	pmap
Syntax	pmap [-r]
Builds	DINK
Boards	Non Excimer/Maximer
Processors	All

The “PMAP” command displays the devices detected and configured during the initialization process.

Arguments:

-r Re-runs the PCI initialization process, configuring newly discovered devices (already configured devices are not changed). This is typically used for devices that are not automatically detected by the DINK PCI enumeration process and/or that need special enablement procedures.

Examples:

```
DINK32 [MPC603e] >> pmap
PCI IO Map:
  No       Range           Inbound   Outbound  Name
  ==  ==================  ========  ========  ====

PCI Mem Map:
  No       Range           Inbound   Outbound  Name
  ==  ==================  ========  ========  ====
```

3.59 PCI Modify

Command	pcimod
Abbreviation	pm
Syntax	pm [-bhw] [-f fno] dev_num reg_addr
Builds	DINK
Boards	All
Processors	All

The “pcimod” command is used to modify the content of a configuration register (reg_addr) of a PCI device (dev_num). The DevNum depends on the PCI slot to which the device is attached, and it can be found by executing the ppr (PCI Device Probe) command. This command first displays the current value of the desired register, then asks the user to enter the new value.

The PCI devices accessible are 0 and 10 to 31.

This command does not return an error if the register requested is a read-only register.

Arguments:

-b	Display and modify NB registers as a byte.
-h	Display and modify NB registers as a halfword.
-w	Display and modify NB registers as a word.
-f fno	Show configuration space for function number “fno” on a multifunction device; default is 0.
dev_num	PCI device to access, 0 or 10..31
reg_addr	Register address (hex index of standard PCI configuration registers)

Examples:

```
DINK32 [MPC750] >> pcimod 11 10
  0x10 0x00000000 Base Address Register 0
New Value? 12345678

DINK32 [MPC750] >> pcidisp 11 10
  0x10 0x12345678 Base Address Register 0
```

3.60 PCI Probe

Command	pciprobe
Abbreviation	ppr
Syntax	ppr
Builds	DINK
Boards	All
Processors	All

The “pciprobe” command causes the PCI bus to be scanned for PCI devices and cards. For each device found, the vendor and device ID are printed, as well as the device classification.

The PCI devices scanned are 0 and 10 to 31.

Arguments:

None

Examples:

```
DINK32 [MPC8240] >> ppr
  ====  ===============  =========  ==============  ===========  =============
  UNIT  PORT.BUS.DEV.FN  VEND/DEV#  CLASS           IDSEL/ADDR   BASE ADDRESS
  ====  ===============  =========  ==============  ===========  =============
    0     0.0.0. 0       1057.0002  Bridge          80000000  MEM: 00000000
...
```

3.61 PHY

Command	phy
Abbreviation	phy
Syntax	phy [-l] [-r]
Builds	All
Boards	All
Processors	All

The “phy” command performs operations on Ethernet PHY devices. With no arguments, the current status of the PHYs is reported.

Arguments:

-r Reset a PHY using the IEEE standard RESET bit.

-l List all known PHYs.

-Examples:

```
DINK32 [MPC750] >> phy
  ENet Port 0 / PHY #8
  ====================
    Auto-Neg: Complete
       Speed: 1000 Mbps
      Duplex: Full
   Crossover: MDIX
       Cable: 80-110 m
        Link: Up
```

3.62 Ping

Command	ping
Abbreviation	ping
Syntax	ping [-f][-n no][-l len] <host_ip>"
Builds	DINK
Boards	Sandpoint, MVP
Processors	All

The “ping” command executes a typical network ping operation.

Arguments:

-f Performs a ‘flood’ ping

-l no Sets the packet length to ‘no’. Default is 64.

-n no Sets the number of pings to perform;.Default is 4.

host_ip IP address to ping.

Examples:

```
DINK32 [MPC750] >> ping 163.11.104.199
PING 163.11.104.199 from 163.11.104.198 : 64 bytes of data.
64 bytes from 163.11.104.199: icmp_seq=0 ttl=20 time=0.004 ms
64 bytes from 163.11.104.199: icmp_seq=1 ttl=20 time=0.004 ms
64 bytes from 163.11.104.199: icmp_seq=2 ttl=20 time=0.004 ms
64 bytes from 163.11.104.199: icmp_seq=3 ttl=20 time=0.004 ms
--- 163.11.104.199 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
```

3.63 PX

Command	e500 performance register display
Abbreviation	px
Syntax	px
Builds	eDINK
Boards	Mars
Processors	e500 MPC85xx

The “px” command will display the names and register numbers of all the e500 performance registers.

Examples:

```
edink[MPC85x0] {9} >>px
          e500 PERFORMANCE MONITOR REGISTERS
  Perf Mon  Number      Register Name
  ========  ======      =============
  UPMC0          0       User Performance Monitor Counter 0 (Reflects PMC0)
  UPMC1          1       User Performance Monitor Counter 1 (Reflects PMC1)
  UPMC2          2       User Performance Monitor Counter 2 (Reflects PMC2)
  UPMC3          3       User Performance Monitor Counter 3 (Reflects PMC3)
  PMC0          16       Supervisor Performance Monitor Counter 0
  PMC1          17       Supervisor Performance Monitor Counter 1
  PMC2          18       Supervisor Performance Monitor Counter 2
  PMC3          19       Supervisor Performance Monitor Counter 3
  UPMLCa0      128       User Local Control A Register  0
  UPMLCa1      129       User Local Control A Register  1
  UPMLCa2      130       User Local Control A Register  2
  UPMLCa3      131       User Local Control A Register  3
  PMLCa0       144       Supervisor Local Control A Register  0
  PMLCa1       145       Supervisor Local Control A Register  1
  PMLCa2       146       Supervisor Local Control A Register  2
  PMLCa3       147       Supervisor Local Control A Register  3
  UPMLCb0      256       User Local Control B Register  0
  UPMLCb1      257       User Local Control B Register  1
  UPMLCb2      258       User Local Control B Register  2
UPMLCb3      259       User Local Control B Register  3
  PMLCb0       272       Supervisor Local Control B Register  0
PMLCb1       273       Supervisor Local Control B Register  1
  PMLCb2       274       Supervisor Local Control B Register  2
  PMLCb3       275       Supervisor Local Control B Register  3
  UPMGC0       384       User Performance Global Control Register 0
  PMGC0        400       Supervisor Performance Global Control Register 0
```

3.64 REGDISP

Command	regdisp
Abbreviation	rd
Syntax	rd [-b] [-h] [-w] [-d] [-v] [-f] [-e] [r[##[+|-##]] [f[##[+|-##]] [v[##[+|-##]] [s[##]][spr_name['.'fieldname] [nb['.' field_name]] [bats]
Builds	DINK
Boards	All
Processors	All

The “rd” command will display the contents of the specified registers. This command offers the user numerous options for viewing different sets or portions of the PowerPC register file, as well as for Northbridge or PCI registers.

In general, all members of a register family can be displayed by first letter of the register family name. Otherwise, one or more of the specified registers are displayed.

Arguments:

The following arguments are all valid for specifying register displays.

-b Display and modify registers as a byte

-h Display and modify registers as a halfword

-w Display and modify registers as a word

-f Display and modify registers as a single-float value (vector and floating-point registers only)

-d Display and modify registers as a double-float value (vector and floating-point registers only)

-v Force display to be ‘verbose’ regardless of RDMODE environment setting

-e Present register values of all the fields of an SPR or NB register as a list of field names and values. It is longer than a simple hexadecimal display, but shorter than the ‘verbose’ option.

r Show entire general-purpose register family

rx Show general-purpose register #x (0 to 31)

rx+ Show general-purpose register #x through r31

rx-ry Show general-purpose register #x through #y

rd f Show entire floating-point family

fx Show floating-point register #x (0 to 31)

fx+ Show floating-point register #x through f31

fx-fy Show floating-point register #x through #y

v Show entire AltiVec vector register family

vx Show AltiVec vector register #x (0 to 31)

vx+ Show AltiVec vector register #x through v31

vx-vy Show AltiVec vector register #x through #y

p Show entire e500 performance register family (eDINK only)

px Show e500 performance register # (0-1024) (eDINK only)

s Show entire special-purpose register family

sx Show special-purpose register # (0-1024)

SPRnameShow special-purpose register by name, e.g. “msr”

sx.fieldShow field of special-purpose register # (0-1024)

SPRname.fieldShow field of special-purpose register by name

nb # Show Northbridge (MPC107, etc) register # (0-0xFF)

bats Show and decode BAT registers.

If no arguments are given, the last register set used (as saved in the “RDLAST” variable) is shown. If RDLAST is not set, nothing is shown.

The general-purpose registers (GPRs) and floating-point registers (FPRs) are always displayed as hexadecimal values, while special-purpose registers (SPRs) and NB registers may be displayed as hexadecimal values, a list of field names, or a visually expanded bit field (called ‘verbose’ mode). Refer to the ‘-e’ and ‘-v’ for the latter two, or the ENV option for RDMODE. Vector prefix registers (VPRs) may be displayed as hex or floating point values, in sizes from byte to word.

After registers are shown, the arguments are saved in the RDLAST environment variable.

Examples:

```
VDINK32[MPC7450] {7} >>rd r1-r2
  R1 =0008FFE0            R2 =00000000

DINK32 [MPC750] >> rd hid0
Hardware Implementation Dependent 0                   SPR #1008
-----------------------------------------------------------------
HID0 : 0x80010080
10000000000000010000000010000000
+------------------------------- : EMCP   : Enable MCP Check
 +------------------------------ : DBP    : Disable Addr/Data Gener
  +----------------------------- : EBA    : Enable Address Parity
   +---------------------------- : EBD    : Enable Data Parity Check
    +--------------------------- : BCLK   : Select Bus Clock
     +-------------------------- : ---    :
      +------------------------- : ECLK   : Enable External Test
       +------------------------ : PAR    : Disable ARTRY/SHD
        +----------------------- : DOZE    : Doze Mode: PLL/TB/
(etc.)

DINK32 [MPC750] >> rd -e HID0
HID0 : 0x80010080
  EMCP=1    DBP=0   EBA=0   EBD=0
```

```
(etc.)

VDINK32[MPC7450] {9} >>rd r23+
  R23=00000000            R24=00000000
  R25=00000000            R26=00000000
  R27=00000000            R28=00000000
  R29=00000000            R30=00000000
  R31=00000000
```

Vector registers have flexible formatting.

```
DINK32 [MPC750] >> rd v2
  V02= 0x00000000_00000000_00000000_00000000
VDINK32[MPC7450] {5} >>rd -f v2
  V02=       0.00000000       0.00000000       0.00000000       0.00000000
DINK32 [MPC750] >> rd v
  V00= 0x00000000_00000000_00000000_00000000
  V01= 0x00000000_00000000_00000000_00000000
...
  V30= 0x00000000_00000000_00000000_00000000
  V31= 0x00000000_00000000_00000000_00000000
DINK32 [MPC750] >> rd -b v2-v4
  V02= 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  V03= 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  V04= 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

DINK32 [MPC750] >> rd nb a8
PICR1 : 0xA800CE42
```

In addition, the “rd” command can display fields of the SPRs and NB register, where defined. To display only a field, append a dot “.” and the name of the field to the SPR register name.

```
DINK32 [MPC750] >> rd msr.emcp
  EMCP = 1
DINK32 [MPC750] >>
```

Lastly, the “rd” command can display and decode all BAT registers (four or eight of each, depending on the processor).

```
DINK32 [MPC750] >> rd bats
IBAT  BLOCK ADDRESS      SIZE  REPLACEMENT ADDR    WIMG  PROT  VALID
====  ================== ====  ==================  ====  ====  =====
 0    00000000..0FFFFFFF 256MB 00000000..0FFFFFFF  0011  R/W   [S,P]
 1    FF000000..FFFFFFFF 256MB FF000000..FFFFFFFF  1011  R/W   [S,P]
 2    --------..-------- ----  --------..--------  ----  ----  [off]
 3    --------..-------- ----  --------..--------  ----  ----  [off]
DBAT  BLOCK ADDRESS      SIZE  REPLACEMENT ADDR    WIMG  PROT  VALID
====  ================== ====  ==================  ====  ====  =====
 0    00000000..0FFFFFFF 256MB 00000000..0FFFFFFF  0011  R/W   [S,P]
 1    FF000000..FFFFFFFF 256MB FF000000..FFFFFFFF  1011  R/W   [S,P]
 2    80000000..8FFFFFFF 256MB 80000000..8FFFFFFF  1111  R/W   [S,P]
 3    --------..-------- ----  --------..--------  ----  ----  [off]

DINK32 [MPC750] >>
```

3.65 REGMOD

Command	regmod
Abbreviation	rm
Syntax	rm[-b|-h|-w] [r[##[+|-##]] [f[##[+|-##]] [v[##[+|-##]] [s[##]][spr_name['.'fieldname] [nb['.' field_name]] '=' value
Builds	DINK
Boards	All
Processors	All

The "rm" command will display the contents of the specified registers and then prompt for a new value to set the register. This command offers the user numerous options for viewing different sets or portions of the PowerPC register file, as well as for Northbridge or PCI registers.

NOTE

Changes to registers only take effect when user programs are executed via 'go' or 'tr', with the exception of the Northbridge PCI registers.

For GPRs and SPRs, the value can also be specified on the command line by following the name with an equals sign and the value to be used. If more than one register is slated for modification, the same supplied value is stored in all the registers.

Arguments:

The following arguments are all valid for specifying register displays.

-b, -h, -w	Set the access size for Northbridge and vector registers. Ignored for register accesses, which are always 32-bits.
rx	Show/modify general-purpose register #x (0 to 31)
rx+	Show/modify general purpose-register #x through r31
rx-ry	Show/modify general purpose-register #x through #y
rd f	Show/modify entire floating-point family
fx	Show/modify floating-point register #x (0 to 31)
fx+	Show/modify floating-point register #x through f31
fx-fy	Show/modify floating-point register #x through #y
v	Show/modify entire AltiVec vector register family
vx	Show/modify AltiVec vector register #x (0 to 31)
vx+	Show/modify AltiVec vector register #x through v31
vx-vy	Show/modify AltiVec vector register #x through #y

s	Show/modify entire special-purpose register family
sx	Show/modify special-purpose register # (0-1024)
SPRname	Show/modify special-purpose register by name, such as “msr”
sx.field	Show/modify field of special-purpose register # (0-1024)
SPRname.field	Show/modify field of special-purpose register by name
nb #	Show/modify Northbridge (MPC107, etc) register # (0-0xFF)

The GPRs and FPRs are always displayed as hexadecimal values. However, SPRs and NB registers may be displayed as hexadecimal values, a list of field names, or a visually expanded bit field (called ‘verbose’ mode). Refer to the ‘-e’ and ‘-v’ for the latter two, or the ENV option for RDMODE. VPRs may be displayed as hex or floating point values, in sizes from byte to word.

Note that special-purpose and Northbridge registers can only be accessed individually. They cannot be accessed as a family or with the "+" or range forms. Northbridge supports -b, -h, -w options for byte, halfword, and word access.

The vector registers are treated in the following manner: since they are 128-bit registers, they are normally treated as a collection of four 32-bit values. Thus, each 32-bit value can be set individually or as a set. Each 32-bit value is indicated by a number separated by the '_' underline character. Each 32-bit value will be shifted left into the vector register.

Example: "rm v02" modifies the contents of vector register 2. A value x will shift 32 bits into v02, x_x will shift 64 bits into v02, and x_x_x, and x_x_x_x will shift correspondingly into v02. Thus, "0_0_0_0" is required to zero out all of v02.

Alternately, the size of the register can be specified, and then DINK will prompt for each particular byte, halfword, or long value.

Examples:

```
VDINK32[MPC7450] {11} >>rm r6
  R06 = 00000000 ? 1234
VDINK32[MPC7450] {13} >>rd r6
  R6 =00001234
VDINK32[MPC7450] {11} >>rm r6=7890
  R06 = 00000000 ? 7890
VDINK32[MPC7450] {13} >>rd r6
  R6 =00007890

VDINK32[MPC7455] {2} >>rm v2
  V02= 0x00000000_00000000_00000000_00000000 ?1
VDINK32[MPC7455] {3} >>rd v2
  V02= 0x00000000_00000000_00000000_00000001
VDINK32[MPC7455] {4} >>rm v2
  V02= 0x00000000_00000000_00000000_00000001 ?abc_34e
VDINK32[MPC7455] {5} >>rd v2
  V02= 0x00000000_00000001_00000ABC_0000034E
VDINK32[MPC7455] {4} >>rm v2
  V02= 0x00000000_00000000_00000000_00000001 ?0_0_0_0
VDINK32[MPC7455] {5} >>rd v2
  V02= 0x00000000_00000000_00000000_00000000
```

```
VDINK32[MPC7455] {12} >>rm -w v2
  V02.0= = 00000002  ?1234
  V02.1= = 00000003  ?5678
  V02.2= = 00000004  ?9abc
  V02.3= = 00000003  ?def0
VDINK32[MPC7455] {12} >>rd v2
  V02= 0x00001234_00005678_00009ABC_0000DEF0
VDINK32[MPC7455] {14} >>rm -b v2
  V02.0= = 00 ?FF
  V02.1= = 00  ?
  V02.2= = 12  ?
  V02.3= = 34  ?
  V02.4= = 00  ?FF
  V02.5= = 00  ?
  V02.6= = 56  ?
  V02.7= = 78  ?
  V02.8= = 00  ?FF
  V02.9= = 00  ?
  V02.10= = 9A  ?
  V02.11= = BC  ?
  V02.12= = 00  ?FF
  V02.13= = 00  ?
  V02.14= = DE  ?
  V02.15= = F0  ?
VDINK32[MPC7455] {14} >>rd v2
  V02= 0xFF001234_FF005678_FF009ABC_FF00DEF0


DINK32 [MPC8241] >> rm nb 70
ADDR. VALUE DESCRIPTION
===== ===== ===========
0x70 0x0000 Power management config. 1
new value ? 1234

DINK32 [MPC8241] >> rd nb 70
ADDR. VALUE DESCRIPTION
===== ===== ===========
0x70 0x1234 Power management config. 1

DINK32 [MPC8241] >> rm nb -h 0xaa
  0xaa  0x1234  ? ABCD
```

3.66 Reset

Command	reset
Abbreviation	rst [-p]
Syntax	reset
Builds	All
Boards	All
Processors	All

The “reset” command attempts to reset the system, either by using the Port 92 reset facility or, if unavailable, by using a software reset (a jump, which is usually unsatisfactory).

Arguments:

p Power down the board instead of reset.

Examples:

```
DINK32 [MPC750] >> rst
Software reset

        <By default,  the system will now restart>
```

3.67 SETBAUD

Command	setbaud
Abbreviation	sb
Syntax	sb [-h] [-k] [rate]
Builds	All
Boards	All
Processors	All

The “setbaud” command sets or displays the baud rate for the host serial port (“-h” option) or the keyboard serial port (“-k” option). “-k” is assumed if not entered.

Arguments:

-k Set baud rate for keyboard port

-h Set baud rate for host port

rate A baud rate; typically 2400, 4800, 9600, 19200, 38400, 57600 or 115200

Other values may be acceptable; an error is shown if the driver does not support the requested value.

If no baud rate is entered, the current baud rate setting for the port is displayed.

Examples:

```
DINK32 [MPC8245] >>setbaud 57600
 Baud rate changing to 57600...BØ

      <NOTE: the user must then change the baud rate on the terminal to
       57600.  Press return a few times to clear any corrupted characters>

DINK32 [MPC8245] >>
```

3.68 SQUIT

Command	sim quit
Abbreviation	sq
Syntax	sq
Builds	eDINK
Boards	None
Processors	All

The “squit” command exits the simulator.

Arguments:

None.

Examples:

```
EDINK32 [E500] >> sq
```

3.69 Show SPRs

Command	show sprs
Abbreviation	sx
Syntax	sx
Builds	DINK
Boards	All
Processors	All

The “sprs” command displays all SPRs valid for the current processor. It is equivalent to “rd s+”.

Arguments:

None.

Examples:

```
DINK32 [MPC7410] >> sx
SPR       Number      Register Name
===       ======      =============
XER         1         FXU Exception Register
LR          8         Link Register
CTR         9         Counter Register
...
DINK32 [MPC7410] >>
```

3.70 STTY

Command	stty
Abbreviation	stty
Syntax	stty [+ | -] [name] [“sane”]
Builds	DINK
Boards	All
Processors	All

The “stty” command is used to control the behavior of the DINK IO library.

Arguments:

“sane” Reset all parameters to defaults

“+” Activate the selected feature

“-” De-activate the selected feature

name An IOCTL parameter, one of the following:

INLCR	If set, converts newlines to carriage returns on input.
ICRNL	If set, converts to carriage returns to newlines on input
IGNCR	If set, ignores carriage returns on input
IGNNL	If set, ignores newlines on input
IECHO	If set, echoes input characters
IUCLC	If set, converts lower-case input characters to upper case
ILCUC	If set, converts lower-case input characters to upper case
ISTRIP	If set, clears MSB bit 7 on input characters
ONLCR	If set, converts newlines to carriage returns on input
OCRNL	If set, converts to carriage returns to newlines on output
ONLRET	If set, on writing a newline, also writes a carriage return
OUCLC	If set, converts lower-case output characters to upper case
OLCUC	If set, converts lower-case input characters to upper case
OSTRIP	If set, clears MSB bit 7 on output characters

If + or - is not used, the indicated name is reported. If no arguments at all are supplied, all settings are reported.

Examples:

```
VDINK32[MPC7455] {1} >>stty
INLCR =0        ICRNL =0        IGNCR =0        IGNNL =0        IECHO =0
IUCLC =0        ILCUC =0        ISTRIP=0        ONLCR =1        OCRNL =0
ONLRET=0        OUCLC =0        OLCUC =0        OSTRIP=0
vdink32[mpc7455] {3} >>stty +ouclc
vdink32[mpc7455] {4} >>he
            sandpoint/mpc107 dink command list
  command             mnemonic    command             mnemonic
  ==================  ========    ==================  ========
  about...              ab        assemble code         as
```

```
  benchmark, code       bm        boot program          bo
...
vdink32[mpc7455] {5} >>stty -ouclc
```

3.71 Switch CPU

Command	switchcpu
Abbreviation	sc
Syntax	sc
Builds	DINK
Boards	MVP
Processors	All

The “sc” command is used to select between the primary (CPU #0) and secondary (CPU #1) processors in a multi-processor environment. Subsequent DINK commands (including “go”) are handled by the selected processor.

Arguments:

None.

Examples:

```
DINK32 [MPC7450#0] >> rd r3
  R03: 0x12341234
DINK32 [MPC7450#0] >> sc
DINK32 [MPC7450#1] >> rd r3
  R03: 0xABCDABCD
DINK32 [MPC7450#1] >>
```

3.72 SX

Command	special-purpose register display
Abbreviation	sx
Syntax	sx
Builds	DINK
Boards	all
Processors	all

The “sx” command will display the names and register numbers of all the special-purpose registers.

Examples:

```
edink[MPC85x0] {36} >>sx
          SPECIAL PURPOSE REGISTERS
  SPR       Number      Register Name
  ===       ======      =============
  XER           1       FXU Exception Register
  LR            8       Link Register
```

```
  CTR           9       Counter Register
  DEC          22       Decrementer
  SRR0         26       Status Save/Restore Register 0
  SRR1         27       Status Save/Restore Register 1
  PID0         48       Processor 0 ID Register
  DECAR        54       Decrementer Auto Reload
  CSRR0        58       Critical Status Save/Restore Register 0
  CSRR1        59       Critical Status Save/Restore Register 1
  DEAR         61       Data Exception Address Register
  ESR          62       Exception Syndrome Register
  IVPR         63       Interrupt Vector Prefix Register
  USPRG0      256       User General Purpose SPR 0
  USPRG3      259       User General Purpose SPR 3
  USPRG4      260       User General Purpose SPR 4
  USPRG5      261       User General Purpose SPR 5
  USPRG6      262       User General Purpose SPR 6
  USPRG7      263       User General Purpose SPR 7
... more...
```

3.73 SYMTAB

Command	symtab
Abbreviation	st
Syntax	st [-c] [-d label]
Builds	DINK
Boards	All
Processors	All

The “symtab” command displays DINK32 symbol table information. Some symbols are built in, created by DINK. Others are collected by the assembler during code entry. These symbols may be used as the address (@symbol) of the branch instruction while entering assembly code.

Arguments:

-c Delete all user-defined symbols

-d name Delete the named symbol. DINK labels cannot be deleted.

If no arguments are present, the symbol table is printed.

Examples:

```
DINK32 [MPC603e] >> as 160000+
0x00160000 0x60000000 nop      br1:xor r1,r2,r3
0x00160000 0x60000000 BRANCH LABEL br1:
0x00160000 0x60000000 nop      xor r3,r4,r5
0x00160004 0x60000000          br2:xor r1,r5,r6
0x00160004 0x60000000 BRANCH LABEL br2:
0x00160004 0x60000000          x
VERIFYING BRANCH LABELS.....
DINK32 [MPC603e] >> ds 160000
0x00160000 0x7c832a78 BRANCH LABEL br1:
0x00160000 0x7c832a78 xor         r03, r04, r05
DINK32 [MPC603e] >> as 160100
```

```
0x00160100 0x60000000          br3:xor r5,r6,r7
0x00160100 0x60000000 BRANCH LABEL br3:
0x00160100 0x60000000          x
VERIFYING BRANCH LABELS.....
DINK32 [MPC603e] >> st
Current list of DINK branch labels:
         KEYBOARD:    0x0
         get_char:    0x1e5e4
       write_char:    0x5fac
        TBaseInit:    0x39c4
   TBaseReadLower:    0x39e8
   TBaseReadUpper:    0x3a04
     CacheInhibit:    0x3a20
    InvEnL1Dcache:    0x3a40
      DisL1Dcache:    0x3a88
    InvEnL1Icache:    0x3aac
      DisL1Icache:    0x3b00
        BurstMode:    0x3bfc
         RamInCBk:    0x3c3c
       RamInWThru:    0x3c7c
        dink_loop:    0x5660
      dink_printf:    0x6368

Current list of USER branch labels:
              br1:    0x160000
              br2:    0x160004
              br3:    0x160100
DINK32 [MPC603e] >> st -d br2
DINK32 [MPC603e] >> st
Current list of DINK branch labels:
         KEYBOARD:    0x0
         get_char:    0x1e5e4
       write_char:    0x5fac
        TBaseInit:    0x39c4
   TBaseReadLower:    0x39e8
   TBaseReadUpper:    0x3a04
     CacheInhibit:    0x3a20
    InvEnL1Dcache:    0x3a40
      DisL1Dcache:    0x3a88
    InvEnL1Icache:    0x3aac
      DisL1Icache:    0x3b00
        BurstMode:    0x3bfc
         RamInCBk:    0x3c3c
       RamInWThru:    0x3c7c
        dink_loop:    0x5660
      dink_printf:    0x6368

Current list of USER branch labels:
              br1:    0x160000
              br3:    0x160100
DINK32 [MPC603e] >> st -c
DINK32 [MPC603e] >> st
Current list of DINK branch labels:
         KEYBOARD:    0x0
         get_char:    0x1e5e4
       write_char:    0x5fac
        TBaseInit:    0x39c4
```

```
   TBaseReadLower:    0x39e8
   TBaseReadUpper:    0x3a04
     CacheInhibit:    0x3a20
    InvEnL1Dcache:    0x3a40
      DisL1Dcache:    0x3a88
    InvEnL1Icache:    0x3aac
      DisL1Icache:    0x3b00
        BurstMode:    0x3bfc
         RamInCBk:    0x3c3c
       RamInWThru:    0x3c7c
        dink_loop:    0x5660
      dink_printf:    0x6368

Current list of USER branch labels:
DINK32 [MPC603e] >>
```

3.74 TAU

Command	tau
Abbreviation	tau
Syntax	tau [-c cal][-w][-fh]
Builds	DINK
Boards	Maximer, MVP, Sandpoint, Yellowknife
Processors	MPC750, MPC755, MPC7410

The “tau” command displays or calibrates the TAU (Thermal Assist Unit). If no option is entered, the current temperature is displayed (with or without calibration). The “^C” notation is intended to (remotely) resemble a degree notation.

