CML-5485

Application Board for Freescale MCF5485 MCU
Hardware User Manual

Axiom Manufacturing • 2813 Industrial Lane • Garland, TX 75041
Email: Sales@axman.com Web: http://www.axman.com

CONTENTS

Cautionary Notes

1) Electrostatic Discharge (ESD) prevention measures should be applied whenever handling this product. ESD damage is not a warranty repair item.

2) Axiom Manufacturing reserves the right to make changes without further notice to any products to improve reliability, function or design. Axiom Manufacturing does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under patent rights or the rights of others.

3) EMC Information on the CML-5485 board:

a) This product as shipped from the factory with associated power supplies and cables, has NOT been tested for requirements of CE and the FCC a **CLASS A** products.

b) This product is designed and intended for use as a development platform for hardware or software in an educational / professional laboratory or as a component in a larger system.

c) In a domestic environment this product may cause radio interference in which case the user may be required to take adequate prevention measures.

d) Attaching additional wiring to this product or modifying the products operation from the factory default as shipped may effect its performance and also cause interference with other apparatus in the immediate vicinity. If such interference is detected, suitable mitigating measures should be taken.

Terminology

This development board applies option selection jumpers. Terminology for application of the option jumpers is as follows:

Jumper on, in, or installed = jumper is a plastic shunt that fits across 2 pins and the shunt is installed so that the 2 pins are connected with the shunt.

Jumper off, out, or idle = jumper or shunt is installed so that only 1 pin holds the shunt, no 2 pins are connected, or jumper is removed. It is recommended that the jumpers be placed idle by installing on 1 pin so they will not be lost.

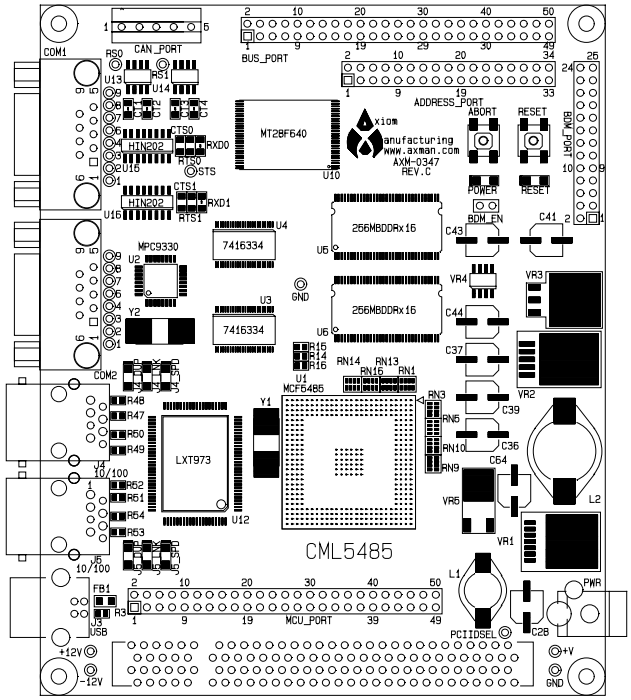
This development board applies option selections indicated CT or R designations that require a razor knife and soldering tool to install or remove. The default closed type connection places an equivalent Jumper Installed type option. Applying the connection can be performed by installing a resistor component or small wire for 0 ohms between the option pads. See the Options sections for more details.

FEATURES

CML-5485 is a low cost development kit for the Freescale MCF5485 ColdFire® microcontroller. Application development is quick and easy with the included Axiom CML-5485 evaluation / application board, DB9 serial cable, Ethernet cable, BDM Cable, and GNU Software Tool Package that includes C compiler and source level debug utilities. The BDM port is compatible with standard ColdFire® BDM / JTAG development cables.

Features:

- MCF5485 CPU
  - 200Mhz operation
  - 32K Byte Ram (on chip)
  - 32K Byte Data and Instruction Caches
  - Double Precision Floating Point Unit (FPU)
  - Memory management Unit (MMU)
  - Security Encryption Controller (SEC)
  - PCI V2.2 bus controller w/ arbitration unit
  - 16 Channel DMA Controller
  - 4 x 16 bit Timers w/ PWM
  - DSPI and IIC Serial Ports
  - 4 x UART Ports with IRDA / DMA capability
  - USB V2.0 device transceiver, 6 endpoints
  - Interrupt Controller
  - BDM / JTAG Port
  - 3.3V / 1.5V operation
- 8M Byte Flash (16 bit, external)
- 64M Bytes DDR RAM (32 bit, external)
- 25Mhz Reference clock, 200Mhz core operation
- 2 x 10/100TX Ethernet Ports w/ RJ45 Connectors,
  LNK, DUP, SPD indicators for each port
  Auto Negotiation and MDIX connections
- 2 x COM Ports (UART0/1) w/ RS232 DB9-S Connectors
- USB type B connector w/ USB device I/O
- 2 x CAN Ports w/ 1 M baud transceivers, 5 pin Term Block
- PCI V2.2 Bus connector, 3.3V bus, 50Mhz maximum.
- MCU Port, 50 pins w/ I/O port signals
- ADDRESS Port, 34 pin Demultiplexed Address BUS signals
- BUS Port, 50 pins, Multiplexed Address and Data signals w/ control signals
- BDM / JTAG Port, 26 Pins, development port.
- RESET switch and indicator
- ABORT (IRQ7) switch
- Regulated +5V, 3.3V, 2.5V and 1.5V power supplies w/ 3.3V indicator
- Supplied with DB9 Serial Cable, Cat 5E Ethernet cable, BDM Development Cable,
- Utility / Support CD, Manuals, and 12V Universal Wall Adapter power supply.



CML-5485

Specifications:

Board Size 4.5” x 5.0”, 8 layers
Power Input: +8 - +24VDC, 12VDC typical
Current Consumption: 200ma @ 12VDC input

CML-5485 is a low cost development system for the Freescale MCF5485 Coldfire® microcontroller. Application development is quick and easy with the included DB9 serial cable, Ethernet cable, Debug firmware monitor, and GNU c compiler with utilities. The BDM port is compatible with standard Coldfire BDM / JTAG interface cables and hosting software, allowing easy application debugging and development.

GETTING STARTED

The CML-5485 single board computer is a fully assembled, fully functional application board for the Freescale MCF5485 microcontroller. Provided with wall plug power supply, Ethernet cable, and serial cable. Support software provided for this development board is for Windows 95/98/NT/2000/XP operating systems.

Development board users should also be familiar with the hardware and software operation of the target MCF5485 device, refer to the provided Freescale User Guide for the device and the Coldfire Reference Manual for details. The development board purpose is to assist the user in quickly developing an application with a known working environment, to provide an evaluation platform, or as a control module for an applied system. Users should be familiar with memory mapping, memory types, and embedded software design for the quickest successful application development.

Software Development

Application development maybe performed by applying the dBUG firmware monitor, or by applying a compatible Coldfire BDM / JTAG cable with supporting host software. The monitor provides an effective and low cost command line debug method.

Software development is best performed with a development tool connected to the BDM port. This provides real-time access to all hardware, peripherals and memory on the board. Development tool software also provides high-level (C/C++) source code debug environment.

The target development environment and procedure for best success is to place software to be tested into RAM memory. Execute software to be tested under dBUG monitor or development tool control, after software is tested and operational, port and program application into FLASH memory to execute new application when power is applied.

Reference Documentation

Reference documents are provided on the support CD in Acrobat Reader® format.

CML5485UM.pdf – This user manual.
MCF5485UM.pdf – MCF5485 Device User Manual
CFPRM.pdf – Coldfire Programmers Reference Manual with instruction set
CML5485_SCH_C.pdf – CML5485 board schematics

CML-5485 Startup

Follow these steps to connect and power on the board for the default dBUG monitor operation.

1) Carefully unpack the CML-5485 and observe ESD preventive measures while handling the CML-5485 development board.

2) Load the ColdFire support CD into the PC and install the AxIDE terminal software from the utilities directory, OR configure HyperTerminal for a direct connection to the PC COM port to be applied for serial communication with the CML-5485 board. Set the baud rate to 19.2K baud, 8 data bits, 1 stop bit, and no parity. Software XON / XOFF flow control should be enabled for flash memory support operations. Use the AxIDE '√' tool bar button to configure the COM port on the PC.

