ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Motorola CMBA002* at our website: ***Click HERE***

MCOREEVSUM/D

November 1997

M•CORE™ EVALUATION SYSTEM

USER'S MANUAL

© Motorola Inc., 1997; All Rights Reserved

Motorola reserves the right to make changes without further notice to any products herein to improve reliability, function or design. Motorola 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 its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

Motorola,  , and M•CORE are registered trademarks of Motorola Inc.

Motorola Inc. is an Equal Opportunity/Affirmative Action Employer.

Contents

Chapter 1. Introducing the M•CORE™ Evaluation System 1-1
1.1 The M•CORE™ Evaluation System. 1-2
1.1.1 Microcontroller Memory Board 1-2
1.1.2 Input/Output Peripheral Board 1-2
1.1.3 Test Interface Board 1-3
1.2 M•CORE Evaluation System Features 1-3
1.3 M•CORE Block Diagram 1-3
1.4 M•CORE Specifications 1-4
1.5 References. 1-5

Chapter 2. Quick Start Guide 2-1
2.1 Configuring the CMB 2-2
2.1.1 RS-232 Connection from PC to the CMB 2-2
2.1.2 Power Supply 2-2
2.1.3 CMB Jumper Header Settings 2-2
2.2 Configuring the IPB 2-3
2.3 Using the TIB 2-3
2.4 Software Installation. 2-4
2.4.1 Installing the SDS Single Step Monitor/Debugger. 2-4
2.4.2 Starting the SDS Single Step Monitor/Debugger 2-4
2.4.3 Installing the Diab Data C Compiler 2-5
2.4.4 Installing the Gnu C Compiler 2-6
2.5 Running the Sample Programs 2-7
2.5.1 EVS Debug Mode. 2-7
2.5.2 Flash Programming Mode 2-9
2.5.3 User Mode. 2-10
2.6 Accessing Peripheral Device Registers 2-11
2.7 Description of Software Programs 2-13

Chapter 3. The Microcontroller Memory Board 3-1
3.1 CMB Options 3-2
3.1.1 Mode Select Jumper (J22) 3-3

3.1.2 MDTACK/MDS Jumper (J21-13 & J21-15) 3-3
3.1.3 TRST Select Jumper (J99) 3-4
3.2 CMB Connectors 3-5
3.2.1 Serial 1 (J24) and Serial 2 (J25) Connectors 3-5
3.2.2 Power Connector (TB1) 3-5
3.2.3 Data Bus Connectors (J10, J11, J13, J14). 3-5
3.2.4 Address Bus Connectors (J4, J5, J7, J8) 3-7

Chapter 4. The Input/Output Peripheral Board 4-1
4.1 Introducing the Input/Output Peripheral Board 4-2
4.1.1 IPB Block Diagram. 4-2
4.1.2 Peripheral Device Slave Units 4-2
4.1.3 General Purpose Inputs and Outputs 4-5
4.1.4 Bus and Control Signals 4-6
4.2 IPB Options. 4-6
4.2.1 QADC Voltage Reference Source (W14, W15). 4-6
4.2.2 General Purpose Input/Output (W1 through W8). 4-7
4.2.3 W10 Jumper 4-8
4.2.4 W13 Jumper 4-8

Chapter 5. Memory Maps and Pinouts 5-1
5.1 IPB Memory Maps 5-2
5.1.1 EVS Base Addresses. 5-2
5.1.2 Peripheral Memory Map. 5-3
5.1.3 Interrupt Controller Memory Map 5-4
5.1.4 GPIO Memory Map 5-5
5.2 Test Interface Board Pinouts. 5-5

Chapter 6. Interrupt Control and PORTF. 6-1
6.1 Interrupt Controller Registers 6-2
6.1.1 Interrupt Source Register 1 (ISR1) 6-2
6.1.2 Interrupt Source Register 2 (ISR2) 6-3
6.1.3 Interrupt Enable Register 1 (IER1). 6-3
6.1.4 Interrupt Enable Register 2 (IER2). 6-4
6.1.5 Fast Interrupt Enable Register (FIER) 6-4
6.2 Controlling Interrupts 6-5
6.3 Interrupt Vector Map 6-5
6.4 Interrupt Controller Port F 6-7
6.4.1 Port F Edge Control Register (PFECR) 6-7
6.4.2 Port F Interrupt Enable Register (PFIER). 6-8
6.4.3 Port F Data Direction Register (DDRF) 6-9
6.4.4 Port F Output Data Register (PORTF) 6-9
6.4.5 Port F Pin Data Register (PORTFP). 6-10
6.4.6 Port F Edge-Detect Register (PORTFE). 6-10

Chapter 1. Introducing the M•CORE™ Evaluation System

In This Chapter

We introduce the M•CORE Evaluation System.

The M•CORE™ Evaluation System 1-2
Microcontroller Memory Board 1-2
Input/Output Peripheral Board 1-2
Test Interface Board 1-3
M•CORE Evaluation System Features 1-3
M•CORE Block Diagram 1-3
M•CORE Specifications 1-4
References 1-5

1.1 The M•CORE™ Evaluation System

The M•CORE Evaluation System allows you to emulate M•CORE, a new family of 32-bit RISC microcontrollers that are optimized for 16-bit external systems. The M•CORE Evaluation System consists of three boards:

- Microcontroller Memory Board (CMB)
- Input/Output Peripheral Board (IPB)
- Test Interface Board (TIB)

All three boards are connected through a ring of connectors which form the modular active probe interconnect (MAPI).

1.1.1 Microcontroller Memory Board

The CMB contains the following components:

- M•CORE microcontroller
- 512 Kbytes of Flash EEPROM
- 512 Kbytes of Fast Static RAM
- EVS Debug Mode
- RS-232 interface
- Reset and interrupt steering logic
- MAPI ring on the bottom side to connect to the IPB

1.1.2 Input/Output Peripheral Board

The IPB emulates the peripheral modules for the M•CORE microcontroller. It contains the following components:

- Two Motorola peripheral devices operating in slave mode:
- One field-programmable gate array (FPGA) programmed to re-map interrupts and provide edge-detect inputs and on-board chip selects
- Eight 74FCT652s providing 64 bits of general purpose inputs or general purpose outputs
- Two PALs for bus matching and chip select steering
- Top-side MAPI connectors for interface with the CMB
- Bottom-side MAPI connectors for interface with the TIB

For more information about the IPB, see Chapter 4, “The Input/Output Peripheral Board.”

1.1.3 Test Interface Board

The TIB provides external access to some of the signals on the MAPI ring, typically for connecting a logic analyzer to the M•CORE Evaluation System. This board is not required for operation of the other two boards.

1.2 M•CORE Evaluation System Features

To assist in the development of an M•CORE controller, the M•CORE Evaluation System includes the following features:

- 512 Kbytes of Flash EEPROM
- 512 Kbytes of Fast Static RAM
- Manual reset switch
- 5V for Flash programming on the CMB
- Direct 5V input power on the CMB
- 5V to 3.3V converter for M•CORE
- 64 bits of general purpose input or output
- Eight edge-detect inputs
- Two slave peripheral devices, each providing:
  - 10-bit queued analog-to-digital converter (QADC)
    - Up to 16 channels of analog input each
    - Four channels programmable as external input triggers
  - Two queued serial modules (QSM)
    - Two queued serial peripheral interfaces (QSPI)
    - Two serial communication interfaces (SCI)
  - Two controller access network interfaces(TouCAN™)
  - Two configurable timer modules, version 4 (CTM4)
    - Four 16-bit modulus counter submodule (MCSM)
    - Two 16-bit free-running counter submodules (FASM)
    - Eight double-action submodules (DASM)
    - Eight pulse width modulation submodules (PWMSM)

1.3 M•CORE Block Diagram

Figure 1-1 illustrates the block diagram of the M•CORE Evaluation System.

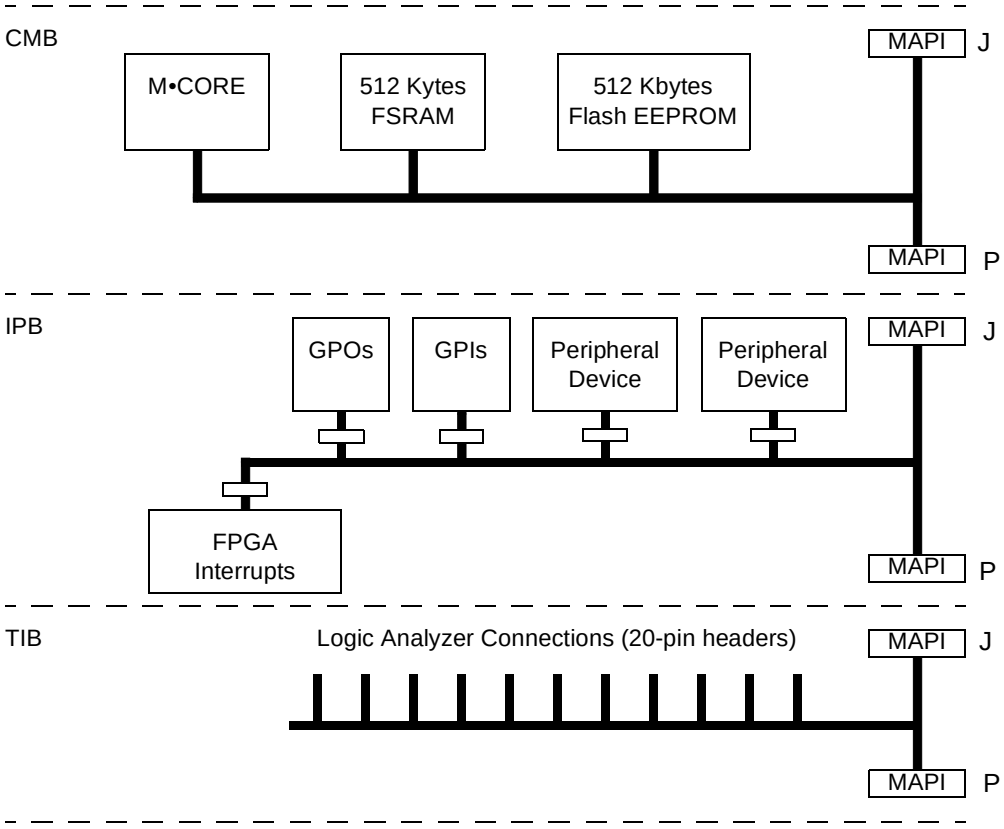


Figure 1-1. M•CORE Evaluation System Block Diagram

1.4 M•CORE Specifications

Table 1-1 lists the M•CORE Evaluation System specifications.