Arguments:

-c val	Calibrate the TAU to the supplied correct temperature (in ^C).
-w	Display the TAU value repeatedly (until a key is pressed)
-fh	Show results in Fahrenheit.

Tau calibration values are always saved in the environment variable TAUCAL, if the environment variable storage is available and has been initialized.

Examples:

```
DINK32 [ARTHUR] >>tau
Tjc = 58 ^C (uncalibrated)
DINK32 [ARTHUR] >>tau -c 18
Tjc = 18 ^C
DINK32 [ARTHUR] >>tau -c 18
TAUCAL = 0x8E001234
DINK32 [ARTHUR] >>tau
Tjc = 18 ^C
DINK32 [ARTHUR] >>tau -fh
Tjc = 32 ^F
```

3.75 Test

Command	test
Abbreviation	te
Syntax	test [k][kl][bt][vga][hwm]
Builds	DINK
Boards	All
Processors	All

The “TE” command is an interface to specialized test code.

Arguments:

k	PS/2 keyboard test
kl	PS/2 keyboard loop.
bt	BIOS timer test
vga	VGA init
hwm	Hardware monitor test (MVP)

Examples:

```
DINK32 [ARTHUR] >>hwm
pVCORE=1.5V  OVDD=2.3V  +3.3V=3.6V  +5.0V=5.2V
```

3.76 Time

Command	time
Abbreviation	rtc
Syntax	time [-wuxb][-d #][-s [date|time]
Builds	All
Boards	MVP, Sandpoint, Yellowknife, Elysium
Processors	All

The “time” command allows setting or displaying the real-time clock available on the MVP, Sandpoint and Yellowknife systems.

Arguments:

-w	The date and time are repeatedly displayed until a key is pressed.
-u	Report UTC time as a 32-bit value (assumes RTC is set to UTC/GMT)
-x	Disable the RTC; time measurement throughout DINK will use ‘jiffy’-based decrementer timing. This is for RTC debugging.
-s val	Set the date and/or time to values specified on line
-b	Report the RTC battery status

-d # Disable DSE mode (# = 0) or enable it (# != 0)

If no argument is entered, the current date and time are displayed.

If a date or time value is entered, it is interpreted as follows:

‘/’ indicates a date value. If only one, the current year is not changed and the value is interpreted as month ‘/’ date; otherwise, it is interpreted as year ‘/’ month ‘/’ date.

‘:’ indicates a time value. If only one, the time is set to hour ‘:’ minute with seconds at zero; if two are detected, seconds are also set. A trailing ‘AM’ or ‘PM’ sets AM/PM mode; otherwise, a 24-hour clock is assumed.

Examples:

```
DINK32 [MPC8245] >> time
2001/04/15 03:38:14 PM
DINK32 [MPC8245] >> rtc -s 02/09/25 10:02:03pm
  2002/09/25 10:02:03PM
DINK32 [MPC8245] >>rtc -w
2002/09/25 10:02:14 PM
2002/09/25 10:02:15 PM
2002/09/25 10:02:16 PM
 ...
```

3.77 Transparent

Command	transparent
Abbreviation	tm
Syntax	tm
Builds	DINK
Boards	MVP, Sandpoint, Yellowknife
Processors	All

The “tm” command will put DINK32 into a transparent mode, which essentially connects the DINK console to the host port for remote access to the host system.

To exit transparent mode, enter control-A.

See Section 3.20, “Do.”

Arguments:

None.

Examples:

```
DINK32 [MPC750] >> tm
 % telnet 128.0.0.1
 ...
 <control-a>
DINK32 [MPC750] >>
```

3.78 Trace

Command	trace
Abbreviation	tr
Syntax	tr {address | +}
Builds	All
Boards	Excimer, Maximer, MVP, Sandpoint, Yellowknife
Processors	All

The “trace” command allows the user to single-step through a user program. The processor will execute a single instruction, and then return control back to DINK32. At that point the user can examine and/or modify the processor context (registers, SPRs, memory, etc.) and possibly continue executing additional instructions.

After “tr +” has been executed, it is often useful to simply enter “.” to repeat the trace command and step through code.

Arguments:

+	then the address of the instruction to execute is derived from bits 0-29 of the SRR0 (Machine Status Save / Restore) register.
address	Trace one instruction from the supplied address.

Example:

```
DINK32 [MPC8240] >> ds 2100
0x00002100 0x7c0802a6 mfspr r00, s0008
DINK32 [MPC8240] >> trace 2100
0x00002104 stw r13, 0xfff8(r01)
DINK32 [MPC8240] >> trace +
0x00002108 add r03, r00, r01
DINK32 [MPC8240] >> .
0x0000210c mfspr r04, s0274
DINK32 [MPC8240] >> tr + ; rd r3
0x00002108 add r03, r00, r01
  R03: 0x00000001
DINK32 [MPC8240] >> .
0x0000210c mfspr r04, s0274
  R03: 0x00000001
```

3.79 Upload

Command	upload
Abbreviation	up
Syntax	up {-kl-h} -b [-o offset] [-p usec] start_addr end_addr
Builds	All
Boards	All
Processors	All

The “upload” command sends memory contents to the serial port in a binary or S-record format.

Arguments:

-k	Send memory to the keyboard/console port
-h	Send memory to the host
-b	Send in binary format. Otherwise, S-records are sent.
-o offset	Add ‘offset’ to the addresses generated for S-records. Does not apply to binary downloads.
-p usec	Add a ‘usec’ delay to each character sent (for binary) or each line (for S-records). This may be needed for slow-capture devices.
start_addr	Starting address to send.
end_addr	Ending address to send.

Example:

```
DINK32 [MPC8240] >> ul 100 200
S00600004844521B
S7150000010060000000600000006000000060000000600000006000000000
...
```

3.80 Virtual Host File

Command	virtual hostfile
Abbreviation	vhf
Syntax	vhf [-r] filename
Builds	VDINK
Boards	None
Processors	All

The “vhf” command specifies the name of the source file that will be used when IO operations are performed to the host port under VDINK. Typically used with the ‘DL’ commands to load S-record or binary files into the virtual memory space.

Arguments:

-r	the file is rewound; that is, the I/O pointer is set to the start of the file. Used to restart I/O operations.
filename	The name of a Unix, Linux or PC filename to be used.

Example:

```
VDINK32 [MPC8240] >> vhf /usr/share/dink32.src
VDINK32 [MPC8240] >> dl -h -o 100000
  3456 records
VDINK32 [MPC8240] >> dl -h -o 100000
```

3.81 Virtual Memory File

Command	virtual memory
Abbreviation	vmem
Syntax	vmem [-w] filename
Builds	VDINK
Boards	None
Processors	All

The “VM” command allows reading or writing the virtual memory space of VDINK to an external file.

Arguments:

-w The virtual memory space is written to the named file.

filename The name of a Unix, Linux, or PC filename that holds an image of the VDINK virtual memory.

If ‘-w’ is not specified, the filename is read and virtual memory is set. “VM” is often used with the “AUTO” ENV variable to load a default setup.

Example:

```
VDINK32 [MPC8240] >> vmem -w /tmp/vmem
VDINK32 [MPC8240] >> env AUTO=vmem /tmp/vmem
```

3.82 VQUIT

Command	virtual quit
Abbreviation	vq
Syntax	vq
Builds	VDINK
Boards	None
Processors	All

The “VQ” command quits the VDINK and returns to the OS.

Arguments:

None.

Examples:

```
VDINK32 [MPC750] >> vq
$
```

3.83 Virtual Simulator

Command	vsim
Abbreviation	vs
Syntax	vsim [-DIrx][-f n][-d addr['-'addr]][-i addr['-'addr]][-n no] [t ['r']['w']]
Builds	VDINK
Boards	None
Processors	All

The "vs" command controls various aspects of the VDINK ISS (instruction set simulator) "VSIM".

Arguments:

-b Display the backtrace stack and related information. VSIM tracks and records each change of flow initiated by branch operations.

-d addr Set a data access breakpoint. If any data access to the address is detected, execution is interrupted.

-D Toggle data access breakpoint enable/disable (settings retained)

-f [0|1] Set the code flow trace flag to 0 or 1. If '1', VSIM prints.

-i Set an instruction access breakpoint

-I Toggle instruction access breakpoint enable/disable (settings retained)

-n ### Set the number of cycles to trace on each TR command. Default is 1, but larger numbers allow 'burst' tracing.

-r Reset the emulation system.

-t [r/w] Set type of data access breakpoint to 'r'=read, 'w'=write, or 'rw' for either.

-v ### Set the simulator verbose level; 0..n where n is fairly quiet and >=2 is quite wordy.

-x Disable IABR/DABR for one cycle, then re-enable. This allows the 'trace' and the 'go' commands to continue across a breakpoint.

Examples:

```
VDINK32 [MPC750] >> vs
Instructions: 0 issued.
        RSRV: none
        IABR: none
        DABR: none
  Flow trace: off
 Trace burst: 1 cycles
     Verbose: 0
VDINK32[MPC7450] {1} >>as 100
  00000100  00000000              .WORD     0x00000000   li r3,1234
  00000104  00000000              .WORD     0x00000000
VDINK32[MPC7450] {2} >>tr 100
  00000104  00000000              .WORD     0x00000000
VDINK32[MPC7450] {4} >>rd r3
  R3 =00001234
VDINK32[MPC7450] {5} >>vs
```

```
Instructions: 1 issued.
        RSRV: none
        IABR: none
        DABR: none
  Flow trace: off
 Trace burst: 1 cycles
     Verbose: 0
VDINK32[MPC7450] {9} >>vs -b
  Stack Information
    Maximum stack depth = 0 levels
    Maximum R1 at blr   = 00000000
    Stack backtrace
```

3.84 Wait

Command	wait
Abbreviation	wa
Syntax	[-k] [nn]
Builds	DINK
Boards	All
Processors	All

The “wait” command halts execution of DINK until the specified number of seconds have elapsed, or until a key is pressed if the ‘-k’ option is used. This command is mostly used in script files.

Arguments:

nn	Number of seconds to pause.
-k	Wait for a key, instead.

Examples:

```
DINK32 [MPC750] >> wait 10
DINK32 [MPC750] >>
```

3.85 Watch

Command	watch
Abbreviation	wat
Syntax	-rst [-e exc] [-m addr]
Builds	DINK
Boards	All
Processors	All

The “watch” command monitors and displays activities repeatedly, until a key is pressed. One monitoring capability is the number of exceptions taken; to accomplish this, the exception handler maintains a table of exception event counts. Each count is incremented on each exception.

Arguments:

-e nnn Monitors a specific exception. The argument is the exception number index, which is generally the exception address shifted right by 8 bits. For example, the interrupt exception address is 0x500, so the monitoring table index is '5'.

-m addr Monitors a memory address. Reports any changes to that address.

-r Resets to zero the exception count table

-s Monitors the exception table. That is, it prints it out.

-t Monitors

Examples:

```
DINK32 [MPC750] >> wat -t
  DEC:01234887 TBU:00000035 TBL:CE4297B3
DINK32 [MPC750] >> wat -e 5
  INT:00000337
DINK32 [MPC750] >>
```

3.86 While

Command	wh
Abbreviation	wh
Syntax	wh [-k] '{' test '}' command [command ...]
Builds	All
Boards	All
Processors	All

The "wh" command executes a DINK command as long as the specified expression is non-zero.

Arguments:

-k Check the keyboard for keypresses between each iteration of a command. If not specified, the loop cannot be interrupted.

Examples:

```
DINK32[MPC7448] {17} >> ex i=0; while { $i < 5; echo $i; md 0-100 ; ex i=$i+5 }
0
  00000000      00000000 00000000 00000000 00000000         ................
  00000010      00000000 00000000 00000000 00000000         ................
  00000020      00000000 00000000 00000000 00000000         ................
  00000030      00000000 00000000 00000000 00000000         ................
  00000040      00000000 00000000 00000000 00000000         ................
  00000050      00000000 00000000 00000000 00000000         ................
  ...
1
  00000000      00000000 00000000 00000000 00000000         ................
  00000010      00000000 00000000 00000000 00000000         ................
  00000020      00000000 00000000 00000000 00000000         ................
  00000030      00000000 00000000 00000000 00000000         ................
  00000040      00000000 00000000 00000000 00000000         ................
```

```
  00000050      00000000 00000000 00000000 00000000         ................
2
  00000000      00000000 00000000 00000000 00000000         ................
  00000010      00000000 00000000 00000000 00000000         ................
  00000020      00000000 00000000 00000000 00000000         ................
  00000030      00000000 00000000 00000000 00000000         ................
  00000040      00000000 00000000 00000000 00000000         ................
  00000050      00000000 00000000 00000000 00000000         ................
...
```

3.87 Shell Escape

Command	escape
Abbreviation	!
Syntax	! [args]
Builds	VDINK
Boards	None
Processors	All

The “!” command escapes the VDINK environment and runs the indicated command. When done, VDINK resumes execution.

Arguments:

args Command to execute.

Examples:

```
VDINK32 [MPC750] >> ! ls v*
virtual_dink_io.c       virtual_dink_s.c        virtual_dink_sim.c
VDINK32 [MPC750] >>
```

Appendix A
History of DINK32 Changes

A.1 Version 13.3, November 06, 2006

1. New platforms and parts supported:
   - Added support for HPCN, MPC8544DS, Arcadia and CDS platforms.
   - Added support for MPC8641D, MPC854x/MPC855x (including MPC8548) processors.
2. New features:
   - Caches state change for DINK32 is now possible through the ca -x option.
   - Cache state can be set from environment variables.
   - Cache invalidate command has been added.
   - Added temperature control code for Taiga.
   - On DINK internal errors, DINK now prints out GPR and other registers and a stack dump. It then halts and prompts for options. User exceptions are not be affected.
   - Parameters used for networking (such as IP addresses) are saved to environment variables.
   - For CDS boards active network presence on the TSEC ports is detected and correct port is automatically selected for networking.
   - Malloc and free now supported and accessible to user programs.
   - Download address for user programs has moved to 0x200000 to make space for current dink code and for future growth space for dink.
3. Other changes:
   - DINK image for ArgoNavis, CDS and Intrepid has grown bigger than 1MBytes requiring restructuring the map for 2MBytes image. When downloading the image to flash make sure it resides in the last 2MBytes of flash.
   - Support for Sandpoint, Mars, MVP has stopped with version 13.2.
   - Support for stand-alone DINK has stopped with version 13.1.1. (There was a stand-alone DINK release with 13.2 but it didn't work).
   - Dink for Freeserve has not been tested for this release.
4. Environment (“env”) changes:
   - Added TOFFSET variable, controls DC offset of thermal sensor.
   - Added TDISABLE variable, disables ATMS.
   - Added FANPWM variable, controls CPU fan speed.
   - Added TSHUTDOWN variable, sets the ATMS critical overheat temperature limit.
5. New shell functions/enhancements

— “dt” command--displays temperature of CPU when ATMS is enabled.
— “ct” command--configures ATMS.

6. Replace boot command with bootm command. Added full support to boot Linux with uImage kernel.

A.2 Version 13.2, July 15, 2005

1. General system initialization
   — MPC7447 and later now default to BTIC disabled.
   — Monitors and counts exception events
   — MPC7447 and later now use extended BAT sizes.
2. Breakpoints
   — Now have an option to lock breakpoints in
3. Downloading
   — Updates line count every 100 lines (at 115200 too much time was spent doing IO)
   — Uses the decrementer instead of the real-time clock
   — Uses char-by-char flow control
4. Environment space
   — Sets up malloc space (env MALLOC=start,size)
5. Generic memory engine (DD/DM/MD/MM/etc.)
   — Adds ability to set write-delay (per byte) to compensate for write recovery time of I^2C EEPROM
   — Accepts 'q' in addition to ESC/x
   — -s to summarized duplicated lines (with '...')
   — -n option to suppress reading memory (write-only)
   — Added i^2cx device driver -- extended address (> 8 bit) I^2C
   — Added phy for CDS/Ely
   — Phy addresses are aligned for halfwords.
6. go_tr1
   — go/tr verbose messages can be enabled independently of the 'global' VERBOSE using the DINKOPT flag 0x08 (i.e. DO=0x8 -- or'd with other options).
7. i^2c
   — Added 'i^2c' command; -p probes I^2C bus, otherwise like 'dd i^2c'
8. meminfo
   — Fixed inability to set SPD at higher baud rates
   — CL=nn allows overriding CL settings.
9. mem_tb
   — Memory test can be stopped with the keyboard.
   — Memory test can set a max # of errors ('-m nnn')

- mt: -l ## had stopped working; fixed
- mt: -v option shows current addr each 1 K.
- mt: -N suppresses IO for faster tests.
- mv fffff000 fffffffe 100000 would fail due to 32-bit transfer overflow. Has been fixed.
- crc was not working (at some point, the CRC32 function return was deleted).
- crc sign/verify functions added
- Memory search can specify a mask for <32-bit comparisons.

10. par_tb
    - Added watch command: monitor memory, timer, interrupts, and exception events pass halt semaphore in transfer_table.
    - Added malloc/free to transfer table
    - fi sets 'AUTO=mt -l 5 -fax 100000-7fffff0' for complete, thorough burn-in before shipping.
11. PCI commands
    - ppr works through bridges.
    - All PCI commands work with multiple PCI ports
12. Flash programming
    - fu -lq command added: reports flash type and size for given address.
    - fu help amended for MPC8240 special case
    - Added ‘-s’ for sector erase option
13. reg_pci
    - Bus extensions added for MVP, CDS, Ely.
14. reg_spr
    - Command line values (i.e. "rm ctr=2") stopped working.
    - Fixed HID0 display for A7/A7pm
    - rd SR[0:15] was intentionally suppressed, for some reason. Fixed.
    - Added dfs bits for a7pm
15. reg_swap
    - Purged write_MPC_reg/read_MPC_reg
16. reg_tb
    - If no args, display RD_LAST
    - If args, set RD_LAST
17. shell
    - Fixed error that negative numbers were not converted
    - Inserts $cmdARG (e.g. $RDARG) value, if any, between command and user-arguments
    - Will not run "AUTO=cmd" when initialization skipped (press a key during startup)
    - dlf -- new command for Excimer 'dl -fl'
    - do -- do (n) times

— while (1); -- while true do rest of line
— for (n); -- do n times
— if (n) -- conditional

A.3 Version 13.1, April 30, 2003

1. New assembler/disassembler
   — Works better
   — All known bugs fixed
   — asm can assemble all dsm generated instructions
   — Entering control-B while in the assembler adds a breakpoint at the current spot.
2. Common parts of “except2.S” and “except2e.S” moved to “common.S” for better control of DINK and eDINK source.
3. New eDINK target for e500 processors, such as the MPC8540 and MPC8560.
4. A complete “cpuinfo” database is used for DINK control.
5. Debug support for DPRINT/DEBUG_ENTRY/DEBUG_EXIT macros provided.
6. Download command (“dl”) enhancements:
   — Now works with no arguments
   — “dl -b” (binary) initial timeout is now 30 seconds.
   — SRR0 is set to the code entry (download address or address + 0x10000 if an ELF header is detected). This allows ‘tr +’ to start execution of downloaded code.
   — New default download address is now 0x100000.
7. Environment (“env”) changes:
   — Added -w ("wipe") option
   — Added L3HELP alias to L2HELP
   — L*HELP has timer option now.
   — Deleted I^2C L2CACHE/L3CACHE options (slow, never was used)
   — Fixed errors with BOOT arguments
   — NVRAM on Sandpoint/Elysium set to 8K-100 bytes (upper portion reserved for network parameters and/or RTC)
8. Apollo6 and Apollo7 MPC7455/57 supported, including L2/L3 cache changes, extended BAT tables, and PLL encoding variations
9. Only MPC8245- and MPC107-based systems set “target_type” for multi-system DINK boots.
10. The MARS Arcadia platform is detected:
    — Support for the VIA 82C686B PIPC/IO
    — TSI320 PCI-to-PCI bridge detection/initialization
11. SMP detection and setting for CPU #1 improved (MVPDINK)
12. Tracing over breakpoints now works with 'tr +' form.
13. Memory initialization:

- Support for 15row/256MB SDRAM devices
- MBEN only set to actual number of detected banks (MVLinux request)
- Fixed bug preventing double-bank SODIMM detection
- I^2C data built dynamically for non-SPD-I^2C PMCs
- Added ‘mi -r’ to force memory controller re-initialization

14. Miscellaneous command changes:
    - “fi” revamped for more platform options
    - Added “-f” flush option to cache command
    - Added -r 'remap' option to pmap (forces PCI remapping to devices that are user-enabled)
    - Added -d “dump” option to PCI register display
15. IOLIB enhancements:
    - Added sprintf(), fprintf(), and printf() interfaces
    - Added MEM: IO device driver for memory-based scripts
    - Unified all PC-16x50-based UART drivers into one file: “uart.c”
    - RTS/CTS flow control is handled for Sandpoint.
16. New RapidIO functions (custom builds)
17. New shell functions/enhancements:
    - “wait” command—waits for time or keyboard for scripts
    - “id” command—usable on more platforms
    - Command line arguments can be entered as hex (default, with or without leading “0x”, decimal (with a leading “&”) or binary (with a leading “%”).
    - “ex” command—expression calculator with no operator precedence supports values (hex/dec/bin as per the above) and +, -, *, /, |, &, ^, ~, <, <=, =, >=, >, !=, >>, <<
    - Command lines can be recalled from the history buffer with a VT52/VT102 up-arrow/down-arrow command. Requires VT emulation to be enabled on the terminal emulator program.
    - Command lines can be edited with VTxx sequences: left and right arrows move left and right in the command line; DEL deletes to the end of the line.

A.4 Version 13.0, November 1, 2002

1. Extensive rewrite to all commands to use the interactive shell and the command line parser it provides.
   The parser and tokenizer toolbox is gone (arg_tb.*, rfs_tb.*, tok.*, tok_tb.*, help.*); adding and modifying commands is vastly simpler.
2. Networking support for the RealTek 8239 is added.
   TELNET to DINK supported.
   TFTP downloads supported.
3. Default baud rate of 115200 changed.
4. Most IO is channeled through “iolib.c”, facilitating IO redirection for VDINK and other platforms.

5. Similarly, most commands use “gme.c” drivers for memory and IO access, allowing VDINK to emulate them. eDINK may also.
6. ENV is now case-insensitive; “VERBOSE” and “verbose” are equivalent.
7. RESET_BASE is no longer needed; the reset base is detected via a ‘blr next’ sequence and is globally saved.
8. More information is made available for Linux via the DINK_TRANSFER table.
9. IDE track-level drivers are added. A tiny filesystem is also provided allowing booting from disk (however, this filesystem is not compatible with anything).
10. More flexible boot options, including disk and network booting.
11. MPC8245-based systems set the secondary flash to 8-bit mode, and the flash algorithms detect and use this. MPC8240 is still 64-bit though.
12. Upon startup, the PCI bus is enumerated and enabled.
13. Flash programming for MVP and PMCs updated with blank check skips, overwrite skips, etc.
14. Support for MVP X3 including background tasks, easier memory optimization
15. Register display (‘rd’) can display FPRs and VPRs using various floating point and/or byte/word-long sizes.
16. The RTC clock setting syntax is free-form and works from the command line, instead of prompting.
17. A final-assembly test (physical assembly of the Sandpoint board) initialization command was added.
18. More memory optimization commands and data were added.
19. Virtual DINK has a simulator added. All INT instructions and most FP and VEC operations supported.
20. PCI bus number selection was replaced with an automatically detected bus number.
21. Support for the BigBend platform was removed.
22. md and mm were replaced with the more flexible dd/dm commands (invisibly). Therefore, the MDMODE option was deleted.
23. Upon startup, memory is always optimized (unless the env is wiped out or a key is pressed), so MEMOPT is no longer a special ENV variable.
24. For MPC745X processors, L2CACHE is always enabled, so the ENV variable is less useful (but still supported).

A.5 Version 12.3, October 25, 2001

1. Flag assembly of unknown SPRs. as 90000 mfspr r14,badsprname
2. Fixed the log and dl -h commands, that is, the host port
   — Restrictions on the dl -h command baud of 19200 or less supported. After the download completes, it displays the error: “ERROR: unrecognized command or symbol,” but in fact there is no error, so ignore the error message.

3. Accommodated four stack spaces, two each for each of two cpus
   — CPU0: set user's stack pointer to 0x20 bytes below the user's load address of 0x90000-32 (8ffe0) == CPU0U_STACK and 10000 bytes above the DINK stack space. So if user uses more than 0x8000 stack, it will impact the other CPU. Set DINK stack pointer to 0x80000-32 (7ffe0) == CPU0_STACK
   — CPU1: (if present) set user's stack pointer to 0x20 bytes below the user's load address of 0x89000-32 (88fe0) == CPU1U_STACK and 0x10000 bytes above the DINK stack space. So if user uses more than 0x8000 stack, it will impact DINK. Set DINK stack pointer to 0x79000-32 (78fe0) == CPU1_STACK.

   NOTE

   - DINK stack space is 0x70000 to 0x80000-32.
   - User space is 0x80000 to 0x90000-32.
   - User can download programs starting at 0x90000.