3) Connect the CML-5485 board COM1 serial port connector to the host PC COM port with the provided 9 pin serial cable.

4) Apply power to the development board by installing the wall plug power supply between a wall outlet and the PWR Jack on the board. The board voltage indicators should turn on at this time. The RESET indicator will flash during power on or Reset switch press.

5) Observe the AxIDE or HyperTerminal window display for the dBUG monitor prompt. Prompt should be similar to the following:

```
Hard Reset
DRAM Size: 64M

Copyright 1995-2003 Freescale, Inc. All Rights Reserved.
ColdFire MCF5485  Firmware v2e.1a.xx (Build XXX on XXX  XX 20XXxx:xx:xx)

Enter 'help' for help.

dBUG>
```

6) The board is ready to use now. See the dBUG monitor section of the user manual for additional monitor information. If BDM / JTAG development port interfaced tools are to be applied, see the BDM PORT section of this manual for more details on installation.

CML-5485 Hardware Configuration and Options

The CML-5485 board provides a basic development or application platform for the MCF5485 microcontroller. Following are descriptions of the main components and options provided on the board.

POWER SUPPLY

Input power is applied by external connection to the PWR power jack. The input supply is provided to the 3.3V primary supply regulator VR1 and the PCI +5V regulator VR5.

VR2 provides 2.5V peripheral and DDRAM supply from the 3.3V supply. VR3 provides the 1.5V core supply from the 3.3V supply. VR4 is the DDRAM termination supply. With +8 to +20VDC applied at the PWR jack, the POWER Indicator should be ON. External fuse of 1A and optional ON / OFF switch should be applied in system applications to the PWR jack input supply.

RESET and RESET_SW

External reset is provided by the RESET switch, LV1 low voltage detector, or user applied connection to the RESET* signal on the BUS_PORT. These external Reset sources activate a 150ms minimum pulse duration to the MCF5485 RSTI* input. If the main 3.3V supply is below operating level, the LV1 voltage detector will cause the MCF5485 to stay in the RESET condition.

The BDM port provides a direct connection to the MCF5485 RSTI* for Reset application by development tools.

Application of RESET will cause the dBUG monitor or user application to initialize the MCF5485. Previous DDRAM memory content and operating state of the MCF5485 will be lost.

INDICATORS

Indication is provided for power supply status, CPU Reset status, and Ethernet status. The indications may be applied to determine proper operation of the development board.

Indicator Summary

INDICATOR	COLOR	OPERATION	DEFAULT CONDITION
POWER	Green	+3.3V power present	ON
RESET	RED	CPU is in RESET state	OFF
LNK	Green	Ethernet channel has Link	ON with Network connected
100	Green	Ethernet channel is operating 100 base	ON if 100 base network, OFF if 10 base network
DUP	Green	Ethernet channel is operating full duplex	ON if full duplex network

ABORT Switch

The ABORT switch provides for manual application of the IRQ7 interrupt signal. This operation will allow the dBUG monitor to stop execution of a user program and maintain the CPU operating state for user examination. After application of the ABORT switch, dBUG will prompt the current program instruction pending. Display is similar to a breakpoint operation.

SYSTEM CLOCK

The system clocks are provided by U2 with 25MHz reference oscillator Y2. U2 provides separate 50MHz primary clocks to the MCF5485, PCI Bus and connector J1, and the BUS_PORT if R63 is installed. A 25Mhz clock from U2 is also provided to the Ethernet PHY device U12. The MCF5485 local bus operates at the 50Mhz primary clock frequency. Internal PLL clock generation of the MCF5485 provides 100MHz DDRAM clock and 200MHz core clock. The MCF5485 USB reference clock of 12MHz is provided by crystal oscillator Y1.

MEMORY

Memory is provided in the MCF5485 device and externally on the development board. The MCF5485 provides 64K bytes of SRAM memory internally. Board memory consists of 64M Bytes of DDRAM on the SDRAM bus and 8M bytes of Flash memory on the FLEX Bus. The debug monitor occupies the first or lower 256K bytes of the board Flash memory. User should refer to the dBUG memory map for default memory allocation and physical locations.

The BUS and ADDRESS Ports maybe applied for Flex Bus expansion off board.

Note: The MCF5485 DDRAM controller must be configured prior to application of the memory space when BDM or JTAG tools are applied.

COMMUNICATION PERIPHERALS

The CML-5485 provides 7 conditioned communication ports, 2 x 10/100TX Ethernet, 2 x RS232 Serial COM, 2 x CAN, and a USB port. User should refer to the respective manual chapter for details of operation and connection of each port.

CML-5485 PORTS AND CONNECTORS

PWR - Power Jack

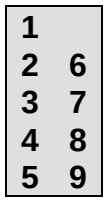
PWR provides the default power input to the board. The power jack accepts a standard 2.0 ~ 2.1mm center barrel plug connector (positive voltage center) to provide the +V supply of +8 to +20 VDC (+12VDC typical).

+Volts, 2mm center

COM1 and COM2 Ports

The COM1 and 2 ports provide standard 9 pin serial connection with RS232 type interface to the MCF5485 UART0 and UART1 peripherals respectfully. The COM1 port is applied by the dBUG monitor at 19.2K baud default. Both ports will connect to a standard PC COM port with a straight through type 9 pin serial cable. Following is the DB9S connection reference.

COM1 or COM2



The **COM-1or 2** port is a DB9 socket connector with RS232 signal levels.
RXD maybe isolated from port by option, see chart below.
CTS / RTS are enabled by options, see chart below.
1,4,6 connected for status null to host

The 1, 4, 6 and 9 pins provide RS232 status signals. These DB9 connector locations are provided access pads behind the connector.

COM PORT OPTIONS

The following table provides COM options and default connections to the MCF5485 Programmable Serial Controllers (PSC).

Port	Signal	Option	Default	MCF5485 Signal
COM1	RXD	RXD0	Closed	PSC0RXD
COM1	RTS	RTS0	Open	PSC0RTS
COM1	CTS	CTS0	Open	PSC0CTS
COM2	RXD	RXD1	Closed	PSC1RXD
COM2	RTS	RTS1	Open	PSC1RTS
COM2	CTS	CTS1	Open	PSC1CTS

Notes:
1) All options are SMT 0805 component size pads.
2) Default closed options must be opened by cutting trace between option pads.
3) To close an option a SMT 0805 resistor of 0 ohms maybe applied.

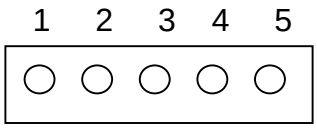
CAN Port

The CAN port provides two physical interface layers for the two channels of MCF5485 FlexCAN Controller Area Network version 2.0B peripheral. The FlexCAN transmit and receive signals are connected to a 3.3V CAN transceivers capable of 1M baud communication (SN65HVD230). Transceiver differential CAN network signals are provided a 62 ohm termination by components RC1 and RC2, and applied to the CAN port connector.

Each CAN transceiver has CAN signal drive control via the **RS0 / 1** test pad on the development board. The RS signal is provided a 1K Ohm pull-down resistor for the maximum signal rate setting. User may refer to the SN65HVD230 data sheet and apply additional signal control at the RS test pad.

The CAN Port connector is a 5 position screw terminal block that will accept discreet wiring for CAN Bus connections.

CAN Port Term Block



CAN PORT Connection and Options Table

CAN Port	Port Signal		MCF5485 Signals	OPTION	Default
1	CAN_HI_0		CANTX0	CT1	Closed
2	CAN_LO_0		CANRX0	CT2	Closed
3	Ground				
4	CAN_HI_1		CANTX1	CT3	Closed
5	CAN_LO_1		CANRX1	CT4	Closed

Notes:
1) All options are SMT 0805 component size pads.
2) Default closed options must be opened by cutting trace between option pads.
3) To close an option a SMT 0805 resistor of 0 ohms maybe applied.

J4 and J5 ETHERNET Ports

These ports provide the 10/100TX Ethernet physical interface connections to the MCF5485 FEC0 and 1 respectfully. Port configuration applies a dual PHY device, the Intel LXT973 10/100TX physical layer transceiver (PHY) for a complete IEEE802.3 interface. Features of the port include Auto MDX cable or connection configuration type detection, 3 status indicators per port, and integrated magnetics RJ45 connectors.