Table 1-1. M•CORE Evaluation System Specifications

Characteristic	Specification
System clock	20 MHz
External clock	20 MHz
MCU I/O ports	HCMOS compatible
Operating temperature	0°C to +40°C
Storage temperature	0°C to +40°C
Relative humidity	0% to 90% (non-condensing)
Power requirements: VDD	 5V @ 2 amp

1.5 References

The following documents provide additional information relevant to developing M•CORE controllers.

- *QSM Reference Manual* (QSMRM/AD) (on CD-ROM)
- *QADC Reference Manual* (QADCRM/AD) (on CD-ROM)
- *CTM4 Configurable Timer Module section* excerpted from MC68336/376UM/AD (on CD-ROM)
- *TouCAN CAN 2.0 Controller Module section* excerpted from MC68336/376UM/AD (on CD-ROM)

The CD-ROM that comes with the M•CORE Evaluation System also contains PDF files of relevant portions of other Motorola manuals concerning the peripheral devices used on the IPB.

Check our web sites at **www.mcu.motsps.com/lit/fam_3xx.htm** and **www.motorola.com/mcore** for addtional information.

Chapter 2. Quick Start Guide

In This Chapter

We lead you through the initial stages of setup and installation for the M•CORE Evaluation System.

Configuring the CMB 2-2
RS-232 Connection from PC to the CMB 2-2
Power Supply 2-2
CMB Jumper Header Settings 2-2
Configuring the IPB 2-3
Using the TIB 2-3
Software Installation 2-4
Installing the SDS Single Step Monitor/Debugger 2-4
Starting the SDS Single Step Monitor/Debugger 2-4
Installing the Diab Data C Compiler 2-5
Installing the Gnu C Compiler 2-6
Running the Sample Programs 2-7
EVS Debug Mode 2-7
Flash Programming Mode 2-9
User Mode 2-10
Accessing Peripheral Device Registers 2-11
Description of Software Programs 2-13

2.1 Configuring the CMB

This section describes the basic set-up and installation of the M•CORE CMB.

The CMB can operate in one of three modes:

EVS Debug	Downloads an application to the FSRAM and operates the EVS using that application.
Flash Programming	Writes to the on-board EEPROM
User	Copies the code in the EEPROM to the FSRAM and operates the EVS using that code.

The settings described below place the CMB in EVS Debug mode. This mode allows you to activate the SDS Monitor/Debugger software and download an application to the EVS from a PC through the serial port.

Figure 3-1 on page 3-2 illustrates the CMB to show the locations referred to in this section.

2.1.1 RS-232 Connection from PC to the CMB

Attach the ten-pin plug of the RS-232 miniature jack cable to socket J25 (Serial 2) on the CMB and the other end to a 9-pin serial port on the PC.

2.1.2 Power Supply

Connect power from a 5V power supply to TB1 using the following procedure:

1. Use 20 or 22 AWG wire for power connections.
2. Trim 1/4 inch (.635 cm) of insulation from each wire.
3. Lift the levers of TB1 to release tension on the contacts.
4. Insert the trimmed portion of each wire into the appropriate contact of TB1 and close the lever. The red lever (TB1-1) is VDD (+5 volts). The black lever (TB1-2) is ground.

2.1.3 CMB Jumper Header Settings

Place a jumper on pins 1 and 2 of jumper J22.

Place a jumper on pins 2 and 3 of the J99 header.

Note: If you are using the CMB without the IPB, then install a jumper across pins 15 (MDTACK) and 13 (MDS) of the J21 header.

2.2 Configuring the IPB

The IPB board attaches to the CMB board through the MAPI ring connectors on the bottom of the CMB. Power is supplied to the IPB through the MAPI ring. The IPB comes from the factory already set up to a default configuration. For more information about IPB configuration options, see Chapter 4, “The Input/Output Peripheral Board.”

2.3 Using the TIB

If you need access to the ports and interrupt I/O of the IPB, you can attach the Test Interface board (TIBA0002) to the bottom of the IPB at the MAPI ring. For information about connections to the Test Interface board, see *Test Interface Board Pinouts* on page 5-5. The Quick Start instructions do not use the Test Interface board.

2.4 Software Installation

These software installation steps supplement the instructions that come with the software.

2.4.1 Installing the SDS Single Step Monitor/Debugger

To install the SDS Single Step Monitor/Debugger, use the following procedure:

1. Insert the CD-ROM into the CD drive.
2. In Windows 3.1, select the File->Run menu item.

   In Windows 95, click the Start Button, then click Run.
3. Type `D:\SETUP` (where D is the drive letter for the CD drive) and press the Enter key.

2.4.2 Starting the SDS Single Step Monitor/Debugger

When you start the SDS Debugger, the debug window is open and the File tab is selected. To use the SDS Debugger with the M•CORE Evaluation System, perform the following steps:

1. Type the location and name of the ***.elf** file to be downloaded.
2. Click the Connection tab and choose the serial port. Under Details select `COM1` (or the appropriate comm port for your system) and `19200 baud`.
3. Click the Processor tab and select M•CORE as the target CPU. The Co-processor field should be set to `None`.
4. Click the Options tab and set the following options by clicking the check box next to each option until a check mark appears:

   Create symbol database
   Load application image
   Reset target
   Execute until main during reset
   Break at exit
5. Disable the Require exact symbol names option by clicking the appropriate check box until the check mark disappears.

2.4.3 Installing the Diab Data C Compiler

To install the Diab Data C Compiler, use the following procedure:

1. Insert the CD-ROM into the CD drive.
2. Select the root directory of the CD-ROM as your current directory. The installation program must see the **dtools** directory in the current directory.
3. From an command prompt window, type one of the following commands.

   For MS-DOS or Windows 3.xx:

   ```
   dtools\MSDOS\install <Enter>
   ```

   For Windows 95 or Windows NT:

   ```
   dtools\WIN32\install <Enter>
   ```

   For OS/2:

   ```
   dtools\OS2\install <Enter>
   ```

   For IRIX:

   ```
   dtools/SGI/install <Enter>
   ```

4. If you have not already done so, call the 1-800 number listed in the documentation to obtain the installation key from the manufacturer.
5. Type the correct response to each of the parameter questions:

   ```
   1. Source directory for the install (default: “current dir”)
   2. Destination directory for Diab tools (default: c:\diab)
   3. Installation key (default: reenter installation key)
   4. First diab tool to be installed. (default: yes)
   5. Second diab tool to be installed. (default: yes)
   etc...

   Enter “y” to continue or “n” to change the settings.
   ```

6. At the command prompt, type the following command:

   ```
   dctrl -t <Enter>
   ```

7. Select the SOFTWARE floating point option.
8. Select `cross - Ramdisk I/O`.

In addition to the standard installation procedures, the following steps are useful.

1. In the **autoexec.bat** file, add the following extension to the path command:

   ```
   ...;C:\diab\4.0b\Win32\bin
   ```

2. Verify the version number of Diab with the following command:

```
dcc -VV
```

2.4.4 Installing the Gnu C Compiler

The Gnu C Compiler can be downloaded from the Motorola web site at **http://www.motorola.com/mcore**.

2.4.4.1 Installation In Standard Directories

The M•CORE toolset binary is compiled and configured to be installed in one of the following locations, depending on the host:

Linux	**/usr/local**
Solaris 2.5	**/opt/mcore**
SunOS 4.1.3	**/usr/local**

Unpack the tar file in the specified directory. For example, to install the Gnu C Compiler in Linux, use the following commands:

```
cd /usr/local
tar xf the_linux_distribution_file.tar
```

Adjust the `PATH` environment relative to the Host. For most uses, **/usr/local/bin** is already be in the path. Solaris 2.5 users should add the **/opt/mcore/bin** directory to the path.

2.4.4.2 Installation In Non-Standard Directories

To install the Gnu C Compiler in an alternate directory, you must perform these additional steps.

1. Choose an alternate directory.
2. Copy the distribution tar file to the alternate directory.
3. Go to the alternate directory:
   `cd` *`/altdir`*
4. Unpack the distribution file:
   `tar xf` *`distribution_file`*

5. Set the environment variables as described below, replacing *altdir* with the alternate directory name:

Variable	Value
add to PATH	*/altdir*/bin
GCC_EXEC_PREFIX	*/altdir*/lib/gcc-lib/mcore-elf/SSRL-2.0/ (The trailing slash is required)
COMPILER_PATH	*/altdir*/mcore-elf/bin
C_INCLUDE_PATH	*/altdir*/mcore-elf/include
CPLUS_INCLUDE_PATH	*/altdir*/mcore-elf/include

6. Export these environment variables to sub-processes (export command in sh and ksh, setenv command in Cshell).

7. Add these definitions to the **.profile** or **.login** files to make the compiler accessible in future login sessions. The env command reveals the current settings for the system environment.

2.5 Running the Sample Programs

The following programs help you become acquainted with various aspects of running the M•CORE Evaluation System. These programs move through the three operating modes. The first two programs use the CMB only. The final operation incorporates both the CMB and the IPB and walks you through the action of accessing individual registers of the IPB modules. This final operation uses the SDS Single-Step Monitor/Debugger to prove the functionality of the IPB modules.

2.5.1 EVS Debug Mode

The getnum program illustrates the use of the EVS Debug mode. It is a C program which:

- Calls an assembly level function.
- Provides an example of keyboard interface while using SDS debug mode.
- Includes all of these functions in one C program.

The separate segments of the seven-segment LED are directly connected to bits of the Global Control register (GCR). Alter LED output by making assembler moves to this register. Figure 2-1 shows which GCR bits correspond to the LED segments.