4. Installed the code to support MVP and dual vger.
   a) Galileo bridge chip memory manager
   b) Galileo bridge chip sdma/uart support
   — Change the *DINK32 Reference Manual* to reflect that we are using BAT3 to support the uart descriptors and buffers at 01c00000 (about 29 MB decimal). If user wants to use this memory space they need to fix except2.s and drivers/galileo_uart/uartMiniDriver.c.
5. Much better cache support with CCs cache management code.
   a) All cache management routines (flush, flush_disable, inval_enable) have been thoroughly proven on MPC603e, MPC8245/MPC8241/MPC8240, MPC750/MPC755, MPC7400/7410, and MPC7455, and may be useful as example code.
   b) L2 (and L3 if applicable) can now be enabled on DINK boot by setting environment variables L2CACHE and L3CACHE. DINK will always use those settings (that is, regmod no longer can change the DINK HID0, L2CR, or L3CR). The prescribed way to change L2CR or L3CR is to change env variables and reboot. The DINK HID0 is programmed to enable the maximum performance features at DINK boot up. (DINK HID0, L2CR, and L3CR are the user's initial state.)
   c) Regmod can modify the user's cache state; and the appropriate flush, disable, invalidate, and enable routines manage the transition between user cache state and DINK cache state on every go_trace and return_from_exception (or user_return).
   d) By enabling VERBOSE=1 in the environment variables the user can observe the transitions between DINK cache state and user cache state and return.
6. PCI MEM aperture set to 256 MB
7. FPRs are set to zero on reset.
8. Corrections for PCI drive strength on PMC cards
9. Set EXTROM and TBEN for MPC8245, plus reduced size write.
10. New rd/rm parser:
    a) 'rd bats' -- shows bats

b) ‘rd r’ -- shows registers in condensed column format.
c) ‘rd SPR.FIELD’ -- shows individual bits of an SPR register.
d) ‘rm SPR.FIELD’ -- shows/modifies individual bits of an SPR register.
e) In non-verbose mode, registers are not force-aligned and are printed in a denser format. ‘ENV RDMODE=e’

11. fu for local flash faster
12. id command extended for L3 support and MPC8245-standalone mode
13. reg_spr.c maintained with a database/structure system instead of individual procedures
14. gpic support and new gpic command
15. Reorganized all the makefiles to have one file for the compiler tool paths
16. Added new demos and utilities directories
17. Auto-detect memory
18. Fixed many bugs and improved register displays

A.6 Version 12.2, February 16, 2001

1. Support MPC8245, MPC7410, and MPC7450
2. Expand more() in help.c to accept ‘x’ for termination
3. Expand help menu for env
4. Two new utilities, dasm and srec2bin
5. Fixed bugs:
   #11: ds 0 as 0x now disassemble as
   #13: tr no longer trashes srr0 and srr1
   #22: ds bc 4,14,-14 and bne cr3,-14 correctly
6. One new demo, generate_s

A.7 Version 12.1, August 30, 1999

1. Improved the flash capability for Yellowknife and Sandpoint in the fu command
2. Reorganized all the demo directories into one high-level directory called “demos” and added makefile_gcc
3. User SPRs are now initialized during boot-up. No need to perform a ‘go’ command to initialize register table. Added a return path through the exception handler for user code to safely return to DINK. The routine is called user_return and is sort of a dummy exception vector that allows the exception handler to take care of all context switching between DINK and USER code.
4. Added “dev epic ISRCnt” to “dev epic” command list. This command allows the user to connect a downloaded Interrupt Service Routine to an epic interrupt vector.
5. PMC ROM support
6. Added memSpeed (memory bus speed) and processor_type (type of processor MPC603, etc) to the dink_transfer_table

A.8 Version 12.0, November 30, 1999

1. Implement a DINK transfer table to dynamically assign DINK functions such as printf, dinkloop, and getchar, in a table so that it is no longer necessary to statically determine the function address and change them in demo or dhrystones or any user program.
2. Configuration (environment variables) are saved in NVRAM for yk/sp, saved in RAM for Excimer and Maximer. New command, ‘env’, manipulates these configurations. Also implements multiple command aliases; however, da and ra are still available.
3. New command, ‘tau’, displays and/or calibrates the thermal assist unit.
4. Faster download and no need to set character delays on the serial line, implemented by turning on the duart FIFO.
5. Turn on both banks of memory in the YellowKnife and Sandpoint, now 32 MB is available on dink32 startup.
6. Improved printf format facilities, including floating point.
7. Most commands can now be placed into quiet mode, and verbose mode can be used with the -v command. Default is verbose on both, same as always, with or without ENV. The ‘-e’ mode expands fields and can be made default with env RDMODE=e. Only Excimer and Maximer require the setup, and RDMODE can be ‘Q’ (quiet), ‘E’ (expand fields), or anything else. On Excimer and Maximer it can be set up with these commands:

   env -c, env rdmode=0
8. The dl command can be placed in silent mode with the “-q”.
9. rd or rm can use these aliases for the memory register, Northbridge (nb), MPC106, MPC107, or MPC8240.
10. Fixed command termination character, ‘x’, so it will not restart if unexpected
11. Fixed problems with double prompts printed on startup with DCACHE
12. Implement a new makefile, makefile_gcc, and conform the DINK code to build with the gcc eabi compatible compiler for processors that implement the PowerPC architecture. Build and load works; all memory features are broken. This will be fixed in the next release.
13. Implemented flash programming for PCI-hosted boot ROM on YK/SP platforms. The command ‘fl -h’ transfers 512k from a specified memory location to the flash.
14. Added share memory between host and agent targets using the address translation unit (ATU).

A.9 Version 11.0.2, June 1, 1999

1. Fixed invalid caching on MPC603. MPC603 does not reset the cache invalidate bits in hardware, so added the facility in software.
2. Detects MPC107
3. About command now reports board and processor identification.
4. Improved the help facility.
5. Added makefiles for the PC, makefile_pc in every directory.

A.10 Version 11.0.1, May 1, 1999 (Not Released)

1. Change the location of stack pointer load/save. DINK code now occupies through 0x0080000.

NOTE

User code must not start earlier than 0x0090000.

2. Fixed vector alignment
3. Fixed VSCR register implementation issue
4. Fixed access issue for registers VRSAVE, RSCR, FPSCR, RTCU, RTCL, and RPA
5. Fixed HID1 display for MPC603e, MPC604e
6. Fixed breakpoint/exception problem broken in rev10.7 for MPC603e
7. Fixed location of exception vectors after EH1200, they were wrong
8. Fixed flushhead in except2.s to work correctly

A.11 Version 11.0, March 29, 1999

1. Added AltiVec support for the MAX processor
2. Added vector registers to register list
3. Added assembler disassembler code for AltiVec mnemonics
4. fl -dsi has been expanded to display the flash memory range for each sector.

A.12 Version 10.7, February 25, 1999

1. Added 1999 to copyright dates
2. Added timeout to flash_write_to_memory, so an unfinished write to flash won't last for ever. It will timeout and issue an error message.
3. Added test all flash write for protected sector and if protected issue an error and refuse the write.
4. Disable transpar,tm from Excimer.
5. Set DCFA bit from 0 to 1 for MAX chips only

A.13 Version 10.6, January 25, 1999

1. Implement the history.c file and allow the about command to use constants for Version, Revision, and Release.
2. Implement the fl –dsi and fl -se commands.
3. Automatically detect flash between Board Rev 2 and 3.
4. Removed the fw -e command from DINK32; it is only available in MDINK32.

A.14 Version 10.5, November 24, 1998

1. Changed default reset address to be -xfff0 for standalone DINK
2. Fixed bugs in trace command

A.15 Version 10.4, November 11, 1998

1. Recaptured 10.3 LED post routine in MDINK
2. Added BMC_BASE_HIGH for MPC8245/MPC8241/MPC8240 to reach the high config registers
3. Added memory test feature during POR
4. Corrected ending address for MPC8245/MPC8241/MPC8240 X4 configuration
5. Added basic support for MPC8245/MPC8241/MPC8240

A.16 Version 10.3, No Date

1. This was never released.

A.17 Version 10.2, September 11, 1998

1. This release is the same as Version 10, Revision 1.

A.18 Version 10.1, September 10, 1999

1. Enable ICACHE and DCACHE

A.19 Version 9.5, August 5, 1998

1. Implement flash commands, fw -e and basic flash erase and write support
2. Split DINK into two types: MDINK (minimal DINK) and DINK.
3. Implement support for Excimer.

A.20 Version 9.4, May 22, 1998

1. Implement L2 Backside Code
2. Turned on DCACHE and ICACHE as default at boot time
3. Added Yellowknife X4 boot code (Map A & B)

A.21 Prior to Version 9.4, Approximately October 10, 1997

1. Merged CHRP and PREP
2. Added W_ACCESS (Word access) H_ACCESS, and B_ACCESS
3. One version of DINK works with all processors: MPC601, MPC603, MPC604, and ARTHUR.

Appendix B
Example Code

The example demos described in this appendix are available on the DINK32 web site. Please note that the following examples are mainly written for the sandpoint system which is no longer supported by the current DINK32 release (version 13..3). To obtain DINK32 for sandpoint, please request DINK32 version 13.2.

Seventeen example directories are included in the DINK32 distribution. These directories include all the source files, makefiles, and readme files(s). All these directories contain examples of using the dynamic DINK addresses as described in Appendix D, "User-Callable DINK Functions."

There is one makefile for each of these demos.

makefile—UNIX metaware

The metaware compiled code will complete by returning to DINK without error.

The 17 demo programs for this release are as follows:

1. agentboot
2. dink_demo
3. Dhrystone - mwnosc
4. l1test and l1teststd
5. l2sizing
6. backsidel2test
7. lab4 (Excimer only)
8. memspeed
9. printtest
10. testfile
11. l3test
12. spinit
13. duart_8245
14. galileo (mvp only)
15. galileo_read_write (mvp only)
16. linpack
17. watch (Excimer only)

B.1 agentboot

The directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run. This example program is meant to demonstrate how to boot an MPC8245/MPC8241/MPC8240/Tsi107™-based PCI agent from agent local memory space on the Sandpoint reference platform.

B.1.1 Background

DINK32 V12.0 and later versions are currently set up so that once the host boots up from the DINK image in ROM, it then configures the agent. Once the agent is configured, its PCI command register is then set and it is allowed to boot up from the same DINK image in ROM. This example code forces the agent to boot up from code that it thinks is in ROM, but that is actually in host local memory space.

B.1.2 In This Directory

- README.txt—this appendix
- main.c—C code routines
- agentboot.s—ASM code routines
- makefile—UNIX makefile
- agentboot.txt—agentboot demo summary file including the source files and readme.

B.1.3 Assumptions

- Running on a Sandpoint reference platform.
- MPC8245/MPC8241/MPC8240/™7-based agent in 32bit PCI slot #4 (Third from PMC). See note 3 below on using alternate PCI slots.
- Running DINK32 V12.0 or later versions.

B.1.4 Usage

- Compile/assemble the code below and link into S-Record format downloadable to 0x100000 using makefile or makefile_pc. Simply type make to use the UNIX makefile, or type make -f makefile_pc to use the PC makefile.
- Download the S-Record to agent local memory using dl -k at the agent's DINK32 command prompt.
- Launch the program using go 100000 at the agent's DINK32 command prompt.
- The program should set up the agent to boot from the agent's local memory space at 0x0100. The agent boot code located there will have the agent write the value 777 (0x309) to agent local memory at 0x4C04. The user can verify this by using DINK to display that memory location by typing md 4c04 at the DINK command prompt.

B.2 Dink_demo

The demo directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run.

B.2.1 Building

The demo can be built with the UNIX command, make -f makedemo. The demo.src file can be downloaded with the DINK32 command dl -k. It can be executed with the DINK32 command, go 100000. Demo will run continuously. It can be stopped by a reset or by pressing any key.

B.2.2 Function Addresses

All DINK function addresses are determined dynamically. See Appendix D, “User-Callable DINK Functions,” for more information.

B.3 Dhrystone - mwnosc

The Dhrystone directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run. The Dhrystone directory has two subdirectories ties, MWnosc and watch. The makefile is contained in the MWnosc directory. This directory contains all the code necessary to build and run a Dhrystone benchmark program. Before starting execution, change the value of HID0 and DBAT1l. DINK32 by default starts the downloaded program with caches off and cache enabled in the DBATS. Change HID0 to 0000cc00 and DBAT1l to 12. Use these commands:
rm hid0 | 0000cc00, rm dbat1l | 12.

B.3.1 Building

The demo can be built with the UNIX command, make. After making the dhrystone src, download the file, dhry.src with the DINK32 command dl -k. Then change the HID0 register to 8000C000 and change the DBAT1L to 12.

There are two makefiles:

makefile—use the UNIX cross compiler tools for the PowerPC architecture.

It can be executed with the DINK32 command, go 100000.

B.3.2 Function Addresses

All DINK function addresses are determined dynamically. See Appendix D, “User-Callable DINK Functions,” for more information.

B.4 l1test and l1teststd

The directory contains source ‘c’ code to perform an L1 cache test.

B.4.1 Building

The example program can be built with the UNIX metaware or GNU GCC.

There is only the default target. Only the metaware makefile is available.

B.5 l2sizing

The directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run. This example program is meant to demonstrate how to detect whether a processor is an MPC740 or MPC750. It also detects the size of the L2 backside cache.

B.5.1 In this Directory

- README.txt—this appendix
- l2sizing1.c—C code routines
- l2sizing2.s—ASM code routines
- l2sizing.h—Header file
- l2sizing.src—Downloadable S-Record
- makefile—UNIX makefile

B.5.2 Assumptions

Running DINK32 V12.0 or later versions.

B.5.3 Usage

- Download the modified DINK32 V12.0 (See Notes section).
- Compile/assemble the code below and link into S-Record format downloadable to 0x100000 using makefile. Simply type make to use the UNIX makefile.
- Download the S-Record to agent local memory using dl -k at the agent's DINK32 command prompt.
- Launch the program using go 100000 at the agent's DINK32 command prompt.

B.5.4 To Build

UNIX: make [clean]

B.5.5 Notes

In order to use DINK's dynamic functions such as printf you must #include dinkusr.h and link dinkusr.s during compilation/link time. Please see Appendix D, "User-Callable DINK Functions," for more info.

B.6 backsideL2test

The directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run. This application will test the L2 cache and exercise the performance monitor. Read the l2test.readme for more information.

B.6.1 Building

The demo can be built with the UNIX command, make. The srecord file can be downloaded with the DINK32 command dl -k. It can be executed with the DINK32 command, go 100000.

makefile—used for this release of DINK32 R12 and beyond

B.6.2 Function Addresses

All DINK function addresses are determined dynamically, see Appendix D, “User-Callable DINK Functions,” for more information.

B.7 lab4 (Excimer only)

The directory contains source files that can be used to build an application that will blink the lights on the Excimer platform when it is downloaded into DINK at address 0x100000 and run. This test will only work on Excimer.

B.7.1 Building

The lab4 can be built with one makefile. It can be executed with the DINK32 command, go 100000. Lab4 will run continuously.

B.7.2 Function Addresses

All DINK function addresses are determined dynamically. See Appendix D, “User-Callable DINK Functions,” for more information.

B.8 memspeed

The directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run. This application demonstrates using the dynamic variable (and dynamic function) capability. The two variables, memSpeed (bus speed), and process_type (Processor type) are available via the dink_transfer_table as described in Appendix D, “User-Callable DINK Functions.”

It prints out the memory bus speed and processor name of the board on which it is executing.

B.8.1 Building

The demo can be built with the UNIX command, make. The memspeed.src file can be downloaded with the DINK32 command dl -k.

It can be executed with the DINK32 command, go 100000.

B.8.2 Function Addresses

All DINK function addresses and the two DINK variable addresses are determined dynamically. See Appendix D, "User-Callable DINK Functions," for more information.

B.9 printtest

The directory contains source files that can be used to build an application that can then be downloaded into DINK at address 0x100000 and run. This application will test the various printf features.

B.9.1 Building

The demo can be built with the UNIX command, make. The printtest.src file can be downloaded with the DINK32 command dl -k.

It can be executed with the DINK32 command, go 100000.

B.9.2 Function Addresses

All DINK function addresses are determined dynamically, see Appendix D, "User-Callable DINK Functions," for more information.

B.10 testfile

This directory contains source files that can be used to build an application, which is an endless loop, that can then be downloaded into DINK at address 0x100000 and run.

B.10.1 Building

The testfile can be built with the UNIX command, makefile. The demo.src file can be downloaded with the DINK32 command dl -k. It can be executed with the DINK32 command, go 100000. testfile will run continuously. It can be used to try out the breakpoint and other features of DINK32.

B.10.2 Function Addresses

All DINK function addresses are determined dynamically, see Appendix D, "User-Callable DINK Functions," for more information.

B.11 l3test

The directory contains source files that can be used to build an application that will initialize and test the L3 cache on those processors that have an L3. This is a simple test to see if the caches have been appropriately flushed, disabled, invalidated, and enabled on the various Freescale Semiconductor processors installed on Sandpoint reference platforms.

B.11.1 Building

The makefile can be invoked with two targets.

l3manage—create the l3 test executable, cachetestp.src.

There are four versions of the makefile:

metaware on unix—makefile

It can be downloaded and executed with the DINK32 commands, dl -k, go 100000. See the README file for more details.

B.11.2 Function Addresses

All DINK function addresses are determined dynamically. See Appendix D, “User-Callable DINK Functions,” for more information.

B.12 SPINIT

The “SPINIT” (pronounced “S-P-Init”) directory contains source files that can be used to build an application that will initialize the Sandpoint board with auto-sizing memory and kernel IO drivers. Unlike the DINK code, SPINIT is streamlined and omits most of the legacy support. It also contains assembly-language drivers for standard PC-16x50 UART (such as the MPC8245 and/or Sandpoint motherboard), allowing early access to console I/O for debugging.

B.12.1 Building

The makefile target is spinit.src. The s-record file code begins at 0x0, allowing it to be loaded in place of DINK. SPINIT cannot be downloaded and executed (at least, it will not do anything useful). A more typical use of SPINIT is to add custom modifications and then to replace DINK, whether with a PromJET flash emulator, a COP debugger download function, or even just a re-programmed flash. A user can use this code as a starting place for their own Sandpoint monitor code.

B.12.2 Status

The SPINIT directory is complete except for interrupt handling. Also, DUART C-language environment is not fully working (most likely bss copy is not right).

B.13 duart_8245

This test is composed of two directories, duart_8245_driver and duart_8245_isr. These directories contain source files that can be used to build an application that will use the DUART in the MPC8245. It will read characters from the input stream and echo them to the output stream via the serial port using the 8245 DUART and PIC interrupt controller. Read the src/readme.txt file for information on running this test.

B.13.1 Building

This code uses two different load addresses, both different from the normal load address. The driver uses 0x001a0000 and the isr uses 0x001c0000 load addresses. The makefile forces these load addresses upon the elf and src files. Read the readme.txt files

The code is contained in two directories:

- driver—contains the code to build the function that echoes the keyboard
- isr—interrupt service routine that is connected to the PIC

1. Build isr_uart1.src in the isr directory:
   - cd isr
   - make
   - isr_uart1.src is built to load at 0x001c0000
2. Build main.src in the driver directory
   - cd driver
   - make
   - main.src is built to load at 0x001a0000
3. Download isr_uart1.src and main.src in DINK
   - dl -k <use the terminal emulator to download isr_uart1.src>
   - dl -k <use the terminal emulator to download main.src>
4. Attach the isr to epic
   - dev epic ISRCnt 24 1c0000
   - dev epic

B.13.2 Usage

Attach the SP com1 output to a terminal emulator for com1.

Attach the 8245 DUART output to a terminal emulator for com2. Once started using the steps outlined in the build instructions above, begin typing into the com2 terminal after the download and performing the dink32 commands above.

B.14 galileo - mvp only

This directory is used to build an except2.s suitable for initializing a board using the Galileo GT642600 Discovery bridge chip. It is designed for simulation on a synopsis board; however, the s-record file could

be downloaded to a board using the Galileo bridge chip. It would require some modifying for the particular bridge chip. This is designed for the Galileo GT64260 Discovery bridge chip. It also contains the capability of running a linpack benchmark. It is called simexcept2.src.

This is only useful for someone with a hardware simulator.

B.14.1 Building

makefile target:

all:

builds an simexcept2.src file and converts it to a simulator input file sram_data.bin the user must still change the load address from @fff00000 to @0000000

B.14.2 Usage

See the readme file for more information.

B.15 galileo_read_write - mvp only

The directory contains source files that can be used to build an application that will read the Galileo GT64260 Discovery bridge chip config register, modify the PIPELINE bit (bit 13), and then write it back.

This program can be modified to read or write any Galileo configuration register by modifying the main function.

B.15.1 Building

There is one way to build it.

- metaware on unix—makefile.

B.15.2 Usage

No input is accepted. Output consists of displaying the register value when read, when modified, and then when written. See the readme file for more information.

B.16 linpack

The directory contains source files that can be used to build an application that will perform the linpack benchmark.

B.16.1 Building

There is one way to build it.

metaware on unix—makefile

B.16.2 Usage

The user is prompted for input and, when it is supplied, then the function will run and produce results. See the readme file for more information.

B.17 watch—Excimer Only

The directory contains source files that can be used to build an application that will read the time base on an Excimer.

B.17.1 Building

There is one way to build it.

metaware on unix—makefile

B.17.2 Usage

Download and execute the program.

Appendix C
Building and Extending DINK

C.1 Obtaining DINK32 Source

Limited source is available for downloading from the Freescale Semiconductor website. Full source can also be requested through the same web site. Due to website rearrangement, the easiest way to locate the DINK32 web home is to go to http://www.freescale.com and search for the keyword “DINK32.”

C.2 Building DINK32

Instructions are supplied with the full source code.

C.3 DINK Software Build Process

DINK32 is a sophisticated debug ROM program. Most hardware specific features such as the specific processor, the memory map, the target platforms, etc. are automatically detected at run time. This flexibility allows a single version of DINK32 to run on different platforms with different processors; for example, the same version of DINK32 will boot the Sandpoint as well as the Excimer and Maximer platforms with all the supported processors.

The ROM device on the Sandpoint system is the plastic leaded chip carrier (PLCC) device. Upgrading the firmware on such a system could be as easy as removing and replacing the old ROM with the new one. The ROM devices on the Excimer and Maximer platforms, however, are the thin, small, surface mount packages (TSSOP). It is not easy to remove such devices on the target hardware for upgrading. To solve this problem, Freescale Semiconductor provides a smaller version of DINK32 called MDINK. The main purpose of MDINK is to download DINK32 or another boot program to ROM, thus providing a robust way for upgrading the firmware.

There are four different versions of DINK:

1. DINK32 provides the capability to download and debug application programs on ‘classic’ cores.
2. eDINK provides the capability to download and debug application programs on e500 cores, plus the ability to simulate the e500 core processor using an instruction set simulator (ISS), an MPC8245/MPC8241/MPC8240 simulator, and an MPC8245/MPC8241/MPC8240 board, Elysium. eDINK is described in the application note AN2236.
3. MDINK32 provides the capability to download and upgrade firmware.
4. VDINK provides simulation of PowerPC processors in a UNIX environment.

All the DINK32 are available in executable form. They are delivered in the following file formats as shown in Table C-1.

Table C-1. DINK32 File Formats

Board	S record	elf
Sandpoint 13.1	dink32.src	dink32k
standalone 13.1	dink32SA.src	n/a
MVP 13.1	mvpdink32.src	mvpdink32
excimer 12.3	dink32.src	n/a
vDINK	vdink32 (Sun OS)	vdink32 (Linux)
eDINK for ISS	edinkCor.src	n/a
eDINK for 8240 ISS	edinkIss.src	n/a
eDINK for Elysium	edinkEly.src	n/a

The source files can be used to build DINK32, VDINK32, or eDINK or MDINK32. The source files are *.c, *.s, and *.h.

Other files are makefile and README.

Freescale Semiconductor uses the Metaware tool set to build MDINK32 and DINK32 in a UNIX environment. The syntax of the makefile, therefore, complies with the make program available on UNIX machines. The command to build DINK32 on a UNIX command line is ‘make dink’, and the command to build MDINK32 is ‘make mdink’.

MDINK32 is a subset of DINK32. Both versions share many source files. Of all the files that contribute to the making of MDINK32, the files that MDINK32 does not share with DINK32 is mpar_tb.c and mhelp.c. DINK32's version of mpar_tb.c is par_tb.c and mhelp.c is help.c. MDINK is only available on an Excimer board; source is no longer supported.

Both can also be built on UNIX with the GNU gcc tool set using makefile_gcc, and on a PC/NT with the Metaware tool set using makefile_pc.

When the “make dink” command is executed, the output files produced by “make” are put in the “obj” directory (See Figure C-1).

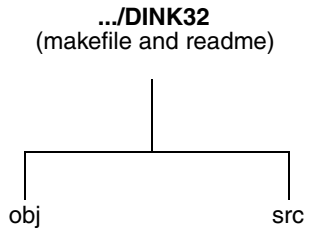

Figure C-1. DINK32/MDINK32 Directory Organization

The directory structure, as shown in Figure C-1, consists of a root directory, which contains the makefile and readme files, as well as the src and obj subdirectories. Instructions for building DINK32 are included in the full release.

C.4 Installation

DINK32 is intended to replace any existing boot code. It is not downloaded and executed but must typically be installed into a flash device.

C.4.1 Sandpoint

Using a ROM burner or in-line ROM emulator, load the dink32.src srecord file or the dink32 executable.

See Section 3.31, “FUPDATE.”

C.4.1.1 Excimer and Maximer

Using the MDINK32 facility running on an Excimer and Maximer board, download the new dink32 with the command dl -fl -o ffc00000, then using the terminal’s ascii download facility, download the dink32 sfile. See Section 3.32, “FW,” and Section 3.20, “Do.”

Steps for downloading a new DINK32 into Excimer or Maximer:

1. Connect the board to the computer by using a null-serial cable to connect port1 from Excimer or Maximer board to com1 on the host computer PC.
2. Start Hyperterminal on a Windows NT PC see Section 2.1.2.1, “Hyperterm on NT,” or a terminal emulator on a Mac see Section 2.1.2.2, “Zterm on Mac.”
3. Reset the Excimer or Maximer board and stop MDINK32 by hitting any key on the keyboard during MDINK32 startup. Perform the command, fw -e, which will erase all of flash memory and recopy MDINK32 to flash. Normally, the MDINK32 flash sector is protected and the copy will be a no operation.
4. When MDINK prompt returns, reset board.
5. Reset the baud rate by doing the following:
   - sb -k 57600
   - Press enter
   - Select ‘disconnect’ icon
   - Select ‘properties’ icon
   - Press ‘configure’ button
   - Change bits per second (baud rate) to 57600
   - Press ‘okay’ button
   - Select ‘connect’ button
   - Press enter
6. Type fl -dsi (only required on MDINK32 V10.6)
7. Type dl -fl -o ffc00000

8. Select pull-down menu ‘transfer’, use option “‘end text file’, and select the dink32.src file from the list of files.

NOTE

Do not use the option ‘send file’.

MDINK32 is not supplied as elf or sfiles on this site. However, all the code (some code is purposefully removed and the object files are substituted) is available to build MDINK32. Loading MDINK32 requires unprotecting sector 15 on the Excimer and Maximer and using some type of emulator to download the code.

Selected DINK32 code is available at this site. Some files are not released in source form, and starting with this release R12.3, no objects are supplied so that DINK32 can not be built.

All the source, including the removed code, is available from the Freescale Semiconductor confidential site and can be obtained from your Freescale Semiconductor salesperson.

C.5 Extending DINK

DINK uses a shell processor to collect various command arguments and pass them to the command function, in addition to managing help. To extend DINK functions by adding new commands, the following procedure is used:

1. The command is written in ‘C’ and linked into the DINK executable. Command arguments are passed to the function and may invoke an argument processor. When the command has completed its task, it returns a status code (SUCCESS or an error code from “errors.h”).
2. A descriptive line is added to the function “shell.c”.