Refer to the LXT973 data sheet for register and operation details.

Ethernet Status Indicators

LNK	Green	Ethernet has Link	ON with Network connected
100	Green	Ethernet is operating 100 base	ON if 100 base network, OFF if 10 base network
DUP	Green	Ethernet is operating full duplex	ON if full duplex network

Note: Ethernet port must be initialized and operating for status indications.

Refer to the CML5485_SCH_C.pdf drawing for details on hardware connections to this port.

MCU_PORT

The MCU PORT provides access to the MCF5485 I/O ports. Ports applied on the board are noted.

	DACK0*	**1**	**2**	DACK1*	
	DREQ0*	**3**	**4**	DREQ1*	
(Note 1)	CANRX1	**5**	**6**	CANTX1	(Note 1)
(Note 1)	CANRX0	**7**	**8**	CANTX0	(Note 1)
	TIN2	**9**	**10**	TOUT2	
	TIN1	**11**	**12**	TOUT1	
	TIN0	**13**	**14**	TOUT0	
	PSC3RTS*	**15**	**16**	PSC3CTS*	
	PSC3RXD	**17**	**18**	PSC3TXD	
	PSC2RXD	**19**	**20**	PSC2TXD	
(Note 2)	PSC1RTS	**21**	**22**	PSC1CTS*	(Note 2)
(Note 2)	PSC1RXD	**23**	**24**	PSC1TXD	(Note 2)
(Note 3)	PSC0RTS*	**25**	**26**	PSC0CTS*	(Note 3)
(Note 3)	PSC0RXD	**27**	**28**	PSC0TXD	(Note 3)
	DSPISOUT	**29**	**30**	DSPISIN	
	DSPISCK	**31**	**32**	DSPICS5	
	DSPICS3	**33**	**34**	DSPICS2	
	DSPICS0	**35**	**36**	SDA	
	SCL	**37**	**38**	PCIBG0*	
	PCIBG1*	**39**	**40**	PCIBG2*	
	PCIBG3*	**41**	**42**	PCIBR0*	(Note 4)
	PCIBR1*	**43**	**44**	PCIBR2*	
	PCIBR3*	**45**	**46**	IRQ7*	
	IRQ6*	**47**	**48**	IRQ5*	
	GND	**49**	**50**	+3.3V	

Notes:
1) These signals have peripheral connection on the CML-5485 board CAN Port.
2) These signals have peripheral connection on the CML-5485 board COM2 Port.
3) These signals have peripheral connection on the CML-5485 board COM1 Port.
4) This signal has peripheral connection on the CML-5485 board J1 PCI Port.

BUS_PORT

The BUS PORT provides access to the MCF5485 FLEX Bus data and control signals. Most signals on the BUS PORT have a peripheral connection on the CML-5485 board. The FBCS0* chip select is dedicated to the on board flash memory. The board must boot from the on board flash.

AD30	**1**	**2**	AD31
AD28	**3**	**4**	AD29
AD26	**5**	**6**	AD27
AD24	**7**	**8**	AD25
AD22	**9**	**10**	AD23
AD20	**11**	**12**	AD21
AD18	**13**	**14**	AD19
AD16	**15**	**16**	AD17
AD14	**17**	**18**	AD15
AD12	**19**	**20**	AD13
A10	**21**	**22**	AD11
AD8	**23**	**24**	AD9
AD6	**25**	**26**	AD7
AD4	**27**	**28**	AD5
AD2	**29**	**30**	AD3
AD0	**31**	**32**	AD1
RSTO*	**33**	**34**	RESET* IN
(Note 1) CLKOUT	**35**	**36**	FBCS1*
FBCS2*	**37**	**38**	FBCS3*
FBCS4*	**39**	**40**	FBCS5*
BWE0*	**41**	**42**	BWE1*
BWE2*	**43**	**44**	BWE3*
R/W*	**45**	**46**	TS*
TA*	**47**	**48**	OE*
+3.3V	**49**	**50**	GND

Notes:

1) The CLKOUT signal must be enabled by installing option resistor R63. The value of R63 should be selected to reduce connection reflections of the 50MHz signal.
2) The default bus size of FBCS0* is 16 bits data on AD16 – AD31.
3) The FLEX BUS operates in Multiplexed address and data mode by default. The address port provides de-multiplexed address signals.

ADDRESS_PORT

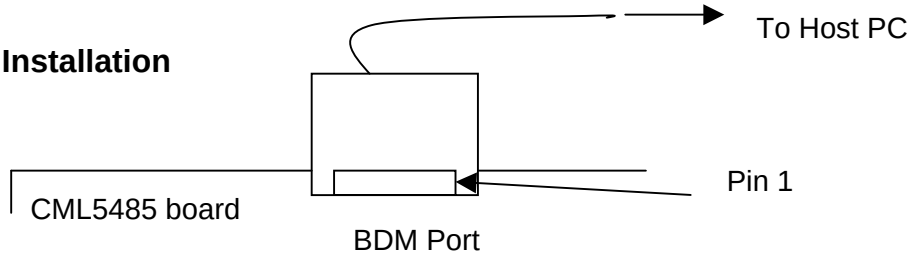
The ADDRESS PORT provides access to the MCF5485 FLEX Bus latched address signals. This port may be applied for bus expansion.

A0	**1**	**2**	A1
A2	**3**	**4**	A3
A4	**5**	**6**	A5
A6	**7**	**8**	A7
A8	**9**	**10**	A9
A10	**11**	**12**	A11
A12	**13**	**14**	A13
A14	**15**	**16**	A15
A16	**17**	**18**	A17
A18	**19**	**20**	A19
A20	**21**	**22**	A21
A22	**23**	**24**	A23
A24	**25**	**26**	A25
A26	**27**	**28**	A27
A28	**29**	**30**	A29
A30	**31**	**32**	A31
+3.3V	**33**	**34**	GND

BDM_PORT

The BDM PORT provides a standard Coldfire BDM / JTAG development port. The **BDM_EN** option provides for the development port mode selection between BDM or JTAG.

Development Cable Installation



BDM_EN

INSTALLED = BDM_PORT is in BDM Mode (Development mode = BDM).

OPEN = BDM_PORT is in JTAG Mode (Development mode = JTAG).

BDM /JTAG Port Connection

		1	**2**	BKPT*
	GND	**3**	**4**	DSCLK
	GND	**5**	**6**	TCLK
(Note 1)	RSTI*	**7**	**8**	DSI
	+3.3V	**9**	**10**	DSO
	GND	**11**	**12**	PST3
	PST2	**13**	**14**	PST1
	PST0	**15**	**16**	DDATA3
	DDATA2	**17**	**18**	DDATA1
	DDATA0	**19**	**20**	GND
		21	**22**	
	GND	**23**	**24**	CLKOUT
	+3.3V	**25**	**26**	TA*

J1 PCI Port

J1 provides a single 3.3V standard 32 bit PCI bus connector, slot 4 connection (0 – 3 slots not applied). The MCF5485 PCI bus is applied to this connector with arbitration signals and external master support. Due to the fixed system frequency only 66Mhz PCI cards should be applied for an operating PCI bus frequency of 50Mhz. Following is the connector and signal detail:

J1 PCI Bus Connector

Notes	PCI Signal	J1 PIN Number		PCI Signal	Notes
From -12V TP connection	-12V	B1	A1	TRST*	Not applied, 4.7K ohm pull-down
Not applied, 4.7K ohm pull-down	TCK	B2	A2	+12V	From +12V TP connection
	GND	B3	A3	TMS	Not applied
Not applied	TDO	B4	A4	TDI	Not applied
	+5V	B5	A5	+5V	
	+5V	B6	A6	INTA*	MCF5485 IRQ6*
MCF5485 IRQ6*	INTB*	B7	A7	INTC*	MCF5485 IRQ6*
MCF5485 IRQ6*	INTD*	B8	A8	+5V	
.1UF connect only	PRSNT1*	B9	A9		Reserved
Reserved		B10	A10	+3.3V	
.1UF connect only	PRSNT2*	B11	A11		Reserved
Card slot Key		B12	A12		Card slot Key
Card slot Key		B13	A13		Card slot Key
Reserved		B14	A14		Reserved
	GND	B15	A15	PCI_RST*	
50MHz	PCICLK	B16	A16	+3.3V	
	GND	B17	A17	PCI_BG4*	
	PCI_BR4*	B18	A18	GND	
	+3.3V	B19	A19		Reserved
	PCI_AD31	B20	A20	PCI_AD30	
	PCI_AD29	B21	A21	+3.3V	
	GND	B22	A22	PCI_AD28	
	PCI_AD27	B23	A23	PCI_AD26	
	PCI_AD25	B24	A24	GND	
	+3.3V	B25	A25	PCI_AD24	
	PCI_BE3*	B26	A26	IDSEL	PCI_AD17 applied, base 0x20000
	PCI_AD23	B27	A27	+3.3V	
	GND	B28	A28	PCI_AD22	
	PCI_AD21	B29	A29	PCI_AD20	
	PCI_AD19	B30	A30	GND	
	+3.3V	B31	A31	PCI_AD18	
	PCI_AD17	B32	A32	PCI_AD16	
	PCI_BE2*	B33	A33	+3.3V	
	GND	B34	A34	PCI_FRAME*	
	PCI_IRDY*	B35	A35	GND	
	+3.3V	B36	A36	PCI_TRDY*	
	PCI_DEVSEL*	B37	A37	GND	
	GND	B38	A38	PCI_STOP*	
LOCK not applied		B39	A39	+3.3V	
	PCI_PERR*	B40	A40		SDONE not applied
	+3.3V	B41	A41		SB0* not applied
	PCI_SERR*	B42	A42	GND	
	+3.3V	B43	A43	PCI_PAR	
	PCI_BE1*	B44	A44	PCI_AD15	
	PCI_AD14	B45	A45	+3.3V	
	GND	B46	A46	PCI_AD13	
	PCI_AD12	B47	A47	PCI_AD11	
	PCI_AD10	B48	A48	GND	
PCI_BR0* applied	MCGEN	B49	A49	PCI_AD9	
	GND	B50	A50	GND	
	GND	B51	A51	GND	
	PCI_AD8	B52	A52	PCI_BE0*	
	PCI_AD7	B53	A53	+3.3V	
	+3.3V	B54	A54	PCI_AD6	
	PCI_AD5	B55	A55	PCI_AD4	
	PCI_AD3	B56	A56	GND	
	GND	B57	A57	PCI_AD2	
	PCI_AD1	B58	A58	PCI_AD0	
	+3.3V	B59	A59	+3.3V	
4.7kK ohm pull-up only	ACK64*	B60	A60	REQ64*	4.7K Ohm pull-up only
	+5V	B61	A61	+5V	
	+5V	B62	A62	+5V	

TROUBLESHOOTING

The CML5485 is fully tested and operational before shipping. If it fails to function properly, inspect the board for obvious physical damage first. Verify the communications setup as described under GETTING STARTED.

The most common problems are improperly configured options or communications parameters.

1. Verify default option settings and RESET the board.

2. Make sure that the RSTI* line is not being held low or the RESET indicator is not on constantly.

3. Verify that your COM communications port is working by substituting a known good serial device or by doing a loop back diagnostic. if you applied a different baud rate with the dBUG SET command, make sure the terminal software is set correctly.

4. Verify the power source, +3.3V and +2.5V Indicators are ON? You should measure a minimum of 9 volts between the GND and +V test pad and GND test pad near the power jack with the standard power supply provided.

5. If no power indications or voltage is found, verify the wall plug connections to AC outlet and the PWR jack power connector.

6. Disconnect all external connections to the board except for COM1 to the PC and the wall plug and check operation again.

7. Contact support@axman.com by email for further assistance. Provide board name and describe problem.

dBUG MONITOR OPERATION

dBUG is a firmware resident development environment operated by the MCF5485 as a primary control program. The monitor provides serial and Ethernet communication for loading and controlling the execution of software under test. User should note that the monitor occupies the first or lower address 256K bytes of the external flash memory. Caution should be applied in the user application not to corrupt the monitor flash memory space. If the monitor is corrupted, a development port tool will be required to restore operation.

dBUG Communication:

Primary user interface to the dBUG monitor is by command lines that are entered into the serial port. These commands are defined in the following table “dBUG Commands”. For serial communications, dBUG requires eight data bits, no parity, and one stop bit, 8N1 with XON/XOFF soft flow control. The default baud rate is 19200 however, this rate can be changed by the user with a “set” command. The command line prompt is “dBUG> “. Any dBUG command may be entered from this prompt. dBUG does not allow command lines to exceed 80 characters. Wherever possible, dBUG displays data in 80 columns or less. dBUG echoes each character as it is typed, eliminating the need for any “local echo” on the terminal side. In general, dBUG is not case sensitive. Commands may be entered either in upper or lower case, depending upon the user’s equipment and preference. Only symbol names require that the exact case be used.

See the dBUG Ethernet Support section in this manual for details on operating this port.

dBUG System Initialization

The act of powering up the board will initialize the system. The processor is reset and dBUG is invoked. dBUG performs the following configurations of internal resources during the initialization:

MCF5485 clock is initialized to 200MHz.

The instruction cache is invalidated and disabled.

The Software Watchdog Timer is disabled and internal timers are placed in a stop condition.

dBUG memory map is configured.

Interrupt Service Support

Interrupt controller register is initialized with unique interrupt level/priority pairs. Please refer to the dBUG source files on the ColdFire support CD for the complete initialization code sequence.

The Vector Base Register, VBR, points to the dBUG Flash memory space. However, a copy of the exception table is made at address $00000000 in SDRAM memory space. To take over an exception vector, the user places the address of the exception handler in the appropriate vector in the vector table located at 0x00000000, and then points the VBR register to 0x00000000. See the MCF5485 User Guide and ColdFire Programmers Reference Manual for more details on applying interrupts.

dBUG Memory Map

0x00000000 0x000003FF	External DDRAM Memory: User Vector table if applied. See Interrupt Support for more information. 1K bytes
0x00000400 0x0001FFFF	External DDRAM Memory: dBUG **reserved** ram space. 128K bytes
0x00020000 0x03FFFFFF	External DDRAM Memory: User ram or development memory space 64M bytes, 32 bits wide, 100Mhz
0x04000000 - 0x3FFFFFFF	Not applied memory space. **reserved**
0x40000000 0x7FFFFFFF	ISBAR: MCF5485 Internal register and peripheral space. Includes internal ram space. Refer to MCF5485 User Manual for details.
0x80000000 - 0xFF7FFFFF	Not applied memory space. **reserved**
0xFF800000 0xFF83FFFF	DBUG Monitor Flash Memory Space. **reserved** 256K bytes
0xFF840000 0xFFFFFFFF	User External Flash Memory Space 7.8M bytes, 16 bits wide

Note: Applying BDM / JTAG development port tools does not require applying the dBUG memory map.

dBUG Commands

After the system initialization, the dBUG waits for a command-line input from the user terminal. When a proper command is entered, the operation continues in one of the two basic modes. If the command causes execution of the user program, the dBUG firmware may or may not be re-entered, at the discretion of the user's program. For the alternate case, the command will be executed under control of the dBUG firmware, and after command completion, the system returns to command entry mode.

dBUG Command Table

MNEMONIC	SYNTAX	DESCRIPTION
ASM	asm <<addr> stmt	Assemble
BC	bc addr1 addr2 length	Block Compare
BF	bf <width> begin end data <inc>	Block Fill
BM	bm begin end dest	Block Move
BR	br addr <-r> <-c count> <-t trigger>	Breakpoint
BS	bs <width> begin end data	Block Search
DC	dc value	Data Convert
DI	di <addr>	Disassemble
DL	dl <offset>	Download Serial
DLDBUG	dldbug	Download dBUG Update
DN	dn <-c> <-e> <-i> <-s> <-o offset> <filename>	Download Network
FL	fl <command> dest <src> size	Flash write or erase
GO	go <addr>	Execute
GT	gt addr	Execute To
HELP	help <command>	Help
IRD	ird <module.register> Internal	Internal Register Display
IRM	irm module.register data	Internal Register Modify
LR	lr <width> addr	Loop Read
LW	lw <width> addr data	Loop Write
MD	md <width> <begin> <end>	Memory Display
MM	mm <width> addr <data>	Memory Modify
MMAP	mmap	Memory Map Display
RD	rd <reg>	Register Display (core)
RM	rm reg data	Register Modify (core)
RESET	reset	Reset
SD	sd	Stack Display (contents)
SET	set <option value>	Set Configurations
SHOW	show <option>	Show Configurations
STEP	step	Step (Over)
SYM	symbol <symb> <-a symb value> <-r symb> <-C| l |s>	Symbol Management
TRACE	trace <num>	Trace (Into)
UP	up begin end filename	Upload binary data
VER	version	Show dBUG Version

During command execution, additional user input may be required depending on the command function. For commands that accept an optional <width> to modify the memory access size, the valid values are:

• B = 8-bit (byte) access • W = 16-bit (word) access • L = 32-bit (long) access

When no <width> option is provided, the default width is “W”, 16-bit.