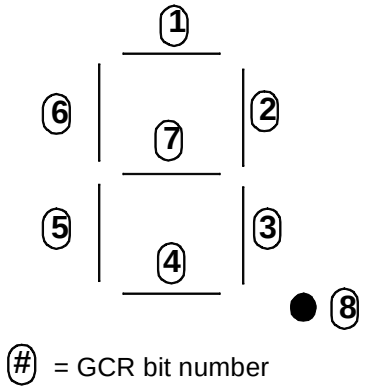


Figure 2-1. GCR to LED Mapping

To run getnum, use the following procedure.

1. Be sure the following files are in the same directory.

- **chario.c**
- **LED.h**
- **asm_mac.c**
- **getnum.c**
- **getnum.bat**

Note: To compile with Gnu C, replace **chario.c** with **printf.c**. Then link the provided **outc.o** file with the other ***.o** files in the final step.

---

2. Place the specified files into a directory that is accessible to the Diab Data controller. Use the `dcc -VV` command to determine the version number of Diab Data and to ensure its accessibility.

3. In the command prompt window, type the following command:

```
getnum <Enter>
```

This command runs the **getnum.bat** file which compiles **getnum.c** and produces the **getnum.elf** file.

4. Set the CMB to EVS debug mode, as instructed in *CMB Jumper Header Settings* on page 2-2.

5. Start the SDS Single-Step Debugger program.

6. From the file selection window, find, select, and open the **getnum.elf** file generated in Step 1.

7. When **getnum.elf** has been loaded, close the debug window.
8. Select the Windows->Command menu item to open the Command window.
9. At the Command window prompt, type `go <Enter>`.
10. In the Command window, type any single digit Hex number (0-F) and press `<Enter>`. The digit is displayed on the seven-segment LED for a brief period of time. Repeat this step until you are satisfied that the program is operating correctly.
11. Type `q <Enter>` to discontinue the getnum program. You can reset the program and run it again at this time.

2.5.2 Flash Programming Mode

The cmbflash program illustrates the use of the Flash Programming mode. In Flash Programming mode, you download software into the CMB EEPROMs.

The showa program downloads an S-record application into the Flash EEPROM which counts from $1 to $F in Hex. The application downloaded by showb counts from $F to $1 in Hex. Alternately downloading these two programs demonstrates that the EEPROMs have been reprogrammed.

To create and download the showa and showb programs, use the following procedure:

1. Be sure the following files are in the same directory and make that directory your current directory:

- **chario.h**
- **chario.o**
- **asm_mac.h**
- **led.h**
- **showa.c**
- **showb.c**
- **showa.bat**
- **showb.bat**
- **zero.s**

2. To produce the **showa.rec** file, type the following command in the DOS command prompt window:

```
showa <Enter>
```

3. To produce the **showb.rec** file, type the following command in the DOS command prompt window:

```
showb <Enter>
```

4. Be sure the CMB is powered off.
5. Connect the RS-232 cable to Serial 1 (closest to the reset switch).
6. Remove the jumper on J22.
7. To download the showa program, type the following command in the DOS command prompt window:

```
cmbflash showa.rec
```

8. When prompted, turn on the power to the CMB board.

   The command prompt window shows the results of EEPROM loading. When the cmbflash program is finished, you are instructed to remove the serial connection and reset the CMB. This reset can be accomplished with either a RESET or removing and adding power to the board.

9. Disconnect power to the CMB.
10. Disconnect the serial cable from the CMB.
11. Reconnect power to the CMB.

    When the power returns to the CMB, you should see the seven-segment LED on the CMB count upwards from $0 to $F in Hex.

12. Repeat Steps 5 through 9, substituting `showb.rec` in Step 5.

    In Step 9, the LED should count downward from $F to $0 in Hex. This countdown provides visual proof that the EEPROM has been reprogrammed.

2.5.3 User Mode

User mode copies the code in the Flash EEPROM to the on-board FSRAM and runs that code. This is actually the mode used in Step 9 of the Flash Programming mode example.

To operate the CMB in User Mode, use the following procedure:

1. Disconnect power to the CMB.
2. Disconnect the serial cable from the CMB.
3. Be sure that the jumper on J22 is not attached.

4. Connect power to the board and check the LED output to see if the program in the EEPROM has been copied to the FSRAM and is running properly.

2.6 Accessing Peripheral Device Registers

You can access the various modules of the IPB using the SDS Single-Step Monitor/Debugger program in EVS Debug mode. Using this mode, you can read specific registers of each module, write to the register to configure and prove access, and then read the register again to confirm that it was updated.

To access the IPB registers, use the following procedure:

1. Disconnect power from the CMB.
2. Attach the IPB to the CMB. Line up the MAPI connectors and the marker triangle and carefully push the boards together at the MAPI ring. J1 (top) of the IPB should align with P1 (bottom) of the CMB board.
3. Connect the serial cable to Serial 2 and to the PC or workstation.

   For the location of the Serial 2 connector, see Figure 3-1, “CMB Layout,” on page 3-2.
4. Reconnect power to the CMB.
5. Start the SDS Single-Step Monitor/Debugger program.
6. In the Debug window, check the box for Debug without a program.
7. Click OK in the Debug window to close it.
8. Open the Command window. The Command window allows direct access to the module address locations.
9. In the Command window, enter the following:

   ```
   read -w 0x1203208 <Enter>
   ```

   Confirm that the return value is `0x0000`. This command reads the Port A Data Direction Register (DDRQA) of the QADC module, which should be set to `0x0000` at reset.
10. In the Command window, enter the following:

    ```
    write 0x1203208 <Enter>
    ```

    This command enters write mode for the DDRQA register.

11. In the Command window, enter the following:

```
0xFFFF. <Enter>
. <Enter>
```

12. In the Command window, enter the following:

```
read 0x1203208 <Enter>
```

This command confirms an expected result of `0xFF00`.

13. Repeat Steps 9 through 12 to confirm the registers listed in Table 2-1.

Table 2-1. Module Registers to Test Read/Write

Register Name	Register Address	Peripheral Device No.	Value at Reset	Value to Write	Expected Value
QADC Data Direction Register (DDRQA)	0x1203208	1	0x0000	0xFFFF	0xFF00
QSM Interrupt level/vector Register (QILR/QIVR)	0x1203C04	1	0x000F	0x3F01	0x3F01
CTM4 Pulse Width Register (PWM5B)	0x120342C	1	0x0000	0x0000 0xFFFF	0x0000 0xFFFF
TouCAN™ Interrupt Mask (IMASK)	0x12030A2	1	0x0000	0xFFFF	0xFFFF
QADC Data Direction Register (DDRQA)	0x1205208	2	0x0000	0xFFFF	0xFF00
QSM Interrupt level/vector Register (QILR/QIVR)	0x1205C04	2	0x000F	0x3F01	0x3F01
CTM4 Pulse Width Register (PWM5B)	0x120542C	2	0x0000	0x0000 0xFFFF	0x0000 0xFFFF
TouCAN™ Interrupt Mask (IMASK)	0x12050A2	2	0x0000	0xFFFF	0xFFFF

The act of reading, writing, and reading each of these registers proves connectivity to one register of each of the modules found in each peripheral device.

2.7 Description of Software Programs

Table 2-2 lists and describes the programs used in the Quick Start activities.

Table 2-2. Program Descriptions

Program Name	Description of Program Function
SHOWA.c SHOWB.c	The basic C code to be compiled for the EEPROM programs. SHOWA.c counts up from $0 to $F, while SHOWB.c counts down from $F to $0.
LED.h	Defines the LED patterns for the function calls.
ASM_MAC.h	Represents the assembly routine called by the C program. While this routine is very simple, it serves as an example of how to input more complex assembly routines.
CHARIO.o	When using the Diab Data C compiler, this program contains character input/output and interrupt functions for the M•CORE Evaluation System. This program is not necessary for the EEPROM download. However, it is necessary for producing a proper interrupt in EVS Debug mode.
CHARIO.h	When using the Diab Data C compiler, this file provides support for the CHARIO.o program. Keep these two files together.
SHOWA.rec SHOWB.rec	The S-record files produced by SHOWA.bat and SHOWB.bat and used with CMBFLASH to program the EEPROMs on the CMB.
SHOWA.bat SHOWB.bat	Batch files to compile and link the SHOWA and SHOWB programs and produce the corresponding S-record files.
ROMA.lnk ROMB.lnk	These files link the C programs. They set the ROM, RAM, and stack addresses for downloading to the EEPROMs.
ZERO.s	Provides the information for interrupt space which overrides the CRT0.o file when properly included into a link file. (As is done with ROMA.lnk.)
CMBFLASH.exe	Prepares S-record files into packets and downloads these packets to the EEPROM, ensuring the SDS program in the EEPROM is not overwritten.
GETNUM.c	Demonstrates keyboard interaction with the seven-segment LED in EVS Debug mode. It calls an assembler routine in a C program. The user can expand on this program as desired.
OUTC.c	When using the Gnu C compiler, this file handles character I/O functions in conjunction with PRINTF.c. You must compile this file with Diab Data C because Gnu does not currently support the PRINTF functions.
PRINTF.c	When using the Gnu C compiler, this file handles I/O and Interrupt functions in conjunction with OUT.c.

Chapter 3. The Microcontroller Memory Board

In This Chapter

We discuss the features and operation of the CMB, including configuration jumper settings and connectors.

CMB Options 3-2
Mode Select Jumper (J22) 3-3
MDTACK/MDS Jumper (J21-13 & J21-15) 3-3
TRST Select Jumper (J99) 3-4
CMB Connectors 3-5
Serial 1 (J24) and Serial 2 (J25) Connectors 3-5
Power Connector (TB1) 3-5
Data Bus Connectors (J10, J11, J13, J14) 3-5
Address Bus Connectors (J4, J5, J7, J8) 3-7

3.1 CMB Options

The CMB provides several configuration options for determining the board's operation. You select the options you want by positioning shunts (jumpers) across the appropriate jumper pins.

Figure 3-1 illustrates the layout of the CMB, including the locations of the header blocks discussed in this chapter.