C.5.1 Writing Commands

To write a new command, it is easiest to use the following template:

```
STATUS my_cmd( DINKCMD *cmd, char *args )
{
    static char *mycmd_help[] = {
     "Syntax: mycmd [-a][-n fno] dno",
      0
    };
    int     b_arg;
    char    b, *str;
    ULONG   x, no;

    b_arg = 0;
    while ((b = arg_getopts( &args, "bn#s:x#", &str, &no )) != 0) {
        switch (b) {
        case '?':   return( more_help( cmd, mycmd_help ) );

        case 'b':   b_arg++;
                    break;

        case 'n':   funcNo = no;
                    break;
        }
```

```
    }

(user commands here)

        return( SUCCESS );
}
```

Each command is passed information via the DINKCMD structure, which is documented below, and the command line arguments. If command arguments are needed, then the DINK function arg_getopts(args, template, str_ptr, long_ptr) is used to extract them as follows:

args	Supplied to the command, passed as-is
template	A string containing the arguments to collect, and the expected format. See below.
str_ptr	A pointer to a ‘char *’. If character strings are collected, the data is stored there.
long_ptr	A pointer to a long variable. If numeric values are collected, the value is stored there. If more than one argument is collected, the value must be transferred elsewhere.

On exit, the “args” character pointer points to the rest of the command line, which is often used by commands for filename collection, etc.

The format of ‘template’ is:

[a-zA-Z]	A letter alone is detected as ‘-a’, ‘-b’ etc. or in series such as ‘-abc’.
[a-zA-Z]’:’	A letter followed by ‘:’ is a string argument, and is stored in the accompanying ‘str_ptr’. The argument is re-used for multiple string arguments. The enclosing switch statement copies or processes the string argument.
[a-zA-Z]’$’	A letter followed by ‘$’ is a hexadecimal numeric argument and is converted from hex and stored in the accompanying ‘long_ptr’. As with ’:’, the argument is reused on subsequent arguments.
[a-zA-Z]’#’	A letter followed by ‘#’ is a decimal numeric argument and is converted from decimal and stored in the accompanying ‘long_ptr’. As with ’:’, the argument is reused on subsequent arguments.

If the argument ‘?’ is detected, the function more_help() is called to show the statically defined text array, in this case, mycmd_help(). Embedding the help text in the function allows the shell() command processor to have the command and the help in one place.

C.5.2 Linking Commands

Once the command has been written, it must be added to the shell processor so DINK can invoke it upon user command. To the file “shell.c”, add the following:

1. The line “extern STATUS my_cmd();” at the top of the file.
2. A line such as:
   { “MYcmd”, “My new command”, C_ALL, my_cmd },
3. In the command table “DINKCMD sh_dink_cmds[]”.

Note that commands must be entered in alphabetical order. The upper-case letters in the command indicate the minimum number of characters that must be matched. Thus, “MYcmd” can be invoked with “mycmd” or “my”, but not “m”.

Appendix D
User-Callable DINK Functions

D.1 General Information

Many library functions such as printf() and memSpeed() are available via the DINK32 debugger. In the past, it has been necessary to ascertain the address of these functions, which change with each compile, from the cross reference listing, and statically set these addresses in the programs that used these features. The demo and dhrystone directories included with the DINK32 distribution contained examples of how to set these static function addresses. With the release of DINK32 V11.1 and V12.0, these addresses are now dynamically ascertained and the user only needs to call a few functions and set up some #defines. This technique is described in this appendix. Users with access to the entire DINK32 source base can modify or add DINK32 functions. DINK32 global variables can also be ascertained from this table. R12.1 includes the two global variables, memSpeed, and process_type.

D.2 Methodology and Implementation

This method is implemented with a static structure that is filled with the current functions address during link time. The table is allocated in the file par_tb.c. Only users with access to this file can change the contents of the table, thereby determining which DINK32 functions are available. par_tb.c is only available via the Freescale Semiconductor sales office; it is not included on the web site. However, all users can use the technique for linking their code with the these DINK32 functions.

The structure is defined in dink.h as dink_exports

```
//---------------------------------------------------------------------------
// Transfer vector -- table of pointers to useful functions within DINK,
//                                      callable through R21.
//---------------------------------------------------------------------------

typedef struct st_dink_exports {                    // @ R[21]
    int     version;                                    // 0
    ULONG   *keyboard;                                  // 4
    int     (*printf)(const char*,...);                 // 8
    int     (*dink_loop)();                             // 12
    unsigned int (*is_char_in_duart)();                 // 16
    unsigned int (*menu)();                             // 20
    unsigned int (*par_about)();                        // 24
    unsigned int (*disassemble)(/*long, long*/);        // 28
    UCHAR   (*get_char)(ULONG);                         // 32
    void    (*write_char)(UCHAR);                       // 36
    ULONG   *speed_mem;                                  // 40
    char    *process_type;                              // 44
    void    (*smp_release_cpu)(unsigned long);          // 48
    int     (*read_pid)();                              // 52
    ULONG   (*pvr_read)();                              // 56
```

```
    ULONG   *memSize;                                   // 60
    ULONG   *memCL;                                     // 64
    MACINFO *MAC_info;                                  // 68
    UCHAR   *args;                                      // 72
    int     (*board_ctl)();                             // 76
    CPUINFO **ci;                                       // 80
    MCAPS   **mi;                                       // 84
    STATUS (*IDE_UsrSupport)();                         // 88
    VOIDPTR (*malloc)( size_t size );                   // 92
    void    (*free)( void *blk );                       // 96
    ULONG   *bg_halt_sema;                              // 100

  //GRENDEL
    void (*mpicISRConnect)(int vector, void (*ISRFUNCPTR)(ULONG context)); // 104
    int  (*mpicSetIntDestination)(int vect, int cpu_num, int pin_id); // 108
    void (*mpicIntEnable)(int Vect);                    // 112
    void (*mpicIntDisable)(int Vect);                   // 116
    int  (*mpicCurTaskPrioSet)(unsigned int prio);   // 120
    void (*CoreExtIntEnable)();                         // 124
    void (*CoreExtIntDisable)();                        // 128

} dink_exports;
```

and populated in par_tb.c as dink_transfer_table.

```
dink_exports dink_transfer_table = {
    6,    //  version 5 introduces the IDE_UsrSupport function
    0,
    (int (*)(const char*,...)) dink_printf,
    dink_loop,
    is_char_in_duart,
    shell_help,
    par_about,
    disassemble,
    dink_getchar,
    dink_putchar,
    &speed_mem,
    &process_type,
    smp_release_cpu,
    read_pid,
    pvr_read,
    &memSize,
    &memCL,
    &MAC_nos,
    NULL,
    board_ctl,
    &cpu_info,
    &mach_info,
    IDE_UsrSupport,
    &malloc,
    &free,
    &bg_halt_sema,
#ifdef GRENDEL
    &mpicISRConnect,
    &mpicSetIntDestination,
    &mpicIntEnable,
    &mpicIntDisable,
```

```
    &mpicCurTaskPrioSet,
    &CoreExtIntEnable,
    &CoreExtIntDisable
#else
    0,
    0,
    0,
    0,
    0,
    0,
    0
#endif
};
```

These are all of the functions and variables that are supported in V13.2. Additional or replacement DINK32 functions and global variables can be added to the table.

This table is allocated and linked into the DINK32 binaries. The user typically downloads his/her program into the starting location of free memory, at this release, address 0x90000. Unfortunately, the user program has no way of determining where the dink_transfer_table is located. Therefore when DINK32 transfers control to the user program, it sets the address of the dink_transfer_table in general-purpose register 21 in go_tr2.s. This register appears to be immune from being used by the compiler prior to the invocation of the user programs start address, usually, main(). Therefore, the user must call the supplied function, set_up_transfer_base, or equivalent, which is described below in G.4. After this call, the address of the dink_transfer_table is available to the user program.

D.3 Setting Up Static Locations

Table D-1 shows some of the functions that are currently supported.

Table D-1. DINK32 Dynamic Names

DINK32 Name	Common Name
Version of table	4
&KEYBOARD	Value of port for Keyboard support
dink_printf	Printf
dink_loop	DINK32 idle function
`is_char_in_duart`	Return 1 if character has been sent
shell.help	Entry point for DINK32 menu function
par_about	Entry point for DINK32 about function
disassemble	Entry point for DINK32 disassemble function
dink_get_char	get_char - get next character from com port
dink_put_char	put_char - send character to com port
memSpeed	Address of global variable memSpeed
process_type	Address of global variable process_type
smp_release_cpu	cpu

Table D-1. DINK32 Dynamic Names (continued)

DINK32 Name	Common Name
read_pid	Read process ID
pvr_read	Read PVR
memSize	Memory size
memCL	Memory burst rate
MVP_MACs	External MAC address for MPV
NULL	--
board_ctl	Various board functions
cpu_info	CPU information
mach_info	Board information

To change or add any new DINK32 functions or variables, one must change the dink_transfer_table.

To use any of these functions in user code, define the user code function name to be the DINK function name. For example, to link the user code printf to the DINK32 printf function, #define printf dink_printf, to link the user code put_char to DINK32 write_char, #define put_char writechar. See the directories demo and dhrystone for examples of setting up these #define statements. See the directory memspeed for an example of how to use dynamic global variables.

D.4 Using the Dynamic Functions

These functions are implemented via the assembly language file, dinkusr.s, and the include file dinkusr.h. The user includes (#include) dinkusr.h and links in dinkusr.s during compilation/link time. All of the functions in this table, except set_up_transfer_base, transfer control to the DINK32 function while leaving the link register, lr, unchanged. This effectively transfers control to the DINK32 function and the DINK32 function on completion returns directly to the caller in the user's code. The functions supplied in dinkusr.s are shown in Table D-2.

Table D-2. dinkusr.s Functions

Function Name	Function Definition
set_up_transfer_base	Capture the dink_transfer_table address from r21 and store it into a local memory cell for future use. The user must call this function before using any of the functions below, and it should be called immediately after entry, such as the first statement in main().
dink_printf	DINK32 entry into printf
dink_loop	DINK32 idle loop
`is_char_in_duart`	DINK32 function to detect if DUART has a new character
shell_help	DINK32 display menu function
par_about	DINK32 display about function
disassemble	DINK32 disassemble instruction

Table D-2. dinkusr.s Functions (continued)

Function Name	Function Definition
get_KEYBOARD	Return address of keyboard com port
dink_get_char	DINK32 get next character from the DUART buffer, essentially the keyboard for the user. This function requires the KEYBOARD value, obtained from get_KEYBOARD, as an argument. See G.6 example program _getcannon for an example of the correct way to obtain this value.
dink_put_char	DINK32 put character to the output buffer.
get_memSpeed	Returns the integer value of memSpeed example: int val; val=get_memSpeed();
get_process_type	Returns the character value of process_type example: char type; type=get_process_type();
smp_release_cpu	Holds jump address for this cpu
read_pid	Read the PID of this CPU
memSize	Returns the size of memory
memCL	Memory burst rates
NULL	Unused
board_ctl	Various control featers of the board
cpu_info	CPU information like the name, cache sizes, AltiVect.
mach_info	Board information like the BoardID, ports, io drivers.

The simple steps for using these dynamic addresses are as follows:

1. Use DINK32 V11.1 or later.
2. Use #define for local functions to be connected to the DINK32 functions example: #define printf dink_printf
3. The first executable statement in the C code must be: ‘set_up_transfer_base();’.
4. Now, whenever the program calls one of these functions, such as printf, it will transfer control to the equivalent DINK32 function.
5. Or, whenever the program needs a DINK32 global value defined in the table, call the associated get function in dinkusr.s.

D.5 Error Conditions

The only error condition is a trapword exception, which indicates that the dink_transfer_table address is zero. This is caused by one of the following conditions:

1. The user has not called set_up_transfer_base().
2. R21 is getting trashed before set_up_transfer_base() is called.

3. The DINK32 version does not support dynamic functions. DINK32 V11.0.2 was the last version that DID NOT support this feature. Ensure that you are using DINK32 V12.0 or greater.

D.6 Alternative Method for Metaware Only

While printf is fairly straightforward, scanf is more complex. In the drystone directory, a local copy of scanf is supplied in the file, support.c. Scanf and printf can also be emulated in a simpler program when using the metaware compiler. Two metaware functions are supplied to the user to give control to characters that are scanned into and out of the program buffers. Refer to the metaware documentation for more information than is given here.

When the user compiles and links with the -Hsds flag, two functions, int _putcanon(int a), and int _getcanon() are called whenever the user gets or receives a character. Thus, the user can write the simple functions shown below, and scanf and printf will use the DINK32 functions for printf and scanf. In this case, it is not necessary to use #define to change the name of the printf or scanf functions or to write one's own printf or scanf function. It is still necessary to call set_up_transfer_base() as the first statement in program.

```
/*****************************************************************
 *  Functions to capture characters from printf and scanf using
 *  the -Hsds hooks in the metaware compiler
 *  mlo  7/22/99
*****************************************************************/

#include "dinkusr.h"

int _putcanon(int a)
{
/*  grab the character sent by printf in -Hsds and
 *  use it in dink putchar
*/
char c;
        c=a;
        write_char(c);
        return 1;
}

int _getcanon()
{
/*  extract the character received by scanf in -Hsds and use
 *  it in dink putchar
 */
unsigned long key;
        key = get_KEYBOARD();
        return (get_char(key ));
}
```

Appendix E
MPC8240 Drivers

There are four drivers for the MPC8240 integrated peripheral devices.

- DMA—memory controller
- I^2C—serial controller
- I_2O—doorbell controller
- PIC—interrupt controller

Sample code for each of these drivers are in the directory, ‘drivers’, under dink32. Under the drivers directory are six directories, one for each controller. The following sections describe the driver and the sample code. Each driver is discussed in one of the following four appendices.

Note that support for the MPC8240 has stopped with DINK32 version 13.2. Make a special request to receive this version of the software.

- Section E.1, “MPC8240 DMA Memory Controller”
- Section E.2, “MPC8240 I2C Driver Library”
- Section E.3, “MPC8240 I2O Doorbell Driver”
- Section E.4, “MPC8240 PIC Interrupt Driver”

E.1 MPC8240 DMA Memory Controller

This section provides information about the generic application program interface (API) to the DMA driver library, as well as information about the implementation of the DMA driver library internals (DLI) specific to MPC8240, MPC8245, and MPC8241.

E.1.1 Background

The intended audience for this document is assumed to be familiar with the DMA protocol. This is a companion document to the MPC8240, MPC8245, and MPC8241 specifications and other documentation, which collectively give details of the DMA protocol and the MPC8240, MPC8245, and MPC8241 implementation. This document provides information about the software written to access the DMA interface for MPC8240, MPC8245, and MPC8241. This software is intended to assist in the development of higher-level applications software that uses the DMA interface.

NOTE

At the time of Revision 13.2, the DMA driver software is currently under development. The only mode that is functional is a direct transfer (chaining is not yet implemented). Only transfers to and from local memory have been tested. Controlling a remote agent processor is not yet implemented. Of the various DMA transfer control options implemented in MPC8240, MPC8245, and MPC8241, the only ones currently available in this release of the DMA library are source address, destination address, length, channel, interrupt steering and snoop enable.

E.1.2 Overview

This document consists of these parts:

- An application program interface (API) that provides a very simple, "generic," application-level, programmatic interface to the DMA driver library that hides all details of the MPC8240/MPC8245/MPC8241-specific implementation of the interface (that is, the control register, status register, embedded utilities memory block, etc.). Features provided by the MPC8240/MPC8245/MPC8241 implementation that may or may not be common with other implementations (that is, not "generic" DMA operations) are made available to the application; however, the interface is controlled by passing parameters defined in the API rather than the application having to have any knowledge of the MPC8240/MPC8245/MPC8241 implementation (i.e., registers, embedded utilities memory block, etc.) The API will be expanded to include chaining mode and additional DMA transfer control features in future releases.
- DMA API functions showing the following:
  - How the function is called (that is, function prototype), parameter definitions, possible return values, and a brief description of what the function does.
  - An explanation of how the functions are used by an application program (via examples of DINK32 usage)
- A DMA driver library internals (DLI) that provides information about the lower-level software accessing the MPC8240/MPC8245/MPC8241-specific implementation of the DMA interface
- DMA DLI functions showing the following:
  - how the function is called (that is, function prototype)
  - parameter definition possible
  - return values
  - brief description of what the function does

E.1.3 DMA Application Program Interface (API)

API functions description:

The DMA API function prototypes, defined return values, and enumerated input parameter values are declared in drivers/dma/dma_export.h.

The functions are defined in the source file drivers/dma/dma1.c.

- DMA_Status DMA_Initialize(int(*app_print_function)(char*,...));
  app_print_function is the address of the optional application's print function, otherwise NULL if not available
- Return: DMA_Status return value is either DMA_SUCCESS or DMA_ERROR.

Description:

Configure the DMA driver prior to use, as follows:

The optional print function, if supplied by the application, must be similar to the C standard library printf library function. It accepts a format string and a variable number (zero or more) of additional arguments. This optional function may be used by the library functions to report error and status condition information. If no print function is supplied by the application, the application must provide a NULL value for this parameter, in which case the library will not attempt to access a print function.

NOTE

Each DMA transfer will be configured individually by the function call that initiates the transfer. If it is desirable to establish a default configuration, these could be added as parameters. Alternately, the first (or most recent) transfer configuration values could also be used to establish defaults.

This function call triggers the DMA library to read the eumbbar so that it is available to the driver, so it is a requirement that the application first call DMA_Initialize before starting any DMA transfers. This could be eliminated if the other functions read the eumbbar if it has not already been done.

```
DMA_Status DMA_direct_transfer( DMA_INTERRUPT_STEER int_steer,
DMA_TRANSFER_TYPE type,
unsigned int source,
unsigned int dest,
unsigned int len,
DMA_CHANNEL channel,
DMA_SNOOP_MODE snoop);
```

- int_steer controls interrupt steering, use defined constants as follows:
  DMA_INT_STEER_LOCAL to steer to local processor
  DMA_INT_STEER_PCI to steer to PCI bus through INTA_
- type is the type of transfer, use defined constants as follows:
  DMA_M2M local memory to local memory (note: at the time of Revision 13.2, this is currently the only one tested)
  DMA_M2P local memory to PCI
  DMA_P2M PCI to local memory
  DMA_P2P PCI to PCI
- source is the source address of the data to transfer
- dest is the destination address, the target of the transfer
- len is the length in bytes of the data
- channel is the DMA channel to use for the transfer, use defined constants as follows:
  MPC8240, MPC8245, and MPC8241 have two channels, zero and one

  - DMA_CHN_1
  - DMA_CHN_0
- snoop controls processor snooping of the DMA channel buffer, use defined constants a follows:
  DMA_SNOOP_DISABLE
  DMA_SNOOP_ENABLE
- Return: DMA_Status return value is either DMA_SUCCESS or DMA_ERROR.

Description:

Initiate the DMA transfer.

This function does not implement any validation of the transfer. It does check the status of the DMA channel to determine if it is okay to initiate a transfer, but the application must handle verification and error conditions via the interrupt mechanisms.

E.1.4 API Example Usage

The ROM monitor program DINK32, at the time of Revision 13.2, uses the DMA API to initiate a direct data transfer in local memory only. The DINK32 program runs interactively to allow the user to transfer a block of data in local memory. DINK32 obtains information from the user as follows: interrupt steering, transfer type, source address of the data, destination (target) address, length of the data to transfer, DMA channel, and snoop control.

Note that the initialization call to configure the DMA interface is made once: the first time the user requests a DMA transfer operation. Each transmit or receive operation is initiated by a single call to a DMA API function. The DINK32 program is an interactive application, so it gives the DMA library access to its own print output function. At the time of Revision 13.2, DINK32 does not implement any handling of interrupts for error handling or completion of transfer verification.

These are the steps DINK32 takes to perform a DMA transfer:

1. Call DMA_Initialize (if first transfer) to identify the optional print function.
2. Call DMA_direct_transfer to transmit the buffer of data.

The following code samples have been excerpted from the DINK32 application to illustrate the use of the DMA API:

```
#define PRINT dink_printf
int dink_printf( unsigned char *fmt, ... )
{
/* body of application print output function, */
}
/* In the function par_devtest, for testing the DMA device interface */
{
/* initialize the DMA handler, if needed */
if ( DMAInited == 0 )
{
DMA_Status status;
if ((status = DMA_Initialize( PRINT ) ) != DMA_SUCCESS )
{
PRINT( "devtest DMA: error in initiation\n" );
return ERROR;
```

```
} else {
DMAInited = 1;
}
}
return test_dma( en_int ); /* en_int is the steering control option */
}
/*****************************************************
* function: test_dma
*
* description: run dma test
*
* note:
* test local dma channel
*****************************************************/
static STATUS test_dma( int en_int )
{
int len = 0, chn = 0;
long src = 0, dest = 0;
int mode = 0;
DMA_SNOOP_MODE snoop = DMA_SNOOP_DISABLE;
DMA_CHANNEL channel;
DMA_INTERRUPT_STEER steer;
/* The default for is en_int = 0 for DMA, this steers the DMA interrupt to the local processor.
If the DINK user puts a '+' on the command line, en_int = 1 and the steering for the DMA interrupt
is to the PCI bus through INTA_. */

steer = ( en_int == 0 ? DMA_INT_STEER_LOCAL : DMA_INT_STEER_PCI );

/* read source and destination addresses, length, type, snoop and channel */
...

/* validate and translate to API defined parameter values */
...

/* call the DMA library to initiate the transfer */
if ( DMA_direct_transfer ( steer, type, (unsigned int)src,
(unsigned int)dest, (unsigned int)len, channel, snoop) != DMA_SUCCESS )
{
PRINT( "dev DMA: error in DMA transfer test\n" );
return ERROR;
}
return SUCCESS;
}
```

E.1.5 DMA Driver Library Internals (DLI)

This information is provided to assist in further development of the DMA library.

All of these functions are defined as static in the source file drivers/dma/dma1.c.

E.1.5.1 Common Data Structures and Values

The following data structures, tables, and status values are defined (see drivers/dma/dma.h unless otherwise noted) for the MPC8240/MPC8245/MPC8241 DMA driver library functions.

```
These are the register offsets in a table of the embedded utilities memory block addresses for
the DMA registers.
#define NUM_DMA_REG 7
#define DMA_MR_REG 0
#define DMA_SR_REG 1
#define DMA_CDAR_REG 2
#define DMA_SAR_REG 3
#define DMA_DAR_REG 4
#define DMA_BCR_REG 5
#define DMA_NDAR_REG 6
```

The table that contains the addresses of the local and remote registers for both DMA channels (defined in drivers/dma/dma1.c):

```
unsigned int dma_reg_tb[][14] = {
/* local DMA registers */
{
/* DMA_0_MR */ 0x00001100,
/* DMA_0_SR */ 0x00001104,
/* DMA_0_CDAR */ 0x00001108,
/* DMA_0_SAR */ 0x00001110,
/* DMA_0_DAR */ 0x00001118,
/* DMA_0_BCR */ 0x00001120,
/* DMA_0_NDAR */ 0x00001124,
/* DMA_1_MR */ 0x00001200,
/* DMA_1_SR */ 0x00001204,
/* DMA_1_CDAR */ 0x00001208,
/* DMA_1_SAR */ 0x00001210,
/* DMA_1_DAR */ 0x00001218,
/* DMA_1_BCR */ 0x00001220,
/* DMA_1_NDAR */ 0x00001224,
},
/* remote DMA registers */
{
/* DMA_0_MR */ 0x00000100,
/* DMA_0_SR */ 0x00000104,
/* DMA_0_CDAR */ 0x00000108,
/* DMA_0_SAR */ 0x00000110,
/* DMA_0_DAR */ 0x00000118,
/* DMA_0_BCR */ 0x00000120,
/* DMA_0_NDAR */ 0x00000124,
/* DMA_1_MR */ 0x00000200,
/* DMA_1_SR */ 0x00000204,
/* DMA_1_CDAR */ 0x00000208,
/* DMA_1_SAR */ 0x00000210,
/* DMA_1_DAR */ 0x00000218,
/* DMA_1_BCR */ 0x00000220,
/* DMA_1_NDAR */ 0x00000224,
},
};

These values are the function status return values:
typedef enum _dmastatus
{
DMASUCCESS = 0x1000,
DMALMERROR,
DMAPERROR,
DMACHNBUSY,
```

```
DMAEOSINT,
DMAEOCAINT,
DMAINVALID,
DMANOEVENT,
} DMAStatus;
```

These structures reflect the bit assignments of the DMA registers.

```
typedef enum dma_mr_bit
{
IRQS = 0x00080000,
PDE = 0x00040000,
DAHTS = 0x00030000,
SAHTS = 0x0000c000,
DAHE = 0x00002000,
SAHE = 0x00001000,
PRC = 0x00000c00,
EIE = 0x00000080,
EOTIE = 0x00000040,
DL = 0x00000008,
CTM = 0x00000004,
CC = 0x00000002,
CS = 0x00000001,
} DMA_MR_BIT;
typedef enum dma_sr_bit
{
LME = 0x00000080,
PE = 0x00000010,
CB = 0x00000004,
EOSI = 0x00000002,
EOCAI = 0x00000001,
} DMA_SR_BIT;
/* structure for DMA Mode Register */
typedef struct _dma_mr
{
unsigned int reserved0 : 12;
unsigned int irqs : 1;
unsigned int pde : 1;
unsigned int dahts : 2;
unsigned int sahts : 2;
unsigned int dahe : 1;
unsigned int sahe : 1;
unsigned int prc : 2;
unsigned int reserved1 : 1;
unsigned int eie : 1;
unsigned int eotie : 1;
unsigned int reserved2 : 3;
unsigned int dl : 1;
unsigned int ctm : 1;
/* if chaining mode is enabled, any time, user can modify the
* descriptor and does not need to halt the current DMA transaction.
* Set CC bit, enable DMA to process the modified descriptors
* Hardware will clear this bit each time, DMA starts.
*/
unsigned int cc : 1;
/* cs bit has dua role, halt the current DMA transaction and
* (re)start DMA transaction. In chaining mode, if the descriptor
* needs modification, cs bit shall be used not the cc bit.
```

```
* Hardware will not set/clear this bit each time DMA transaction
* stops or starts. Software shall do it.
*
* cs bit shall not be used to halt chaining DMA transaction for
* modifying the descriptor. That is the role of CC bit.
*/
unsigned int cs : 1;
} DMA_MR;
/* structure for DMA Status register */
typedef struct _dma_sr
{
unsigned int reserved0 : 24;
unsigned int lme : 1;
unsigned int reserved1 : 2;
unsigned int pe : 1;
unsigned int reserved2 : 1;
unsigned int cb : 1;
unsigned int eosi : 1;
unsigned int eocai : 1;
} DMA_SR;
/* structure for DMA current descriptor address register */
typedef struct _dma_cdar
{
unsigned int cda : 27;
unsigned int snen : 1;
unsigned int eosie : 1;
unsigned int ctt : 2;
unsigned int eotd : 1;
} DMA_CDAR;
/* structure for DMA byte count register */
typedef struct _dma_bcr
{
unsigned int reserved : 6;
unsigned int bcr : 26;
} DMA_BCR;
/* structure for DMA Next Descriptor Address register */
typedef struct _dma_ndar
{
unsigned int nda : 27;
unsigned int ndsnen : 1;
unsigned int ndeosie: 1;
unsigned int ndctt : 2;
unsigned int eotd : 1;
} DMA_NDAR;
/* structure for DMA current transaction info */
typedef struct _dma_curr
{
unsigned int src_addr;
unsigned int dest_addr;
unsigned int byte_cnt;
} DMA_CURR;
```

E.1.6 Function Descriptions of DMA Driver Library Internals

The API function DMA_direct_transfer (described above) accepts predefined parameter values to initialize a DMA transfer. These parameters are used by the DMA driver library functions to set up the

MPC8240/MPC8245/MPC8241 DMA status and mode registers so that the application does not have to interface to the processor on such a low level. A description of the processing performed in the DMA_direct_transfer function and descriptions of the lower level DMA driver library functions follow.