The core ColdFire register set is maintained by dBUG. These are listed below:

• A0 - A7
• D0 - D7
• PC
• SR

All control registers on ColdFire are not readable by the supervisor-programming model, and thus not accessible via dBUG. User code may change these registers, but caution must be exercised as changes may render dBUG inoperable. A reference to “SP” (stack pointer) actually refers to general purpose address register seven, “A7.”

The commands DI, GO, MD, STEP and TRACE are used repeatedly when debugging. dBUG recognizes this and allows for repeated execution of these commands with minimal typing. After a command is entered, simply press <RETURN> or <ENTER> to invoke the command again. The command is executed as if no command line parameters were provided.

User programs are provided access to various dBUG routines by the “Trap 15 Functions". These functions are discussed at the end of this chapter.

dBUG Ethernet Support

Ethernet support by the dBUG monitor is limited to TFTP (Trivial File Transfer Protocol) downloads of object code files. Note that this operation requires an Ethernet TFTP server to be running on the host (usually a PC) attached to the development board. The support CD provides simple TFTP host software (TFTPD32) for a Windows based PC if needed. Prior to using this feature, several parameters must be configured for network downloads to occur. The information that is required and the steps for configuring dBUG are described below.

Required Network Parameters: To perform network downloads, dBUG needs 7 parameters of which 5 are network-related and 2 are download-related. The parameters are listed below, with the dBUG designation following in parenthesis. All computers connected to an Ethernet network running the IP protocol need 4 network-specific parameters. These parameters are:

- Internet Protocol
- IP address for the computer (client IP)
- IP address of the Gateway for non-local traffic (gateway IP)
- Network netmask for flagging traffic as local or non-local (netmask)

In addition, the dBUG network download command requires the following three parameters:

- IP address of the TFTP server (server IP)
- Name of the file to download (filename)
- Type of the file to download (file type of S-record, COFF, ELF, or Image)

Your local system administrator can assign a unique IP address for the board, and also provide you the IP addresses of the gateway, netmask, and TFTP server. Fill out the lines below with this information:

Client IP: ___.___.___.___ (IP address of the board)

Server IP: ___.___.___.___ (IP address of the TFTP server)

Gateway: ___.___.___.___ (IP address of the gateway)

Netmask: ___.___.___.___ (Network netmask)

Configuring dBUG Network Parameters

Once the network parameters are known, the dBUG Monitor must be configured. The following commands are used to configure the network parameters:

set client <client IP>
set server <server IP>
set gateway <gateway IP>
set netmask <netmask>
set mac <addr>

For an example, the TFTP server has IP address of 123.45.67.1. The board is assigned the IP address of 123.45.68.15. The gateway IP address is 123.45.68.250, and the netmask is 255.255.255.0. The MAC address is chosen arbitrarily and is unique. The commands to dBUG are:

set client 123.45.68.15

set server 123.45.67.1

set gateway 123.45.68.250

set netmask 255.255.255.0

set mac 00:CF:52:82:EB:01

The last step is to inform dBUG of the name and type of the file to download. Prior to giving the name of the file, keep in mind that the host TFTP server will require the file to be transferred be in the TFTP server's assigned base directory or a subdirectory from the base directory (depends on security settings). See the TFTP server software setup configuration and help file to locate or change this directory.

A default file name for network downloads is maintained by dBUG. To change the default file name, use the command:

set filename <filename>

When using the Ethernet network for download, either S-record, COFF, ELF, or Image files may be downloaded. A default file type for network downloads is maintained by dBUG as well. To change the default file type, use the command:

set filetype <S record | coff | elf | image>

Continuing with the above example, the Coldfire C compiler produces an executable COFF file, 'example.out'. This file is copied to the TFTP server base directory for download to dBUG. Note: If the TFTP server base directory is assigned to the compiler output files directory on the host PC, no file copy to another directory is required.

Change the dBUG default filename and file type with the commands:

set filename example.out

set filetype coff

Finally, perform the network download with the '**dn**' command. The network download process uses the configured IP addresses and the default filename and file type for initiating a TFTP download from the TFTP server.

Appendix 1: dBUG Command Set

ASM - Assembler

Usage: ASM <<addr> stmt>

The ASM command is a primitive assembler. The <**stmt**> is assembled and the resulting code placed at <**add**r>. This command has an interactive and non-interactive mode of operation.

The value for address <addr> may be an absolute address specified as a hexadecimal value, or a symbol name. The value for stmt must be valid assembler mnemonics for the CPU.

For the interactive mode, the user enters the command and the optional <addr>. If the address is not specified, then the last address is used. The memory contents at the address are disassembled, and the user prompted for the new assembly. If valid, the new assembly is placed into memory, and the address incremented accordingly. If the assembly is not valid, then memory is not modified, and an error message produced. In either case, memory is disassembled and the process repeats.

The user may press the <Enter> or <Return> key to accept the current memory contents and skip to the next instruction, or a enter period to quit the interactive mode.

In the non-interactive mode, the user specifies the address and the assembly statement on the command line. The statement is the assembled, and if valid, placed into memory, otherwise an error message is produced.

Examples:

To place a NOP instruction at address 0x0001_0000, the command is:

```
asm 10000 nop
```

To interactively assembly memory at address 0x0040_0000, the command is:

```
asm 400000
```

BC - Block Compare

Usage: BC addr1 addr2 length

The BC command compares two contiguous blocks of memory on a byte by byte basis. The first block starts at address **addr1** and the second starts at address **addr2**, both of **length** bytes.

If the blocks are not identical, the address of the first mismatch is displayed. The value for addresses addr1 and addr2 may be an absolute address specified as a hexadecimal value or a symbol name. The value for length may be a symbol name or a number converted according to the user defined radix (hexadecimal by default).

Example:

To verify that the data starting at 0x20000 and ending at 0x3_0000 is identical to the data starting at 0x8_0000, the command is:

bc 20000 80000 10000

BF - Block Fill

Usage: BF<width> begin end data <inc>

The BF command fills a contiguous block of memory starting at address **begin**, stopping at address **end**, with the value **data**. <**Width**> modifies the size of the data that is written. If no <width> is specified, the default of word sized data is used.

The value for addresses begin and end may be an absolute address specified as a hexadecimal value, or a symbol name. The value for data may be a symbol name, or a number converted according to the user-defined radix, normally hexadecimal.

The optional value <**inc**> can be used to increment (or decrement) the data value during the fill.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To fill a memory block starting at 0x2_0000 and ending at 0x4_0000 with the value 0x1234, the command is:

bf 20000 40000 1234

To fill a block of memory starting at 0x20000 and ending at 0x4_0000 with a byte value of 0xAB, the command is:

bf.b 20000 40000 AB

To zero out the BSS section of the target code (defined by the symbols bss_start and bss_end), the command is:

bf bss_start bss_end 0

To fill a block of memory starting at 0x2_0000 and ending at 0x4_0000 with data that increments by 2 for each <width>, the command is:

bf 20000 40000 0 2

BM - Block Move

Usage: BM begin end dest

The BM command moves a block of memory starting at address **begin** and stopping at address **end** to the new address **dest**. The BM command copies memory as a series of bytes, and does not alter the original block.

The values for addresses begin, end, and dest may be absolute addresses specified as hexadecimal values, or symbol names. If the destination address overlaps the block defined by begin and end, an error message is produced and the command exits.

Examples:

To copy a block of memory starting at 0x4_0000 and ending at 0x7_0000 to the location 0x200000, the command is:

bm 40000 70000 200000

To copy the target code's data section (defined by the symbols data_start and data_end) to 0x200000, the command is:

bm data_start data_end 200000

NOTE: Refer to "upuser" command for copying code/data into Flash memory.