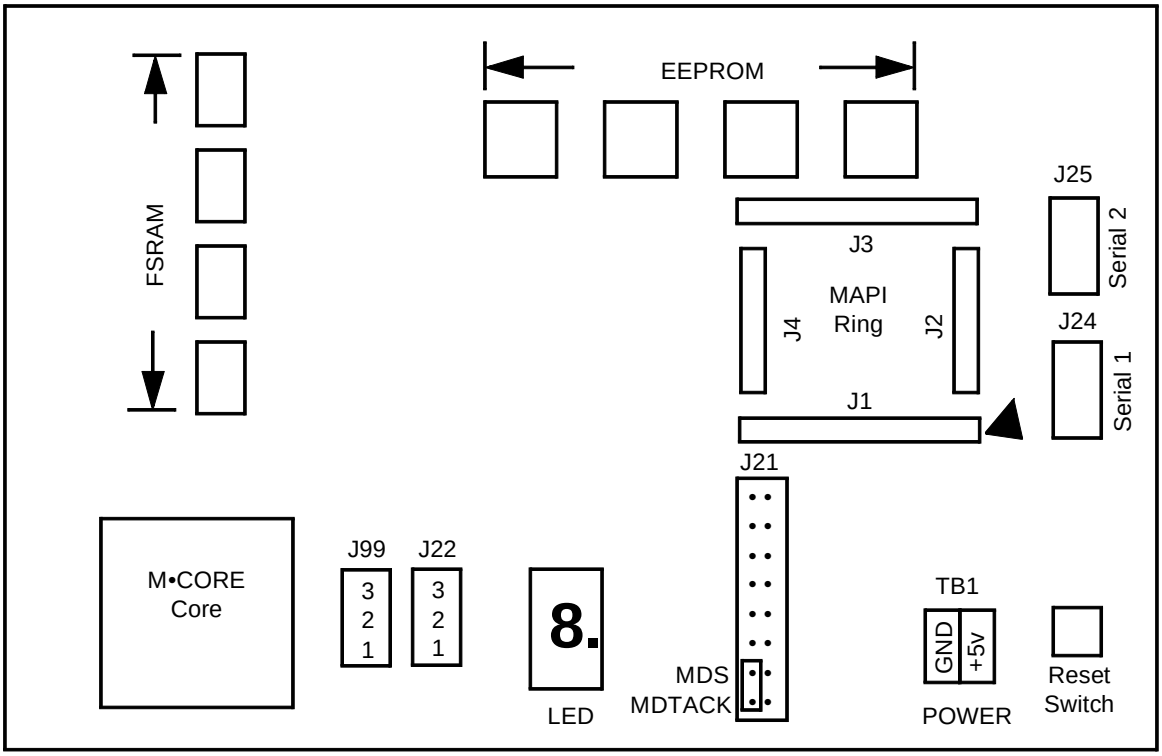
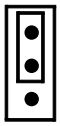
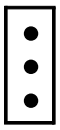
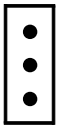


Figure 3-1. CMB Layout

3.1.1 Mode Select Jumper (J22)

The CMB operates in one of the modes listed in Table 3-1.

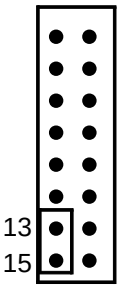
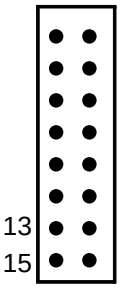
Table 3-1. Mode Select (J22) Jumper Positions

Position	Description
3 2 1 	**EVS Debug mode**—In EVS Debug mode, you use debugger software to download code to the FSRAM.
3 2 1 	**EEPROM Programming mode**—In EEPROM Programming mode, you use standalone software to burn code into the on-board EEPROM. Serial 1 must be connected to a PC.
3 2 1 	**User mode**—In User mode, the CMB copies the code in the EEPROM to FSRAM and runs that code. Serial 1 must be disconnected.

3.1.2 MDTACK/MDS Jumper (J21-13 & J21-15)

Use the MDTACK/MDS jumper to tell the CMB if it is being used with or without the IPB board, as shown in Table 3-2.

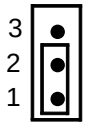
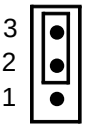
Table 3-2. MDTACK (J21-13 & J21-15) Jumper Positions

Position	Description
	CMB in stand-alone operation
	CMB with IPB

3.1.3 $\overline{\text{TRST}}$ Select Jumper (J99)

Use the $\overline{\text{TRST}}$ Select jumper to map either the ONCE $\overline{\text{TRST}}$ signal or the $\overline{\text{RESET}}$ signal to $\overline{\text{TRST}}$, as shown in Table 3-3.

Table 3-3. $\overline{\text{TRST}}$ Select (J99) Jumper Positions

Position	Description
	Map ONCE $\overline{\text{TRST}}$ signal to $\overline{\text{TRST}}$.
	Map $\overline{\text{RESET}}$ to $\overline{\text{TRST}}$.

3.2 CMB Connectors

The CMB provides the following connectors used with the various modes of operation. Refer to Figure 3-1 to locate the connectors.

3.2.1 Serial 1 (J24) and Serial 2 (J25) Connectors

The Serial 1 (J24) and Serial 2 (J25) connectors provide standard nine-pin RS-232 connections to the CMB. Both connectors have the same pinout, illustrated in Figure 3-2. The mode of operation determines which connection to use, as listed in Table 3-4.

Table 3-4. CMB Serial Connections

Operating Mode	Serial Connector
EVS Debug	Serial 2 (J25)
EEPROM Programming	Serial 1 (J24)
User	No serial connection

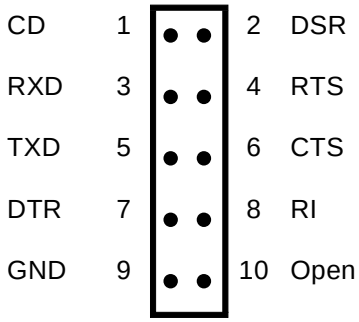


Figure 3-2. Serial Connector Pinout

3.2.2 Power Connector (TB1)

The CMB receives power at the TB1 connector. TB1 is a two lead connector: TB1-1 (red) for +5V and TB1-2 (black) for ground.

3.2.3 Data Bus Connectors (J10, J11, J13, J14)

The CMB provides read-only access to the M•CORE data bus through the J10, J11, J13, and J14 connectors, as shown in Table 3-5.

Table 3-5. CMB Data Bus Connection

Connector	Pin	Data Bus Bit
J14	15	0
J14	13	1
J14	11	2
J14	9	3
J14	7	4
J14	5	5
J14	3	6
J14	1	7
J13	15	8
J13	13	9
J13	11	10
J13	9	11
J13	7	12
J13	5	13
J13	3	14
J13	1	15
J11	15	16
J11	13	17
J11	11	18
J11	9	19
J11	7	20
J11	5	21
J11	3	22
J11	1	23
J10	15	24
J10	13	25
J10	11	26
J10	9	27
J10	7	28
J10	5	29
J10	3	30
J10	1	31

3.2.4 Address Bus Connectors (J4, J5, J7, J8)

The CMB provides read-only access to the M•CORE address bus through the J4, J5, J7, and J8 connectors, as shown in Table 3-6.

Table 3-6. CMB Address Bus Connection

Connector	Pin	Address Bus Bit
J8	15	0
J8	13	1
J8	11	2
J8	9	3
J8	7	4
J8	5	5
J8	3	6
J8	1	7
J7	15	8
J7	13	9
J7	11	10
J7	9	11
J7	7	12
J7	5	13
J7	3	14
J7	1	15
J5	15	16
J5	13	17
J5	11	18
J5	9	19
J5	7	20
J5	5	21
J5	3	22
J5	1	23
J4	15	24
J4	13	25
J4	11	26
J4	9	27
J4	7	28
J4	5	29
J4	3	30
J4	1	31

Chapter 4. The Input/Output Peripheral Board

In This Chapter

We discuss the features and operation of the IPB, including configuration jumper settings and connectors.

Introducing the Input/Output Peripheral Board 4-2
IPB Block Diagram 4-2
Peripheral Device Slave Units 4-2
General Purpose Inputs and Outputs 4-5
Bus and Control Signals 4-6
IPB Options 4-6
QADC Voltage Reference Source (W14, W15) 4-6
General Purpose Input/Output (W1 through W8) 4-7

4.1 Introducing the Input/Output Peripheral Board

The IPB emulates the peripheral modules for the M•CORE. The IPB contains the following modules and components:

- Two Motorola peripheral devices operating in slave mode: Each peripheral device provides the following modules:
  - CAN 2.0B controller module (TouCAN)
  - Queued analog-to-digital controller (QADC)
  - Queued serial module (QSM)
  - Configurable timer module, version 4 (CTM4)
- One field-programmable gate array (FPGA) programmed to re-map interrupts and provide edge-detect inputs and on-board chip selects
- Eight 74FCT652s providing 64 bits of general purpose input (GPI) or general purpose output (GPO)
- Jumpers for selecting latched output or input for the general purpose input/output
- Two PALs for bus matching and chip select steering
- Top-side MAPI connectors for interface with the CMB
- Bottom-side MAPI connectors for interface with the TIB

4.1.1 IPB Block Diagram

Figure 4-1 illustrates the block diagram of the IPB.

4.1.2 Peripheral Device Slave Units

This section gives more details about the capabilities of each Motorola peripheral device slave unit. For more detailed information about each module, see the documents listed in *References* on page 1-5.