This is a description of the DMA_direct_transfer processing, which initiates a simple direct transfer:

1. Read the mode register (MR) by calling DMA_Get_Mode.
2. Set the values in the mode register as follows:
   IRSQ is set from the int_steer parameter
   if steering DMA interrupts to PCI, set EIE and EOTIE
   the other mode controls are currently hard coded:
   PDE cleared
   DAHS = 3; however this is ignored because DAHE is cleared
   SAHS = 3; however this is ignored because SAHE is cleared
   PRC is cleared
   DL is cleared
   CTM is set (direct mode)
   CC is cleared
3. Validate the length of transfer value, report error and return if too large.
4. Read the current descriptor address register by calling DMA_Poke_Desp.
5. Set the values in the CDAR as follows:
   SNEN is set from the snoop parameter
   CTT is set from the type parameter
6. Write the CDAR by calling DMA_Bld_Desp, which checks the channel status to ensure it is free.
7. Write the source and destination address registers (SAR and DAR) and the byte count register (BCR) by calling DMA_Bld_Curr, which maps them according to channel and host, and ensure the channel is free.
8. Write the mode register by calling DMA_Set_Mode.
9. Begin the DMA transfer by calling DMA_Start, which ensures the channel is free and then clears and sets the mode register channel start (CS) bit.
10. The proceeding steps 6 through 9 are done in a sequence so that each call must return a successful status prior to executing the following step. The status is checked and error conditions are reported at this point if all did not execute successfully.
11. If this point is reached, the DMA transfer was initiated successfully, return success status.

These are descriptions of the DMA library functions reference above in the DMA_direct_transfer processing steps.

```
DMAStatus DMA_Get_Mode( LOCATION host,
unsigned eumbbar,
unsigned int channel,
DMA_MR *mode);
host is LOCAL or REMOTE, only LOCAL is currently tested
```

- eumbbar is EUMBBAR for LOCAL or PCSRBAR for REMOTE
- channel is DMA_CHN_0 or DMA_CHN_1

- mode is a pointer to the structure (DMA_MR) to receive the mode register contents
- Return value is DMASUCCESS or DMAINVALID

Description:

Read the DMA mode register.

```
DMAStatus DMA_Poke_Desp( LOCATION host,
unsigned eumbbar,
unsigned int channel,
DMA_CDAR *desp);
host is LOCAL or REMOTE, only LOCAL is currently tested
```

- eumbbar is EUMBBAR for LOCAL or PCSRBAR for REMOTE
- channel is DMA_CHN_0 or DMA_CHN_1
- desp is a pointer to the structure (DMA_CDAR) to receive the CDAR contents
- Return value is DMASUCCESS or DMAINVALID

Description:

Read the current descriptor address register (CDAR) specified by host and channel.

```
DMAStatus DMA_Bld_Desp( LOCATION host,
unsigned eumbbar,
unsigned int channel,
DMA_CDAR *mode);
host is LOCAL or REMOTE, only LOCAL is currently tested
```

- eumbbar is EUMBBAR for LOCAL or PCSRBAR for REMOTE
- channel is DMA_CHN_0 or DMA_CHN_1
- desp is a pointer to the structure (DMA_CDAR) holding the CDAR control bits
- Return value is DMASUCCESS, DMACHNBUSY or DMAINVALID

Description:

Set the current descriptor address register (CDAR) specified by host and channel to the given values.

```
DMAStatus DMA_Bld_Curr( LOCATION host,
unsigned eumbbar,
unsigned int channel,
DMA_CURR *desp);
```

- host is LOCAL or REMOTE—only LOCAL is currently tested
- eumbbar is EUMBBAR for LOCAL or PCSRBAR for REMOTE
- channel is DMA_CHN_0 or DMA_CHN_1
- desp is a pointer to the structure (DMA_CURR) holding the source, destination and byte count
- Return value is DMASUCCESS, DMACHNBUSY or DMAINVALID

Description:

Set the source address register (SAR), destination address register (DAR) and byte count register (BCR) specified by host and channel to the given values.

```
DMAStatus DMA_Start( LOCATION host,
unsigned eumbbar,
unsigned int channel);
```

- host is LOCAL or REMOTE, only LOCAL is currently tested

- eumbbar is EUMBBAR for LOCAL or PCSRBAR for REMOTE
- channel is DMA_CHN_0 or DMA_CHN_1
- Return value is DMASUCCESS, DMACHNBUSY or DMAINVALID

Description:

Start the DMA transfer on the specified host and channel. Ensure the channel is free, then clear and set the CS bit in the mode register. That 0 to 1 transition triggers the DMA transfer.

E.2 MPC8240 I^2C Driver Library

This section provides information about the generic application program interface (API) to the I2C driver library as well as information about the implementation of the MPC8240/MPC8245/MPC8241-specific I2C driver library internals (DLI).

E.2.1 Background

The intended audience for this document is assumed to be familiar with the I2C bus protocol. This is a companion document to the MPC8240/MPC8245/MPC8241 specification and other documentation which collectively give details of the I2C protocol and the MPC8240/MPC8245/MPC8241 implementation. This document provides information about the software written to access the I2C interface. This software is intended to assist in the development of higher-level applications software that uses the I2C interface.

NOTE

At the time of Revision 13.2, the I2c driver software is currently under development. The only modes that are functional are the master-transmit and master-receive in polling mode.

E.2.2 Overview

This section consists of the following parts:

- An application program interface (API) that provides a very simple, generic, application-level, programmatic interface to the I2C driver library that hides all details of the MPC8240/MPC8245/MPC8241-specific implementation of the I2C interface (that is, the control register, status register, embedded utilities memory block, etc.).
- I2C API functions showing the following:
  - How the function is called (i.e., function prototype)
  - Parameter definition
  - Possible return values
  - Brief description of what the function does
  - An explanation of how the functions are used by an application program (DINK32 usage employed as examples)
- An I2C driver library internals (DLI) that provides information about the lower-level software accessing the MPC8240/MPC8245/MPC8241-specific implementation of the I2C interface.

- I2C DLI functions showing the following:
  - How the function is called (i.e., function prototype)
  - Parameter definition
  - Possible return values
  - Brief description of what the function does

E.2.3 I2C Application Program Interface (API)

E.2.3.1 API Functions Description

The I2C API function prototypes, defined return values, and enumerated input parameter values are declared in drivers/i2c/i2c_export.h.

The functions are defined in the source file drivers/i2c/i2c1.c.

```
I2C_Status I2C_Initialize( unsigned char addr,
I2C_INTERRUPT_MODE en_int,
int (*app_print_function)(char *,...));
```

- addr is the MPC8240/MPC8245/MPC8241 I2C slave device address
- en_int controls the I2C interrupt enable status: I2C_INT_ENABLE = enable, I2C_INT_DISABLE = disable
- app_print_function is the address of the optional application's print function, otherwise NULL if not available
- Return: I2C_Status return value is either I2C_SUCCESS or I2C_ERROR.

Description:

Configure the I2C library prior to use, as follows:

The interrupt enable should be set to I2C_INT_DISABLE, the I2C library, at the time of Revision 13.2, only supports polling mode.

The slave address can be set to the I2C listening address of the device running the application program, but the DLI does not yet support the application's device responding as an I2C slave to another I2C master device.

The optional print function, if supplied by the application, must be similar to the C standard library printf library function. It accepts a format string and a variable number (zero or more) of additional arguments. This optional function may be used by the I2C library functions to report error and status condition information. If no print function is supplied by the application, the call to I2C_Initialize must provide a NULL value for this parameter, in which case the I2C library will not attempt to access a print function.

```
I2C_Status I2C_do_transaction( I2C_INTERRUPT_MODE en_int,
I2C_TRANSACTION_MODE act,
unsigned char i2c_addr,
unsigned char data_addr,
int len,
char *buffer,
I2C_STOP_MODE stop,
```

```
int retry,
I2C_RESTART_MODE rsta);
```

- en_int controls the I2C interrupt enable status (currently use I2C_INT_DISABLE only, polling mode)
- act is the type of transaction: I2C_MASTER_RCV or I2C_MASTER_XMIT
- i2c_addr is the I2C address of the slave device
- data_addr is the address of the data on the slave device
- len is the length in bytes of the data
- buffer is a pointer to the buffer that contains the data (xmit mode) or receives the data (rcv mode)
- stop controls sending an I2C STOP signal after completion (currently use I2C_STOP only)
- retry is the timeout retry value (currently ignored)
  rsta controls I2C restart (currently use I2C_NO_RESTART only)
- Return: I2C_Status return value is either I2C_SUCCESS or I2C_ERROR.

Description:

Act as the I2C master to transmit (or receive) a buffer of data to (or from) an I2C slave device.

At the time of Revision 13.2, this function only implements a simple master-transmit or a master-receive transaction. It does not yet support the application retaining I2C bus ownership between transactions, operating in interrupt mode, or acting as an I2C slave device.

E.2.3.2 API Example Usage

The ROM monitor program DINK32 uses the I2C API in both currently implemented modes: master-transmit and master-receive. The DINK32 program runs interactively to allow the user to transmit or receive a buffer of data from an I2C device at address 0x50 on the MPC8240/MPC8245/MPC8241 PMC card. DINK32 obtains information from the user as follows: read/write mode, I2C device address for the data (this is the address of the data on the I2C device, not the I2C bus address of the device itself, which is hard-coded in DINK32), the raw data (if in write mode), and the length of the data to transfer to or from the device. Note that the initialization call to configure the I2C interface is actually made only once, the first time the user requests an I2C transmit or receive operation. Each transmit or receive operation is performed by a single call to an I2C API function. The DINK32 program is an interactive application, so it gives the I2C library access to its own print output function.

These are the steps DINK32 takes to perform a master-transmit transaction:

1. Call I2C_Initialize (if needed) to set the MPC8240/MPC8245/MPC8241 I2C address, polling mode, and identify the optional print function.
2. Call I2C_do_transaction to transmit the buffer of data.

These are the steps DINK32 takes to perform a master-receive transaction in polling mode:

1. Call I2C_Initialize (if needed) to set the MPC8240/MPC8245/MPC8241 I2C address, polling mode, and identify the optional print function.
2. Call I2C_do_transaction to receive the buffer of data.

The following code samples have been excerpted from the DINK32 application to illustrate the use of the I2C API:

```
#define PRINT dink_printf
int dink_printf( unsigned char *fmt, ... )
{
/* body of application print output function, see Appendix ??? */
}
/* In the function par_devtest, for testing the I2C device interface */
{
/* initialize the I2C handler to I2C address 48, if needed */
if ( I2CInited == 0 )
{
I2C_Status status;
if ((status = I2C_Initialize( 48, en_int, PRINT ) ) != I2C_SUCCESS )
{
PRINT( "devtest I2C: error in initiation\n" );
return ERROR;
} else {
I2CInited = 1;
}
}
return test_i2c( action, en_int );
}
static unsigned char rcv_buffer[BUFFER_LENGTH] = { 0 };
static unsigned char xmit_buffer[BUFFER_LENGTH] = { 0 };
/*****************************************************
* function: test_i2c
*
* description: run i2c test by polling the device
*
* note:
* Test i2c device on PMC card, 0x50 serial EPROM.
* The device test data is currently only printable characters.
*
* This function gets some data from the command line, validates it,
* and calls the I2C library function to perform the task.
*****************************************************/
static STATUS test_i2c( int act, int en_int )
{
int retry = 800, len = 0, rsta = 0, addr = 0;
unsigned char eprom_addr = 0x50;
/* read transaction address */
... addr ...
/* read # of bytes to transfer */
... len ...

/* validate the data address, length, etc. */
...
/* If transmitting, get the raw data into the transmit buffer */
... xmit_buffer[] ...
/* read built-in I2C device on MPC8240/MPC8245/MPC8241 PMC card */
if ( act == DISPLAY_TAG )
{
if ( I2C_do_transaction ( en_int, I2C_MASTER_RCV, eprom_addr, addr,
len, rcv_buffer, I2C_STOP, retry, I2C_NO_RESTART ) != I2C_SUCCESS )
{
PRINT( "dev I2C: error in master receive test\n" );
```

```
return ERROR;
} else {
rcv_buffer[len] = 0; /* ensure NULL terminated string */
PRINT( "%s",rcv_buffer); /* expecting only printable data */
PRINT( "\n" );
}
}
/* write to built-in I2C device on MPC8240/MPC8245/MPC8241 PMC card */
if ( act == MODIFY_TAG )
{
if ( I2C_do_transaction ( en_int, I2C_MASTER_XMIT, eprom_addr, addr,
len, xmit_buffer, I2C_STOP, retry, I2C_NO_RESTART ) != I2C_SUCCESS )
{
PRINT( "dev I2C: error in master transmit test\n" );
return ERROR;
}
}
return SUCCESS;
}
```

E.2.4 I2C Driver Library Internals (DLI)

This information is provided to assist in further development of the I2C library to enable the application to operate as an I2C slave device, interrupt enabled mode, bus retention between consecutive transactions, correct handling of device time-out, no slave device response, no acknowledgment, I2C bus arbitration loss, etc.

All of these functions are defined as static in the source file drivers/i2c/i2c1.c.

E.2.4.1 Common Data Structures and Values

These data structures and status values are defined (see drivers/i2c/i2c.h) for the MPC8240/MPC8245/MPC8241 I2C driver library functions:

These are the offsets in the embedded utilities memory block for the I2C registers.

```
#define I2CADR 0x00003000
#define I2CFDR 0x00003004
#define I2CCR 0x00003008
#define I2CSR 0x0000300C
#define I2CDR 0x00003010
typedef enum _i2cstatus
{
I2CSUCCESS = 0x3000,
I2CADDRESS,
I2CERROR,
I2CBUFFFULL,
I2CBUFFEMPTY,
I2CXMITERROR,
I2CRCVERROR,
I2CBUSBUSY,
I2CALOSS,
I2CNOEVENT,
} I2CStatus;
```

These structures reflect the bit assignments of the I2C registers.

```
typedef struct _i2c_ctrl
{
unsigned int reserved0 : 24;
unsigned int men : 1;
unsigned int mien : 1;
unsigned int msta : 1;
unsigned int mtx : 1;
unsigned int txak : 1;
unsigned int rsta : 1;
unsigned int reserved1 : 2;
} I2C_CTRL;
typedef struct _i2c_stat
{
unsigned int rsrv0 : 24;
unsigned int mcf : 1;
unsigned int maas : 1;
unsigned int mbb : 1;
unsigned int mal : 1;
unsigned int rsrv1 : 1;
unsigned int srw : 1;
unsigned int mif : 1;
unsigned int rxak : 1;
} I2C_STAT;
Values to indicate receive or transmit mode.
typedef enum _i2c_mode
{
RCV = 0,
XMIT = 1,
} I2C_MODE;
```

E.2.5 MPC8240/MPC8245/MPC8241 I2C Driver Library Internals Function Descriptions

```
I2CStatus I2C_Init( unsigned int eumbbar,
unsigned char fdr,
unsigned char addr,
unsigned int en_int);
```

- eumbbar is the address of the embedded utilities memory block
- fdr is the frequency divider value used to set the I2C clock rate
- addr is the MPC8240/MPC8245/MPC8241 I2C slave device address
- en_int controls the I2C interrupt enable status: 1 = enable, 0 = disable
- Return: I2CStatus return value is always I2CSUCCESS.

Description:

Set the frequency divider (I2CFDR:FDR), listening address (I2CADR:[7:1]), and interrupt enable mode (I2CCR:MIEN).

```
I2C_CTRL I2C_Get_Ctrl( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block
- Return: I2C_CTRL is the contents of the I2C control register (I2CCR)

Description:

Read the I2C control register.

```
void I2C_Set_Ctrl( unsigned int eumbbar, I2C_CTRL ctrl);
```

- eumbbar is the address of the embedded utilities memory block
- ctrl is the contents of the I2C control register (I2CCR)
- Return: none

Description:

Set the I2C control register.

```
I2CStatus I2C_put( unsigned int eumbbar,
unsigned char rcv_addr,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag,
unsigned int is_cnt );
```

- eumbbar is the address of the embedded utilities memory block
- rcv_addr is the receiver's I2C device address
- buffer_ptr is pointer to the data buffer to transmit
- length is the number of bytes in the buffer
- stop_flag: 1 - signal STOP when buffer is empty
- 0 - don't signal STOP when buffer is empty
- is_cnt: 1 - this is a restart, don't check MBB
- 0 - this is a not restart, check MBB
- Returns: Any defined status indicator

Description:

Set up to send a buffer of data to the intended rcv_addr. If stop_flag is set, after the whole buffer is sent, generate a STOP signal provided that the receiver does not signal the STOP in the middle. Caller is the master performing transmitting. If no STOP signal is generated at the end of the current transaction, the master can generate a START signal to another slave address.

NOTE

The function does not actually perform the data buffer transmit. It just sets up the DLI global variables to control the transaction and calls I2C_Start to send the slave address out on the I2C bus in transmit mode. The application must check the return status to find out if the bus was obtained, then enter a loop of calling I2C_Timer_Event to poll the I2C handler to actually perform the transaction one byte at a time, while checking the return status to determine if there were any errors and whether the transaction has completed.

```
I2CStatus I2C_get( unsigned int eumbbar,
unsigned char sender_addr,
unsigned char *buffer_ptr,
unsigned int length,
```

```
unsigned int stop_flag,
unsigned int is_cnt );
```

- eumbbar is the address of the embedded utilities memory block
- sender_addr is the sender's I2C device address
- buffer_ptr is pointer to the data buffer to transmit
- length is the number of bytes in the buffer
- stop_flag: 1 - signal STOP when buffer is empty
- 0 - don't signal STOP when buffer is empty
- is_cnt: 1 - this is a restart, don't check MBB
- 0 - this is a not restart, check MBB
- Returns: Any defined status indicator

Description:

Set up to receive a buffer of data from the desired sender_addr. If stop_flag is set, when the buffer is full and the sender does not signal STOP, generate a STOP signal. Caller is the master performing receiving. If no STOP signal is generated, the master can generate a START signal to another slave address.

NOTE

The function does not actually perform the data buffer receive, it just sets up the DLI global variables to control the transaction and calls I2C_Start to send the slave address out on the I2C bus in receive mode. The application must check the return status to find out if the bus was obtained, then enter a loop of calling I2C_Timer_Event to poll the I2C handler to actually perform the transaction one byte at a time, while checking the return status to determine if there were any errors and if the transaction has completed.

```
I2CStatus I2C_Timer_Event( unsigned int eumbbar, I2CStatus (*handler)( unsigned int ) );
```

- eumbbar is the address of the embedded utilities memory block
- handler is a pointer to the function to call to handle any existing status event,
- Returns: I2CNOEVENT if there is no completed event, the I2CSR MIF bit is not set results from call to the handler function if there was a pending event completed

Description:

In polling mode, I2C_Timer_Event can be called to check the I2C status and call the given (or the default: I2C_ISR) handler function if the I2CSR MIF bit is set.

```
I2CStatus I2C_Start( unsigned int eumbbar,
unsigned char slave_addr,
I2C_MODE mode,
unsigned int is_cnt );
```

- eumbbar is the address of the embedded utilities memory block
- slave_addr is the I2C address of the receiver
- mode: XMIT(1) - put (write)
- RCV(0) - get (read)

- is_cnt: 1 - this is a restart, don't check MBB
- 0 - this is a not restart, check MBB
- Returns: Any defined status indicator

Description:

Generate a START signal in the desired mode. Caller is the master. The slave_addr is written to bits 7:1 of the I2CDR and bit 0 of the I2CDR is set to 0 for mode = XMIT or 1 for mode = RCV. A DLI-global variable MasterRcvAddress is set if mode = RCV (used by I2C_ISR function).

```
I2CStatus I2C_Stop( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block
- Returns: Any defined status indicator

Description:

Generate a STOP signal to terminate the master transaction.

```
I2CStatus I2C_Master_Xmit( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block
- Returns: Any defined status indicator

Description:

Master sends one byte of data to slave receiver. The DLI global variables ByteToXmit, XmitByte, and XmitBufEmptyStop are used to determine which data byte, or STOP, to transmit. If a data byte is sent, it is written to the I2CDR. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CSR.RXAK = 0 I2CCR.MSTA = 1 I2CCR.MTX = 1

```
I2CStatus I2C_Master_Rcv( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block
- Returns: Any defined status indicator

Description:

Master receives one byte of data from slave transmitter. The DLI global variables ByteToRcv, RcvByte, and RcvBufFulStop are used to control the accepting of the data byte or sending of a STOP if the buffer is full. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CCR.MSTA = 1 I2CCR.MTX = 0

```
I2CStatus I2C_Slave_Xmit( unsigned int eumbbar );
```

NOTE

Untested.

- eumbbar is the address of the embedded utilities memory block
- Returns: I2CSUCCESS if data byte sent
  I2CBUFFEMPTY if no data in sending buffer

Description:

Slave sends one byte of data to requesting master. The DLI global variables ByteToXmit, XmitByte, and XmitBuf are used to determine which byte, if any, to send. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CSR.RXAK = 0 I2CCR.MSTA = 0 I2CCR.MTX = 1

```
I2CStatus I2C_Slave_Rcv(unsigned int eumbbar );
```

NOTE

Untested.

- eumbbar is the address of the embedded utilities memory block
- Returns: I2CSUCCESS if data byte received
  I2CBUFFFULL if buffer is full or no more data expected

Description:

Slave receives one byte of data from master transmitter. The DLI global variables ByteToRcv, RcvByte, and RcvBufFulStop are used to control the accepting of the data byte or to set the acknowledge bit (I2CCR.TXAK) if the expected number of bytes have been received. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CCR.MSTA = 0 I2CCR.MTX = 0

```
I2CStatus I2C_Slave_Addr( unsigned int eumbbar );
```

NOTE

Untested.

- eumbbar is the address of the embedded utilities memory block
- Returns: I2CADDRESS if asked to receive data
  results from call to I2C_Slave_Xmit if asked to transmit data

Description:

Process slave address phase. Called from I2C_ISR. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MAAS = 1

```
I2CStatus I2C_ISR(unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block
- Returns:
  - — I2CADDRESS if address phase for master receive
  - — Results from call to I2C_Slave_Addr if being addressed as slave (untested)
  - — Results from call to I2C_Master_Xmit if master transmit data mode
  - — Results from call to I2C_Master_Rcv if master receive data mode
  - — Results from call to I2C_Slave_Xmit if slave transmit data mode (untested)
  - — Results from call to I2C_Slave_Rcv if slave receive data mode (untested)
  - — I2CSUCCESS if slave has not acknowledged, generated STOP (untested)

— I2CSUCCESS if master has not acknowledged, wait for STOP (untested)
— I2CSUCCESS if bus arbitration lost (untested)

Description:

Read the I2CCR and I2CSR to determine why the I2CSR.MIF bit was set which caused this function to be called. Handle condition, see above in possible return values. This function is called in polling mode as the handler function when an I2C event has occurred. It is intended to be a model for an interrupt service routine for polling mode, but this is untested and the design has not been reviewed or confirmed. This function may only be called when the following condition is met: I2CSR.MIF = 1.

NOTE

This function is tested only for the master-transmit and master-receive in polling mode. It is probably not tested even in those modes for situations when the slave does not acknowledge, bus arbitration is lost, or buffers overflow, etc.

E.2.5.1 DLI Functions Written but Not Used and Not Tested

```
I2CStatus I2C_write( unsigned int eumbbar,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag );
```

- eumbbar is the address of the embedded utilities memory block
- buffer_ptr is pointer to the data buffer to transmit
- length is the number of bytes in the buffer
- stop_flag: 1 - signal STOP when buffer is empty
- 0 - don't signal STOP when buffer is empty
- Returns: Any defined status indicator

Description:

Send a buffer of data to the requiring master. If stop_flag is set, after the whole buffer is sent, generate a STOP signal provided that the requiring receiver does not signal the STOP in the middle. Caller is the slave performing transmitting.

```
I2CStatus I2C_read( unsigned int eumbbar,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag );
```

- eumbbar is the address of the embedded utilities memory block
- buffer_ptr is pointer to the data buffer to transmit
- length is the number of bytes in the buffer
- stop_flag: 1 - signal STOP when buffer is empty
- 0 - don't signal STOP when buffer is empty
- Returns: any defined status indicator

Description:

Receive a buffer of data from the sending master. If stop_flag is set, when the buffer is full and the sender does not signal STOP, generate a STOP signal. Caller is the slave performing receiving.

E.2.6 I2C Support Functions

```
unsigned int get_eumbbar( );
```

- Returns: base address of the embedded utilities memory block

Description:

In the *MPC8245 Integrated Processor Reference Manual*, see the section, “Embedded Utilities Memory Block,” in the “Address Maps” chapter, and the section, “Configuration Register Summary,” in the “Configuration Registers” chapter, for information about the embedded utilities memory block base address register. This function is defined in kahlua.s.

```
unsigned int load_runtime_reg( unsigned int eumbbar,
& nbsp; unsigned int reg);
```

- eumbbar is the address of the embedded utilities memory block
- reg specifies the register: I2CDR, I2CFDR, I2CADR, I2CSR, I2CCR
- Returns: register content

Description:

The content of the specified register is returned. This function is defined in drivers/i2c/i2c2.s.

```
unsigned int store_runtime_reg( unsigned int eumbbar,
& nbsp; unsigned int reg,
& nbsp; unsigned int val);
```

- eumbbar is the address of the embedded utilities memory block
- offset specifies the register: I2CDR, I2CFDR, I2CADR, I2CSR, I2CCR
- val is the value to be written to the register
- Return: no return value used, it should be declared void.

Description:

The value is written to the specified register. This function is defined in drivers/i2c/i2c2.s

E.3 MPC8240 I_2O Doorbell Driver

E.3.1 I_2O Description of Doorbell Communication Between Agent and Host

The sequence of events that transpire during communication via the I_2O doorbell registers between host and agent applications running on MPC8240, MPC8245, and MPC8241 is described in this section. This implementation enables basic doorbell communication. It can be expanded to include other

MPC8240/MPC8245/MPC8241 message unit activity via the message registers and the I_2O message queue.

E.3.1.1 System Startup and Memory Map Initialization

An understanding of the agent's embedded utilities memory block base address register (EUMBBAR) and peripheral control and status registers base address register (PCSRBAR) is important for I_2O doorbell communication because both host and agent use the agent's inbound and outbound doorbell registers and message unit status and control registers. The host accesses the agent's registers via the agent's PCSR and the agent accesses its own registers via its own EUMB. It is worth noting that some registers, such as the doorbell registers, can be accessed via either the PCSR or the EUMB. Other registers, such as the message unit's status and interrupt mask registers, can only be accessed via one or the other of the PCSR or EUMB, but not both. The I_2O library functions require the caller to provide the base address (which will be either the PCSR or the EUMB) and a parameter indicating which is used. In the DINK32 environment, functions are provided to obtain both of these base addresses: get_kahlua_pcsrbar() and get_eumbbar(). The methods of setting and obtaining the PCSR and EUMB base addresses are application-specific, but the register offsets and bit definitions of the registers are specified for the MPC8240/MPC8245/MPC8241 chip memory map B and will be the same for all applications. Details about the register offsets within the EUMB and PCSR as well as bit definitions within registers are found in the *MPC8245 Integrated Processor Reference Manual* and the *MPC8240 Integrated Processor User's Manual*.

When the host and agent come up running the DINK32 application, the host application assigns the agent's PCI address for the PCSR and writes it in the agent's PCSRBAR by calling config_kahlua_agent(). The agent application initializes its own EUMBBAR [this actually happens in the KahluaInit() function, defined in .../kahlua.s] and inbound and outbound address translation windows. This is done during initialization in the main() function, main.c.