BR - Breakpoints

Usage: BR addr <-r> <-c count> <-t trigger>

The BR command inserts or removes breakpoints at address **addr**. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name. Count and trigger are numbers converted according to the user-defined radix, normally hexadecimal.

If no argument is provided to the BR command, a listing of all defined breakpoints is displayed.

The **-r** option to the BR command removes a breakpoint defined at address addr. If no address is specified in conjunction with the -r option, then all breakpoints are removed.

Each time a breakpoint is encountered during the execution of target code, its count value is incremented by one. By default, the initial count value for a breakpoint is zero, but the **-c** option allows setting the initial count for the breakpoint.

Each time a breakpoint is encountered during the execution of target code, the count value is compared against the trigger value. If the count value is equal to or greater than the trigger value, a breakpoint is encountered and control returned to **dBUG**. By default, the initial trigger value for a breakpoint is one, but the **-t** option allows setting the initial trigger for the breakpoint.

If no address is specified in conjunction with the -c or -t options, then all breakpoints are initialized to the values specified by the -c or -t option.

Examples:

To set a breakpoint at the C function main() (symbol _main; see “symbol” command), the command is:

```
br _main
```

When the target code is executed and the processor reaches main(), control will be returned to **dBUG**.

To set a breakpoint at the C function bench() and set its trigger value to 3, the command is:

```
br _bench -t 3
```

When the target code is executed, the processor must attempt to execute the function bench() a third time before returning control back to **dBUG**.

To remove all breakpoints, the command is:

```
br -r
```

BS - Block Search

Usage: BS<width> begin end data

The BS command searches a contiguous block of memory starting at address **begin**, stopping at address **end**, for the value **data**. <**Width**> modifies the size of the data that is compared during the search. If no <width> is specified, the default of word sized data is used.

The values for addresses begin and end may be absolute addresses specified as hexadecimal values, or symbol names. The value for data may be a symbol name or a number converted according to the user-defined radix, normally hexadecimal.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To search for the 32-bit value 0x1234_5678 in the memory block starting at 0x4_0000 and ending at 0x7_0000:

```
bs.l 40000 70000 12345678
```

This reads the 32-bit word located at 0x0004_0000 and compares it against the 32-bit value 0x1234_5678. If no match is found, then the address is incremented to 0x0004_0004 and the next 32-bit value is read and compared.

To search for the 16-bit value 0x1234 in the memory block starting at 0x0004_0000 and ending at 0x0007_0000:

```
bs 40000 70000 1234
```

This reads the 16-bit word located at 0x4_0000 and compares it against the 16-bit value 0x0000_1234. If no match is found, then the address is incremented to 0x0004_0002 and the next 16-bit value is read and compared.

DC - Data Conversion

Usage: DC data

The DC command displays the hexadecimal or decimal value **data** in hexadecimal, binary, and decimal notation.

The value for data may be a symbol name or an absolute value. If an absolute value passed into the DC command is prefixed by ‘0x’, then data is interpreted as a hexadecimal value. Otherwise data is interpreted as a decimal value.

All values are treated as 32-bit quantities.

Examples:

To display the decimal and binary equivalent of 0x1234, the command is:

```
dc 0x1234
```

To display the hexadecimal and binary equivalent of 1234, the command is:

```
dc 1234
```

DI - Disassemble

Usage: DI <addr>

The DI command disassembles target code pointed to by **addr**. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name.

Wherever possible, the disassembler will use information from the symbol table to produce a more meaningful disassembly. This is especially useful for branch target addresses and subroutine calls.

The DI command attempts to track the address of the last disassembled opcode. If no address is provided to the DI command, then the DI command uses the address of the last opcode that was disassembled.

The DI command is repeatable.

Examples:

To disassemble code that starts at 0x0004_0000, the command is:

```
di      40000
```

To disassemble code of the C function main(), the command is:

```
di      _main
```

DL - Download Console

Usage: DL <offset>

The DL command performs an S-record download of data obtained from the console or serial port. The value for offset is converted according to the user-defined radix, normally hexadecimal.

If **offset** is provided, then the destination address of each S-record is adjusted by offset.

The DL command checks the destination download address for validity. If the destination is an address outside the defined user space, then an error message is displayed and downloading aborted.

If the destination address is in the user flash memory space, the flash will be programmed but not erased. See the FL command for flash erasing.

If the S-record file contains the entry point address, then the program counter is set to reflect this address.

Examples:

To download an S-record file through the serial port, the command is:

```
dl
```

To download an S-record file through the serial port, and add an offset to the destination address of 0x40000, the command is:

```
dl      0x40000
```

After the DL command is invoked, the user should select file transfer or upload and send the S-record file form the host. The host serial terminal software should apply XON/XOFF flow control to the transfer if the target memory is Flash space to allow flash programming time delays. Alternate method is to download with an offset into SDRAM memory space and then apply the FL command to program the flash memory space.

DLDBUG – Download dBUG (update)

Usage: dldbug

The dldbug command is used to update the dBUG image in Flash memory. When updates to the MCF5485 dBUG are available, the update S-record may be downloaded into the flash from the console or serial port similar to the DL command. The user is prompted for verification before performing the operation (note case sensitivity here). XON/XOFF serial flow control must be applied when loading the new S-record. Use this command with extreme caution, as any error can render dBUG useless!

DN - Download Network

Usage: DN <-c> <-e> <-i> <-s> <-o offset> <filename>

The DN command downloads code from the network. The DN command will handle files that are S-record, COFF, ELF or Image formats. The DN command uses Trivial File Transfer Protocol (TFTP) to transfer files from a network host.

In general, the type of file to be downloaded and the name of the file must be specified to the DN command. The **-c** option indicates a COFF download, the **-e** option indicates an ELF download, the **-i** option indicates an Image download, and the **-s** indicates an S-record download. The **–o** option works only in conjunction with the -s option to indicate an optional offset for S-record download. The filename is passed directly to the TFTP server and therefore must be a valid filename on the server.

If neither of the -c, -e, -i, -s or filename options are specified, then a default filename and file type will be used. Default filename and file type parameters are manipulated using the SET and SHOW commands.

The DN command checks the destination download address for validity. If the destination is an address outside the defined user space, then an error message is displayed and downloading aborted.

For ELF and COFF files that contain symbolic debug information, the symbol tables are extracted from the file during download and used by **dBUG**. Only global symbols are kept in **dBUG**. The **dBUG** symbol table is not cleared prior to downloading, so it is the user's responsibility to clear the symbol table as necessary prior to downloading.

If an entry point address is specified in the S-record, COFF or ELF file, the program counter is set accordingly.

Examples:

To download an S-record file with the name "srec.out", the command is:

```
dn    -s    srec.out
```

To download a COFF file with the name “coff.out”, the command is:

```
dn    -c    coff.out
```

To download a file using the default file type with the name “bench.out”, the command is:

```
dn    bench.out
```

To download a file using the default filename and file type, the command is:

```
dn
```

FL – Flash Load or Erase

Usage: FL <command> dest <src> <size>

The FL command is used to erase and write both the MCF5485 internal flash and the external flash memory, and display flash device sector information. Erase or write operations must be performed in even sector block sizes. The write command will erase all of the associated flash sectors prior to writing. If the write destination address or byte count range does not provide an even sector boundary, dBUG will prompt the user to continue.

The MCF5485 internal flash destination address (F0000000 – F007FFFC) must be long word (4 byte) aligned and the byte count must be in long word (4 byte) multiples.

External flash destination address (FFF40000 – FFFFFFFE) must be word aligned (2 byte) and the byte count must be in word (2 byte) or even number multiples.

To download S-record files directly into the flash, please see the DL command.

Examples:

To view the flash device sector information, the command is:

```
fl
```

To erase 0x10000 (64K) bytes of internal flash starting at 0xF0000000, the command is:

```
fl erase F0000000 10000
```

To copy 0x4000 (16K) bytes of data from internal SRAM (0x20000000) to external flash at 0xFFE40000, the command is:

```
fl write FFE40000 20000000 4000
```

Note that the above command will cause dBUG to prompt the user to continue due to the addressed sector size is larger than 0x4000 bytes in this case. User should type in “YES” to continue or any other key to stop the operation.

Sector range values:

1000 = 4K bytes, 2000 = 8K bytes, 8000 = 32K bytes, 10000 = 64 K bytes.

To erase the complete internal flash, the command is:

fl erase F0000000 80000

To erase all user sectors of the external flash, the command is:

fl erase FFE40000 1C0000

GO – Execute user code

Usage: GO <addr>

The GO command executes target code starting at address **addr**. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name.

If no argument is provided, the GO command begins executing instructions at the current program counter.

When the GO command is executed, all user-defined breakpoints are inserted into the target code, and the context is switched to the target program. Control is only regained when the target code encounters a breakpoint, illegal instruction, or other exception that causes control to be handed back to **dBUG**.

The GO command is repeatable.

Examples:

To execute code at the current program counter, the command is:

```
go
```

To execute code at the C function main(), the command is:

```
go _main
```

To execute code at the address 0x00040000, the command is:

```
go 40000
```

GT - Execute To Address

Usage: GT addr

The GT command inserts a temporary software breakpoint at **add**r and then executes target code starting at the current program counter. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name.

When the GT command is executed, all breakpoints are inserted into the target code, and the context is switched to the target program. Control is only regained when the target code

encounters a breakpoint, illegal instruction, or an exception which causes control to be handed back to **dBUG**.

Examples:

To execute code up to the C function bench(), the command is:

gt _bench

To execute code up to the address 0x00080004, the command is:

gt 80004

IRD - Internal Register Display

Usage: IRD <module.register>

This command displays the internal registers of the different modules inside the MCF5485. In the command line, module refers to the module name where the register is located and register refers to the specific register to display.

The registers are organized according to the module to which they belong. The available modules on the MCF5485 are:

SCM, CS0, CS1, CS2, CS3, CS4, CS5, CS6, GPIO, QSPI, DMA0, DMA1, DMA2, DMA3, UART0, UART1, UART2, SDRAMC, TIMER0, TIMER1, TIMER2, TIMER3, FEC, CAN, I2C, WDT, PIT0, PIT1, PIT2, PIT3, QADC, GPTA, GPTB, RESET, CCM, PMM, CLOCK, EPORT, CFM, INTC0, and INTC1.

Refer to the MCF5485 user's manual for more information on these modules and the registers they contain.

Example:

ird cs0.csar

IRM - Internal Register Modify

Usage: IRM module.register data

This command modifies the contents of the internal registers of different modules inside the MCF5485. In the command line, module refers to the module name where the register is located and register refers to the specific register to modify. The data parameter specifies the new value to be written into the register.

The registers are organized according to the module to which they belong. The available modules on the MCF5485 are:

SCM, CS0, CS1, CS2, CS3, CS4, CS5, CS6, GPIO, QSPI, DMA0, DMA1, DMA2, DMA3, UART0, UART1, UART2, SDRAMC, TIMER0, TIMER1, TIMER2, TIMER3, FEC, CAN, I2C, WDT, PIT0, PIT1, PIT2, PIT3, QADC, GPTA, GPTB, RESET, CCM, PMM, CLOCK, EPORT, CFM, INTC0, and INTC1.

Refer to the MCF5485 user's manual for more information on these modules and the registers they contain.

Example:

To modify the CSAR register in the CS1 chip select module to the value 0x4000, the command is:

```
irm cs1.csar 4000
```

HELP - Help

Usage: HELP <command>

The HELP command displays a brief syntax of the commands available within **dBUG**. In addition, the address of where user code may start is given. If command is provided, then a brief listing of the syntax of the specified command is displayed.

Examples:

To obtain a listing of all the commands available within **dBUG**, the command is:

```
help
```

To obtain help on the breakpoint command, the command is:

```
help br
```

LR - Loop Read

Usage: LR <width> addr

The LR command continually reads the data at **addr** until a key is pressed. The optional **<width>** specifies the size of the data to be read. If no <width> is specified, the command defaults to reading word sized data.

Example:

To continually read the word data from address 0xFFF2_0000, the command is:

```
lr FFF20000
```

LW - Loop Write

Usage: LW <width> addr data

The LW command continually writes data to **addr**. The optional width specifies the size of the access to memory. The default access size is a word.

Examples:

To continually write the data 0x1234_5678 to address 0x0002_0000, the command is:

```
lw.l 20000 12345678
```

Note that the following command writes 0x78 into memory:

```
lw.b 20000 78
```

MD - Memory Display

Usage: MD <width> <begin> <end>

The MD command displays a contiguous block of memory starting at address begin and stopping at address end. The values for addresses begin and end may be absolute addresses specified as hexadecimal values, or symbol names. Width modifies the size of the data that is displayed. If no **<width>** is specified, the default of word sized data is used.

Memory display starts at the address begin. If no beginning address is provided, the MD command uses the last address that was displayed. If no ending address is provided, then MD will display memory up to an address that is 128 beyond the starting address.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To display memory at address 0x0040_0000, the command is:

```
md 400000
```

To display memory in the data section (defined by the symbols data_start and data_end), the command is:

```
md data_start
```

To display a range of bytes from 0x00040000 to 0x0005_0000, the command is:

```
md.b 40000 50000
```

To display a range of 32-bit values starting at 0x0004_0000 and ending at 0x0005_0000:

```
md 40000 50000
```

MM - Memory Modify

Usage: MM<width> addr <data>

The MM command modifies memory at the address **addr**. The value for addr may be an absolute address specified as a hexadecimal value, or a symbol name. Width specifies the size of the data that is modified. If no **<width**> is specified, the default of word sized data is used. The value for **data** may be a symbol name, or a number converted according to the user-defined radix, normally hexadecimal.

If a value for data is provided, then the MM command immediately sets the contents of addr to data.

If no value for data is provided, then the MM command enters into a loop. The loop obtains a value for data, sets the contents of the current address to data, increments the address according to the data size, and repeats. The loop terminates when an invalid entry for the data value is entered, for instance a period '.'.

This command first aligns the starting address for the data access size, and then increments the address accordingly during the operation. Thus, for the duration of the operation, this command performs properly-aligned memory accesses.

Examples:

To set the byte at location 0x0001_0000 to be 0xFF, the command is:

```
mm.b 10000 FF
```

To interactively modify memory beginning at 0x0001_0000, the command is:

```
mm 10000
```

MMAP - Memory Map Display

Usage: mmap

This command displays the memory map information for the evaluation board. The information displayed includes the type of memory, the start and end address of the memory, and the port size of the memory. The display also includes information on how the Chip-selects are used on the board.

Here is an example of the output from this command:

Type	Start	End	Port size
SDRAM	0x00000000	0x003FFFFF	32-bit
SRAM (int)	0x00000000	0x003FFFFF	32-bit
ISPBAR	0x40000000	0x7FFFFFFF	32-bit
FLASH (int)	0xF0000000	0xF007FFFF	32-bit
FLASH (ext)	0xFFE00000	0xFFFFFFFF	16-bit

```
Protected       Start          End
-------------------------------------------------------------
dBUG code       0xFFE00000     0xFFE3FFFF
dBUG data       0x00000400     0x0000FFFF

Chip Selects
-----------------
CS0  Flash
```

RD - Register Display

Usage: RD <reg>

The RD command displays the register set of the target. If no argument for **reg** is provided, then all registers are displayed. Otherwise, the value for reg is displayed.

dBUG preserves the registers by storing a copy of the register set in a buffer. The RD command displays register values from the register buffer.

Examples:

To display only the program counter:

```
rd pc
```

To display all the registers and their values, the command is:

```
rd
```

Here is an example of the output from this command:

PC: 00000000 SR: 2000 [t.Sm.000...xnzvc]

An: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 01000000

Dn: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

RM - Register Modify

Usage: RM reg data

The RM command modifies the contents of the register **reg** to data. The value for reg is the name of the register, and the value for data may be a symbol name, or it is converted according to the user-defined radix, normally hexadecimal.

dBUG preserves the registers by storing a copy of the register set in a buffer. The RM command updates the copy of the register in the buffer. The actual value will not be written to the register until target code is executed.

Example:

To change program counter to contain the value 0x2000_8000, the command is:

```
rm pc 20008000
```

RESET - Reset the Board and dBUG

Usage: RESET

The RESET command resets the board and **dBUG** to their initial power-on states.