4.1.2.1 CAN 2.0B Controller Access Network Module (TouCAN)

- Full implementation of CAN protocol specification, version 2.0 A and B
- 16 receive/transmit message buffers of 0 to 8 bytes
- Independent mask registers for message buffers 14 and 15
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- 16-bit free-running timer for message time stamping
- Low power sleep mode with programmable wake-up on bus activity

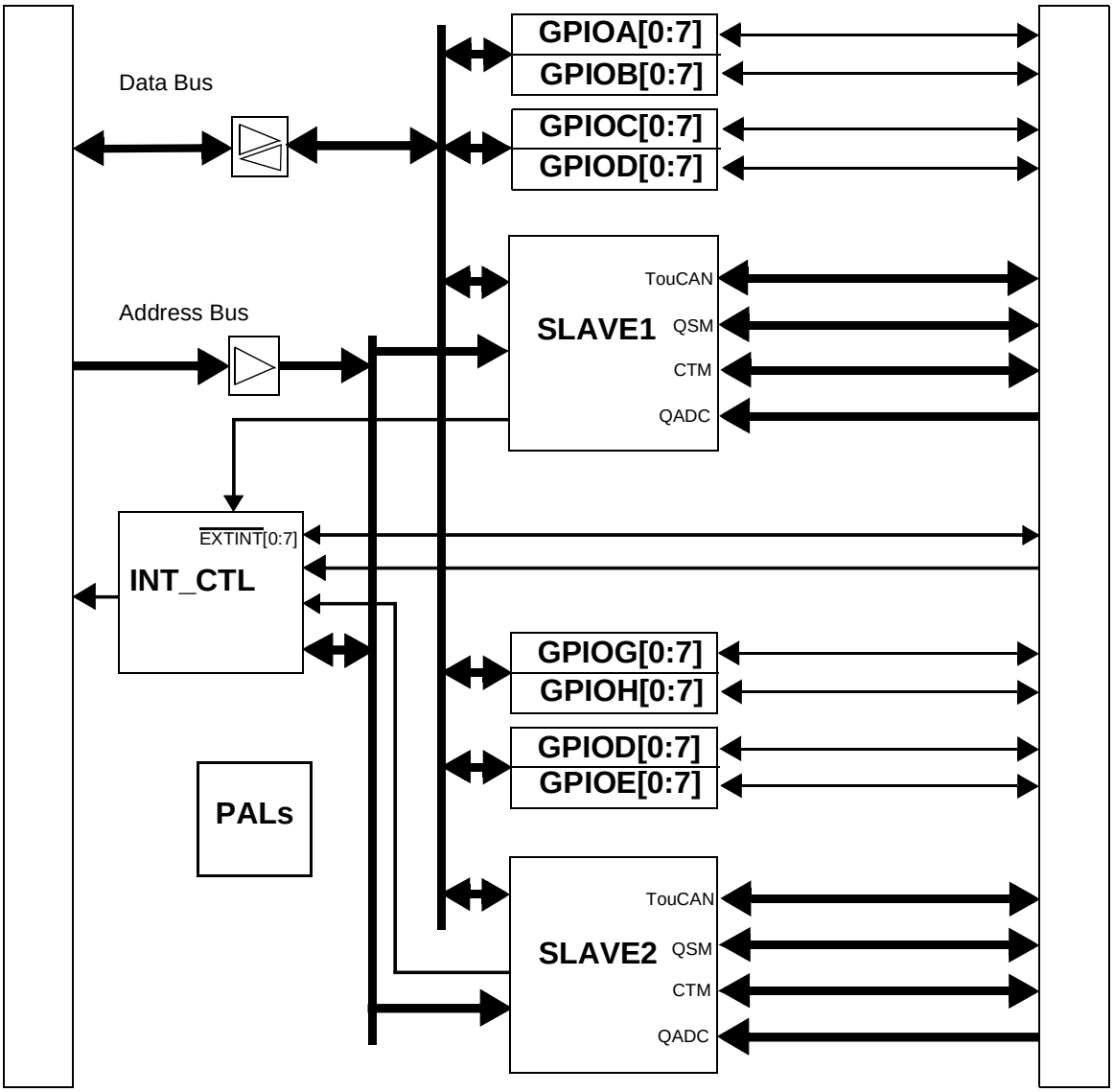


Figure 4-1. IPB Block Diagram

4.1.2.2 10-Bit Queued Analog-to-Digital Converter (QADC)

- 16 channels internally; up to 44 directly accessible channels with external multiplexing
- Six automatic channel selection and conversion modes
- Two channel scan queues of variable length, each with a variable number of subqueues
- 40 result registers and 3 result alignment formats
- Programmable input sample time
- Direct control of external multiplexers

4.1.2.3 Queued Serial Module (QSM)

- Enhanced serial communications interface (SCI)
  - Modulus baud rate generator
  - Parity detection
- Queued serial peripheral interface (QSPI)
  - 80-byte static RAM to perform queued operations
  - Up to 16 automatic transfers
  - Continuous cycling, 8 to 16 bits transfer, LSB or MSB first
  - Dual function I/O pins

4.1.2.4 Configurable Timer Module, Version 4 (CTM4)

4.1.2.4.1 Two 16-bit Modulus Counter SubModules (MCSM)

- An enhanced FCSM
- Composed of:
  - One 16-bit modulus latch
  - One 16-bit loadable up-counter
  - Counter loading logic
  - Clock selector
  - Selectable time-base bus drivers
  - Interrupt interface

4.1.2.4.2 One 16-bit Free-running Counter SubModule (FCSM)

- 16-bit counter with an associated clock source selector
- Selectable time-base bus drivers
- Control registers
- Status bits
- Interrupt interface

4.1.2.4.3 Four Double-Action SubModules (DASM)

- Allows two 16-bit input capture or two 16-bit output compare functions to occur automatically without software intervention
- Input edge detector can be programmed to trigger the capture function on user specified edges
- Output flip-flop can be set by one of the output compare functions and reset by the other one

- Interrupt requests can optionally be generated by the input capture and the output compare functions
- User can select one of the 2 incoming time bases for the input capture and output compare functions
- Composed of:
  - Two timing channels (A & B)
  - One output flip-flop
  - One input edge detector
  - Control logic
  - Interrupt interface

4.1.2.4.4 Four Pulse-Width Modulation SubModules (PWMSM)

- Allows pulse width modulated signals to be generated over a wide range of frequencies, independently of other CTM4 output signals
- Output pulse width duty cycle can vary from 0% to 100% with 16 bits of resolution
- Minimum pulse width is twice the MCU system clock period
- Composed of:
  - Output flip-flop with output polarity control
  - Clock prescaler and selection logic
  - 16-bit up-counter
  - Two registers to hold the current and next pulse width values
  - Two registers to hold the current and next pulse period values
  - Pulse width comparator

4.1.3 General Purpose Inputs and Outputs

The IPB uses 74FCT652s to generate the GPIOs. The 74FCT652s are 8-bit registered bus transceivers configured to be registered on writes with loop-back on reads. Each 74FCT652 has a corresponding jumper location which, if the jumper is installed, configures that set of GPIOs as an input port with loop-back read. If the jumper is not installed, the group is strictly output.

4.1.4 Bus and Control Signals

All bus and control signals are buffered before being fed to the peripheral devices. A PAL controls the steering of chip select on the IPB while another PAL handles the arbitration for access to the peripheral devices. The PAL generates the control logic required for the hand shaking between the CPU and the peripheral devices.

4.2 IPB Options

The IPB provides several configuration options for determining the board's operation. You select the options you want by positioning shunts (jumpers) across the appropriate jumper pins.

Figure 4-2 illustrates the layout of the IPB, including the locations of the header blocks discussed in this chapter.

4.2.1 QADC Voltage Reference Source (W14, W15)

The W14 and W15 headers select the source of the low-level and high-level analog voltage references, as shown in Table 4-1 and Table 4-2.

Table 4-1. Low Voltage Reference Source Jumper (W14)

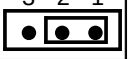
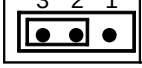
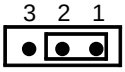
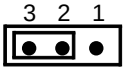
W14	Description
	On-board low voltage reference
	External low voltage reference through MAPI P2-12

Table 4-2. High Voltage Reference Source Jumper (W15)

W15	Description
	External high voltage reference through MAPI P3-6
	On-board high voltage reference

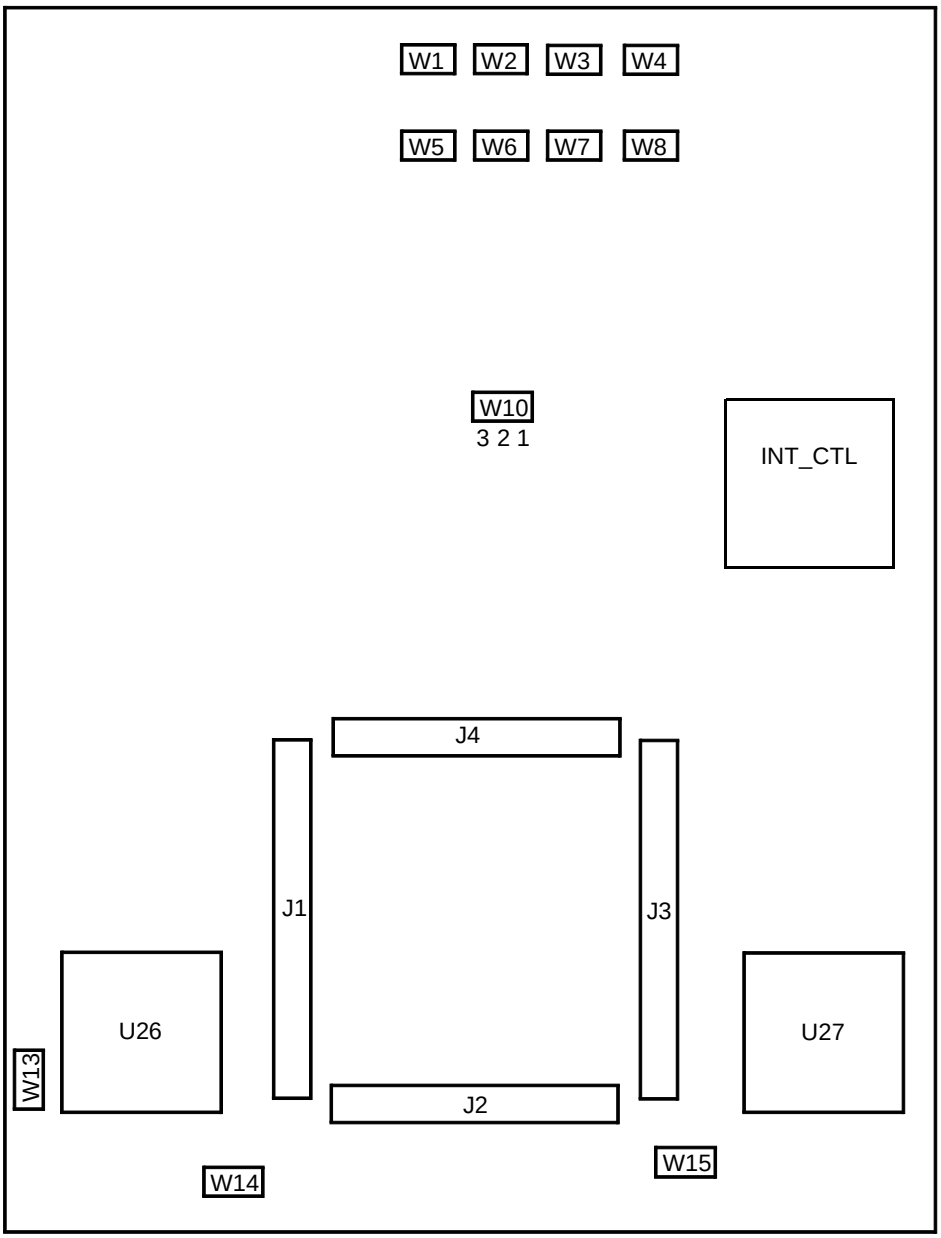


Figure 4-2. IPB Board Layout

4.2.2 General Purpose Input/Output (W1 through W8)

The W1 through W8 header blocks specify whether each set of GPIO signals operate as outputs or inputs, as shown in Table 4-3. Table 4-4 shows which header controls which set of GPIO signals.