```
/*
            ** Try to enable an MPC8240/MPC8245/MPC8241 slave device. This is only enabled for
Map B.
             */
             if (address_map[0] == 'B')

                 if (target_mode == 0)

                      /* probe PCI to see if we have an MPC8240/MPC8245/MPC8241 */
                      if (pciKahluaProbe( KAHLUA_ID_LO, VENDOR_ID_HI, &target_add
r)==1)

                         PRINT("Host ....\n");
                         config_kahlua_agent( );
                         }
                     }
                 else if (target_type == (( KAHLUA_ID_LO << 16 ) | VENDOR_ID_HI ))
PRINT("Agent ....\n");
                         /* Inbound address translation */
                         sysEUMBBARWrite(L_ATU_ITWR, ATU_BASE|ATU_IW_64K);
                         pciRegSet(PCI_REG_BASE, PCI_LMBAR_REG, PCI_MEM_ADR);
                         /* Outbound address translation */
```

```
                sysEUMBBARWrite(L_ATU_OTWR, 0x100000|ATU_IW_64K);
                sysEUMBBARWrite(L_ATU_OMBAR, 0x81000000);
                }
        }
```

E.3.1.2 Interrupt Service Routines: I_2O_ISR_host() and I_2O_ISR_agent()

There is a fundamental difference in the interrupt service routine (ISR) for the host and agent: the I_2O_ISR_agent function only has to handle inbound message unit interrupts, but the I_2O_ISR_host must handle any possible interrupt from an MPC8240/MPC8245/MPC8241 (Kahlua) agent, not limited to the agent's outbound message unit. The ISRs implemented at present just check for doorbell activity. If a doorbell event occurred, the ISR prints out a simple message including the doorbell register content and the doorbell register is cleared. Otherwise, the ISR prints a message that it was unable to determine the cause of the interrupt. The I_2O_ISR_agent function checks the inbound message interrupt status register (IMISR) to determine the cause of the message unit interrupt. The message unit interrupt can occur because of doorbell, message register, or message queue activity. The ISR will distinguish and handle the interrupt accordingly; but at first stage implementation, only doorbell interrupts will be handled.

The I_2O library function I_2OInMsgStatGet() is used to read the IMISR. It returns the content of the IMISR after applying the mask value in the inbound message interrupt mask register (IMIMR) and clears the status register. The I_2O library function I_2ODBGet() is used to read the IDBR. It returns the content and clears the register. Similarly, the I_2O_ISR_host function checks the agent's outbound message interrupt status register (OMISR) to determine if the cause of the interrupt was due to the agent's outbound doorbell. It is important to note that the I_2O_ISR_host must be expanded to check for any kind of expected interrupt from the agent, not just message unit interrupts. The I_2O library function I_2OOutMsgStatGet() is used to read the OMISR. It returns the content of the OMISR after applying the mask value in the outbound message interrupt mask register (OMIMR) and clears the status register. The I_2O library function I_2ODBGet() is used to read the ODBR. It returns the content and clears the register.

The two functions I_2O_ISR_host() and I_2O_ISR_agent() are defined in the source file .../drivers/I_2O/I_2O1.c and are linked into the libdriver.a library. For testing, they are currently manually called when requested by the user in the function test_I_2O(). Eventually, the host and agent will register an interrupt service routine (ISR) with their respective embedded programmable interrupt controller (PIC) systems. Details about how to register the ISRs with PIC are not yet specified. It may take the form of a function call to an PIC-provided function that accepts a pointer to the ISR function. Alternately, it may be integrated by the linker by placing a reference to the ISR functions in some configuration table. When the integration takes place, this document will be updated to reflect the details. The code for the entire I_2O_ISR_host function follows. Note that, at the time of Revision 13.2, the only type of interrupt that is handled is doorbell interrupt from the message unit, but there are comments in the code indicating where to check for other causes of interrupts. The code can be found in I_2O1.c.

E.3.1.3 Enable Doorbell Interrupts

Since the agent is servicing the inbound doorbell, the agent enables it by calling the I_2O library function I_2ODBEnable(), which clears the inbound doorbell interrupt mask (IDIM) bit in the inbound doorbell interrupt mask register (IMIMR). The IMIMR is at offset 0x104 in the agent's embedded utilities memory block (EUMB), whose address is in the agent's EUMBBAR. Similarly, since the host is servicing the

agent's outbound doorbell, the host enables it by calling the I_2O library function I_2ODBDisable(), which clears the outbound doorbell interrupt mask (ODIM) bit in the agent's outbound message interrupt mask register (OMIMR). The OMIMR is at offset 0x34 in the agent's PCSR block, whose address is in the agent's PCSRBAR at offset 0x14 in the agent's configuration registers.

The address of the agent's configuration registers are known by the host and are accessible from the PCI bus. At present, the user interface in DINK32 allows the user to set or clear the ODIM or IDIM bit. The functions I_2ODBEnable() and I_2ODBDisable() are defined in .../drivers/I_2O/I_2O1.c to perform this task. See the code in test_I_2O() for a usage example. It is interesting to note that the observed behavior of the MPC8240/MPC8245/MPC8241 chip with regard to message unit registers is not dependant on the ODIM and IDIM bit settings Even if the ODIM or IDIM mask bits are set, writes to the affected doorbell are not blocked and the appropriate bit is set in the message unit's status register. It is up to software to apply the mask to the status register to determine whether or not to take any action. The interrupt should not occur if the mask bit is set, but this has not yet been tested.

E.3.1.4 Writing and Reading Doorbell Registers

The functions I_2ODBPost() and I_2ODBGet() are defined in .../drivers/I_2O/I_2O1.c to write a bit pattern to or return the contents of the agent's inbound and outbound doorbell registers. Note that the agent application accesses both inbound and outbound doorbell registers via its own EUMB and the host application accesses these same doorbell registers via the agent's PCSR. See the code in test_I_2O() for usage examples.

E.3.1.4.1 Host Rings an Agent via Agent's Inbound Doorbell

The host application calls the I_2O library function I_2ODBPost() to write the bit pattern to the agent's inbound door bell register (IDBR). If the inbound doorbell is enabled, this generates a message unit interrupt to the agent processor and the agent's PIC unit will execute the I_2O_ISR_agent function to determine the cause of the message unit interrupt and handle it appropriately. If the inbound doorbell is not enabled, no interrupt is generated; but the doorbell and the status register bit are still set. The agent application reads the IDBR by calling the I_2O library function I_2ODBGet(). This clears the IDBR.

E.3.1.4.2 Agent Rings a Host via Agent's Outbound Doorbell

The agent application calls the I_2O library function I_2ODBPost() to write the bit pattern to the agent's outbound door bell register (ODBR). If the outbound doorbell is enabled, this causes the outbound interrupt signal INTA_ to go active which interrupts the host processor. After the ISR is integrated into the PIC unit, this mechanism is documented here. If the outbound doorbell is not enabled, no interrupt is generated; but the doorbell and the status register bit are still set. The host application reads the ODBR by calling the I_2O library function I_2ODBGet(). This clears the ODBR.

Sample application code: The following is sample code from the DINK32 function test_i2o() in device.c that provides examples of how the I_2O library functions can be used by an application. When this section of code is entered, the DINK32 user interface has already set the local variables 'mode' and 'bit'. Mode reflects the user request. Bit is the doorbell bit number to set. Mode = 4 to manually run the ISRs for testing prior to integration with PIC.

```
/* different depending on if DINK = is running on host or agent */
   if (target_mode  0)
  {
    /*  running on host */
     unsigned int kahlua_pcsrbar  get_kahlua_pcsrbar();
    /* PRINT("kahlua's pcsrbar   0x%x\n",kahlua_pcsrbar); */
    switch (mode)
    {
      case 0:
      /* read agent's  outbound DB register and print it out */
      db_reg_content  I2ODBGet(REMOTE,kahlua_pcsrbar);
      PRINT("Agent's  outbound doorbell register: 0x%x\n",db_reg_content);
      break;

      case = 1:
      /* set  agent's inbound doorbell register */
      db_reg_content  1 << bit;
      I2ODBPost(REMOTE,kahlua_pcsrbar,db_reg_content);
      break;

      case = 2:
      /* enable  agent's outbound DB register interrupts */
      if (I2ODBEnable(REMOTE,kahlua_pcsrbar,0) ! = I2OSUCCESS)
PRINT("Cannot enable agent's outbound doorbell interrupt.\n");
      else
         PRINT("Enabled agent's outbound doorbell interrupt.\n");
       break;

      case = 3:
      /*  disable agent's outbound DB register interrupts */
      if (I2ODBDisable(REMOTE,kahlua_pcsrbar,0) ! = I2OSUCCESS)
        PRINT("Cannot disable agent's outbound doorbell interrupt.\n");
      else
         PRINT("Disabled agent's outbound doorbell interrupt.\n");
       break;

      #ifdef DBG_I2O
      case 4:
      I2O_ISR_host();
      break;
      #endif
    }
  }
  else
  {
/* running on agent */
    /*  PRINT("kahlua's eumbbar  0x%x\n",eumbbar); */
    switch (mode)
    {
      case 0:
      /* read agent's inbound  DB register and print it out */
      db_reg_content  I2ODBGet(LOCAL,eumbbar);
      PRINT("Agent's  inbound doorbell register: 0x%x\n",db_reg_content);
      break;

      case = 1:
      /* set  agent's outbound doorbell register */
```

```
      db_reg_content  1  << bit;
      I2ODBPost(LOCAL,eumbbar,db_reg_content);
       break;

      case = 2:
      /* enable  agent's inbound DB register interrupts */
      if  (I2ODBEnable(LOCAL,eumbbar,3) ! I2OSUCCESS)
         PRINT("Cannot enable agent's inbound doorbell interrupt.\n");
       else
PRINT("Enabled agent's inbound doorbell interrupt.\n");
       break;

      case = 3:
      /*  disable agent's inbound DB register interrupts */
      if  (I2ODBDisable(LOCAL,eumbbar,3) ! I2OSUCCESS)
         PRINT("Cannot disable agent's inbound doorbell  interrupt.\n");
       else
        PRINT("Disabled agent's inbound doorbell interrupt.\n");
       break;

      #ifdef DBG_I2O
      case 4:
      I2O_ISR_agent();
      break;
      #endif
    }
  }
```

E.3.1.4.3 Descriptions of the I_2O Library Functions

```
I2OSTATUS I2ODBEnable ( LOCATION loc,unsigned int base,unsigned int in_db )
```

- loc = LOCAL or REMOTE: Use LOCAL if called from agent, REMOTE if called from host. This controls the use of the base parameter as PCSR (if REMOTE) or EUMB (if LOCAL) and selection of outbound (if REMOTE) or inbound (if LOCAL) mask registers.
- base is the base address of PCSR or EUMB.
- in_db is used for LOCAL to control enabling of doorbell and/or machine check
- Returns: I2OSUCCESS

Description:

Enable the specified doorbell interrupt by clearing the appropriate mask bits.

```
I2OSTATUS I2ODBDisable( LOCATION loc,unsigned int base,unsigned int in_db )
```

- Same as I2ODBEnable, but it disables the specified interrupts by setting the mask bits.

```
unsigned int I2ODBGet( LOCATION loc,unsigned int base)
```

- loc = LOCAL or REMOTE: Use LOCAL if called from agent, REMOTE if called from host. This controls the use of the base parameter as PCSR (if REMOTE) or EUMB (if LOCAL) and selection of outbound (if REMOTE) or inbound (if LOCAL) doorbell registers.
- base is the base address of PCSR or EUMB.
- Returns:Contents of agent's inbound (if loc = LOCAL) or outbound (if loc REMOTE) doorbell register.

Description:

Returns content of specified doorbell register and clears the doorbell register.

```
void I2ODBPost( LOCATION loc,unsigned int base,unsigned int msg )
```

- loc = LOCAL or REMOTE: Use LOCAL if called from agent, REMOTE if called from host. This controls the use of the base parameter as PCSR (if REMOTE) or EUMB (if LOCAL) and selection of outbound (if REMOTE) or inbound (if LOCAL) doorbell registers.
- base is the base address of PCSR or EUMB
- msg is the 32 bit value written to the specified doorbell register

Description:

The 32 bit value is written to the specified doorbell register.

```
I2OSTATUS I2OInMsgStatGet(unsigned int eumbbarI2OIMSTAT *val)
```

- eumbbar is the base address of the agent's EUMB
- *val receives the agent's inbound message interrupt status register
- Returns: I2OSUCCESS

Description:

The agent's inbound message interrupt status register (IMISR) content is masked by the agent's inbound message interrupt mask register (IMIMR) and placed in the address given in the val parameter. The IMISR is cleared.

```
I2OSTATUS I2OOutMsgStatGet(unsigned int pcsrbar,I2OOMSTAT *val)
```

- pcsrbar is the base address of the agent's PCSR
- *val receives the agent's outbound message interrupt status register
- Returns: I2OSUCCESS

Description:
The agent's outbound message interrupt status register (OMISR content is masked by the agent's outbound message interrupt mask register (OMIMR) and placed in the address given in the val parameter. The OMISR is cleared.

E.3.2 I2C Driver Library

This section provides information about the generic application program interface (API) to the I2C driver library as well as information about the implementation of the MPC8240/MPC8245/MPC8241-specific I2C driver library internals (DLI).

E.3.2.1 Background

The intended audience for this document is assumed to be familiar with the I2C bus protocol. This is a companion document to the MPC8240, MPC8245, and MPC8241 specification and other documentation which collectively give details of the I2C protocol and the MPC8240, MPC8245, and MPC8241 implementation. This document provides information about the software written to access the I2C

interface. This software is intended to assist in the development of higher-level applications software that uses the I2C interface.

NOTE

At the time of Revision 13.2, the I2C driver software is currently under development. The only modes that are functional are the master-transmit and master-receive in polling mode.

E.3.2.2 Overview

This document consists of these parts:

- An application program interface (API) which provides a very simple, generic, application-level programmatic interface to the I2C driver library that hides all details of the MPC8240/MPC8245/MPC8241-specific implementation of the I2C interface (that is, the control register, status register, embedded utilities memory block, etc.).
- I2C API functions showing the following:
  - How the function is called (that is, function prototype)
  - Parameter definition possible
  - Return values
  - A brief description of what the function does
  - An explanation of how the functions are used by an application program (DINK32 usage employed as examples)
- An I2C driver library internals (DLI) that provides information about the lower-level software that is accessing the MPC8240/MPC8245/MPC8241-specific implementation of the I2C interface.
- I2C DLI functions showing the following:
  - How the function is called (that is, function prototype)
  - Parameter definition
  - Possible return values
  - Brief description of what the function does

E.3.2.3 I2C Application Program Interface (API)

E.3.2.3.1 API Functions Description

The I2C API function prototypes, defined return values, and enumerated input parameter values are declared in drivers/i2c/i2c_export.h. The functions are defined in the source file drivers/i2c/i2c1.c.

```
I2C_Status I2C_Initialize( unsigned char addr, I2C_INTERRUPT_MODE en_int, int
(*app_print_function)(char *,...));
addr is the MPC8240/MPC8245/MPC8241 chip's I2C slave device address
```

- en_int controls the I2C interrupt enable status: I2C_INT_ENABLE = enable, I2C_INT_DISABLE = disable
- app_print_function is the address of the optional application's print function, otherwise NULL if not available

- Return: I2C_Status return value is either I2C_SUCCESS or I2C_ERROR.

Description:

Configure the I2C library prior to use as follows:

- The interrupt enable should be set to I2C_INT_DISABLE, the I2C library, at the time of Revision 13.2, only supports polling mode.
- The slave address can be set to the I2C listening address of the device running the application program, but the DLI does not yet support the application's device responding as an I2C slave to another I2C master device.
- The optional print function, if supplied by the application, must be similar to the C standard library printf library function. It accepts a format string and a variable number (zero or more) of additional arguments. This optional function may be used by the I2C library functions to report error and status condition information. If no print function is supplied by the application, the call to I2C_Initialize must provide a NULL value for this parameter, in which case the I2C library will not attempt to access a print function.

```
I2C_Status I2C_do_transaction( I2C_INTERRUPT_MODE en_int,
I2C_TRANSACTION_MODE act,
unsigned char i2c_addr,
unsigned char data_addr,
int len,
char *buffer,
I2C_STOP_MODE stop,
int retry,
I2C_RESTART_MODE rsta);
Where:
```

- en_int controls the I2C interrupt enable status (currently use I2C_INT_DISABLE only, polling mode)
- act is the type of transaction: I2C_MASTER_RCV or I2C_MASTER_XMIT
- i2c_addr is the I2C address of the slave device
- data_addr is the address of the data on the slave device
- len is the length in bytes of the data
- buffer is a pointer to the buffer that contains the data (xmit mode) or receives the data (rcv mode)
- stop controls sending an I2C STOP signal after completion (currently use I2C_STOP only)
- retry is the timeout retry value (currently ignored)
- rsta controls I2C restart (currently use I2C_NO_RESTART only)
- Return: I2C_Status return value is either I2C_SUCCESS or I2C_ERROR.

Description:

Act as the I2C master to transmit (or receive) a buffer of data to (or from) an I2C slave device.

At the time of Revision 13.2, this function only implements a simple master-transmit or a master-receive transaction. It does not yet support the application retaining I2C bus ownership between transactions, operating in interrupt mode, or acting as an I2C slave device.

E.3.2.3.2 API Example Usage

The ROM monitor program DINK32 uses the I2C API in both currently implemented modes: master-transmit and master-receive. The DINK32 program runs interactively to allow the user to transmit or receive a buffer of data from an I2C device at address 0x50 on the MPC8240/MPC8245/MPC8241 PMC card. DINK32 obtains information from the user as follows: read/write mode, I2C device address for the data (this is the address of the data on the I2C device, not the I2C bus address of the device itself, which is hard-coded in DINK32), the raw data (if in write mode), and the length of the data to transfer to or from the device. Note that the initialization call to configure the I2C interface is actually made only once: the first time the user requests an I2C transmit or receive operation. Each transmit or receive operation is performed by a single call to an I2C API function. The DINK32 program is an interactive application, so it gives the I2C library access to its own print output function.

These are the steps DINK32 takes to perform a master-transmit transaction:

1. Call I2C_Initialize (if needed) to set the MPC8240/MPC8245/MPC8241 I2C address, polling mode, and identify the optional print function.
2. Call I2C_do_transaction to transmit the buffer of data.

These are the steps DINK32 takes to perform a master-receive transaction in polling mode:

1. Call I2C_Initialize (if needed) to set the MPC8240/MPC8245/MPC8241 I2C address, polling mode, and identify the optional print function.
2. Call I2C_do_transaction to receive the buffer of data.

The following code samples have been excerpted from the DINK32 application to illustrate the use of the I2C API from par_devtest in device.c:

```
#define PRINT dink_printf
int dink_printf( unsigned char *fmt, ... )
{
/* body of application print output function, */
}
/* In the function par_devtest, for testing the I2C device interface */
{
/* initialize the I2C handler to I2C address 48, if needed */
if ( I2CInited == 0 )
{
I2C_Status status;
if ((status = I2C_Initialize( 48, en_int, PRINT ) ) != I2C_SUCCESS )
{
PRINT( "devtest I2C: error in initiation\n" );
return ERROR;
} else {
I2CInited = 1;
}
}
return test_i2c( action, en_int );
}
static unsigned char rcv_buffer[BUFFER_LENGTH] = { 0 };
static unsigned char xmit_buffer[BUFFER_LENGTH] = { 0 };
/*****************************************************
* function: test_i2c
*
* description: run i2c test by polling the device
```

```
*
* note:
* Test i2c device on PMC card, 0x50 serial EPROM.
* The device test data is currently only printable characters.
*
* This function gets some data from the command line, validates it,
* and calls the I2C library function to perform the task.
*****************************************************/
static STATUS test_i2c( int act, int en_int )
{
int retry = 800, len = 0, rsta = 0, addr = 0;
unsigned char eprom_addr = 0x50;
/* read transaction address */
... addr ...
/* read # of bytes to transfer */
... len ...

/* validate the data address, length, etc. */
...
/* If transmitting, get the raw data into the transmit buffer */
... xmit_buffer[] ...
/* read built-in I2C device on the MPC8240/MPC8245/MPC8241 PMC card */
if ( act == DISPLAY_TAG )
{
if ( I2C_do_transaction ( en_int, I2C_MASTER_RCV, eprom_addr, addr,
len, rcv_buffer, I2C_STOP, retry, I2C_NO_RESTART ) != I2C_SUCCESS )
{
PRINT( "dev I2C: error in master receive test\n" );
return ERROR;
} else {
rcv_buffer[len] = 0; /* ensure NULL terminated string */
PRINT( "%s",rcv_buffer); /* expecting only printable data */
PRINT( "\n" );
}
}
/* write to built-in I2C device on the MPC8240/MPC8245/MPC8241 PMC card */
if ( act == MODIFY_TAG )
{
if ( I2C_do_transaction ( en_int, I2C_MASTER_XMIT, eprom_addr, addr,
len, xmit_buffer, I2C_STOP, retry, I2C_NO_RESTART ) != I2C_SUCCESS )
{
PRINT( "dev I2C: error in master transmit test\n" );
return ERROR;
}
}
return SUCCESS;
}
```

E.3.2.4 I2C Driver Library Internals (DLI)

This information is provided to assist in further development of the I2C library to enable the application to operate as an I2C slave device, interrupt enabled mode, bus retention between consecutive transactions, correct handling of device time out, no slave device response, no acknowledgment, I2C bus arbitration loss, etc.

All of these functions are defined as static in the source file drivers/i2c/i2c1.c.

E.3.2.4.1 Common Data Structures and Values

These data structures and status values are defined (see drivers/i2c/i2c.h) for the MPC8240/MPC8245/MPC8241 I2C driver library functions:

These are the offsets in the embedded utilities memory block for the I2C registers.

```
#define I2CADR 0x00003000
#define I2CFDR 0x00003004
#define I2CCR 0x00003008
#define I2CSR 0x0000300C
#define I2CDR 0x00003010
typedef enum _i2cstatus
{
I2CSUCCESS = 0x3000,
I2CADDRESS,
I2CERROR,
I2CBUFFFULL,
I2CBUFFEMPTY,
I2CXMITERROR,
I2CRCVERROR,
I2CBUSBUSY,
I2CALOSS,
I2CNOEVENT,
} I2CStatus;
These structures reflect the bit assignments of the I2C registers.
typedef struct _i2c_ctrl
{
unsigned int reserved0 : 24;
unsigned int men : 1;
unsigned int mien : 1;
unsigned int msta : 1;
unsigned int mtx : 1;
unsigned int txak : 1;
unsigned int rsta : 1;
unsigned int reserved1 : 2;
} I2C_CTRL;
typedef struct _i2c_stat
{
unsigned int rsrv0 : 24;
unsigned int mcf : 1;
unsigned int maas : 1;
unsigned int mbb : 1;
unsigned int mal : 1;
unsigned int rsrv1 : 1;
unsigned int srw : 1;
unsigned int mif : 1;
unsigned int rxak : 1;
} I2C_STAT;
Values to indicate receive or transmit mode.
typedef enum _i2c_mode
{
RCV = 0,
XMIT = 1,
} I2C_MODE;
```

E.3.2.4.2 MPC8240/MPC8245/MPC8241 I2C Driver Library Internals Function Descriptions

```
I2CStatus I2C_Init( unsigned int eumbbar,
unsigned char fdr,
unsigned char addr,
unsigned int en_int);
```

- eumbbar is the address of the embedded utilities memory block.
- fdr is the frequency divider value used to set the I2C clock rate.
- addr is the MPC8240/MPC8245/MPC8241 chip's I2C slave device address.
- en_int controls the I2C interrupt enable status: 1 = enable, 0 = disable.
- Return: I2CStatus return value is always I2CSUCCESS.

Description:

Set the frequency divider (I2CFDR:FDR), listening address (I2CADR:[7:1]), and interrupt enable mode (I2CCR:MIEN).

```
I2C_CTRL I2C_Get_Ctrl( unsigned int eumbbar );
```

: eumbbar is the address of the embedded utilities memory block

- Return: I2C_CTRL is the contents of the I2C control register (I2CCR)

Description:

Read the I2C control register.

```
void I2C_Set_Ctrl( unsigned int eumbbar, I2C_CTRL ctrl);
```

- eumbbar is the address of the embedded utilities memory block
- ctrl is the contents of the I2C control register (I2CCR)
- Return: none

Description:

Set the I2C control register.

```
I2CStatus I2C_put( unsigned int eumbbar,
unsigned char rcv_addr,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag,
unsigned int is_cnt );
```

- eumbbar is the address of the embedded utilities memory block
  rcv_addr is the receiver's I2C device address
- buffer_ptr is pointer to the data buffer to transmit
- length is the number of bytes in the buffer
- stop_flag: 1 - signal STOP when buffer is empty
- 0 - don't signal STOP when buffer is empty
- is_cnt: 1 - this is a restart, don't check MBB

- 0 - this is a not restart, check MBB
- Returns: Any defined status indicator

Description:

Set up to send a buffer of data to the intended rcv_addr. If stop_flag is set, after the whole buffer is sent, generate a STOP signal provided that the receiver does not signal the STOP in the middle. Caller is the master performing transmitting. If no STOP signal is generated at the end of the current transaction, the master can generate a START signal to another slave address.

NOTE

The function does not actually perform the data buffer transmit. It just sets up the DLI global variables to control the transaction and calls I2C_Start to send the slave address out on the I2C bus in transmit mode. The application must check the return status to find out if the bus was obtained, then enter a loop of calling I2C_Timer_Event to poll the I2C handler to actually perform the transaction one byte at a time, while checking the return status to determine if there were any errors and whether the transaction has completed.

```
I2CStatus I2C_get( unsigned int eumbbar,
unsigned char sender_addr,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag,
unsigned int is_cnt );
```

- eumbbar is the address of the embedded utilities memory block.
- sender_addr is the sender's I2C device address
  buffer_ptr is pointer to the data buffer to transmit.
- length is the number of bytes in the buffer.
- stop_flag: 1—signal STOP when buffer is empty.
- 0—do not signal STOP when buffer is empty.
- is_cnt: 1—this is a restart. Do not check MBB.
- 0—this is a not restart. Check MBB.
- Returns: any defined status indicator

Description:

Set up to receive a buffer of data from the desired sender_addr. If stop_flag is set, when the buffer is full and the sender does not signal STOP, generate a STOP signal. Caller is the master performing receiving. If no STOP signal is generated, the master can generate a START signal to another slave address.

NOTE

The function does not actually perform the data buffer receive. It just sets up the DLI global variables to control the transaction and calls I2C_Start to send the slave address out on the I2C bus in receive mode. The application must check the return status to find out if the bus was obtained, then enter a loop of calling I2C_Timer_Event to poll the I2C handler to actually perform the transaction one byte at a time, while checking the return status to determine if there were any errors and if the transaction has completed.