The RESET command executes the same sequence of code that occurs at power-on. If the RESET command fails to reset the board properly, cycle the power or press the RESET button.

Examples:

To reset the board and clear the **dBUG** data structures, the command is:

```
reset
```

SET - Set Configurations

Usage: SET <option value>

The SET command allows the setting of user-configurable options within **dBUG**. With no arguments, SET displays the options and values available. The SHOW command displays the settings in the appropriate format. Note that some configuration items will not take effect until a Reset has occurred. The standard set of options is listed below.

baud - This is the baud rate for the first serial port on the board. All communications between **dBUG** and the user occur using 19200 bps, eight data bits, no parity, and one stop bit, 8N1, with no flow control.

base - This is the default radix for use in converting a number from its ASCII text representation to the internal quantity used by **dBUG**. The default is hexadecimal (base 16), and other choices are binary (base 2), octal (base 8), and decimal (base 10).

client - This is the network Internet Protocol (IP) address of the board. For network communications, the client IP is required to be set to a unique value, usually assigned by your local network administrator.

server - This is the network IP address of the machine which contains files accessible via TFTP. Your local network administrator will have this information and can assist in properly configuring a TFTP server if one does not exist.

gateway - This is the network IP address of the gateway for your local subnetwork. If the client IP address and server IP address are not on the same subnetwork, then this option must be properly set. Your local network administrator will have this information.

netmask - This is the network address mask to determine if use of a gateway is required. This field must be properly set. Your local network administrator will have this information.

filename - This is the default filename to be used for network download if no name is provided to the DN command.

filetype - This is the default file type to be used for network download if no type is provided to the DN command. Valid values are: “srecord”, “coff”, and “elf”.

mac - This is the Ethernet Media Access Control (MAC) address (a.k.a hardware address) for the evaluation board. This should be set to a unique value, and the most significant nibble should always be even.

Examples:

To set the baud rate of the board to be 38400, the command is:

set baud 38400

NOTE: See the SHOW command for a display containing the correct formatting of these options. See the dBUG Ethernet support for additional details on network settings.

SHOW - Show Configurations

Usage: SHOW <option>

The SHOW command displays the settings of the user-configurable options within **dBUG**. When no option is provided, SHOW displays all options and values.

Examples:

To display the current baud rate of the board, the command is:

show baud

To display all options and settings, the command is:

show

Here is an example of the output from a show command:

```
dBUG> show
base: 16
baud: 19200
server: 192.0.0.1
client: 192.0.0.2
gateway: 0.0.0.0
netmask: 255.255.255.0
filename: test.srec
filetype: S-Record
ethaddr: 00:CF:52:49:C3:01
```

STEP - Step Over

Usage: STEP

The STEP command can be used to “step over” a subroutine call, rather than tracing every instruction in the subroutine. The ST command sets a temporary software breakpoint one instruction beyond the current program counter and then executes the target code.

The STEP command can be used to “step over” BSR and JSR instructions.

The STEP command will work for other instructions as well, but note that if the STEP command is used with an instruction that will not return, i.e. BRA, then the temporary breakpoint may never be encountered and **dBUG** may never regain control.

Examples:

To pass over a subroutine call, the command is:

```
step
```

SYMBOL - Symbol Name Management

Usage: SYMBOL <symb> <-a symb value> <-r symb> <-c|l|s>

The SYMBOL command adds or removes **symbol** names from the symbol table. If only a symbol name is provided to the SYMBOL command, then the symbol table is searched for a match on the symbol name and its information displayed.

-a option adds a symbol name and its value into the symbol table.

-r option removes a symbol name from the table.

-c option clears the entire symbol table.

-l option lists the contents of the symbol table.

-s option displays usage information for the symbol table.

Symbol names contained in the symbol table are truncated to 31 characters. Any symbol table lookups, either by the SYMBOL command or by the disassembler, will only use the first 31 characters. Symbol names are case-sensitive.

Symbols can also be added to the symbol table via in-line assembly labels and Ethernet downloads of ELF formatted files.

Examples:

To define the symbol “main” to have the value 0x0004_0000, the command is:

```
symbol -a main 40000
```

To remove the symbol “junk” from the table, the command is:

symbol -r junk

To see how full the symbol table is, the command is:

symbol -s

To display the symbol table, the command is:

symbol –l

TRACE - Trace Into

Usage: TRACE <num>

The TRACE command allows single-instruction execution. If **num** is provided, then num instructions are executed before control is handed back to **dBUG**. The value for num is a decimal number.

The TRACE command sets bits in the processors' supervisor registers to achieve single-instruction execution, and the target code executed. Control returns to **dBUG** after a single-instruction execution of the target code.

This command is repeatable.

Examples:

To trace one instruction at the program counter, the command is:

tr

To trace 20 instructions from the program counter, the command is:

tr 20

UP – Upload network

Usage: UP begin end filename

The UP command transfers binary data stored in memory on the development board to the host via the network connection. The transfer applies the same network settings as the DN command but provides the new file name on the host.

Examples:

To send memory data from 0x00020000 to 0x0002FFFF as file name test.bin to the network host, the command is:

up 20000 2FFFF test.bin

VERSION - Display dBUG Version

Usage: VERSION

The VERSION command displays the version information for **dBUG**. The **dBUG** version, build number and build date are all given.

The version number is separated by a decimal, for example, “v 2b.1c.1a”.

The version date is the day and time at which the entire **dBUG** monitor was compiled and built.

Examples:

To display the version of the **dBUG** monitor, the command is:

ver

TRAP #15 Functions

An additional utility within the dBUG firmware is a function called the TRAP 15 handler. This function can be called by the user program to utilize various routines within the dBUG, perform a special task, and to return control to the dBUG. This section describes the TRAP 15 handler and how it is used.

There are four TRAP #15 functions. These are: **OUT_CHAR**, **IN_CHAR**, **CHAR_PRESENT**, and **EXIT_TO_dBUG**.

OUT_CHAR

This function (function code 0x0013) sends a character, which is in lower 8 bits of D1, to terminal. Assembly example:

/* assume d1 contains the character */

```
move.l    #$0013,d0    Selects the function
TRAP      #15          The character in d1 is sent to terminal
```

C example:

```
void board_out_char (int ch)
{
 /* If your C compiler produces a LINK/UNLK pair for this routine,  then use the following code which takes this into account */
#if    l
/* LINK a6,#0          -- produced by C compiler */
asm (“ move.l 8(a6),d1”);          /* put ‘ch’into d1 */
```

```
asm (“ move.l #0x0013,d0”);        /* select the function */

asm (“ trap #15”);                 /* make the call */

/* UNLK a6              -- produced by C compiler */

#else

/*  If C compiler does not produce a LINK/UNLK pair, the use the following code */

asm (“ move.l 4(sp),d1”);          /* put ‘ch’into d1 */

asm (“ move.l #0x0013,d0”);        /* select the function */

asm (“ trap #15”);                 /* make the call */

#endif

}
```

IN_CHAR

This function (function code 0x0010) returns an input character (from terminal) to the caller. The returned character is in D1.

Assembly example:

```
        move.l      #$0010,d0   Select the function

        trap        #15         Make the call, the input character is in d1.
```

C example:

```
int board_in_char (void)

{

        asm (“ move.l #0x0010,d0”);        /* select the function */

        asm (“ trap #15”);                 /* make the call */

        asm (“ move.l d1,d0”);             /* put the character in d0 */

}
```

CHAR_PRESENT

This function (function code 0x0014) checks if an input character is present to receive. A value of zero is returned in D0 when no character is present. A non-zero value in D0 means a character is present.

Assembly example:

```
        move.l      #$0014,d0       Select the function

        trap        #15             Make the call,  d0 contains the response (yes/no).
```

C example:

```
int board_char_present (void)

{

        asm (" move.l #0x0014,d0");     /* select the function */

        asm (" trap #15");              /* make the call */

}
```

EXIT_TO_dBUG

This function (function code 0x0000) transfers the control back to the dBUG, by terminating the user code. The register contents are preserved.

Assembly example:

```
        move.l      #$0000,d0       Select the function

        trap        #15             Make the call,  exit to dBUG.
```

C example:

```
void board_exit_to_dbug (void)

{

        asm (" move.l #0x0000,d0");     /* select the function */

        asm (" trap #15");              /* exit and transfer to dBUG */

}
```