Table 4-3. General Purpose Input/Output Jumpers (W1 through W8)

W1–W8	Description
	Jumper off specifies GPIO signals as latched output only. This is the default position for W2, W4, W6, and W8.
	Jumper on specifies GPIO signals as input only. This is the default position for W1, W3, W5, and W7.

Table 4-4. GPIO Control Header Mapping

GPIO Set	Header
GPIOA	W2
GPIOB	W6
GPIOC	W3
GPIOD	W5
GPIOE	W4
GPIOG	W1
GPIOH	W7
GPIOK	W8

4.2.3 W10 Jumper

The W10 header must have a jumper on pins 1 and 2, as shown in .

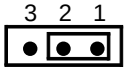


Figure 4-3. W10 Jumper Position

4.2.4 W13 Jumper

The W13 header is reserved for factory use.

Chapter 5. Memory Maps and Pinouts

In This Chapter

We provide the memory maps and connector pinouts for the M•CORE Evaluation System.

IPB Memory Maps 5-2
EVS Base Addresses 5-2
Peripheral Memory Map 5-3
Interrupt Controller Memory Map 5-4
GPIO Memory Map. 5-5
Test Interface Board Pinouts 5-5

5.1 IPB Memory Maps

To use the IPB successfully, you need to know where information is stored in the various memory components. This section contains the relevant memory maps.

5.1.1 EVS Base Addresses

The M•CORE Evaluation System uses the base addresses listed in Table 5-1.

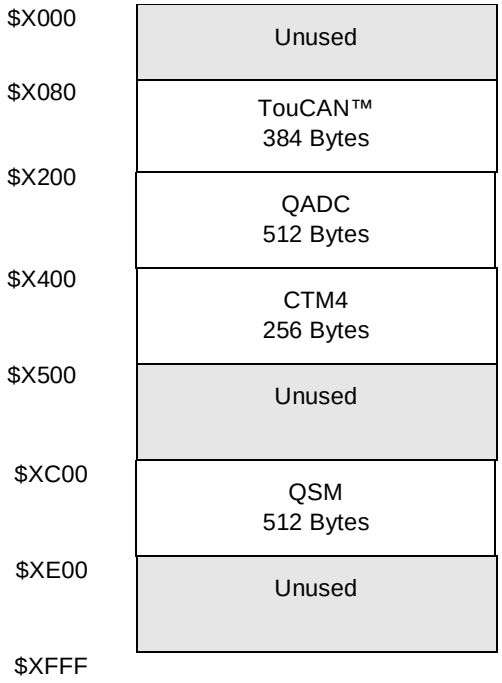
Table 5-1. Base Addresses

Base Address	Device
0x00000000	Flash EEPROM
0x00200000	FSRAM
0x00400000	UART1 (Serial 1)
0x00600000	UART2 (Serial 2)
0x00800000	Swap[1]
0x01200000	Interrupt Controller[2]
0x01200800	GPIOs[2]
0x01203000	Peripheral Device 1[2]
0x01205000	Peripheral Device 2[2]

[1]Bit 0 of the Swap register allows the Flash EEPROM and FSRAM base addresses to be swapped. Setting Bit 0 swaps the addresses; clearing bit 0 leaves the addresses as shown. After RESET, the monitor sets this bit. This bit is write only.
[2]These devices are 16-bit access only.

5.1.2 Peripheral Memory Map

Figure 5-1 illustrates the memory map for a peripheral device.



Note:
X = 3 for Peripheral Device 1
X = 5 for Peripheral Device 2

Figure 5-1. Peripheral Address Map

5.1.3 Interrupt Controller Memory Map

Table 5-1 lists the offset addresses for the Interrupt Controller registers. For detailed information about these registers, see Chapter 6, “Interrupt Control and PORTF.”

Table 5-1 Interrupt Controller Memory Map

Offset Address	Read/Write	Register
0000	Read Only	Interrupt Source 1 (ISR1)
0002	Read Only	Interrupt Source 2 (ISR2)
0004	R/W	Interrupt Enable 1 (IER1)
0006	R/W	Interrupt Enable 2 (IER2)
0008		Reserved
000A	R/W	Fast Interrupt Enable (FIER)
0100	R/W	Port F Edge Control (PFECR)
0102–0108		Reserved
010A	R/W	Port F Interrupt Enable (PFIER)
010C	R/W	Port F Data Direction (DDRF)
010E	R/W	Port F Output Data (PORTF)
0110	R/W	Port F Pin Data (PORTFP)
0112	R/W	Port F Edge Detect (PORTFE)
0114–07FF		Reserved

5.1.4 GPIO Memory Map

Table 5-2 lists the offset addresses for the GPIO registers.

Table 5-2. GPIO Memory Map

Offset Address	Read/Write	Register
0800–0801	R/W	GPIOA & GPIOB
0802–080F		Reserved
0810–0811	R/W	GPIOC & GPIOD
0812–081F		Reserved
0820–0821	R/W	GPIOE & GPIOK
0822–082F		Reserved
0830–0831	R/W	GPIOG & GPIOH
0832–083F		Reserved
0840–1FFF		Reserved

5.2 Test Interface Board Pinouts

Figure 5-2 and Figure 5-3 show the connector layout with signals names for the left and right halves of the Test Interface board (TIBA0002), respectively.

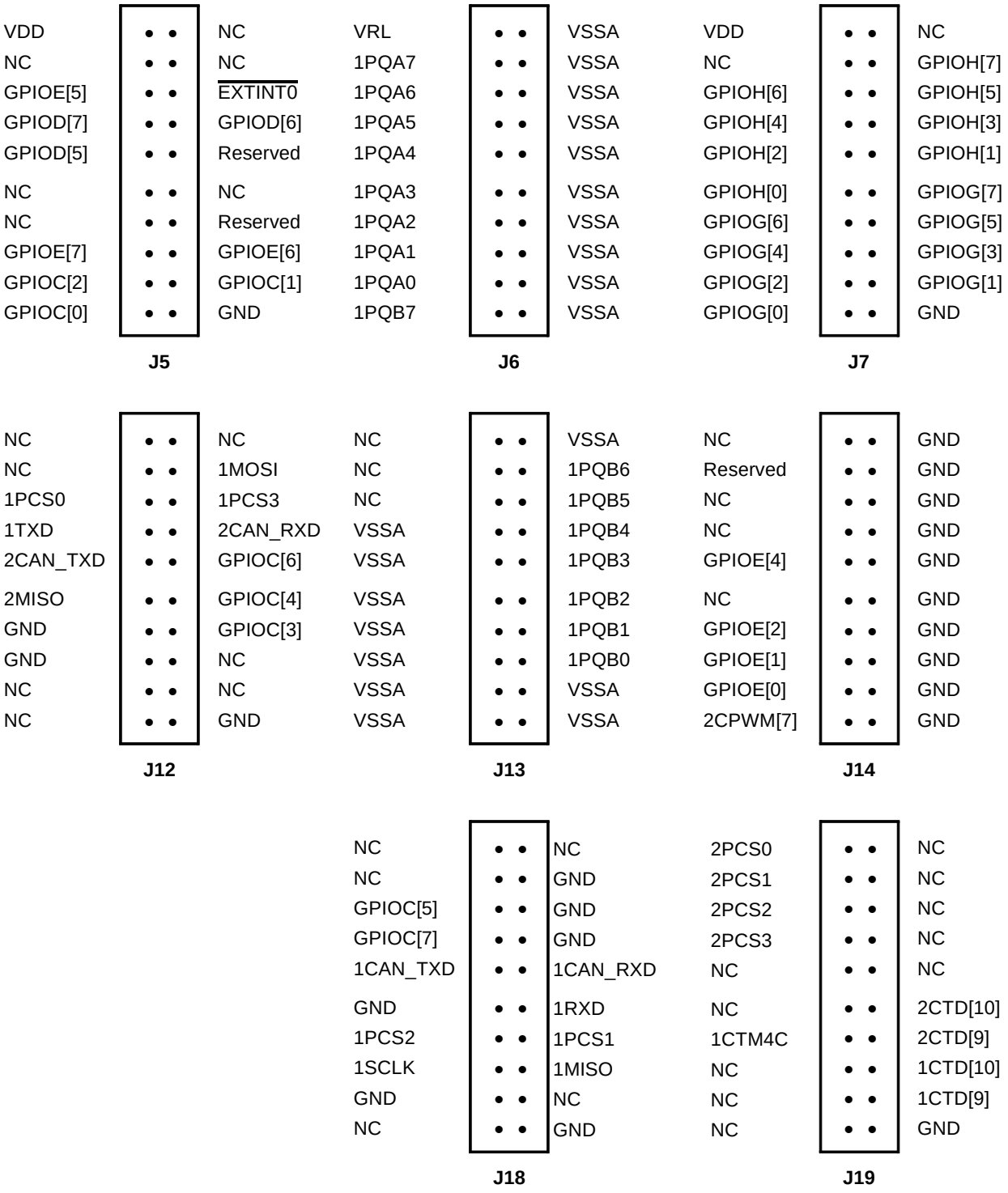


Figure 5-2. TIB Connector Layout (Left Half)