```
I2CStatus I2C_Timer_Event( unsigned int eumbbar, I2CStatus (*handler)( unsigned int ) );
```

- eumbbar is the address of the embedded utilities memory block
- handler is a pointer to the function to call to handle any existing status event,
- Returns: I2CNOEVENT if there is no completed event, the I2CSR MIF bit is not set results from call to the handler function if there was a pending event completed

Description:

In polling mode, I2C_Timer_Event can be called to check the I2C status and call the given (or the default: I2C_ISR) handler function if the I2CSR MIF bit is set.

```
I2CStatus I2C_Start( unsigned int eumbbar,
unsigned char slave_addr,
I2C_MODE mode,
unsigned int is_cnt );
```

- eumbbar is the address of the embedded utilities memory block
- slave_addr is the I2C address of the receiver
- mode: XMIT(1)—put (write)
- RCV(0)—get (read)
- is_cnt: 1—this is a restart, don't check MBB
- 0—this is a not restart, check MBB
- Returns: any defined status indicator

Description:

Generate a START signal in the desired mode. Caller is the master. The slave_addr is written to bits 7:1 of the I2CDR and bit 0 of the I2CDR is set to 0 for mode = XMIT or 1 for mode = RCV. A DLI-global variable MasterRcvAddress is set if mode = RCV (used by I2C_ISR function).

```
I2CStatus I2C_Stop( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block.
- Returns: any defined status indicator

Description:

Generate a STOP signal to terminate the master transaction.

```
I2CStatus I2C_Master_Xmit( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block.

- Returns: Any defined status indicator

Description:

Master sends one byte of data to slave receiver. The DLI global variables ByteToXmit, XmitByte, and XmitBufEmptyStop are used to determine which data byte, or STOP, to transmit. If a data byte is sent, it is written to the I2CDR. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CSR.RXAK = 0 I2CCR.MSTA = 1 I2CCR.MTX = 1

```
I2CStatus I2C_Master_Rcv( unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block.
- Returns: any defined status indicator

Description:

Master receives one byte of data from slave transmitter. The DLI global variables ByteToRcv, RcvByte, and RcvBufFulStop are used to control the accepting of the data byte or sending of a STOP if the buffer is full. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CCR.MSTA = 1 I2CCR.MTX = 0

```
I2CStatus I2C_Slave_Xmit( unsigned int eumbbar );
```

NOTE

Untested.

- eumbbar is the address of the embedded utilities memory block
- Returns: I2CSUCCESS if data byte sent I2CBUFFEMPTY if no data in sending buffer

Description:

Slave sends one byte of data to requesting master. The DLI global variables ByteToXmit, XmitByte, and XmitBuf are used to determine which byte, if any, to send. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CSR.RXAK = 0 I2CCR.MSTA = 0 I2CCR.MTX = 1

```
I2CStatus I2C_Slave_Rcv(unsigned int eumbbar );
```

NOTE

Untested.

- eumbbar is the address of the embedded utilities memory block.
- Returns: I2CSUCCESS if data byte received I2CBUFFFULL if buffer is full or no more data expected.

Description:

Slave receives one byte of data from master transmitter. The DLI global variables ByteToRcv, RcvByte, and RcvBufFulStop are used to control the accepting of the data byte or setting the acknowledge bit (I2CCR.TXAK) if the expected number of bytes have been received. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MCF = 1 I2CCR.MSTA = 0 I2CCR.MTX = 0

```
I2CStatus I2C_Slave_Addr( unsigned int eumbbar );
```

NOTE

Untested.

- eumbbar is the address of the embedded utilities memory block.
- Returns: I2CADDRESS if asked to receive data
  results from call to I2C_Slave_Xmit if asked to transmit data.

Description:

Process slave address phase. Called from I2C_ISR. This function may only be called when the following conditions are met: I2CSR.MIF = 1 I2CSR.MAAS = 1

```
I2CStatus I2C_ISR(unsigned int eumbbar );
```

- eumbbar is the address of the embedded utilities memory block.
- Returns:
  - I2CADDRESS if address phase for master receive results from call to I2C_Slave_Addr if being addressed as slave (untested)
  - results from call to I2C_Master_Xmit if master transmit data mode
  - results from call to I2C_Master_Rcv if master receive data mode
  - results from call to I2C_Slave_Xmit if slave transmit data mode (untested)
  - results from call to I2C_Slave_Rcv if slave receive data mode (untested)
  - I2CSUCCESS if slave has not acknowledged, generated STOP (untested)
  - I2CSUCCESS if master has not acknowledged, wait for STOP (untested)
  - I2CSUCCESS if bus arbitration lost (untested)

Description:

Read the I2CCR and I2CSR to determine why the I2CSR.MIF bit was set which caused this function to be called. Handle condition; see above in possible return values. This function is called in polling mode as the handler function when an I2C event has occurred. It is intended to be a model for an interrupt service routine for polling mode, but this is untested and the design has not been reviewed or confirmed. This function may only be called when the following condition is met: I2CSR.MIF = 1

NOTE

This function is tested only for the master-transmit and master-receive in polling mode. It is probably not tested even in those modes for situations when the slave does not acknowledge or bus arbitration is lost or buffers overflow, etc.

The following DLI functions were written but not used and not tested:

```
I2CStatus I2C_write( unsigned int eumbbar,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag );
```

- eumbbar is the address of the embedded utilities memory block.
- buffer_ptr is pointer to the data buffer to transmit.

- length is the number of bytes in the buffer.
- stop_flag: 1—signal STOP when buffer is empty.
- 0—don't signal STOP when buffer is empty.
- Returns: any defined status indicator.

Description:

Send a buffer of data to the requiring master. If stop_flag is set, after the whole buffer is sent, generate a STOP signal provided that the requiring receiver does not signal the STOP in the middle. Caller is the slave performing transmitting.

```
I2CStatus I2C_read( unsigned int eumbbar,
unsigned char *buffer_ptr,
unsigned int length,
unsigned int stop_flag );
```

- eumbbar is the address of the embedded utilities memory block.
- buffer_ptr is pointer to the data buffer to transmit.
- length is the number of bytes in the buffer.
- stop_flag: 1—signal STOP when buffer is empty.
- 0—don't signal STOP when buffer is empty.
- Returns: any defined status indicator.

Description:

Receive a buffer of data from the sending master. If stop_flag is set, when the buffer is full and the sender does not signal STOP, generate a STOP signal. Caller is the slave performing receiving.

E.3.2.4.3 I2C Support Functions

```
unsigned int get_eumbbar( );
```

- Returns: base address of the embedded utilities memory block

Description:

In the *MPC8245 Integrated Processor Reference Manual*, see the section, “Embedded Utilities Memory Block,” in the “Address Maps” chapter, and the section, “Configuration Register Summary,” in the “Configuration Registers” chapter, for information about the embedded utilities memory block base address register. This function is defined in kahlua.s.

```
unsigned int load_runtime_reg( unsigned int eumbbar,
unsigned int reg);
```

- eumbbar is the address of the embedded utilities memory block.
- reg specifies the register: I2CDR, I2CFDR, I2CADR, I2CSR, I2CCR.
- Returns: register content

Description:

The content of the specified register is returned. This function is defined in drivers/i2c/i2c2.s.

```
unsigned int store_runtime_reg( unsigned int eumbbar,
unsigned int reg,
unsigned int val);
```

- eumbbar is the address of the embedded utilities memory block.
- offset specifies the register: I2CDR, I2CFDR, I2CADR, I2CSR, I2CCR.
- val is the value to be written to the register
- Return: No return value used. It should be declared void.

Description:

The value is written to the specified register. This function is defined in drivers/i2c/i2c2.s.

E.4 MPC8240 PIC Interrupt Driver

This appendix describes the sample PIC driver source code provided in this DINK32 release and its usage on the Sandpoint reference platform running DINK32.

E.4.1 General Description

PIC is the programmable interrupt controller feature implemented on Freescale Semiconductor's MPC8245/MPC8241/MPC8240, and on Tundra Semiconductor's Tsi107™. It is derived from the open Programmable Interrupt Controller (PIC) Register Interface Specification R1.2 developed by AMD and Cyrix. PIC provides support for up to five external interrupts or one serial-style interrupt line (supporting 16 interrupts), four internal logic-driven interrupts (DMA0, DMA1, I^2C, I_2O), four global timers, and a pass-through mode. MPC8245 and MPC8241 also add DUART support. Please refer to the "Programmable Interrupt Controller (PIC) Unit" chapter of the MPC8245/MPC8241/MPC8240 reference manual, or to Tundra Semiconductor's *Tsi107 PowerPC Host Bridge User Manual*, for a more in-depth description of PIC. For simplicity, all manual references in this appendix will be to the *MPC8240 Integrated Processor User's Manual*; however, the reader should refer to the corresponding manual for their particular device.

E.4.2 PIC Specifics

Unlike other embedded features of the MPC824x and MPC107 such as DMA and I_2O, the PIC unit is accessible from the local processor only. The control and status registers of this unit cannot be accessed by external PCI devices. The PIC registers are accessed as an offset from the embedded utilities memory block (EUMB). The PIC unit supports two modes: mixed and pass-through.

The DINK32 PIC driver sample code demonstrates PIC in mixed mode and also error checks for pass-through mode in case external interrupts are enabled with no interrupt handler setup. Both direct and serial mode are implemented in DINK32; but when PIC is initialized through DINK32, the default PIC setup is dependant on the Sandpoint board version. However, the default setups can be overridden using the "dev epic" command set.

The Sandpoint board version is detected by a loop-back feature implemented on SuperIO of the Sandpoint X3 (see the *Sandpoint Microprocessor Evaluation System User's Manual*). DINK32 will attempt to detect this loop-back feature. If the loop-back test is successful, the board version is Sandpoint X3. If the test is unsuccessful, the board version is Sandpoint X2. PIC on a Host PMC will default to serial mode on a Sandpoint X3 and will default to direct mode on a Sandpoint X2. PIC on an agent PMC in a PCI slot will default to direct mode on either Sandpoint board version.

The PIC registers are in little-endian format. If the system is in big-endian mode, the bytes must be appropriately swapped by software. DINK32 is written for big-endian mode, and the sample code referred to in this appendix performs the appropriate byte swapping.

E.4.2.1 Embedded Utilities Memory Block (EUMB)

The EUMB is a block of local and PCI memory space allocated to the control and status registers of the embedded utilities. The embedded utilities are the messaging unit (I_2O), DMA controller, PIC, I^2C, and ATU. The MPC8245 adds the DUART to the EUMB. The local memory map location of the EUMB is controlled by the embedded utilities memory block base address register (EUMBBAR). The PCI bus memory map location of the EUMB is controlled by the peripheral control and status registers base address register (PCSRBAR). Since PIC is only accessible from local memory, only the EUMBBAR is of concern for this appendix.

Please refer to the following sections in the *MPC8245 Integrated Processor User's Manual*:

- "Embedded Utilities Memory Block," in the "Address Maps" chapter
- "Embedded Utilities Memory Block Base Address Register—0x78" in the "Configuration Registers" chapter
- "Configuration Register Access" in the "Configuration Registers" chapter

E.4.2.2 PIC Register Summary

The PIC register map occupies a 256 Kilobyte range of the EUMB. All PIC registers are 32 bits wide and reside on 128-bit address boundaries. The PIC registers are divided into four distinct areas whose address offsets are based on the EUMB location in local memory controlled by the value in the EUMBBAR configuration register.

The PIC address offset map areas:

- 0x4_1000 - 0x4_10F0: Global PIC register map
- 0x4_1100 - 0x4_FFF0: Global timer register map
- 0x5_0000 - 0x5_FFF0: Interrupt source configuration register map
- 0x6_0000 - 0x6_0FF0: Processor-related register map

In the *MPC8245 Integrated Processor Reference Manual*, refer to the section, "PIC Register Summary," in the "Programmable Interrupt Controller (PIC) Unit" chapter, for the complete PIC register address map table. Refer to the "Register Definitions" section of the "Programmable Interrupt Controller (PIC) Unit" chapter, for all register definitions.

E.4.2.3 PIC Modes

- Pass-through mode

This mode provides a mechanism to support alternate interrupt controllers such as the 8259 interrupt controller architecture. Pass-through is the default mode of the PIC unit.

- Mixed mode

This mode supports two subsequent interrupt modes, either a serial interrupt mode (up to 16 serial interrupt sources) or a direct interrupt mode (up to 5 direct interrupt sources).

In the *MPC8245 Integrated Processor Reference Manual*, see Sections 11.3–11.5 in the “Programmable Interrupt Controller (PIC) Unit” chapter for more on PIC modes.

E.4.3 Drivers/Epic Directory Structure

DINK32/drivers/epic

- epic.h: contains all PIC register address macros and all function declarations.
- epic1.c: contains all C language routines.
- epic2.s: contains all Assembly language routines.
- epicUtil.s: contains Assembly routines to load and store to registers in the EUMB
- makefile: used by the DINK32 makefile to build this directory into a driver library
- Readme.txt: a text version of this appendix

E.4.4 PIC Cross-Reference Table Structure

The following table is defined in epic1.c in order to cross-reference interrupt vector numbers with the corresponding interrupt vector/priority register address and interrupt service routine address:

```
/* Register Address Offset/ Vector Description /ISR Addr cross-reference table */
struct SrcVecTable SrcVecTable[MAXVEC] =
{
  { EPIC_EX_INT0_VEC_REG, "External Direct/Serial Source 0",                  0x0},
{ EPIC_EX_INT1_VEC_REG, "External Direct/Serial Source 1",                    0x0},
{ EPIC_EX_INT2_VEC_REG, "External Direct/Serial Source 2",                    0x0},
{ EPIC_EX_INT3_VEC_REG, "External Direct/Serial Source 3",                    0x0},
{ EPIC_EX_INT4_VEC_REG, "External Direct/Serial Source 4",                    0x0},
{ EPIC_SR_INT5_VEC_REG,  "External Serial Source 5",                          0x0},
{ EPIC_SR_INT6_VEC_REG,  "External Serial Source 6",                          0x0},
{ EPIC_SR_INT7_VEC_REG,  "External Serial Source 7",                          0x0},
{ EPIC_SR_INT8_VEC_REG,  "External Serial Source 8",                          0x0},
{ EPIC_SR_INT9_VEC_REG,  "External Serial Source 9",                          0x0},
{ EPIC_SR_INT10_VEC_REG, "External Serial Source 10",                         0x0},
{ EPIC_SR_INT11_VEC_REG, "External Serial Source 11",                         0x0},
{ EPIC_SR_INT12_VEC_REG, "External Serial Source 12",                         0x0},
{ EPIC_SR_INT13_VEC_REG, "External Serial Source 13",                         0x0},
{ EPIC_SR_INT14_VEC_REG, "External Serial Source 14",                         0x0},
{ EPIC_SR_INT15_VEC_REG, "External Serial Source 15",                         0x0},
{ EPIC_TM0_VEC_REG,               "Global Timer Source 0",                    0x0},
{ EPIC_TM1_VEC_REG,               "Global Timer Source 1",                    0x0},
{ EPIC_TM2_VEC_REG,               "Global Timer Source 2",                    0x0},
```

```
{ EPIC_TM3_VEC_REG,                "Global Timer Source 3",                    0x0},
{ EPIC_I2C_INT_VEC_REG,  "Internal I2C Source",                                0x0},
{ EPIC_DMA0_INT_VEC_REG, "Internal DMA0 Source",                               0x0},
{ EPIC_DMA1_INT_VEC_REG, "Internal DMA1 Source",                               0x0},
{ EPIC_MSG_INT_VEC_REG,  "Internal Message Source",                            0x0}
{ EPIC_DUART1_INT_VEC_REG,  "DUART Ch1 Source",                                0x0},
{ EPIC_DUART2_INT_VEC_REG,  "DUART Ch2 Source",                                0x0}

};
Each of the 26 entries conforms to the following:

{       "vector/priority register address offset",
        "text description",
        "Interrupt Service Routine address"                 }.
```

The first column of the structure contains the macro for each of the 26 interrupt vector/priority register address offsets in PIC. The middle column is the text description of the interrupt vector, and the last column is the address of the registered interrupt service routine (ISR) for each interrupt vector. Currently the structure is initialized such that each vector ISR address is 0x0. This can be modified such that each defaults to a "catch all ISR" address instead of 0x0. As each interrupt vector is set up, an ISR must be registered with PIC via the epicISRConnect() routine in the epic1.c source file. This routine takes the ISR function name and stores the address of that function in the ISR address structure location corresponding to the interrupt vector number. Although each interrupt's vector/priority register allows the vector number to range from 0 to 255, this structure limits the vector number range to 0–25. So as each interrupt's vector/priority register is set up, the 8-bit vector field value must match the vector number location in the structure.

E.4.5 PIC Sample Routines

The PIC sample routines are contained in the epic1.c and epic2.s files. All C language routines are in epic1.c and all Assembly language routines are in epic2.s. These routines, along with the structure described in E.4.4, “PIC Cross-Reference Table Structure,” can be used as sample code for systems using the PIC Unit. E.4.6, “PIC Commands in DINK32,” describes how these routines are used by DINK32.

E.4.5.1 Low-Level Routines

The following routines are in the epic2.s source file:

- External interrupt control routines:
  - CoreExtIntEnable(): enables external interrupts by setting the MSR[EE] bit.
  - CoreExtIntDisable(): disables external interrupts by clearing the MSR[EE] bit.
- Low-level exception handler:
  - epic_exception():
    Save the current (interrupted) programming model/state
    Calls epicISR() to service the interrupt
    Restore the programming model/state and
    RFI back to interrupted process

E.4.5.2 High-Level Routines

The following routines are in the epic1.c source file:

E.4.5.2.1 PIC Initialization Routines

epicInit(): initialize the PIC Unit by:

- Setting the reset bit in the global configuration register which does the following:
  - — Disables all interrupts
  - — Clears all pending and in-service interrupts
  - — Sets PIC timers to base count
  - — Sets the value of the processor current task priority to the highest priority (0xF), thus disabling interrupt delivery to the processor
  - — Resets spurious vector to 0xFF
  - — Defaults to pass-through mode
  - — Sets the PIC operation mode to default mixed or serial mode (versus pass-through or 8259-compatible mode), depending on Sandpoint board version.
  - — If IRQType (input) is Direct IRQs:
  - — IRQType is written to the SIE bit of the PIC interrupt configuration register (ICR)
  - — clkRatio is ignored
  - — If IRQType is Serial IRQs:
  - — both IRQType and clkRatio will be written to the ICR register

epicCurTaskPrioSet(): Change the current task priority value

epicIntISRConnect(): Register an ISR with the PIC unit cross-reference table

E.4.5.2.2 High-Level Exception Handler

epicISR(): this routine is a catch all for all PIC related interrupts:

- perform IACK (interrupt acknowledge) to get the vector number
- check if the vector number is a spurious vector
- cross-reference vector ISR (interrupt service routine) from table
- call the vector ISR
- perform EOI (end of interrupt) for the interrupt vector

E.4.5.2.3 Direct/Serial Register Control Routines

epicIntEnable(): enable an interrupt source

epicIntDisable(): disable and interrupt source

epicIntSourceConfig(): configure and interrupt source

E.4.5.2.4 Global Timer Register Control Routines

epicTmBaseSet(): set the base count value for a timer

epicTmBaseGet(): get the base count value for a timer

epicTmCountGet(): get the current counter value for a timer

epicTmInhibit(): inhibit counting for a timer

epicTmEnable(): enable counting for a timer

E.4.6 PIC Commands in DINK32

The following commands are typed from the DINK32 command line to control the PIC unit.

- help dev epic - Display usage of PIC commands
- dev epic - Display content and addresses of PIC registers and current task priority
- dev epic ex - “dev epic” command example uses
- dev epic init - Initialize the PIC unit to default mode
  Sandpoint X2 default is direct mode
  Sandpoint X3 default is serial mode
  Agent in a host/agent setup defaults to direct mode
- dev epic init [Mode(0|1)] [Ratio(1-7)] - Initialize the PIC unit (this calls the epicInit() routine)
- dev epic ta [0-15]- Change the processor task priority register
- dev epic en [Vector(0-23)] - Enable a particular interrupt vector
- dev epic dis [Vector(0-23)] - Disable a particular interrupt vector
- dev epic con [Vector(0-23)] - Print content of a source vector/priority register
- dev epic con [Vector(0-23) Polarity(0|1) Sense(0|1) Priority (0-15)]
  — Program the source vector/priority register
- dev epic tmbase [Timer(0-3)] - Display a timer current count register
- dev epic tmbase [Timer(0-3)] Count(hex value) Inhibit(0|1)
  — Set, enable/disable a time base count register
- dev epic tmcnt [Timer(0-3)] - Display a timer current count register
- dev epic tmdis [Timer(0-3)] - Inhibits counting for a timer
- dev epic tmen [Timer(0-3)] - Enables counting for a timer
- dev epic ISRCnt [Vector(0-23) Address]
  — Manually link an ISR to an interrupt vector
- dev epic eie- External interrupt enable
- dev epic eid- External interrupt disable

Example:

dev epic init - Initialize PIC unit to default mode.

dev epic init 0 7 - Initialize PIC unit to serial mode with a clock ratio of 7.

dev epic init 1 - Initialize PIC unit to direct mode.

dev epic en 1 - Enable interrupt vector 1

dev epic ta 10 - Set the processor task priority register to 10

dev epic dis 5 - Disable interrupt vector 5

dev epic con 2- Print the configuration of interrupt vector 2

dev epic con 7 1 0 5- Configure the source Vector/Priority
register of vector 7 to have the following properties:
Polarity = 1
Sense = 0
Priority = 5

dev epic tmbase 0 - Display timer 0 base count register

dev epic tmbase 0 7fff 0
Set timer 0 base count register to 0x7fff and enable counting to proceed

dev epic tmcnt 1 - display timer 1 current count register

dev epic tmdis 2 - Inhibit counting on timer 2

dev epic tmen 3 - Enable counting on timer 3

dev epic ISRCnt 1 90000 - Set the ISR address for vector 1 to 0x90000

E.4.7 PIC Unit Startup

When the system comes up running DINK32, the EUMBBAR is configured such that the EUMB is located at an offset of 0xFC00_0000 from local memory. The PIC unit is untouched by the DINK32 initialization routines and is left in its default state of pass-through mode. External interrupts are also left untouched and left in the default state of disabled. The following list shows the necessary routine calls needed to utilize the PIC unit:

- Initialize the PIC unit
  - epicInit()
- For each interrupt vector to be used:
  - epicSourceConfig()
  - epicISRConnect()
  - epicIntEnable()
- Set the processor current task priority
  - epicCurTaskPrioSet()
- Enable external interrupts
  - CoreExtIntEnable()

E.4.8 External Interrupt Exception Path in DINK32

The path of an external interrupt exception in DINK32 begins at the 0x500 interrupt exception vector. All DINK32 exception vector locations are set up in the same manner which is to save the exception type and pass the exception handling to a catch all exception handler. This handler is called handle_ex and is located in the except2.s DINK32 source file.

In the handle_ex handler, a check is performed to see if the exception was a 0x500 and if DINK32 is running on an MPC8240, MPC8245 or MPC107. If the two conditions are true, the exception handling is passed to the PIC low-level interrupt handler, epic_exception() located in the epic2.s source file. Epic_exception() handles any necessary context switching and saving of state before calling the PIC high-level interrupt handler, epicISR() located in the epic1.c source file.

NOTE

Currently, epic_exception() first checks the mode of the PIC unit. If in pass-through mode, an error message is printed stating that the PIC unit is in pass-through mode and must be initialized.

EpicISR() acknowledges the interrupt by calling the epicIACK() which returns the vector number of the interrupting vector source. This vector number is then compared to the spurious vector value located in the PIC spurious vector register. If the interrupting vector is a spurious vector, the interrupt is ignored and state is restored to the interrupted process. If the interrupting vector is a valid interrupt, then the vector number is used to reference the vector ISR from the cross-reference table. The vector ISR is then called to service the particular interrupt. Once the ISR completes and returns, an end-of-interrupt is issued by calling epicEOI(). Control then returns to epic_exception.

Epic_exception() finishes by restoring state and performs an RFI (return from interrupt) back to the interrupted process.

E.4.9 Example Usage on Sandpoint Reference Platform

At the time of Revision 13.2, the PIC driver source code currently defaults to a demonstration mode on the Sandpoint platform. The demo code is located in the epicInit() routine and allows for an interactive demonstration of external interrupts. On Sandpoint X2, the external interrupts can be manually demonstrated on IRQ lines 1 and 2, using the motherboard slide switches, S5 and S6. However, these switches are not available on Sandpoint X3. The use of global timers 0 and 1, DMA0, and the message unit if in a host/agent setup are available on both Sandpoint board versions. A debug mode is also provided and is controlled by the -DEPICDBG compiler directive in the makefile located in the PIC source directory. The compiler directive allows the driver code to be much more verbose and informative when exercising the PIC unit features in the debug state.

Although manual interrupt demonstration is not available on Sandpoint X3, the platform does implement serial interrupt logic, making it very useful in testing serial interrupts using PIC.

E.4.9.1 Sandpoint X2 Reference Platform

The Sandpoint X2 reference platform provides a means to test external interrupts via two slide switches (S5 and S6) located on the motherboard. Although these switches can be manipulated to demo the PIC unit,

this is not the intended function of the switches. The intended usage of these switches is described in the document titled, "Sandpoint PPMC Processor PCI Mezzanine Card Host Board Technical Summary."

Switch S5 manipulates a 5V signal that originates from the interrupt output line of the Winbond Southbridge chip in the center of the motherboard. With S5 slid to the left, a 5V signal is passed on; with S5 slid right, a 0V signal is passed on. The PIC IRQ0-4 interrupt lines can be configured to be active-low or active-high triggered.

Switch S6 specifies to which IRQ line (IRQ1 or IRQ 2) the interrupt signal from S5 is passed. With the S6 slid right, IRQ1 is selected. With S6 slid left, IRQ2 is selected.

E.4.9.2 Initializing PIC on Sandpoint X2

Initializing PIC requires that DINK32 be running on a Sandpoint X2 system with an MPC8240, MPC8245, or MPC107 PMC module. From the DINK32 command line, initialize the PIC unit by typing the PIC initialization “dev epic init” command. DINK32 will respond with initialization messages and will be ready to handle external interrupts. The user may now also manipulate the S5 and S6 switches to trigger interrupts on the IRQ1 and IRQ2 lines on the Sandpoint X2. The global timers can now be manipulated to generate timed interrupts. The message unit (I_2O) can be used if in a host/agent setup. DMA0 can be used in an interrupt-driven manner to transfer blocks of data. Of course, while all these external interrupts are being handled, DINK32 continues to run and will accept user input at the command line, while simultaneously writing status to the terminal.

Host PIC initialization on Sandpoint X2 running DINK32 in a non-host/agent setup:

```
Host...
Agent in PCI Slot4 detected.
DINK32 [MPC7400] >>
DINK32 [MPC7400] >>dev epic init
Initialize EPIC to Default Mode

Kahlua agent in Slot: 4 -> IRQ: 3.

EPIC: Disable External Interrupts
EPIC: Reseting...  Mixed Mode...  Direct Mode

EPIC: Configuring EPIC to Sandpoint X2 default mode...

EPIC: Slides switches, S5 and S6, can be used to
      manually trigger interrupts on IRQ1 and IRQ2.
EPIC: IRQ1   Configure...  Connect ISR...  Enable
EPIC: IRQ2   Configure...  Connect ISR...  Enable

EPIC: Agent detected on IRQ 3
EPIC: Use 'help dev i2o' to manipulate interrupts
EPIC: IRQ3    Configure...  Connect ISR...  Enable

EPIC: Enable/disable Timers using 'dev epic tmen/tmdis (0-3)'
EPIC: Timer0  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer1  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer2  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer3  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable

EPIC: Use 'help dev dma' to manipulate DMA0 interrupt
```

```
EPIC: DMA0    Configure...  Connect ISR...  Enable

EPIC: Lower Current Task Priority
EPIC: Enable External Interrupts in MSR

DINK32 [MPC7400] >>
```

Agent EPIC initialization on Sandpoint running DINK32 in a Host/Agent setup:

```
Agent ....
DINK32 [MPC8240] >>
DINK32 [MPC8240] >>dev epic init
Initialize EPIC to Default Mode

EPIC: Disable External Interrupts
EPIC: Reseting...  Mixed Mode...  Direct Mode

EPIC: Configuring EPIC to Sandpoint X2 default mode...

EPIC: Host/Agent setup detected
EPIC: Use 'help dev i2o' to manipulate interrupts
EPIC: Message Unit Interrupt  Configure...  Connect ISR...  Enable

EPIC: Enable/disable Timers using 'dev epic tmen/tmdis (0-3)'
EPIC: Timer0  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer1  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer2  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer3  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable

EPIC: Use 'help dev dma' to manipulate DMA0 interrupt
EPIC: DMA0    Configure...  Connect ISR...  Enable

EPIC: Lower Current Task Priority
EPIC: Enable External Interrupts in MSR

DINK32 [MPC8240] >>
```

E.4.9.3 Running the PIC on Sandpoint X3

Initializing PIC requires that DINK32 be running on a Sandpoint X3 system with an MPC8240, MPC8245, or MPC107 PMC module. From the DINK32 command line, initialize the PIC unit by typing the PIC initialization “dev epic init” command. DINK32 will respond with initialization messages and will be ready to handle external interrupts. The global timers can now be manipulated to generate timed interrupts. The message unit (I_2O) can be used if in a host/agent setup. DMA0 can be used in an interrupt -riven manner to transfer blocks of data. Of course, while all these external interrupts are being handled, DINK32 continues to run and will accept user input at the command line, while simultaneously writing status to the terminal.

Host PIC initialization on Sandpoint X3 running DINK32 in a non Host/Agent setup:

```
Host...
Agent in PCI Slot3 detected.
DINK32 [MPC8240] >>
DINK32 [MPC8240] >>dev epic init
Initialize EPIC to Default Mode
```

```
Kahlua agent in Slot: 3 -> IRQ: 4.

EPIC: Disable External Interrupts
EPIC: Reseting...  Mixed Mode...  Serial Mode

EPIC: Configuring EPIC to Sandpoint X3 default mode...

EPIC: For Serial mode on Sandpoint X3 make sure
      switch S2[3,4,5]=On=Right.

EPIC: Agent detected on IRQ 4
EPIC: Use 'help dev i2o' to manipulate interrupts
EPIC: IRQ4   Configure...  Connect ISR...  Enable

EPIC: IRQ0   Configure...  Connect ISR...  Enable
EPIC: IRQ1   Reserved
EPIC: IRQ2   Configure...  Connect ISR...  Enable
EPIC: IRQ3   Configure...  Connect ISR...  Enable
EPIC: IRQ5   Configure...  Connect ISR...  Enable
EPIC: IRQ6   Configure...  Connect ISR...  Enable
EPIC: IRQ7   Configure...  Connect ISR...  Enable
EPIC: IRQ8   Configure...  Connect ISR...  Enable
EPIC: IRQ9   Configure...  Connect ISR...  Enable
EPIC: IRQ10-15  Reserved

EPIC: Enable/disable Timers using 'dev epic tmen/tmdis (0-3)'
EPIC: Timer0  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer1  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer2  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer3  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable

EPIC: Use 'help dev dma' to manipulate DMA0 interrupt
EPIC: DMA0    Configure...  Connect ISR...  Enable

EPIC: Lower Current Task Priority
EPIC: Enable External Interrupts in MSR

DINK32 [MPC8240] >>
```

Agent EPIC initialization on Sandpoint running DINK32 in a Host/Agent setup:

```
Agent ....
DINK32 [MPC8240] >>dev epic init
Initialize EPIC to Default Mode

EPIC: Disable External Interrupts
EPIC: Reseting...  Mixed Mode...  Direct Mode

EPIC: Configuring EPIC to Sandpoint X3 default mode...

EPIC: Host/Agent setup detected
EPIC: Use 'help dev i2o' to manipulate interrupts
EPIC: Message Unit Interrupt  Configure...  Connect ISR...  Enable

EPIC: Enable/disable Timers using 'dev epic tmen/tmdis (0-3)'
EPIC: Timer0  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer1  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
EPIC: Timer2  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable
```

```
EPIC: Timer3  Configure...  Base Count = 0xFFFFFF...  Connect ISR...  Enable

EPIC: Use 'help dev dma' to manipulate DMA0 interrupt
EPIC: DMA0    Configure...  Connect ISR...  Enable

EPIC: Lower Current Task Priority
EPIC: Enable External Interrupts in MSR

DINK32 [MPC8240] >>
```

E.4.10 Code and Documentation Updates

For the most up-to-date versions of the PIC sample driver code and this appendix/document, please visit the Freescale Semiconductor DINK32 website.

Appendix F
S-Record Format Description

F.1 General Format

An S-record is a file that consists of a sequence of specially formatted ASCII character strings. Each line of the S-record file adheres to the same general format (with some variation of the specific fields) and must be 78 bytes or fewer in length. A typical S-record file might look like this:

```
S0100000777269746 56D656D2E73726563AA
S219070000740000007000000003D20DEAD6129BEEF3C60000060E0
S2190700156300003CC0004060C600007D20192E7CE0182E7C07FC
S21907002A4800408200143863040 07C0330004180FFE848000059
S20907003F004800000068
S804070000F4
```

This information is an encoding of data to be loaded into memory by an S-record loader. The address at which the data is loaded is determined by the information in the S-record. The data is verified through the use of a checksum located at the end of each record. Each record in a file should be followed by a linefeed.

The general format of an S-record is as follows:

```
Type            char[2]
Count           char[2]
Address         char[4,6, or 8]
Data            char[0-64]
Checksum        char[2]
```

Note that the fields are composed of characters. Depending on the field, these characters may be interpreted as hexadecimal values or as ASCII characters. Typically, the values in the ‘Type’ field are interpreted as characters, while the values in all other fields are interpreted as hex digits.

Type: Describes the type of S-record entry. There are S0, S1, S2, S3, S5, S7, S8, and S9 types. This information is used to determine the format of the remainder of the characters in the entry. The specific format for each S-record type is discussed in the next section.

Count: When the two characters comprising this field are interpreted as a hex value, ‘Count’ indicates the number of remaining character pairs in the record.

Address: These characters are interpreted as a hex address. They indicate the address where the data is to be loaded into memory. The address may be interpreted as a 2, 3, or 4 byte address, depending on the type of record. 2-byte addresses require 4 characters, 3-byte addresses require 6 characters, and 4-byte addresses require 8 characters.

Data: This field can have anywhere from 0 to 64 characters, representing 0-32 hexadecimal bytes. These values will be loaded into memory at the address specified in the address field.

Checksum: These 2 characters are interpreted as a hexadecimal byte. This number is determined as follows: Sum the byte values of each pair of hex digits in the count, address, and data fields of the record. Take the one's complement. The least significant byte of the result is used as the checksum.

F.2 Specific Formats

Each of the record types has a slightly different format. These are all derived from the general format specified above and are summarized in Table F-1.

Table F-1. SRecord Formats

Type	Description
S0	Contains header information for the S-record. This data is not actually loaded into memory. The address field of an S0 record is unused and will contain 0x0000. The data field contains the header information, which is divided into several sub-fields: char[20] module name char[2] version number char[2] revision number char[0-36] text comment Each subfield is composed of ASCII characters. These are paired and interpreted as one-byte hex values in the case of the revision number and version number fields. For the module name and text comment fields these values should be interpreted as hexadecimal values of ASCII characters.
S1	The address field is interpreted as a 2-byte address. The data in the record is loaded into memory at the address specified.
S2	The address field is interpreted as a 3-byte address. The data in the record is loaded into memory at the address specified.
S3	The address field is interpreted as a 4-byte address. The data in the record is loaded into memory at the address specified.
S5	The address field is interpreted as a 2-byte value which represents a count of the number of S1, S2, and S3 records previously transmitted. The data field is unused.
S7	The address field is interpreted as a 4-byte address and contains the execution start address. The data field is unused.
S8	The address field is interpreted as a 3-byte address and contains the execution start address. The data field is unused.
S9	The address field is interpreted as a 2-byte address and contains the execution start address. The data field is unused.

F.3 Examples

Following are some sample S-record entries broken into their parts with a short explanation:

```
Example 1: S0100000777269746 56D656D2E73726563AA
Separated: S0-10-0000-77726974656D656D2E73726563-AA
```

•Type: S0 - this is a header record •Count: 10 - interpreted as 0x10; indicates that 16 character pairs follow •Address: 0000 - interpreted as 0x0000. The address field for S0 is always 0x0000. •Data: Since this is a header record, the information can be interpreted in a number of ways. It doesn't really matter since you usually don't use this field for anything interesting. •Checksum: AA - the checksum

```
Example 2: S21907000074000000700000003D20DEAD6129BEEF3C60000060E0
```

```
Separated: S2-19-070000-74000000700000003D20DEAD6129BEEF3C60000060-E0
```

•Type: S2 - the record consists of memory-loadable data and the address should be interpreted as 3 bytes •Count: 19 - interpreted as 0x19; indicates that 25 character pairs follow •Address: 070000 - data will be loaded at address 0x00070000 •Data: Memory loadable data representing executable code •Checksum: E0 - checksum

```
Example 2: S804070000F4
Separated: S8-04-070000-F4
```

- Type: S8—this is the record with the execution start address; it also indicates the end of the s-record
- Count: 04—interpreted as 0x04; indicates that 4 character pairs follow
- Address: 070000—execution will begin at 0x00070000
- Data: None—this field is unused for S8 records.
- Checksum: F4—checksum

F.4 Summary of Formats

Table F-2 summarizes the length (in characters, bytes) of each field for the different S-record types. It is useful as a reference when parsing records manually during debug.

Table F-2. Summary of Formats in Bytes

Type	Count	Address	Data	Checksum
S0	2	N/A	0-60	2
S1	2	2 byte address	0-64	2
S2	2	3 byte address	0-64	2
S3	2	4 byte address	0-64	2
S5	2	2 byte count	0	2
S7	2	4 byte execution address	0	2
S8	2	3 byte execution address	0	2
S9	2	4 byte execution address	0	2

Appendix G
Networking Support

This section describes internal mechanism and usage of DINK networking functionality. The purpose is to allow DINK users to access and download code to the reference platforms more conveniently. Current TCP/IP stack in DINK implements:

- TFTP client
- Non-concurrent single user TELNET server
- PING program based on ICMP
- ARP server

Before using any network functions, user may use a RealTek8139 PCI NIC into one of the PCI slots or use an internal networking interface such as eTSEC, then run ‘ni -i’ command from DINK shell. This command will scan PCI bus for acceptable NIC type. Follow the prompt instruction to fill out card information and the network configurations.

If TELNET console is to be used, an environment variable ‘NETIO’ must be created by typing ‘env NETIO=1’. Then power cycle Sandpoint.

TFTP download can be invoked by using ‘-nw ‘ option in the download command. There is no difference between using a serial console or a remote TELNET console.

The following is an example of how to use TELNET console after setting up the NIC .

Firstly, use ni command to verify network status

```
DINK32[MPC7410] {14} >>ni
  NETWORK INFORMATION
    PCI CARD
      Type 8139/10EC on slot 15

    SETTINGS
      SERVER(TFTP) : 16. 11.105.182
      GATEWAY      : 16. 11.105.254
      NETMASK      : 255.255.255.  0
      DHCP         :   0.  0.  0.  0
      CLIENT(DINK) : 16. 11.105.163

    DHCP: Disabled
```

Then run “telsrv” from DINK32 to start the telnet server.

From a remote host computer, run TELNET client to the DINK IP address shown above. For example, from a DOS console, do.

```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.
C:\>telnet 16.11.105.163
```

No login is required. The initial prompt is ‘DINK32-TELNET >>’

To close the connection gracefully, disconnect from client Telnet window.

To re-direct console back to the serial type in Control-D (i.e. hold control key and the letter D at the same time).

G.1 Troubleshooting Hints

Problem 1. Connection lost after about a minute.

Solution: If you are using Windows-based host, this is due to the TCP keep-alive signal sent by Windows host to check if DINK connection is still alive. However, DINK will not respond to this keep-alive polling, (starting with an ARP request). This is due to the fact that DINK TCP/IP is designed to be purely based on polling such that, when running a benchmark via a network, there will be no unpredictable performance impact. The workaround is to disable Windows keep-alive polling by setting MAC address of the NIC card as a static entry in Windows ARP cache. This problem does not occur in BSD style systems, such as Solaris, Linux, and MacOS. Do the following three steps in the Windows command console to disable keep-alive polling.

Obtain the MAC address of the RealTek NIC by doing Telnet to Sandpoint then exit normally by disconnecting from the Telnet client console. There is no exit command from DINK shell.

Type ‘arp -a’ to verify the DINK IP and MAC address is marked as ‘dynamic’

Type ‘arp -s xxx.xxx.xxx.xxx zz:zz:zz:zz:zz:zz’ to set the entry as static. (xxx is the IP address; zz is the MAC address).

Problem 2. Remote machine cannot connect.

Solution: This does not happen very often; however, if it does, try again and the connection will be established.

G.2 Known Limitations in Telnet Console

- Does not allow serial console download
- Does not allow initialize NIC when connection is active
- Does not support simultaneous close
- Does not support certain cache tests which may cause unexpected snoop behavior on Ethernet DMA engine

Appendix H
Converting Dink32 to Little-Endian

H.1 General Information

This information is based on a little-endian version of DINK, V7.0 10/8/97 called DINKLE. The makefile is included in this appendix, the other files from this version are not required to understand this appendix but can be requested from risc10@freescale.com. Following the instructions below and having access to this DINKLE version can facilitate the conversion of any version of DINK32 to a little-endian version.

H.2 Preparation

The reset vector EH100S: at 0x00000100 and 0xfff00100 is extracted from except2.s and copied to a new file called reset.s. The system_reset code, which includes the copy DINK32 from ROM to RAM is extracted from except2.s and copied to a new file called reset1.s. Finally, the rest of except2.s is copied to a new file called except2l.s. This is necessary, as described later, because the reset vector and reset code must run in big-endian and the rest of the code must run in little-endian.

Thus, the two files (reset.s and reset1.s) are compiled as big-endian, and the rest are compiled as little-endian. The linker will then link these mixed mode files into a single executable.

These two Assembly language files need to be compiled as big-endian. Use the metaware assembler option -lb, which is the default.

- reset.s
- reset1.s

These three Assembly language files need to be compiled as little-endian. Use the metaware assembler option -le.

- except2l.s
- reg_swap.s
- go_tr2.s

All the C language files need to be compiled as little-endian. Use the metaware option flag -HL for little-endian compilation, the default is -HB big-endian.

The CC and two assembler commands for metaware are as follows:

- CC = /...path.../metaware/bin/hcppc -HL -Hnocopyr -c -Hsds
- ASOPTL = -big_si -le
  ASL = /...path.../metaware/bin/asppc -c $(ASOPTL)
- ASOPTB = -big_si -be
  ASB = /...path.../metaware/bin/asppc -c $(ASOPTB)

The order of compilation and linking is as follows:

- reset.o
- except2l.o
- reset1.o
- except1.o
- go_tr1.o
- go_tr2.o
- reg_swap.o
- All the rest of the C files.

The makefile included in Section H.3.3, “DINKLE V7.0 10/8/97 makefile,” is useful to understand the linking order.

H.3 Explanation

It is critical to understand that the processors and peripheral logic all come out of reset in big-endian. Therefore, the first code that is run (this is the reset code located in reset.s and reset1.s) will be compiled in big-endian. The reason why the reset vector is separated from the other code and other exception handlers (found in except2l.s) is that it is desirable to run the other handlers in little-endian mode so they will be assembled with the little-endian assembler. The linker will then link the files in the following order: reset.s, except2l.s, reset1.s ..... which are big-endian, little-endian, and big-endian, respectively. All of the other files are compiled as little-endian.

H.3.1 Two Important Considerations

The first involves the copy algorithm and the second involves the little-endian swap code.

H.3.2 Copy Algorithm

In DINK, we copy ROM contents to RAM before jumping to the RAM image. The compiler has compiled the rest of DINK as "TRUE" little-endian. Little endian on processors that implement the PowerPC architecture is not a "TRUE" little-endian but rather a munged little-endian scheme (see the *Programming Environments Manual* for more details). The fact that the processor really expects BIG ENDIAN data at little-endian addressing is accomplished by the unmunging of data during the copy algorithm (use of stwbr instructions). The copy algorithm is found in except2.s, which has been copied along with all of system_reset to except2l.s

The code is shown here.

```
//now copy DINK in ROM to RAM. ROM image is compiler little endian
//so,we have to swap the byte and muge the address by K.O.

        addis   r8,r0,0 //use this for copy ROM to RAM
        ori     r8,r8,4 //use for Munge address
lp1:
        lwz     r10,0(r4) //read word from eprom.
        lwz     r11,4(r4) //muge the address
```

```
        stwbrx    r10,r8,r3 //byte swap
        stwbrx    r11,r0,r3 //byte swap

/* original big endian code which is now replaced
* stwx    r5,0,r3 //store word into dram.
* lwzx    r7,0,r3 //load word from dram.
* cmp     0,0,r7,r5 // check to see if dram got written
* bne     error_dram_init
*/

         addi    r4,r4,8 //go to next double word of eprom and dram.
        addi    r3,r3,8
        addic.  r6,r6,-8 //decrement word from 256k block
//-- set cr0 on this one for branching.
        bgt     lp1 //if count>0, then loop
```

NOTE

The memory, which is local to the processor, always has big-endian ordered data in its physical memory locations (just as the values in the onboard registers and onboard caches are still in big-endian order). The byte lane swap is accomplished at the PCI interface.

H.3.2.1 Little-Endian Swap Instruction Sequence

The peripheral logic is switched to little-endian first (before the processor) in reset1.s (that is, the specific programming of the bridge chip to be in little-endian mode). There is a period when the processor and the peripheral logic will not be in the same endian mode. This period should be minimized, because the addresses that the user thinks they are executing from may not actually be correct. This is not intuitive. In the V7.0 code this is not a problem (and is not dealt with) because the instructions that are being accessed, when the peripheral logic has been switched to little-endian and the processor is still in big-endian, are already in local memory in big-endian format and the processor has not switched yet. If, however, the processor was running from code in PCI, then there is an issue because the ‘unmunge’ logic in the peripheral chip has just been turned ‘on’ but the processor is not yet munging addresses.

There is an elegant way of handling this. Place a sequence (approximately 30) of ‘ORI R0,R0,0x60’ opcodes in the code stream after switching the peripheral logic to little-endian. After these ‘ori’ instructions begin, little-endian code modules that have DUPLICATE opcodes can run until the processor can be put into little-endian mode (that is, until the RFI executes).

NOTE

The ori r0,r0,0x60 opcode is used because it does not matter whether the bytes are read as big-endian or little-endian (that is, they are the same opcode, a relatively innocuous no-op). 0x60000060 is still 0x60000060 in big- or little-endian mode.

After this assembly code (which has duplicate instructions), ‘regular’ compiler-generated, little-endian modules may be located.

NOTE

The duplicated instructions handle the fact that the address is temporarily "unmunged," so it is coming out as 0x04, 0x00, 0x0C, 0x08, etc. Instead of duplicating instructions, the user could alternate a no-op with the real instruction or reverse the opcode in memory (not recommended for clarity).

H.3.3 DINKLE V7.0 10/8/97 makefile

This makefile will only work with V7.0, it is included here only for illustrative purposes.

```
DEBUG   =

OPTIM   =

CC      = /risc/tools/pkgs/metaware/bin/hcppc -HL  -Hnocopyr -c -Hsds -fsoft #-Hlist

CCobj   = $(CC) $(DEBUG) $(OPTIM)

PREP    = $(CC) -P

# Assembler used to build the .s files (for the board version)

ASOPTL   = -big_si -le
ASOPTB   = -big_si -be

ASL      = /risc/tools/pkgs/metaware/bin/asppc -c $(ASOPTL)
ASB      = /risc/tools/pkgs/metaware/bin/asppc -c $(ASOPTB)
# Linker to bring .o files together into an executable.

LKOPT    =  -Bbase=0 -xm -e system_reset -Bnoheader -Bhardalign -xo=dink32.src -q -Qn -Cglobals
-Csections -Csymbols -Ccrossref
LINK    =  /risc/tools/pkgs/metaware/bin/ldppc $(LKOPT) > xref.txt

# DOS Utilities

DEL     = rm
COPY    = cp
LIST    = ls -l

# These are the modules which have to do with DINK's registers.
REGISTERS = reg_tb.o reg_spr.o

# These are the modules which have to do with DINK's memory access routines.
MEMORY = mem_tb.o

# These are the modules which have to do with the DINK parser.
PARSER = tok_tb.o arg_tb.o rfs_tb.o par_tb.o toks.o

# These are the modules which have to do with the error checking
# and reporting.
ERRORS = errors.o err_tb.o

# These are the modules which have to do with the downloader including
# DINK's compression routines.
DOWNLOAD = downld.o dc_tb.o
```

```
#
# These are the modules which have to do with the input output to the
# board level stuff.
INPUTOUTPUT = duart.o board.o

# These are the modules which have to do with DINK's assembler/disassembler.
ASMDSM = asm.o dsm.o

# These are for the exceptions in the DINK32 system.
EXCEPTIONS = reset.o except2l.o reset1.o except1.o

# These are for the Go and Trace routines. Please note that the EXCEPTIONS are
# very important for the Go/Tr operations.
GOTRACE = go_tr1.o go_tr2.o reg_swap.o

# These are the modules which have to do with DINK's help and breakpoints.
MISC = help.o brk_pts.o sublib.o

# These are the modules which have to do with the main loop and

# initialization of DINK32.
DINKMAIN = main.o print.o
DINKASM   =       $(EXCEPTIONS) $(GOTRACE)
DINKWORKERS =$(REGISTERS) $(MEMORY) $(DOWNLOAD) $(ASMDSM) $(MISC)
DINKINTERFACE = $(PARSER) $(ERRORS) $(INPUTOUTPUT)
DINKOBJECTS = $(DINKASM) $(DINKMAIN) $(DINKWORKERS) $(DINKINTERFACE)
DCOMPOBJECTS = dc_tb_unix.o dc_unix.o

dink32:  $(DINKOBJECTS)
                $(LINK) $(DINKOBJECTS) $(LIBS) -o dink32.src

clean:
        $(DEL) -f *.o *.lst *.map dink32.src dcomp zz.*

#.s.o:
#       $(PREP) $*.i
#       $(AS) $*.s

reset.o:  $(INC_ALL) $(INC_ASM)  reset.s
        $(ASB) reset.s
reset1.o: $(INC_ALL) $(INC_ASM)  reset1.s
        $(ASB) reset1.s
except2l.o : $(INC_ALL) $(INC_ASM) except2l.s
        $(ASL)  except2l.s
reg_swap.o:  $(INC_ALL) $(INC_ASM) reg_swap.s
        $(ASL)  reg_swap.s
go_tr2.o: $(INC_ALL) $(INC_ASM) go_tr2.s
        $(ASL)   go_tr2.s

.c.o:
        $(CCobj) $*.c

INC_ALL = config.h
INC_C    = dink.h
INC_TOK = tok_tb.h toks.h
INC_GEN = errors.h cpu.h
```

```
INC_ASM = dink_asm.h yellowknife.h

reg_tb.o: $(INC_ALL) $(INC_C) $(INC_TOK) $(INC_GEN) reg_tb.c reg_tb.h
reg_spr.o: $(INC_ALL) $(INC_C) $(INC_TOK) $(INC_GEN) reg_spr.c reg_tb.h
mem_tb.o: $(INC_ALL) $(INC_C) $(INC_TOK)$(INC_GEN) mem_tb.c
tok_tb.o: $(INC_ALL) $(INC_C) $(INC_TOK) tok_tb.c
arg_tb.o: $(INC_ALL) $(INC_C) $(INC_TOK) $(INC_GEN) arg_tb.c rfs_tb.h
rfs_tb.o:$(INC_ALL) $(INC_C)  $(INC_GEN) rfs_tb.c rfs_tb.h
par_tb.o: $(INC_ALL) $(INC_C) $(INC_TOK) par_tb.c errors.h
toks.o:  $(INC_ALL) $(INC_C) $(INC_TOK) toks.c errors.h
err_tb.o : $(INC_ALL) $(INC_C) $(INC_TOK) err_tb.c err_tb.h
errors.o : $(INC_ALL) $(INC_C) $(INC_GEN) errors.c
help.o : $(INC_ALL) $(INC_C) $(INC_TOK) help.c arg_tb.h rfs_tb.h errors.h
brk_pts.o :$(INC_ALL) $(INC_C) $(INC_GEN) brk_pts.c brk_pts.h
sublib.o :$(INC_ALL) $(INC_C) $(INC_GEN) sublib.c
netrix1.o : $(INC_ALL)                              netrix1.c
netrix.o : $(INC_ALL)                               netrix.c
except1.o : $(INC_ALL) $(INC_C)             except1.c
#except2.o : $(INC_ALL) $(INC_ASM)          except2.s
#go_tr2.o : $(INC_ALL) $(INC_ASM)           go_tr2.s
go_tr1.o : $(INC_ALL) $(INC_C)              go_tr1.c
#reg_swap.o : $(INC_ALL) $(INC_ASM)         reg_swap.s
dc_tb.o :$(INC_ALL) $(INC_C) $(INC_GEN) dc_tb.c
downld.o : $(INC_ALL) $(INC_C) $(INC_GEN)downld.c
duart.o: $(INC_ALL) $(INC_C) $(INC_GEN) duart.c duart.h
print.o : $(INC_ALL) $(INC_C) $(INC_GEN) print.c
board.o : $(INC_ALL) $(INC_C) $(INC_GEN) board.c duart.h
asm.o :  $(INC_ALL) $(INC_C) $(INC_GEN) asm.c asm_dsm.h
dsm.o :  $(INC_ALL) $(INC_C) $(INC_GEN) dsm.c asm_dsm.h
main.o : $(INC_ALL) $(INC_C) $(INC_TOK) main.c errors.h arg_tb.h reg_tb.h duart.h
```