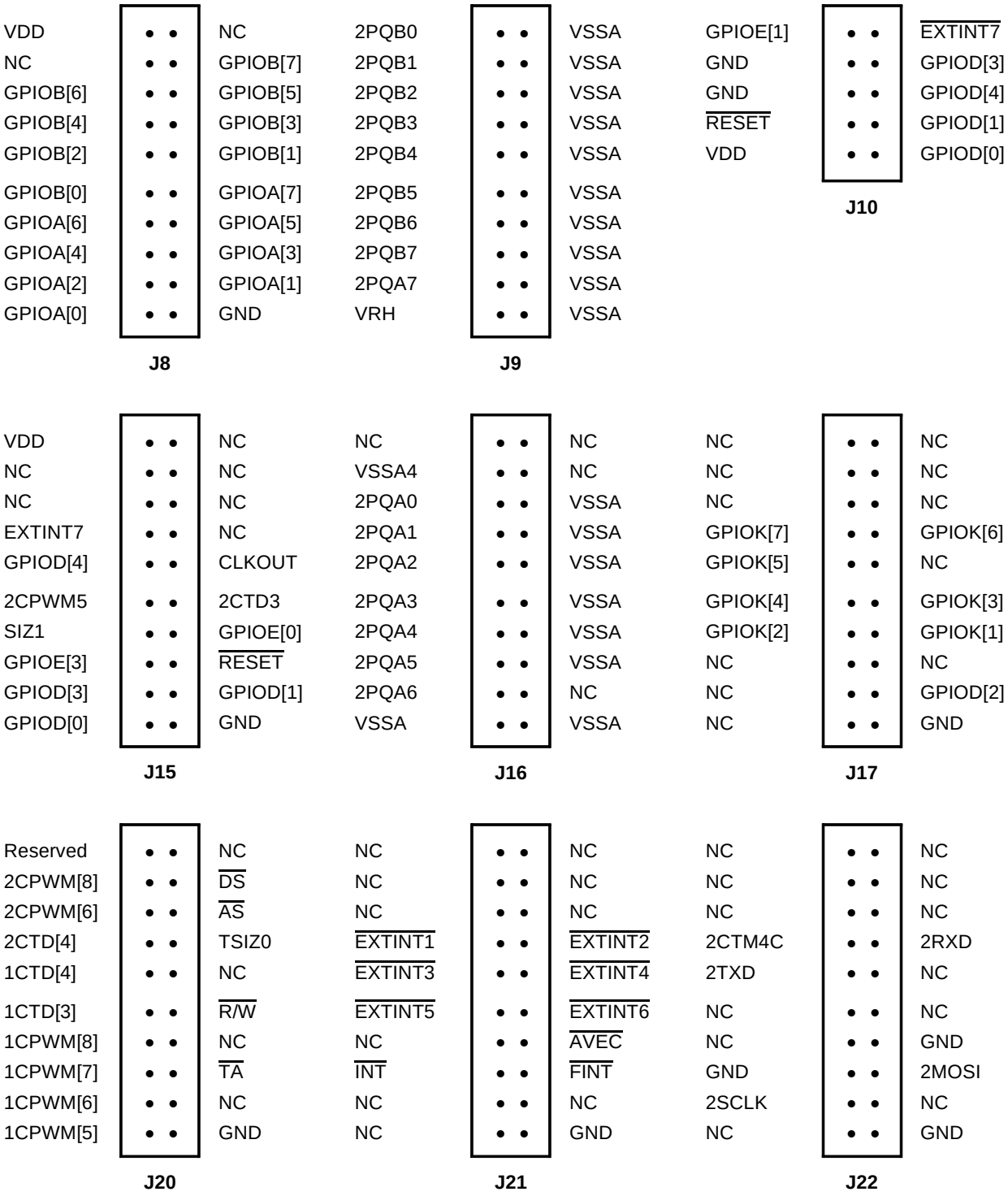


Figure 5-3. TIB Connector Layout (Right Half)

Chapter 6. Interrupt Control and PORTF

In This Chapter

We provide the information needed to control interrupt processing in the M•CORE Evaluation System and to utilize the 8 bits of Port F.

Interrupt Controller Registers 6-2
Interrupt Source Register 1 (ISR1) 6-2
Interrupt Source Register 2 (ISR2) 6-3
Interrupt Enable Register 1 (IER1) 6-3
Interrupt Enable Register 2 (IER2) 6-4
Fast Interrupt Enable Register (FIER) 6-4
Controlling Interrupts 6-5
Interrupt Vector Map 6-5
Interrupt Controller Port F 6-7
Port F Edge Control Register (PFECR) 6-7
Port F Interrupt Enable Register (PFIER) 6-8
Port F Data Direction Register (DDRF) 6-9
Port F Output Data Register (PORTF) 6-9
Port F Pin Data Register (PORTFP) 6-10
Port F Edge-Detect Register (PORTFE) 6-10

6.1 Interrupt Controller Registers

The interrupt controller on the M•CORE Evaluation System provides the following registers for controlling and processing interrupt requests.

6.1.1 Interrupt Source Register 1 (ISR1)

This register and the Interrupt Source Register 2 return the values of the interrupt source signals. These registers are read only. Interrupts must be cleared at their source.

Address: $01200000

	15	14	13	12	11	10	9	8	
	S2_IRQ7	S2_IRQ6	S2_IRQ5	S2_IRQ4	S2_IRQ3	S2_IRQ2	S2_IRQ1	UARTINT2	R
RESET:	X	X	X	X	X	X	X	X	

	7	6	5	4	3	2	1	0	
	Reserved								R
RESET:	0	0	0	0	0	0	0	0	

Bits 7:0 Reserved—Writes to these reserved bits have no effect and reads return zeros.

Bits 15:8 Interrupt Source Bits—These bits and the bits in Interrupt Source Register 2 identify the source of the interrupt.

0 = Interrupt pending.

1 = No interrupt pending.

6.1.2 Interrupt Source Register 2 (ISR2)

This register and the Interrupt Source Register 1 return the values of the interrupt source signals. These registers are read only. Interrupts must be cleared at their source.

Address: $01200002

	15	14	13	12	11	10	9	8	
	EXTINT7	EXTINT6	EXTINT5	EXTINT4	EXTINT3	EXTINT2	EXTINT1	EXTINT0	R
RESET:	X	X	X	X	X	X	X	X	

	7	6	5	4	3	2	1	0	
	S1_IRQ7	S1_IRQ6	S1_IRQ5	S1_IRQ4	S1_IRQ3	S1_IRQ2	S1_IRQ1	UARTINT1	R
RESET:	X	X	X	X	X	X	X	X	

Bits 15:0 Interrupt Source Bits—The bits in this register and in the lower byte of Interrupt Source Register 1 identify the source of the interrupt.

6.1.3 Interrupt Enable Register 1 (IER1)

Address: $01200004

	15	14	13	12	11	10	9	8	
	S2_IRQ7	S2_IRQ6	S2_IRQ5	S2_IRQ4	S2_IRQ3	S2_IRQ2	S2_IRQ1	UARTINT2	R
RESET:	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	Reserved								R
RESET:	0	0	0	0	0	0	0	0	

Bits 7:0 Reserved—Writes to these reserved bits have no effect and reads return zeros.

Bits 15:8 Interrupt Enable Bits—The interrupt controller generates a distinct vector number to be presented to the M•CORE VEC[n] pins. Each enabled and non-masked interrupt source causes the CPU to fetch an interrupt vector address from the exception vector table using the following offset from the vector base register:

```
[decimal] offset (128 + (4 x <bit_number>))
```

0 = Interrupt is masked and is not presented to the VEC[n] pins.

1 = Interrupt is not masked.

6.1.4 Interrupt Enable Register 2 (IER2)

Address: $01200006

	15	14	13	12	11	10	9	8	
	EXTINT7	EXTINT6	EXTINT5	EXTINT4	EXTINT3	EXTINT2	EXTINT1	EXTINT0	R/W
RESET:	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	S1_IRQ7	S1_IRQ6	S1_IRQ5	S1_IRQ4	S1_IRQ3	S1_IRQ2	S1_IRQ1	UARTINT1	R/W
RESET:	0	0	0	0	0	0	0	0	

Bits 15:0 Interrupt Enable Bits—The interrupt controller generates a distinct vector number to be presented to the M•CORE VEC[n] pins. Each enabled and non-masked interrupt source causes the CPU to fetch an interrupt vector address from the exception vector table using the following offset from the vector base register:

```
[decimal] offset (128 + (4 x <bit_number>))
```

0 = Interrupt is masked and is not presented to the VEC[n] pins.

1 = Interrupt is not masked.

6.1.5 Fast Interrupt Enable Register (FIER)

Address: $0120000A

	15	14	13	12	11	10	9	8	
	EXTINT7	EXTINT6	EXTINT5	EXTINT4	EXTINT3	EXTINT2	EXTINT1	EXTINT0	R/W
RESET:	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	Reserved								R/W
RESET:	0	0	0	0	0	0	0	0	

The M•CORE Evaluation System uses fast interrupt (FINT) priority only for the external interrupts. The remaining interrupts cannot have a FINT priority.

EXTINT[7:0] Fast Interrupt Enable

0 = Interrupt request pending configured as a normal interrupt source.

1 = Interrupt request pending configured as a fast interrupt source.

6.2 Controlling Interrupts

To set up the M•CORE Evaluation System peripheral devices for interrupt control, use the following procedure:

1. Enable test mode by writing $0001 to the CREG register (address $0120XA38) in the SIM.
2. Enable Show IRQ by writing $0320 to the SCIMTR register (address $0120XA02). Enabling Show IRQ forces information on the internal interrupt request lines to be driven out the external IRQ pins.
3. Assign the peripheral device's Port F to I/O by writing $0000 to the PFPAR register (address $0120XA1E).
4. Configure the peripheral device's Port F pins for output by writing $00FF to the DDRF register (address $0120XA1C).
5. Enable interrupts from the desired onboard peripheral device module.
6. Set the interrupt level for that module to the appropriate IRQ line.
7. In the Interrupt Controller, enable the interrupt bit for that IRQ line.
8. Write the address for the interrupt service routine to interrupt vector table.

Notes:

1. In the register addresses for this procedure, X = 3 for peripheral slave device 1 and X = 5 for peripheral slave device 2.
2. You must complete this procedure before you enable the peripheral device interrupts through the Interrupt Enable registers (IER1 and IER2).
3. This procedure deals with the Port F pins of the peripheral devices. Do not confuse these Port F pins with the Interrupt Controller Port F.

6.3 Interrupt Vector Map

When an interrupt occurs, the corresponding bit is set in the Interrupt Source registers at offsets 0000 (bits 15:0) and 0002 (bits 31:16). Internal logic then encodes an interrupt vector for the highest level of interrupt present (FINT has priority over INT). Table 6-1 lists the interrupts by their bit location in the Interrupt Source registers.

Table 6-1. Interrupt Vector Map

Bit	Vector Offset	Interrupt Type	Input Pin	Function
31	$0FC	FINT/INT	EXTINT[7]	Port F bit 7/IRQ7
30	$0F8	FINT/INT	EXTINT[6]	Port F bit 6/IRQ6
29	$0F4	FINT/INT	EXTINT[5]	Port F bit 5/IRQ5
28	$0F0	FINT/INT	EXTINT[4]	Port F bit 4/IRQ4
27	$0EC	FINT/INT	EXTINT[3]	Port F bit 3/IRQ3
26	$0E8	FINT/INT	EXTINT[2]	Port F bit 2/IRQ2
25	$0E4	FINT/INT	EXTINT[1]	Port F bit 1/IRQ1
24	$0E0	FINT/INT	EXTINT[0]	Port F bit 0/IRQ0
23	$0DC	INT	S1_IRQ[7]	User defined*
22	$0D8	INT	S1_IRQ[6]	User defined*
21	$0D4	INT	S1_IRQ[5]	User defined*
20	$0D0	INT	S1_IRQ[4]	User defined*
19	$0CC	INT	S1_IRQ[3]	User defined*
18	$0C8	INT	S1_IRQ[2]	User defined*
17	$0C4	INT	S1_IRQ[1]	User defined*
16	$0C0	INT	UARTINT1	UART #1 Interrupt
15	$0BC	INT	S2_IRQ[7]	User defined*
14	$0B8	INT	S2_IRQ[6]	User defined*
13	$0B4	INT	S2_IRQ[5]	User defined*
12	$0B0	INT	S2_IRQ[4]	User defined*
11	$0AC	INT	S2_IRQ[3]	User defined*
10	$0A8	INT	S2_IRQ[2]	User defined*
9	$0A4	INT	S2_IRQ[1]	User defined*
8	$0A0	INT	UARTINT2	UART #2 Interrupt
7	$09C	INT		Reserved
6	$098	INT		Reserved
5	$094	INT		Reserved
4	$090	INT		Reserved
3	$08C	INT		Reserved
2	$088	INT		Reserved
1	$084	INT		Reserved
0	$080	INT		Reserved
* Programmed through the peripheral device				

6.4 Interrupt Controller Port F

Eight digital I/O pins are available for control functions. Each pin includes edge-detect logic which can function as an interrupt request generator. These pins are configured as falling edge-detect during reset. They may be programmed as rising edge-detect. All I/O pins are configured as inputs at reset and can be programmed as either input or output pins through the data direction register (DDRF).

I/O pins have both an data (output) register (PORTF) and a pin state register (PORTFP) to monitor and control the state of its pins. Writes to PORTF cause the data to be stored in the register and to be driven to the corresponding pins when they are programmed as outputs. A read of PORTF returns the current value of PORTF, regardless of the actual state of the pins. A read of PORTFP returns the current state of the corresponding pins, regardless of whether they are input or output. Writes to PORTFP have no effect.

Two additional registers define edge sensitivity and provide edge detection status. The Port F Edge Control Register (PFECR) controls the edge-detection for the I/O pins. The Port F Edge-Detect Register (PORTFE) indicates that a transition has occurred on an I/O pin.

The Port F Interrupt Enable Register (PFIER) controls whether or not an edge-detection on an I/O pin causes an interrupt.

6.4.1 Port F Edge Control Register (PFECR)

This register controls the edge-detect function for the Port F pins.

Address: $01200100

	15	14	13	12	11	10	9	8	
	Reserved								R/W
Reset:	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0	R/W
Reset:	0	0	0	0	0	0	0	0	

PF[x] PORTF Pin Edge Detect—This bit controls the function of the corresponding pin.

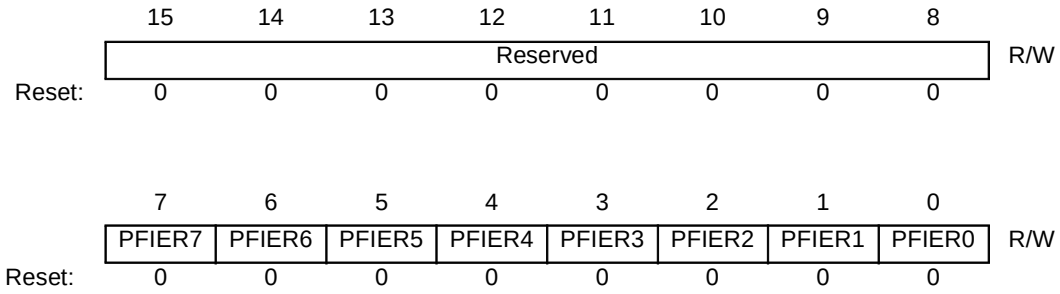
0 = Digital I/O with falling edge detect

1 = Digital I/O with rising edge detect

6.4.2 Port F Interrupt Enable Register (PFIER)

The bits in this register individually enable interrupt requests generated by the corresponding bit in the edge-detect register. If a PFIER bit is 1 and the corresponding bit in the PORTFE register is set (or later becomes set when the specified edge occurs on the corresponding pin), the interrupt controller generates an interrupt request at the interrupt priority level specified by the interrupt vector map. If the PFIER bit is written to 0, the interrupt controller negates the interrupt request.

Address: $0120010A



PFIER[7:0] PORT F Interrupt Enable—This bit controls the function of the corresponding pin.

0 = Interrupt disabled

1 = Interrupt enabled

6.4.3 Port F Data Direction Register (DDRF)

The bits in this register control the direction of the Port F pin drivers.

Address: $0120010C



DDRF[7:0] PORT F Data Direction (Read/Write)—Setting any bit in this register to 1 configures the corresponding pin as an output. Clearing any bit in this register to 0 configures the corresponding pin as an input.

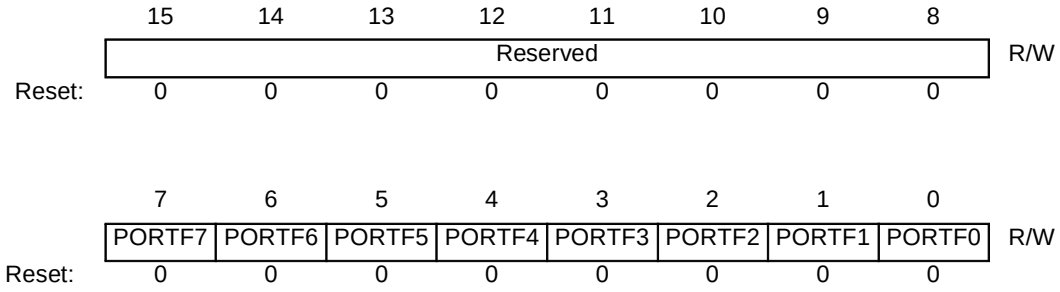
0 = Input

1 = Output

Note that interrupt requests can be generated by programming the PORTF output data register when the DDRF selects output.

6.4.4 Port F Output Data Register (PORTF)

Address: $0120010E



PORTF[7:0] PORT F Output Data (Read/Write)—The PORTF register stores the data to be driven on the Port F output pins. Reading this register returns the current value of the PORTF register, not the states of the Port F pins.

6.4.5 Port F Pin Data Register (PORTFP)

Address: $01200110

	15	14	13	12	11	10	9	8	
	Reserved								R/W
Reset:	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	PORTFP7	PORTFP6	PORTFP5	PORTFP4	PORTFP3	PORTFP2	PORTFP1	PORTFP0	R/W
Reset:	0	0	0	0	0	0	0	0	

PORTFP[7:0] PORT F Pin Data (Read Only)—When read, PORTFP reflects the current state of the Port F pins. Writes to PORTFP have no effect.

6.4.6 Port F Edge-Detect Register (PORTFE)

Address: $01200112

	15	14	13	12	11	10	9	8	
	Reserved								R/W
Reset:	0	0	0	0	0	0	0	0	

	7	6	5	4	3	2	1	0	
	PORTFE7	PORTFE6	PORTFE5	PORTFE4	PORTFE3	PORTFE2	PORTFE1	PORTFE0	R/W
Reset:	0	0	0	0	0	0	0	0	

PORTFE[7:0] PORT F Edge Detect (Read/Write)—The Port F Edge-Detect register (PORTFE) indicates that the programmed transition has occurred on Port F input or output pins. The edge-detect bits in PORTFE are set if the specified edge is detected on the corresponding Port F pin. If a pin transition occurs as a result of changing the PFECR, an erroneous edge-detect may be detected. This condition must be cleared before enabling interrupts (PFIER).

Interrupts can be generated after the proper transition has been detected. An interrupt request is generated whenever the enable bit in the Port F Interrupt Enable Register (PFIER) for the corresponding pin is 1. The bit in the PORTFE register (if the programmed edge is detected) or in the PFIER register can be set in any order to generate the interrupt request.

Once set, the Edge-Detect Flag bits remain set, regardless of the subsequent state of the corresponding pin or changes in PFECR programming, until the bit is cleared by software or a reset. To clear an edge-detect flag, the bit must be read first and then written to 0. Flags which are 0 when the register is read are unaffected by the write operation. Also, if the edge detect logic detects another edge after the flag was read as a 1 and before a 0 is written to clear it, the flag cannot be cleared until the flag is read as a 1 again and written to a 0. Writing 1 to a bit has no effect. Only writes of 0 are valid, when permitted, to clear the bit(s).

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP