

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 *Molex / Innovative Integration cM6x* at our website: ***Click HERE***

M6x/cM6x Development Package Manual

The M6x/cM6X Development Package Manual was prepared by the technical staff of Innovative Integration, November 6, 2002.

For further assistance contact:

Innovative Integration
2655 Park Center Drive
Simi Valley , California 93065

PHONE: (805) 520-3300
FAX: (805) 579-1730
email:techsprt@innovative-dsp.com
Website:www.innovative-dsp.com

This document is copyright 2001 by Innovative Integration. All rights are reserved.

VSS\M62\documents\Hw-Sw Manual\M6x.book

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

List of Tables 7
List of Figures 9

CHAPTER 1 *Introduction* 11

A Note about this Manual 12

CHAPTER 2 *Installation Instructions* 13

Host Hardware Requirements 13
Software Installation 13
Tools Registration 16
Hardware Installation 18
JTAG Emulator Hardware Installation (for DSP boards Only, the JTAG is NOT needed with ChicoPlus) 18
Baseboard Installation (for DSP boards AND ChicoPlus) 18
Code Composer Studio Setup 20

CHAPTER 3 *Integrated Development Environment* 23

The Texas Instruments C Compiler Toolset 23
C Compiler Toolset Usage 24
Code Composer Studio 24
Editor 24
Debugger 24

CHAPTER 4 *Support Applets* 25

The Terminal Emulator "UniTerminal" 25
The COFF File UniDownloader 34
Using the Application 34
Command Line Switches 34
Manual Operation 35
The COFF File Dump Utility 36

CHAPTER 5 *Developing Target Code* 39

Introduction 39
Components of Target Code (.c, .asm, .cmd) 40
Edit-Compile-Test Cycle using Code Composer Studio 40
A Simple Code Composer Studio Project 40
Build Options (M62, Q62, SBC62 Boards) 42
Build Options (M67, Q67, SBC67 Boards) 44
Build Options (SBC6711 Boards) 46

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Build Options (M44 Boards) 48
Build Options (SBC32 Boards) 50
Automatic projectfile creation 53
Rebuilding a Project 53
Running the Target Executable 53
Anatomy of a Target Program 54
Use of Library Code 55
Compiling/Assembling/Linking Outside Code Composer Studio 55
The Next Step: Developing Custom Code 56

CHAPTER 6 *Developing Host Code* 57

Dynamic Link Library 57
Sample Host Programs 58
DspComponent. 60
Host Example Programs for the SBC671x Baseboard. 61
Overview 61
MessageTest. 62
The Target Application 64
VcMessage Test 65

CHAPTER 7 *Creating Target Software* 69

C Code Development 69
C Compiler 69
C Library Reference 70
M6x Zuma Toolset Libraries 70
M6x Hardware Interaction 73
Digital Input/Output 74
Timers 76
Example Target Programs for the M6x 80
HELLO 80
TEST 80

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 8 *Target DSP Peripheral Libraries* **83**

CHAPTER 9 *Host DLL Reference* **91**

CHAPTER 10 *DOS Environment Requirements* **95**

CHAPTER 11 *M6x/cM6x Hardware* **97**

M6x/cM6x Hardware Functions **97**
Memory Map **98**
M6x Hardware Initialization Requirements **99**
External Memory **100**
M6x OMNIBUS **100**
M6x OMNIBUS Memory Mapping ***101***
OMNIBUS Power ***102***
FIFOPort I/O Expansion **103**
Transmitting and Receiving FIFOPort Data ***104***
Monitoring FIFO Status ***104***
FIFOPort Reset ***106***
FIFOPort Enable ***106***
Controlling the FIFOPort Programmable Almost-full Flag ***106***
Timer I/O and the FIFOPort ***107***
Designing External Hardware for use with the FIFOPort ***107***
FIFOPort Timing ***108***
Serial Ports **109**
Timers **110**
On-chip Timers ***110***
16-bit Timers ***111***
Using the External Timers as Digital Signal Counters ***112***
AD9850 Direct Digital Synthesizer ***114***
Digital I/O **114**
Digital I/O Timing ***115***
External Mux Control **116**
Interrupts **117**
JTAG Test Bus **121**
M6x PCI Bus Features **122**
PCI Bus I/O and Memory Map ***122***
M6x Bootstrapping ***123***
Multi-Card Timing Synchronization **124**

CHAPTER 12 *Appendices* ***127***

Board Layout **127**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector pinouts 129
JP17, JP18, JP21, JP22, P1, P2 - OMNIBUS I/O Connectors (M6x only) **129**
JP17, JP18, JP21, JP22, JP32, JP33 - OMNIBUS I/O Connectors (cM6x only) **130**
JP19, 20, 23, 24, 34, 35 - OMNIBUS Bus Connectors **131**
JP14 – Digital I/O Connector **133**
JP31 – Miscellaneous Digital I/O Connector **134**
JP15, JP16 – Processor Serial Port Connectors **134**
JP11 – JTAG Debugger Connector **135**
JP30 – FIFOPort Connector **136**
JP32 - SyncLink **137**
TMS320C6x01 Limitations and Errata 138
Processor Speed Limitations and External Memory **138**
Texas Instruments Device Errata **139**

CHAPTER 13 *Troubleshooting* **143**

Initialization Problems 143
The system does not recognize my board(s) **143**
Borland C++Builder Problems 144
The Builder Help Index system is blank **144**
I created an EXE file and when I try to run it, the system requires a DLL which I don't have. **144**
What DLLs do I have to deploy with my newly created executable? **145**
For Windows 9x, the system directory is the **C:\windows\system** *directory. For Windows NT, it is* **C:\Winnt\system32** *directory.* **145**
How do I know what DLLs my executable is dependent on? **145**
When I run my newly created application, the graphed data within my TIIGraph object stays red and doesn't graph. **145**
DSP Hardware Problems 146
The I/O seems like it is not connected or doesn't work **146**
How can I tell what version of logic I am using? **146**
ChicoPlus Hardware Problems 146
Updating the Logic without the VSProm Loader Applet **146**
The I/O seems like it is not connected or doesn't work **147**
The timer frequencies I can generate do not allow me to sample at the rates I need. **147**
How can I tell what version of logic I am using? **148**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

List of Tables

TABLE 1. Zuma Toolset Source Directories 70
TABLE 2. Zuma Toolset Support Subdirectories 71
TABLE 3. Texas Instruments Standard Library Functions 73
TABLE 4. M6x External Peripheral Memory Map. 74
TABLE 5. Digital I/O Access Memory Location 75
TABLE 6. Digital I/O Direction Configuration 75
TABLE 7. Digital I/O Latch Configuration 76
TABLE 8. Digital I/O Library Functions 76
TABLE 9. C Language Timer Functions 77
TABLE 10. STDIO Driver Functions 78
TABLE 11. Generic DLL Function List 91
TABLE 12. Required Disk Directory Structure for I.I. Development Tools 96
TABLE 13. M6x External Memory Map 99
TABLE 14. M6x Bus Control Register Initialization Values. 100
TABLE 15. M6x I/O Bus Memory Mapping 101
TABLE 16. I/O Bus Power Ratings 102
TABLE 17. Receive FIFOPort Level Status Register Definition 105
TABLE 18. Transmit FIFOPort Level Status Register Definition 105
TABLE 19. FIFOPort Timing Parameters 109
TABLE 20. External Timer Control Registers 111
TABLE 21. AD9850 Control Registers 114
TABLE 22. Digital I/O Control Registers 115
TABLE 23. Digital I/O Port Timing Parameters 116
TABLE 24. TERM Function Memory Map. 116
TABLE 25. External Interrupt Input Control Registers 117
TABLE 26. Interrupt Source 4 and 5 Select Register Values, M6x 118
TABLE 27. Interrupt Source 6 and 7 Select Register Values, M6x 118
TABLE 28. Interrupt Source 4 and 5 Select Register Values, cM6x 118
TABLE 29. Interrupt Source 6 and 7 Select Register Values, cM6x 119
TABLE 30. Interrupt Polarity Control Register Definition 121
TABLE 31. HPI Port PCI Bus Mapping 123
TABLE 32. SyncLink Signal Selection Register Bit Definitions 124
TABLE 33. SyncLink Destination Selection Register Bit Definitions 125
TABLE 34. OMNIBUS I/O Connector Pinouts (M6x) 129
TABLE 35. OMNIBUS I/O Connector Pinouts (cM6x) 130
TABLE 36. I/O Module Bus Connectors (cM6x) 132
TABLE 37. I/O Module Bus Connectors (cM6x) 133
TABLE 38. Digital I/O Connector 133
TABLE 39. Miscellaneous Digital I/O Connector 134

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 40. Processor Serial Port Connector 135
TABLE 41. JTAG Debugger Connector 135
TABLE 42. FIFOPort Connector 136
TABLE 43. SyncLink Connector 137
TABLE 44. 143
TABLE 45. Possible Timer Frequencies for Input = 49.152 MHz 148

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

List of Figures

FIGURE 1. Terminal Emulator "UniTerminal" Applet 26
FIGURE 2. UniTerminal File Menu 27
FIGURE 3. UniTerminal DSP Menu 28
FIGURE 4. UniTerminal Option Menu 29
FIGURE 5. UniTerminal Display Options Tab 29
FIGURE 6. UniTerminal DSP Options Tab 30
FIGURE 7. UniTerminal Sound Options Tab 31
FIGURE 8. UniTerminal Support Options Tab 32
FIGURE 9. UniTerminal Help Menu 32
FIGURE 10. The COFF Dump Utility 36
FIGURE 11. COFF Dump Utility Output 36
FIGURE 12. Creating a New Project in Code Composer Studio 41
FIGURE 13. Adding Files to a Code Composer Studio Project 41
FIGURE 14. Code Composer Studio Project Window 42
FIGURE 15. Code Composer Studio Compiler Build Options 43
FIGURE 16. Code Composer Studio Linker Build Options 44
FIGURE 17. Code Composer Studio Build Results Window 44
FIGURE 18. Code Composer Studio Compiler Build Options 45
FIGURE 19. Code Composer Studio Build Results Window 46
FIGURE 20. Code Composer Studio Compiler Build Options 47
FIGURE 21. Code Composer Studio Linker Build Options for Non-Bios Project 48
FIGURE 22. Code Composer Studio Build Results Window 48
FIGURE 23. Code Composer Studio Compiler Build Options 49
FIGURE 24. Code Composer Studio Assembler Build Options 49
FIGURE 25. Code Composer Studio Linker Build Options 50
FIGURE 26. Code Composer Studio Build Results Window 50
FIGURE 27. Code Composer Studio Compiler Build Options 51
FIGURE 28. Code Composer Studio Assembler Build Options 51
FIGURE 29. Code Composer Studio Linker Build Options 52
FIGURE 30. Code Composer Studio Build Results Window 52
FIGURE 31. The MessageTest Example 63
FIGURE 32. The VclMessage Example 66
FIGURE 33. M6x/cM6x Block Diagram 98
FIGURE 34. FIFOPort Block Diagram 103
FIGURE 35. Receive FIFOPort Level Status Register 104
FIGURE 36. Transmit FIFOPort Level Status Register 105
FIGURE 37. FIFOPort Daughterboard Mechanical Dimensions 108
FIGURE 38. FIFOPort Timing 108

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FIGURE 39. Serial Port Daughterboard Mechanical Dimensions 110
FIGURE 40. Digital I/O Port Timing 116
FIGURE 41. Interrupt Polarity Control Register 121
FIGURE 42. OMNIBUS I/O Connector Pin Configuration 131
FIGURE 43. SyncLink Connector Pin Orientation 137
FIGURE 44. Removing the FLASH 147

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 1

Introduction

This document describes the Zuma software development environment for Innovative Integration (I.I.) digital signal processor (DSP) cards. The environment comes complete with ANSI compliant C/C++ code Compiler, Assembler, Linker, Debugger, and Windows interface software and represents the most complete package available for DSP code creation for Texas Instruments DSP processors.

Each Developer's Package consists of four major features:

- TMS320-based DSP board.
- Texas Instruments Floating Point C Compiler/Assembler toolset.
- Code Composer JTAG-based hardware-assisted debugger.
- Zuma software toolset including:

  DSP Peripheral Library - supporting on-board peripherals and DSP functions, with full source code.

  Custom 32-bit Windows 9x/NT/2000 compatible dynamic link library (DLL) - which utilizes a custom, 32-bit, Ring 0/Kernel-mode device driver for host PC software application development.

  Host Support Applets - for automatic program download, terminal emulation, COFF file dumping and on-board flash programming.

  Sample Applications - showing Host PC as well as target DSP coding techniques.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

This manual discusses installation issues and includes full documentation on all Innovative Integration software tools (please see the accompanying manuals for specific information on the T.I. toolset or Code Composer Studio software packages). Installation is discussed first, followed by brief introductions to each of the software packages and instructions on their use. General software development issues are presented, and a tutorial on DSP software development, particularly as it relates to the integrated use of the software packages included in this kit, are also discussed. References are given for the peripheral libraries and host DLL packages in the Appendices.

A Note about this Manual

Certain typography conventions are used in this manual to indicate user operations, file types, etc., as follows:

- Windows application menu commands are identified and presented as pipe-delimited strings indicating the menu entries which are being discussed. For example, the Load Program menu item under the File menu in the Code Composer package would be named by the following string:

  File | Load Program

  Computer readable files and keyboard input/output are represented in Courier font, with user input in bold. For example, a program file will be referred to by name as

  ```
  C:\SBC6711\TALKER\TALKER.OUT
  ```

  while user input and commands look like

  ROM MYPROG.OUT

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 2

Installation Instructions

Thank you for purchasing from Innovative Integration! We appreciate your business. This chapter describes the software and hardware installation procedure for Windows 2K or XP. Development under other versions of Windows are not supported at this time, though drivers are provided to allow runtime operation under Win9x, 2K and XP. .

Do *NOT* install the hardware card into your system at this time. This will follow the software installation.

Host Hardware Requirements

The software development tools require an IBM or 100% compatible Pentium IV-class or higher machine for proper operation. An Intel-brand processor CPU is *strongly recommended*, since AMD and other "clone" processors are not compatible with the Intel MMX and SIMD instruction-set extensions which the developement software utilizes extensively to improve processing performance within a number of its components. The host system must have at least 128 Mbytes of memory (256MB recommended), 100 Mbytes available hard disk space, and a CDROM drive. Windows 2000 or XP (referred to herein simply as *Windows*) is required to run the developer's package software, and are the target operating systems for which host software development is supported.

Software Installation

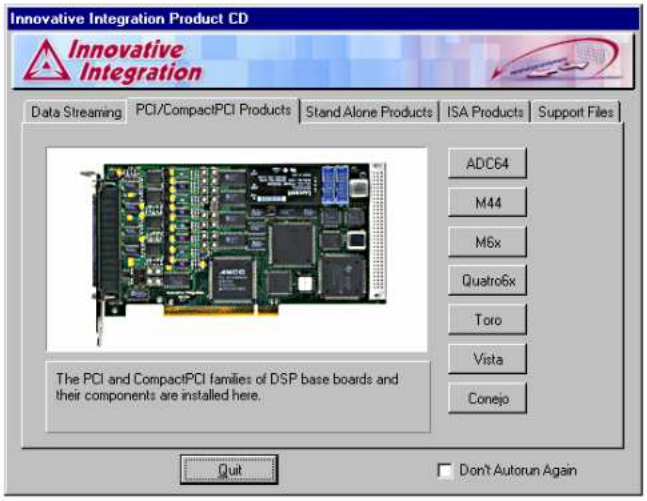
The development package has an installation program that will guide you through the installation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Note: Before installing the host development libraries (VCL components or MFC classes), you must have Microsoft MSVC and/or Borland C++ Builder installed on your machine, depending on which of these IDEs you plan to use for Host development. If you are planning on using one of these environments, it is imperative that these environments be tested and known-operational before proceeding with the library installation. Additionally, you must install Code Composer Studio prior to installation of the development package software if you have a DSP board. (This will not be necessary if you have purchased a streaming board such as ChicoPlus.) If these items are not yet installed, then the installation program will not permit installation of the associated development libraries. However, drivers and DLLs-only may be installed, to facilitate field deployment.

Win2K/XP Users Only. You must have Administrator Privileges to install and run the software/hardware onto your system, refer to the Windows documentation for details on how to get these privileges.

To begin the installation, start the host operating system and insert the installation CD. The installation should autostart after you put it into the CD. If the CD does not auto start, click on the **Start** button, then **Run**. Enter the path to the SETUP.EXE program located at the root of your CD-ROM drive, i.e. E:\SETUP.EXE. The setup program will run. Select the appropriate tab. From there, select the appro-



priate baseboard button, depending on the type of hardware being installed. The baseboard-specific install screen will automatically come up. You should see a screen similar to the following. **Note:** If you



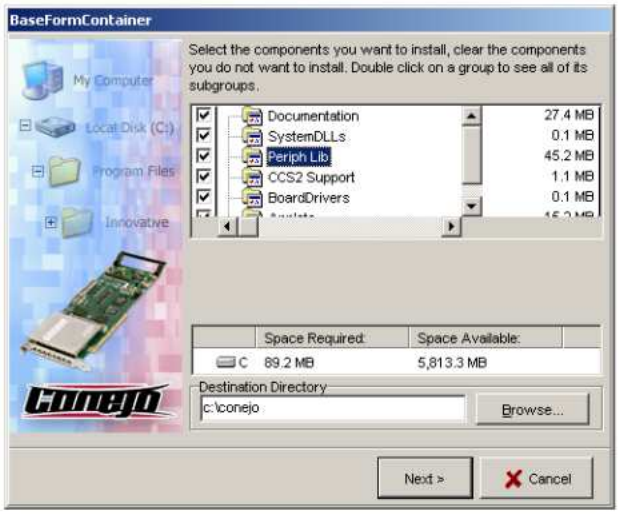
are attempting a full installation with development libraries for a DSP board and have not already installed Code Composer and your Host IDE (Borland Builder or MSVC), you should do so before pro-

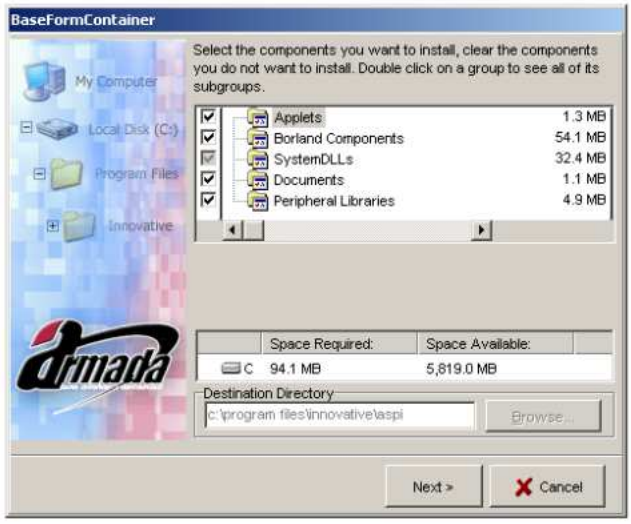
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ceeding with the installation and NOT click Install at the first selection window. In this case, click the Exit button to terminate. If you are deploying a completed application and need driver and DLL support files only to be installed, proceed. There will be a chance to check drivers-only and DLLs-only in each of the sub-installs.

In the example above, each of the check boxes have sub-installs associated with them, and will open a sub-install screen window with another set of check boxes allowing you to select files to be installed. For example, the first sub-install for "Conejo - (including Applets, Examples and Drivers" is shown below.

The second image below, is for "Armada - (including C++Builder or MS Visual C++ support)". It follows a similar pattern and presents another set of choices as illustrated.





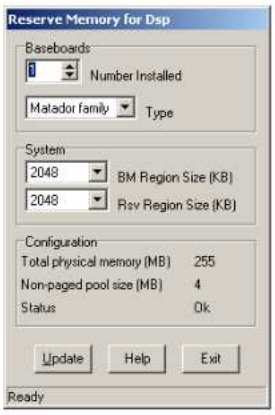
You may customize your installation, including or omitting components shown in these sub-install menus. At this point in each sub-install, check the desired components to be installed and click the 'Next>' button.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The host examples included in this installation are written in C++ using Borland Builder or Microsoft MSVC, and C on the target side using Code Composer Studio (C++ on some targets). They are provided to illustrate the various features of the board and how these features are utilized.

Bus Master Memory Reservation Applet. Near the conclusion of the installation process, **ReserveMem.exe** will run. This will allow you to set the memory size needed for the busmastering to occur properly. This applet may be run from the start menu later if you need to change the parameters.

For optimum preformace each Matador Family Baseboard requires 2 MB of memory to be reserved for its use. To reserve this memory, the registry must be updated using the ReserveMem applet. Simply select the **Number of Baseboards** you have on your system, click **Update** and the applet will update the registry for you. If at any time you change the number of boards in your system, then you must invoke this applet found in **Start | Programs | <target board> | ReserveMemDsp**.

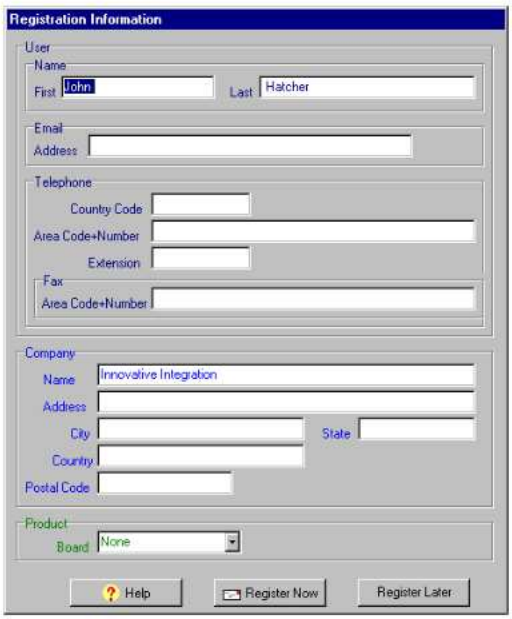


After updating the system exit the applet by clicking the **exit** button to resume the installation process.

Tools Registration

Before beginning DSP and Host software development, you must register your installtion with Innovative Integration. Technical support will not be provided until registration is successfully completed. Additionally, some of the development applets will not operate until unlocked with a passcode provided during the registration process. To Register, click Start | Program Files | Vista | New User to start the *NewUser.exe* registration application. The registration form below will be displayed:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

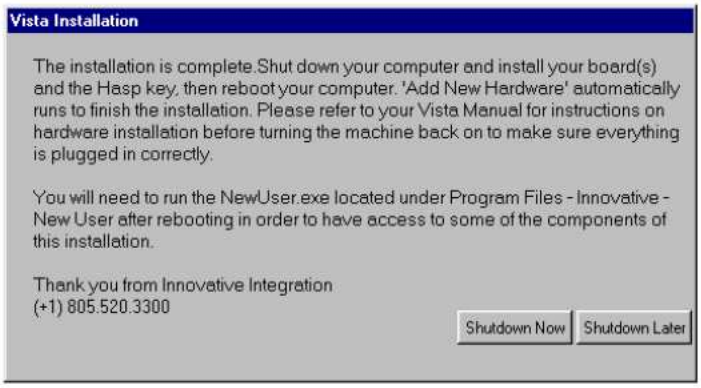


You should fill it out completely and return it to Innovative, either via email or fax. Upon receipt, Innovative will provide access codes to enable downloading of update software from our website, telephone hotline support and unrestricted applet access.

At this point you have finished the installation process and you are ready to begin development. Exit with the **OK** button.

Finally, the following screen appears. At this point, you should shut down your computer. This will also allow you to physically place the board(s) into the machine in addition to allowing newly installed components to be initialized

.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Hardware Installation

The software components of the Development Package have been installed. To proceed with the Development Package Kit installation, it will be necessary to configure and install your hardware.

JTAG Emulator Hardware Installation (for DSP boards Only, the JTAG is NOT needed with ChicoPlus)

First, the emulator hardware must be configured and installed into your PC. The emulator hardware is described in the table below:

Type	Features
Pod-based	Uses a special ribbon cable with integrated line drivers to connect the target DSP emulation signals to the JTAG debugger card. Usable on 3.3 volt or 5 volt designs. (Including 'C54x and 'C6x)

PCI Pod-Based Emulator Installation

To install the PCI pod based emulator, follow the instructions below:

1. Shut down Windows and power-off the host system.
2. Perform the board installation in an "ESD" or static safe workstation.
3. Power off the host system and **touch the chassis** of the host computer system to dissipate any static charge.
4. Remove the card from its protective static-safe shipping container, being careful to handle the card only by the edges.
5. Touch the chassis of the PC to dissipate any built up static charge.
6. Securely install the JTAG board in an available PCI slot in the host computer.
7. Connect the JTAG pod to the host-pod cable. Connect the host-pod cable to the connector located on the end bracket of the JTAG PCI plug-in board.

Baseboard Installation (for DSP boards AND ChicoPlus)

To install the baseboard:

1. Shut down Windows and power-off the host system.
2. Perform the board installation in an "ESD" or static safe workstation.
3. Power off the host system and **touch the chassis** of the host computer system to dissipate any static charge.
4. Remove the card from its protective static-safe shipping container, being careful to handle the card only by the edges.
5. Touch the chassis of the PC to dissipate any built up static charge.
6. Connect the 14-pin connector on the JTAG PCI pod to the DSP board JTAG connector **(ChicoPlus users skip this step.).**
7. Securely install the baseboard into an available PCI slot in the host computer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A Few Considerations BEFORE Power-up. Double-check everything before applying power. Are the JTAG (if needed) and baseboard cards seated correctly in the slot?

It can't be overemphasized: double check your cabling BEFORE connection to the baseboard. Also, don't cut corners and hot plug the cables. This can cause latch-up of components on the card and damage them irreparably. Be aware that the cables to analog inputs are an important part of keeping the signals clean and noise-free. Shielded cables and differential inputs (where applicable) help to control and reduce the noise.

After Power-up. After completing the installation, boot your PC into Windows.

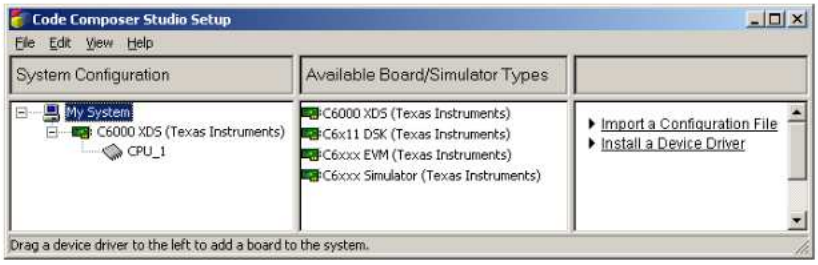
The target cards are plug and play compliant, which Windows should detect and auto-configure at start-up. Under rare circumstances, Windows will fail to auto-install the device-drivers for the JTAG (if needed) and baseboards. If this happens, please refer to the "TroubleShooting" section.

The next section is NOT used with the Chico Family products. If your board is a Chico Family product, you are DONE.

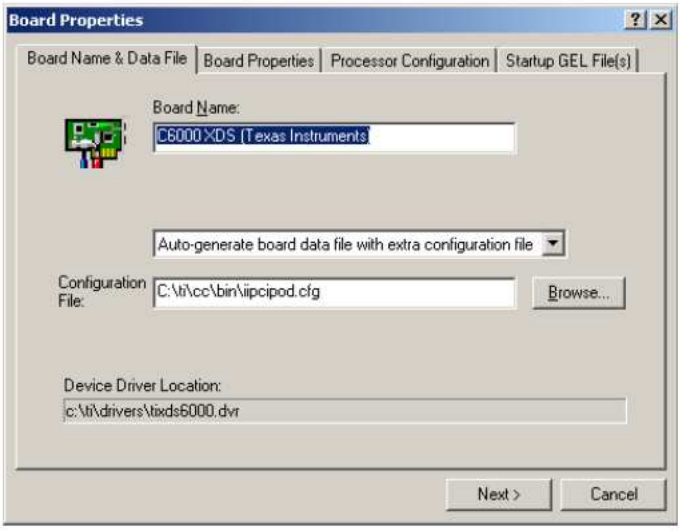
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Code Composer Studio Setup

To setup Code Composer Studio and activate the Innovative-supplied DSP board driver, the Code Composer Setup Utility must be run. Code Composer Studio setup must be configured to use the XDS510 driver for the C6000 after removing the default simulator driver from the System Configuration. This driver is named C6000 XDS within the Code Composer setup utility. In the figure below, it is the first item listed in the Available Board/Simulator Types column of the Setup program.



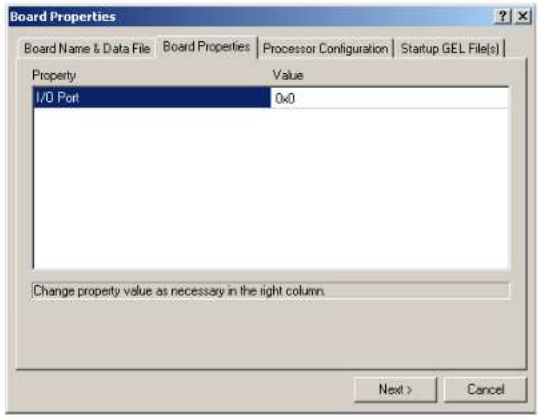
Click this driver from the “Available Board/Simulator Types” control within the setup utility and drag it into the “System Configuration” control. Then, right-click on this C6000 XDS object to invoke the Properties Dialog for the driver.



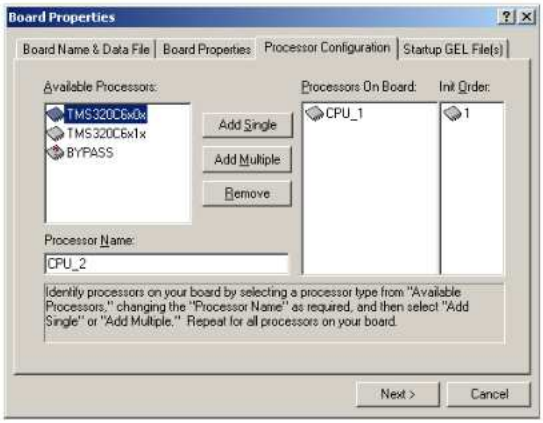
Under the “Board Name & Data File” tab, the board name edit box should list "C6000 XDS (Texas Instruments)". The “Configuration File” combo box should be changed to "Auto-generate board data file with extra configuration file". The “Configuration File” edit box should be changed to "<drive>:\ti\cc\bin\IIPciPod.cfg", where <drive> is the letter for the drive onto which CCS 2.0 was installed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

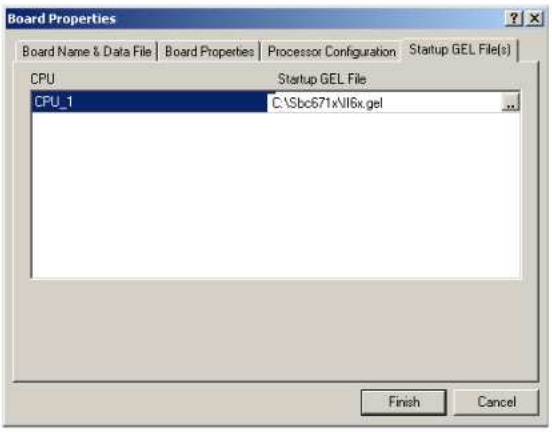
Under the "Board Properties" tab, the I/O port value for the driver should be set to virtual device address "0".



Under the "Processor Configuration" tab, one or more processors of the appropriate type (C6x0x or C6x1x) should be added to the "Processors on the Board" list box.



Under the Startup Gel Files tab, the Startup Gel File combo box should be the board-specific initialization GEL script for the target board. Innovative supplies an appropriate, board-specific GEL initialization file for each product located in the root Zuma toolset directory. For example, for the Vista, this should be set to c:\Vista\II6x.gel.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Finally, the setup configuration should be saved. After saving the configuration and shutting down the setup tool, Code Composer Studio should launch successfully. If you encounter difficulty launching CCS 2.0, Run the JtagDiag.exe utility provided in your Zuma toolset to reset the debugger interface. Then, run the MatadorDownload.exe utility and press Boot, to boot a default target application. Then restart Code Composer Studio.

As a consequence of these steps, a new ccBrd0.dat file will be created. This file has been customized for the particular target being used.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 3

Integrated Development Environment

The C Developer's Package consists of several software tools, integrated to work together to provide a complete DSP design environment for Innovative Integration DSP boards. This section discusses the tools included in the development package and gives descriptions of each applets features and use. A brief introduction is given regarding the software programs provided and their use within the Developer's Package. The user is referred to the individual manuals accompanying these software products for complete documentation.

The Texas Instruments C Compiler Toolset

The compiler supplied with the Developer's Package is the Texas Instruments (T.I.) Floating Point C/C++ Compiler toolset for the DSP target board. The compiler runs under Windows as a cross compiler, generating executable applications for the DSP processor which are then downloaded and executed using the other tools in the Developer's Package. The compiler is ANSI C compatible and supports nearly all standard C functions. Additional libraries provided with the Developer's System include C standard I/O and peripheral drivers for the A/D, D/A, bit-I/O and timers. Assembly language may also be mixed with C code for higher performance where required.

Typical application programs will consist of one or more C (`.C`), header (`.H`), and Assembly language (`.ASM`) source files, as needed. Additionally, target program generation requires use of a linker command file (`.CMD`) which specifies the memory map for the target and optionally includes commands defining the libraries to be linked into the final application.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Users of the Code Composer Studio editor/debugger will also employ make (`.MAK`), workspace (`.WSP`) and special Code Composer-specific script files (`.GEL`). The example programs included in the Developer's Package illustrate the use of these files also and give example files to use as a basis for custom DSP applications.

C Compiler Toolset Usage

The C compiler may be run directly from a DOS Prompt window under Windows as described in the TI toolset documentation. Also included in the installation directory are make files useful for manually rebuilding applications programs within the DOS environment. To rebuild an example or library from the command line, execute **`gmake -f <filename.mak>`**.

Code Composer Studio

Code Composer Studio is a flexible, high-performance, integrated code generation environment developed by Texas Instruments and bundled into the Innovative Integration Zuma toolset. For complete documentation to the features of Code Composer Studio package, refer to the accompanying *Code Composer Studio Manual* provided. If the user wishes to compile outside of Code Composer Studio (or has not purchased the package), these make files may be used from the DOS command line to rebuild individual project files or the entire target file set.

Editor

Code Composer Studio supports code editing and emulates the most popular editing packages (CUA, etc.). Code Composer Studio is also a Window editor. Custom DSP code development can take place entirely within the Code Composer Studio environment using its project management tools to place source files, libraries and linker command files into projects (`.MAK`) in order to build executables. The example programs included in the Developer's Package each have a Code Composer project file (`.MAK`) associated with them which may be used to re-compile the example.

Debugger

Code Composer Studio is a software program for high-level TI C and Assembly Language debugging which supports high-performance, JTAG or MPSD-based hardware assisted debugging directly on the target DSP. This allows the user access to the internal register set, peripherals, and bus of the target board in order to load, run, and debug applications. Also integrated into the Code Composer Studio software package is a code management subsystem for editing files as well as creating and compiling DSP projects.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 4

Support Applets

The Developer's Package includes a number of support applications supporting general DSP development: the terminal emulator (`UNITERMINAL.EXE`), the COFF file downloader (`UNIDOWNLOAD.EXE`), the COFF file display utility (`COFFDUMP.EXE`), the PromImage utility (PROMIMAGE.EXE) and the FLASH prom programming facility (`BURN.EXE`). This section describes the functionality of each of the applications and their use within the development system.

The functions provided by each of the applications may be configured through menu selections available within each of the applets themselves. See the discussion below for applet-specific parameters.

The Terminal Emulator "UniTerminal"

The terminal emulator "UniTerminal" provides a C language-compatible, standard I/O terminal emulation facility for interacting with the `stdio` library running on an Innovative Integration target DSP processor. Display I/O calls such as `printf()`, `scanf()`, and `getchar()` are routed between the DSP target and the Host UniTerminal applet where ASCII output data is presented to the user via a terminal emulation window and host keyboard input data is transmitted back to the DSP. The terminal emulator works almost identically to console-mode terminals common in DOS and Unix systems, and provides an excellent means of accessing target program data or providing a simple user interface to control target application operation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

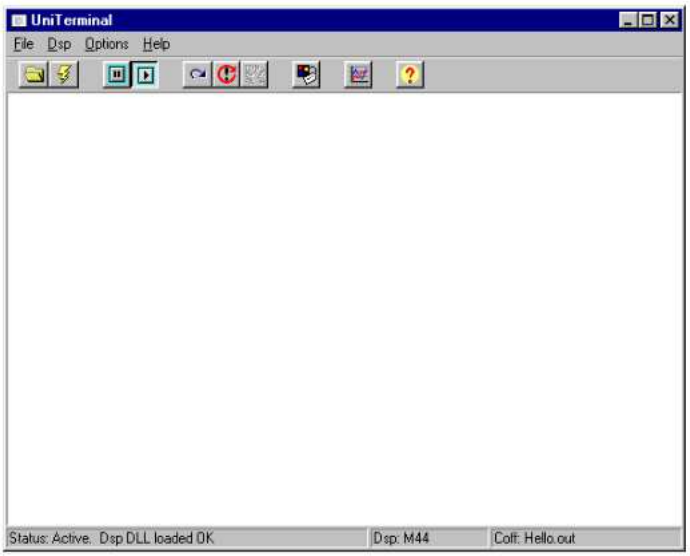


FIGURE 1. Terminal Emulator "UniTerminal" Applet

The terminal emulator is straightforward to use. The terminal emulator will respond to stdio calls automatically from the target DSP card and should be running before the DSP application is executed in order for the program run to proceed normally. The DSP program execution will be halted automatically at the first stdio library call if the terminal emulator is not executing when the DSP application is run, since standard I/O uses hardware handshaking, except on RS232 bus-based targets. The stdio output is automatically printed to the current cursor location (with wraparound and scrolling), and console keyboard input will also be displayed as it is echoed back from the target.

The terminal emulator also supports Windows file I/O using the library routines `fopen()`, `fclose()`, `fread()`, `fwrite()`, `fseek()`, and `fflush()`. Refer to the `Zuma.hlp` file for the types and usage of these library functions as their usage is not 100% ANSI compliant.

Important Note: Before using UniTerminal, you must register your Zuma Toolset. Until you do so, usage will be restricted to a 20-day trial period for UniTerminal and other applets contained in the Zuma Toolset. To register, fill out the contents of the Registration Form, then click on the Register Now button. This will print a Registration report which, must be faxed to Innovative Integration. Innovative Integration will E-mail you an Access Code, which must be typed into the Registration Form for all the Zuma features to be enabled.

Terminal Emulator Menu Commands. The terminal emulator provides several menus of commands for controlling and customizing its functionality. These functions are available on the menu bar, located at the top of the UniTerminal main window. Speed button equivalents for each of the menu options are also available on the button bar located immediately beneath the menu bar. The following is a description of each menu entry available in the terminal emulator, and its effects.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

File Menu:

.

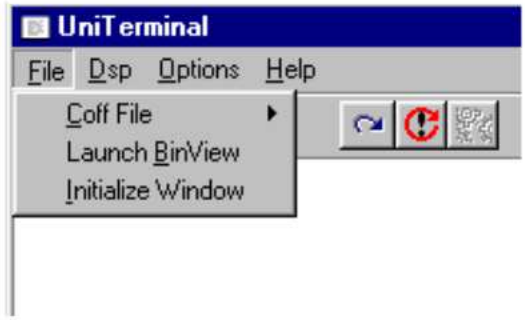


FIGURE 2. UniTerminal File Menu

- File | COFF File | Download - provides for COFF (Common Object File Format) program downloads from within the terminal emulator. When selected, a file requester dialog box is opened and the full pathname to the COFF filename to be downloaded is selected by the user. Clicking "Open" in the file requester once a filename has been selected will cause the requester to close and the file to be downloaded to the target and executed. Clicking "Cancel" will abort the file selection and close the requester with no download taking place.

Quatro6x Users: Terminal supports downloading of either Coff files (`.OUT`) or multi-processor out (`.MPO`) files. The (`.MPO`) files provide a means of downloading separate (`.OUT`) files to multiple processors simultaneously, which greatly simplifies the task of synchronizing execution in a multi-processor environment.

- File | COFF File | Reload - Reloads and executes the COFF file last downloaded to the target. It provides a fast means to re-execute the application program most recently loaded into the target board.

NOTE: File | COFF File | Download and File | COFF File | Reload functions physically resets the target DSP (in order to initiate the target Talker program) prior to the download. When using the terminal emulator in conjunction with the Code Composer debugger, use Code Composers File | Load Program facility to download executable code to the target rather than the terminal emulator's download facility. Since the Code Composer mechanism does not physically reset the target during the download, it is not reliant on the target Talker to perform the download.

If you attempt to download using the COFF Download menu within the terminal emulator while using Code Composer, you may receive the diagnostic dialog box, which indicates that Code Composer has *halted* the target processor via the JTAG hardware link. While in this halted state, the terminal emulator cannot invoke the Talker program on the target DSP in order to perform the software download. To correct this problem, execute the Debug | Run Free menu command from within Code Composer to release the DSP from JTAG control. Afterwards, clear the terminal emulator error message dialogs and retry the terminal emulator COFF Download.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- File | Launch Binview – launches Binview, so that you may use it to examine the contents of a binary windows data file.

- File | Initialize Window - Resets the application window. Use this feature to reset the application if emulation ceases and will not resume, as an alternative to restarting the application. Emulation can be disrupted by outside programs, such as the JTAG-based Code Composer debugger.

DSP Menu:

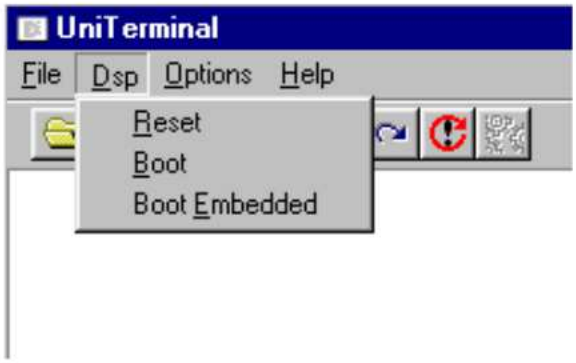


FIGURE 3. UniTerminal DSP Menu

- Dsp | Reset - causes the terminal emulator to assert, then de-assert the target's physical reset pin, bringing the target board into a cold-start, uninitialized condition. This function is also available via the speed button toolbar. This is a special utility function which is not needed in normal operation.

- Dsp | Boot - performs the Dsp | Reset function, then boots the Talker program on the target. On plug-in cards, the Talker program is downloaded into target program memory, the target is reset (launching the Talker program), then UniTerminal polls for the Talker sign-on command via the stdio mailbox. On stand-alone, single-board targets, the target is reset which launches Talker from onboard Flash memory, then UniTerminal polls for the Talker sign-on command via the stdio mailbox. This function is also available via the speed button toolbar. This is a special utility function which is not needed in normal operation.

- Dsp | Boot Embedded - performs the Dsp | Reset function, simultaneously driving the CTS line inactive which causes single board targets to launch the application code embedded into sector one of their onboard Flash memory. This function is also available via the speed button toolbar. This is a special utility function which is only useful for single-board DSP users when testing embedded application programs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Option Menu:

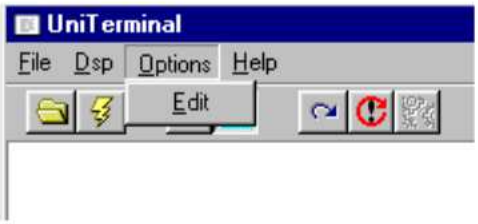


FIGURE 4. UniTerminal Option Menu

- Options | Edit - invokes the user options dialog. The four, tabbed pages on the dialog box allow customization of display settings, dsp type, use of sound and specification of the location of support applets, such as BinView. This function is also available via the speed button toolbar.

Display Options Tab:

The display options tab on the Options dialog (seen below) contains controls to allow user-customization of the appearance of UniTerminal.

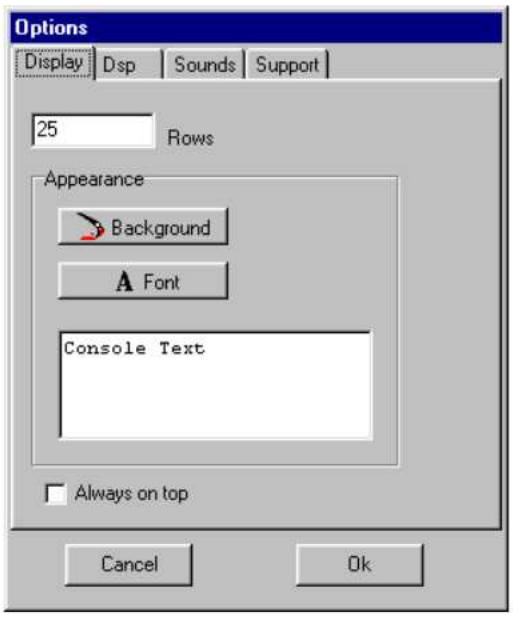


FIGURE 5. UniTerminal Display Options Tab

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Rows edit box controls the number of rows of text available within the terminal emulator window.

The Appearance group box contains separate buttons which are used to adjust the background color and font of the text within the terminal emulation window.

The Always On Top check box controls whether UniTerminal is forced to remain a foreground application, even when it loses keyboard focus. This is useful when running stdio-based code from within the Code Composer environment, when it's preferable to make terminal visible at all times. The terminal will remain atop other windows when this entry is checked. Select the entry again to uncheck and allow the terminal emulator window to be obscured by other windows.

Display Options Tab:

The Dsp tab on the Options dialog (seen below) contains controls to allow selection of the type and number of the target board being serviced by this instance of UniTerminal.



FIGURE 6. UniTerminal DSP Options Tab

The Type combo box specifies the type of target board being serviced by this instance of UniTerminal.

The Target combo box specifies the number of the target board being serviced by this instance of UniTerminal.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Boot On Startup check box controls whether the target processor is booted when UniTerminal is initially invoked. Disable this check box when using UniTerminal in conjunction with Code Composer, to avoid resetting the target and corrupting the JTAG state upon UniTerminal invokation.

The Init JTAG on Boot check box controls whether the JTAG is reset when the DSP board is booted. If this box is checked, the JTAG reset will disconnect any prior logical connections upon running UniTerminal. Disable this check box when using any debugger other then Innovation Intergration's. This is it to prevent any communication problem between the target board and debugger.

Sound Options Tab:

The Sounds tab on the Options dialog contains controls to specify whether UniTerminal provides audible indicators during its operation.

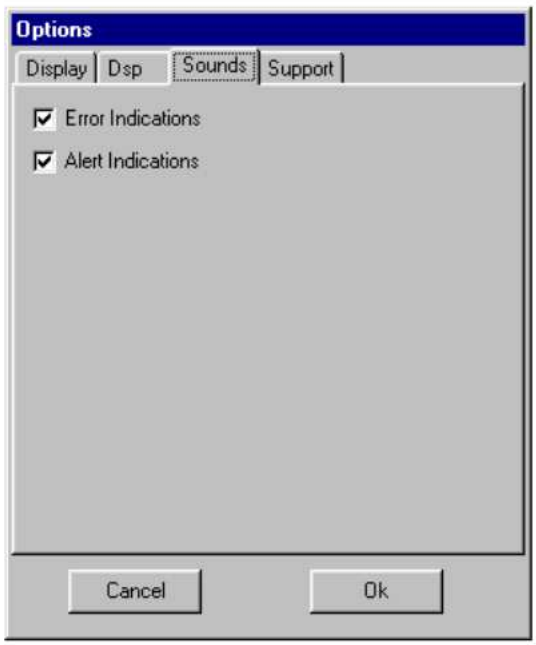


FIGURE 7. UniTerminal Sound Options Tab

The Error Indications check box controls whether sounds are played upon receipt of errors during emulation.

The Alert Indications check box controls whether sounds are played upon receipt of errors during emulation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Support Options Tab:

The Support tab on the Options dialog contains controls to allow specification of the location of companion support applets used by UniTerminal.



FIGURE 8. UniTerminal Support Options Tab

The Binview Button is used to browse to the location of the BinView binary data viewer applet. UniTerminal communicates directly (using TCP/IP, which must be installed) with the BinView applet when target applications attempt to display graphical data files.

Help Menu:

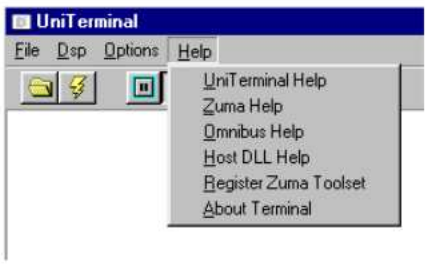


FIGURE 9. UniTerminal Help Menu

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Help menu provides convenient access to the technical resources/Help Files provided in the Zuma Toolset.

- Help | Register Zuma Toolset - invokes a dialog box to be filled out and sent to Innovative Integration

- Help | About Terminal - invokes a dialog displaying copyright and version information about UniTerminal, plus information pertaining to the use of Host resources by the target DSP board.Use this information when contacting Innovative Integration for technical support regarding UniTerminal.

Terminal Emulator Command Line Switches. The terminal emulator also provides the following command line switches to further modify program behavior. The switches must be supplied via the command line or within Windows shortcut properties (see the Installation section for more information), and will override the default behavior of the applet.

Multiple instances of UniTerminal may be invoked simultaneously in order to support installations utilizing multiple target boards. Instances of UniTerminal, after the first loaded instance must be configured via command line switches in order to properly communicate with their associated target.

- -**b***boardtype* - Use the -b switch to force an instance of UniTerminal to communicate with a specific type of target board, boardtype. Supported boardtypes are listed below:

Board Type	Supported Product(s)	Communication Path
None	No DSP selected	N/A
ADC64	ADC64 DSP Board	PCI Bus
M44	M44 DSP Board	PCI Bus
PC31	PC31 DSP Board	ISA Bus
PC32	PC32 DSP Board	ISA Bus
PC44	PC44 DSP Board, up to two installed TIM processors	PCI Bus
PCI32	PCI32 DSP Board	PCI Bus
PCI44	PCI44 DSP Board, up to three installed TIM processors	PCI Bus
M6x	M62/M67 DSP Board	PCI Bus
Q6x	Q62/Q67 DSP Board with four C6x processors	PCI Bus
SBC31	SBC31 stand-alone DSP Board via serial port	COM1: or COM2:
SBC32	SBC32 stand-alone DSP Board via serial port	COM1: or COM2:
SBC54	SBC54 stand-alone DSP Board via serial port	COM1: or COM2:
SBC6x	SBC62/SBC67 stand-alone DSP Board via serial port	COM1: or COM2:
SBC6711	SBC6711 stand-alone DSP Board	USB

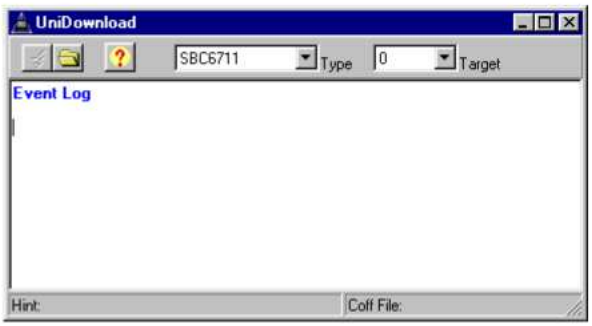
- -**t***X* - Use the -t to allow the user to force the UniTerminal to interact with a specified target number. This switch is particularly useful in multi-board installations to create instances of the emulator for targets other than target 0. See the Installation section for more information on multi-board installations. The *X* parameter specifies the logical target number with which to communicate. **NOTE:** For single-board targets, specify target 0 for boards connected via COM1 and target 1 for boards connected via COM2.

- *"filespec."* - Use the *"filespec."* field, which includes both the path and file name as a unit, to allow the user to force the terminal emulator to download the specified file to the target DSP board, as soon as the terminal emulator is loaded. This field is particularly useful in situations where the UniTerminal is "shelled to" from within an other Host applications to facilitate the automatic execution of target applications employing standard I/O.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The COFF File UniDownloader

The UniDownload application provides a means of downloading debugged programs to a target DSP. The utility is useful in situations where a host application program capable of performing the DSP application download operation is unavailable or unnecessary.



Using the Application

The application may be operated through the use of buttons on the toolbar located at the top of the applet window, or through the use of command line switches. The latter technique is more common, since this allows the applet to be invoked within the Windows startup folder to cause DSP application execution to commence when Windows boots.

Command Line Switches

The following command line switches are supported. Each switch consumes an in line argument, which must be present for proper operation.

Switch Argument

-**b** name

-**f** "filename"

-**t** number

The -**b** switch is used to specify the name of the target DSP to which code is to be downloaded. The following DSP board names are acceptable: None, ADC64, M44, PC31, PC32, PC44, PCI32, PCI44, SBC31, SBC32, SBC54, M6x, Q6x, SBC6x, SBC6711 Case is significant and the name must be spelled exactly as shown. **Note:** The command line must be formatted such that a space character is present between the switch and its argument

The -**f** switch is used to specify the name of the COFF executable image to be downloaded to the target. The filename parameter should be a full unambiguous pathspec to be image file.

The -**t** switch is used to specify the target instance number, 0..15. In a single target system, zero should be supplied.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Manual Operation

The applet may be operated manually using the buttons located at the top of the applet window. The operation of each of the buttons is described below.



The file open button is used to browse to and subsequently download a COFF image file located somewhere on the local machine.



The reload button is used to re-download the last COFF executable image which was successfully downloaded.



The help button is used to invoke the help documentation associated to this application.



The board type combo box is used to change the type of DSP target to which code is to be downloaded. The following DSP types are supported: None, ADC64, M44, PC31, PC32, PC44, PCI32, PCI44, SBC31, SBC32, SBC54, M6x, Q6x, SBC6x, SBC6711.



The target number combo box is used to change he instance of the DSP target to which code is to be downloaded.



The event log is used to display the history of the status of manual operations which have been performed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The COFF File Dump Utility

The COFF downloader utility provides users with the capability to generate a report detailing the memory usage of target DSP programs generated using the TI tool set.

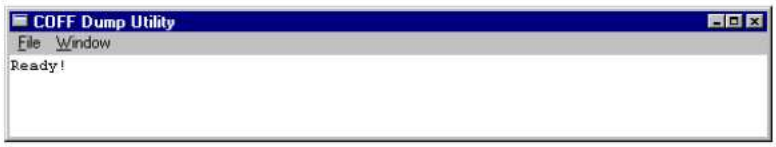


FIGURE 10. The COFF Dump Utility

COFFDUMP.EXE parses through COFF files stored in files on the hard disk and ascertains the complete memory consumption by the DSP program. Memory usage for each of the sections defined in the applications command file are tabularized and the results are written to the Windows NotePad scratch buffer. If desired, Notepad can then be used to write the data to disk or to a printer.

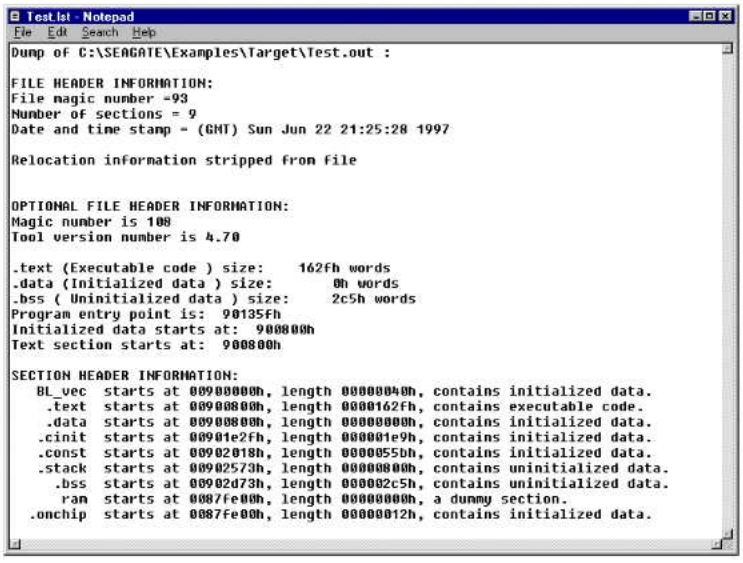


FIGURE 11. COFF Dump Utility Output

COFF Dump Utility Menu Commands. The following is a brief description of commands available from the COFF Downloader menus:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

File Menu

- File | Dump – Involves the standard Windows file selector window for COFF output files (`.OUT`). Parses through selected file and writes diagnostic dump of contents of executable image to NotePad scratch buffer.

- File | Exit - exits the dump utility program.

Window Menu

- Window | About - presents program copyright and version information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 5

Developing Target Code

Introduction

The Innovative Integration (I.I.) Zuma Toolset allows users of I.I. DSP processor boards to develop complete executable applications suitable for use on the target platform. The environment suite consists of the TI Optimizing C Compiler, Assembler and Linker, the Code Composer debugger and code authoring environment as well as I.I.'s custom Windows applets (such as the `UNITERMINAL.EXE` terminal emulator).

Code Composer Studio is the default package used to automate executable build operations within Innovatives Zuma Toolsets, simplifying the edit-compile-test cycle. Source is edited, compiled, and built within Code Composer Studio, then downloaded to the target and tested within either the Code Composer Studio debugger or via the Zuma terminal emulator.

On C6x platforms, such as Innovatives M6x, SBC6x, SBC6711, and Quatro6x, Code Composer Studio may be used for both code authoring and code debugging. Details of constructing projects for use on Innovative DSP platforms using Code Composer Studio are provided in this chapter.

Do not confuse the creation of target applications (code running on the target DSP processor) with the creation of host applications (code running on the host platform). The TI tools generate code for the TI DSP processors, and are a separate toolset from that needed to create applications for the host platform (which would consist of some native compiler for the host processor, such as Microsoft's Visual C++ or Borland Builder C++ for IBM compatibles). To create a completely turn-

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

key application with custom target and host software, *two* programs must be written for *two* separate compilers. While I.I. supports the use of Microsoft C/C++ for generation of host applications under Windows with sample applications and libraries, we do not supply the host tools as part of the Development Environment. For more information on creating host applications, see the section in this manual on host code development.

This section supplies information on the use of the development environment in creating custom or semicustom target DSP software. It is not intended as a primer on the C language. For information on C language basics, consult one of the C primer books available at your local bookstore. The definitive reference to the C language is The C Programming Language, by B. Kernighan and D. Ritchie (Prentice Hall. Englewood Cliffs, NJ. 1988).

Components of Target Code (.c, .asm, .cmd)

In general, DSP applications written in TI C require at least two files: a `.c` file (or "source" file) containing the C source code for the application, and a `.cmd` file (or "linker command" file) which contains the target-specific build data needed by the linker. There may also be one or more `.asm` assembler source files, if the user has coded any portions of the application in assembly language.

Edit-Compile-Test Cycle using Code Composer Studio

Nearly every computer programming effort can be broken down into a three step cycle commonly known as the edit-compile-test cycle. Each iteration of the cycle involves editing the source (either to create the original code or modify existing code), followed by compiling (which compiles the source and creates, or builds, the executable object file), and finally downloading and testing the result to see if it functions in the desired fashion. In the Innovative Intergration development system these stages are accomplished within the Code Composer program.

By using Code Composer Studio, these stages of the programming cycle are accomplished entirely within the integrated Studio environment. The project features of Code Composer Studio support component file editing and compilation stages, along with allowing the executable result to be downloaded and tested on the target hardware. This fully integrated programmers environment is more user-friendly then the basic command line interface, which comes standard with the TI tools.

A Simple Code Composer Studio Project

The following sequence illustrates the creation of a project to build the "`Hello World!`" program from within Code Composer Studio.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

First, start Code Composer Studio. Select `Project | New` from the Project menu and you will see the following dialog:

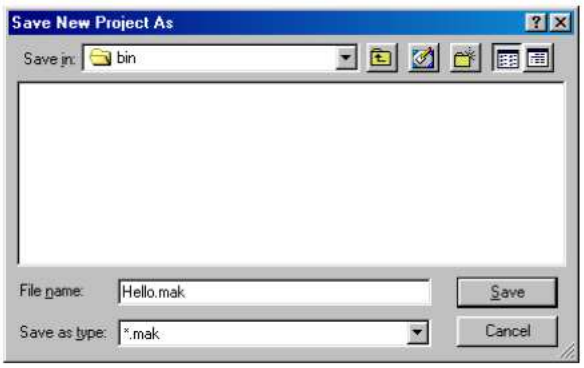


FIGURE 12. Creating a New Project in Code Composer Studio

Browse to the directory in which you would like to create the new project (your working directory) and then type the name of the new project. In this example, the working directory is c:\ti\bin and the project name is hello.mak. In the standard developers package, you may browse into the `<I.I. Target Board>\EXAMPLES\TARGET` directory.

Next, open the `Project | Add Files` to Project dialog box to add files to the project. Add the `HELLO.C file from the C:\<I.I. Target Board>\Examples\Target` directory and the `GENERIC.CMD` file from the `C:<I.I. Target Board>` directory to the project. Remember to choose the correct file type then when you have selected the file to add, click **<Open>** button.



FIGURE 13. Adding Files to a Code Composer Studio Project

SBC6x Only: When building an application to be burned into the Flash ROM, the `EMBED.CMD` file (located `C:\<I.I. Target Board>\Examples\Target)` should be added to the project instead of `GENERIC.CMD` (refer to Chapter 7 "Building Flash Programmable Application").

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

It is imperative that you add an appropriate command file to the Code Composer Studio project. The `generic.cmd` command file describes the memory map of the target hardware, without which the linker will be unable to place executable sections into appropriate memory regions for debugging. That is, the memory map for the target DSP specified in the `generic.cmd` file will be used to link the project output file. If you wish, you may copy the contents of the `generic.cmd` file (located in the root of the Zuma toolset) into your working directory, rename it appropriately and add the modified cmd file to your project instead.

The library files will be required, but do not add them directly into the project like the hello.c and generic.cmd files. Rather, manually type the desired libraries needed to link the project into the Project | Options | Linker tab when instructed to do so later within this chapter.

Next, you may optionally open the files in the project by double-clicking on their names within the Project window.

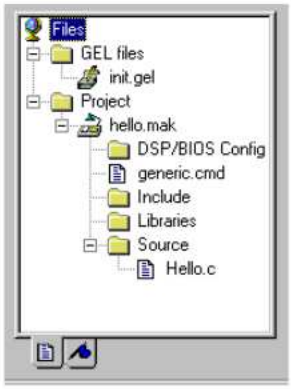


FIGURE 14. Code Composer Studio Project Window

Next, you must configure the project compiler settings so that when Hello.c is compiled, the appropriate memory model and switches are used.

Build Options (M62, Q62, SBC62 Boards)

Click on `Project | Build Options` to open the Build Options dialog box, then click on the Compiler tab to show the current compiler options. The required Build options are described in the following text, all other settings are optional and are described in the Code Composer Studio documentation.

Configure the compiler options to use the following settings: In the category column, choose Basic and click on the Generate Debug Info combination box and select "Full Symbolic Debug". Then click on the Opt Level combination box and select "-02: Function". Next, in the category column, choose Advanced, click on the Endianness combination box and select "Big Endian". Then click on the Mem-

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ory Models combination box and select “Far Calls & Aggregate Data”. In the category column, choose Feedback, and click on the Banners combination box and select “No Banner”. In the category column, choose Files and in the Obj Directory box type “C:\M6x\Examples\Target”. Again in the category column, choose Preprocessor and in the Include Search Path box type C:\M6x\Include\Target”. When finished, the compiler dialog screen should look very similar like one below.

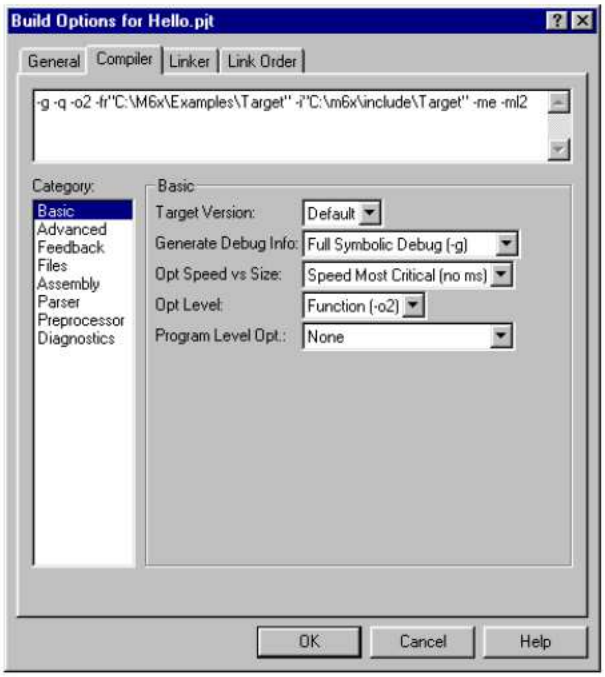


FIGURE 15. Code Composer Studio Compiler Build Options

Now, click on the Linker tab and configure the linker build options. In the category column, choose Basic. In this screen, make sure the Exhaustively Read Libraries has been selected. Then in the Output Filename box and Map Filename box type the project name. Click on the Autoint Model combination box and select “Run-time Autoinitialization”. Next, set the Heap Size to 0x400 bytes and the Stack Size to 0x800 bytes, the Heap & Stack size may change depending on your application’s needs. In the Library Search Path, type “..\..\..\lib\Target. Now, add stdio.lib; periph.lib; dsp.lib; and rts6201e.lib into the Include Libraries edit box (in that order). Then on the Include Libraries edit box line, add either init.lib for non-DSB Bios applications or biosinit.lib for DSP Bios applications, between the periph.lib and dsp.lib entries. When finished, the linker dialog screen should look like one below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

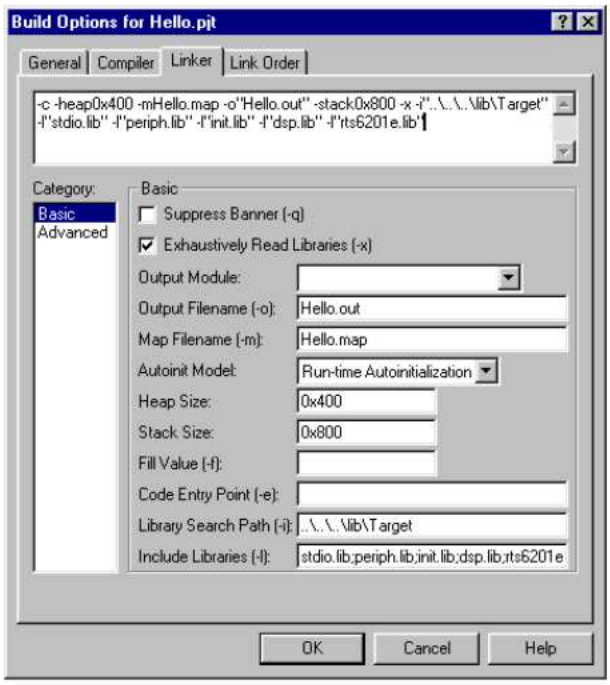


FIGURE 16. Code Composer Studio Linker Build Options

Once all the build options have been set, rebuild your project by clicking Project|Rebuild All in the Code Composer Studio menu bar. If errors are encountered in one or more source files, they are listed in the output window. You may visit and repair each error by either double clicking on each error in the Output window.

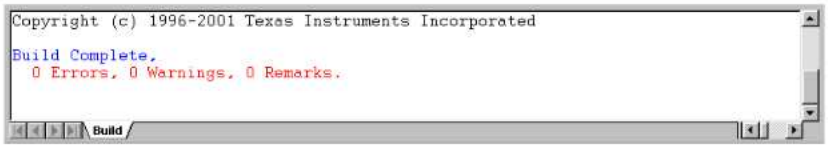


FIGURE 17. Code Composer Studio Build Results Window

Build Options (M67, Q67, SBC67 Boards)

Click on `Project | Options` to open the Build Options dialog box, then click on the Compiler tab to show the current compiler options. The required Build options are described in the following text, all other settings are optional and are described in the Code Composer Studio documentation.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Configure the compiler options to use the following settings: In the category column, choose Basic and click on the Target Version combination box, and select "670x". Click on the Generate Debug Info combination box and select "Full Symbolic Debug". Then click on the Opt Level combination box and select "-02: Function". Next, in the category column, choose Advanced, click on the Endianness combination box and select "Big Endian". Then click on the Memory Models combination box and select "Far Calls & Aggregate Data". In the category column, choose Feedback, and click on the Banners combination box and select "No Banner". In the category column, choose Files and in the Obj Directory box type "C:\M6x\Examples\Target". Again in the category column, choose Preprocessor and in the Include Search Path box type C:\M6x\Include\Target". When finished, the compiler dialog screen should look very similar like one below.

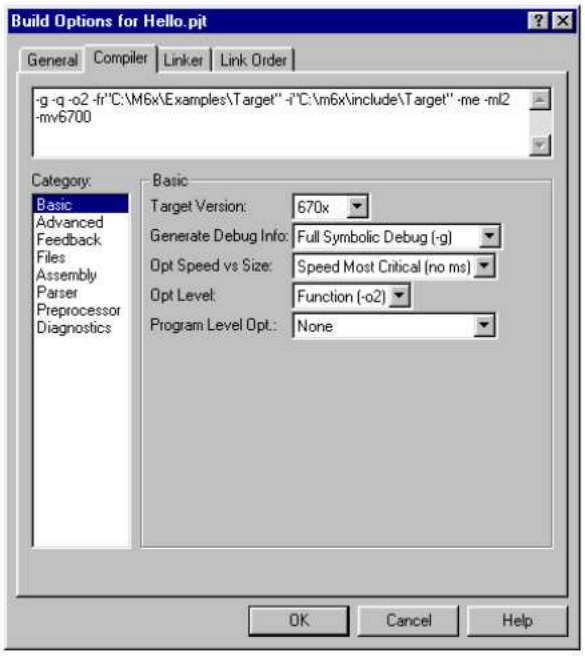
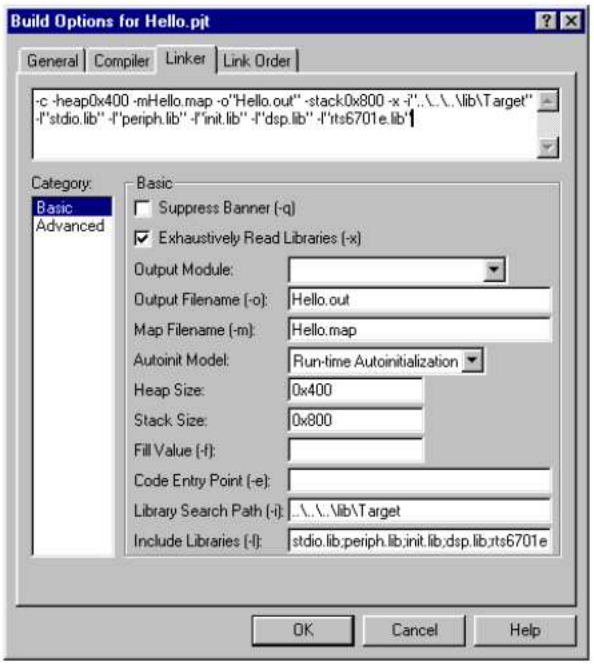


FIGURE 18. Code Composer Studio Compiler Build Options

Now, click on the Linker tab and configure the linker build options. In the category column, choose Basic. In this screen, make sure the Exhaustively Read Libraries has been selected. Then in the Output Filename box and Map Filename box type the project name. Click on the Autoint Model combination box and select "Run-time Autoinitialization". Next, set the Heap Size to 0x400 bytes and the Stack Size to 0x800 bytes, the Heap & Stack size may change depending on your application's needs. In the Library Search Path, type "..\..\..\lib\Target. Now, add stdio.lib; periph.lib; dsp.lib; and rts6201e.lib into the Include Libraries edit box (in that order). Then on the Include Libraries edit box line, add either init.lib for non-DSB Bios applications or biosinit.lib for DSP Bios applications, between the periph.lib and dsp.lib entries. When finished, the linker dialog screen should look like one below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Once all the build options have been set, rebuild your project by clicking Project|Rebuild All in the Code Composer Studio menu bar. If errors are encountered in one or more source files, they are listed in the output window. You may visit and repair each error by either double clicking on each error in the Output window.

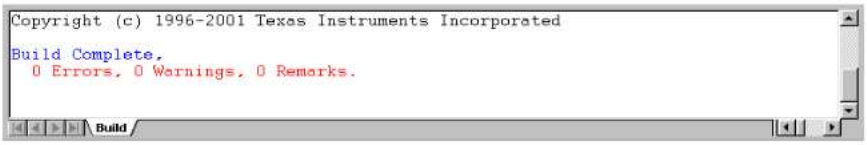


FIGURE 19. Code Composer Studio Build Results Window

Build Options (SBC6711 Boards)

Click on `Project | Build Options` to open the Build Options dialog box, then click on the Compiler tab to show the current compiler options. The required Build options are described in the following text, all other settings are optional and are described in the Code Composer Studio documentation.

Configure the compiler options to use the following settings: In the category column, choose Basic, click on the Target Version combination box, and select “671x”. Then click on the Opt Level combina-

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

tion box and select "-02: Function". Next, in the category column, choose Advanced, click on the Endianness combination box and select "little Endian". Then click on the Memory Models combination box and select "Far Calls & Aggregate Data". Again in the category column, choose Feedback, and click on the Banners combination box and select "No Banner or File Names". Under the Files category the Obj Directory can be added. When finished, the compiler dialog screen should look very similar like one below.

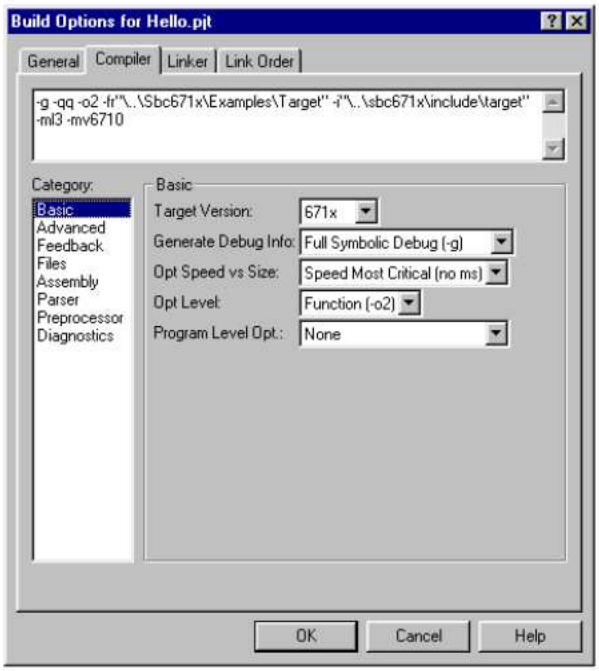


FIGURE 20. Code Composer Studio Compiler Build Options

Next, click on the Linker tab and configure the linker build options as follows. In the Output Module combination box select "Relocatable Executable". Then, click on the Autoint Model combination box and select "Run-time Autoinitialization". Next, set the Heap Size to 0x400 bytes and the Stack Size to 0x800 bytes, the Heap & Stack size may change depending on your application's needs. Make sure the Exhaustively Read Libraries has been selected. Now, add periph.lib, init.lib, stdio.lib, dsp.lib, and rts6700.lib into the Include Libraries edit box (in that order). Then on the Include Libraries edit box line, add either init.lib for non-DSB Bios applications or biosinit.lib for DSP Bios applications, between the periph.lib and stdio.lib entries. When finished, the linker dialog screen should look like one below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

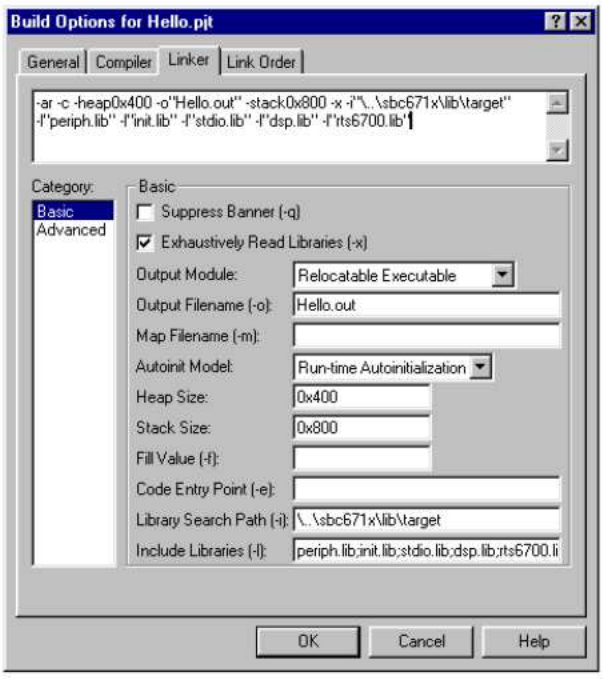


FIGURE 21. Code Composer Studio Linker Build Options for Non-Bios Project

Once all the build options have been set, rebuild your project by clicking Project|Rebuild All in the Code Composer Studio menu bar. If errors are encountered in one or more source files, they are listed in the output window. You may visit and repair each error by either double clicking on each error in the Output window.

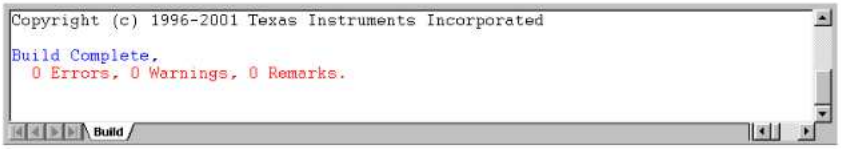


FIGURE 22. Code Composer Studio Build Results Window

Build Options (M44 Boards)

Click on `Project | Options` to open the Build Options dialog box, then click on the Compiler tab to show the current compiler options.

To configure the compiler options type the following command line in the compiler option box: "**`-gq -v40 -mn -o2 -x2 -frC:\M44\Examples\Target`**" When finished, the compiler dialog screen should look exactly like one below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

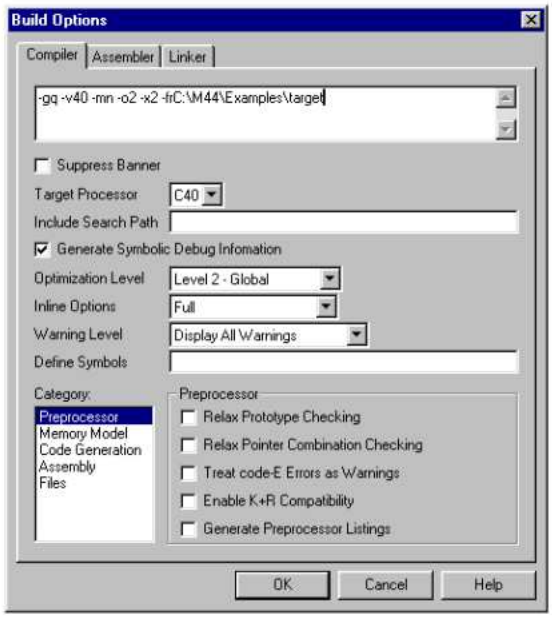


FIGURE 23. Code Composer Studio Compiler Build Options

Next, click on the Assembler tab and configure the assembler build options. In the assembler option box type the assembler build option "`-r`". When finished, the assembler dialog screen should look like one below.

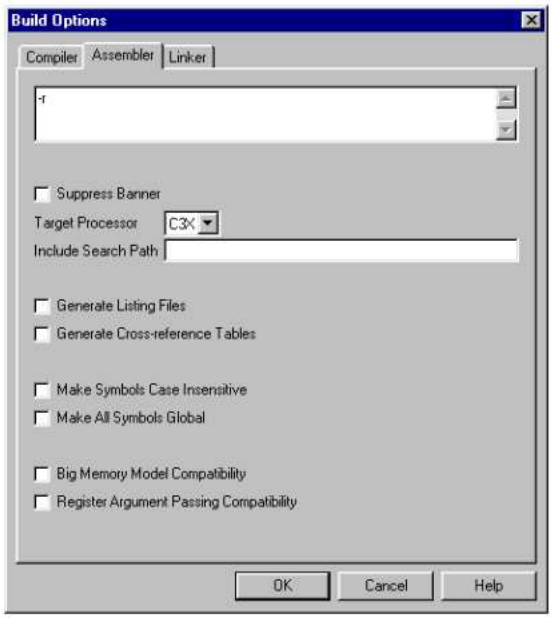


FIGURE 24. Code Composer Studio Assembler Build Options

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Finally, click on the Linker tab and configure the linker build options as follows. In the Output Module combination box, select "Relocatable Executable". In the Output Filename box, type in the name of the output file. Then in the Map Filename combination box, select "ROM Autoinitialization Mode". Next, set the Heap Size to 0x400 bytes and the Stack Size to 0x200 bytes, the Heap & Stack size may change depending on your application's needs. Make sure the Exhaustively Read Libraries have been selected. Now, add stdio.lib; periph.lib; dsp.lib; prts40.lib; and rts30.lib into the Include Libraries edit box (in that order). When finished, the linker dialog screen should look like one below.

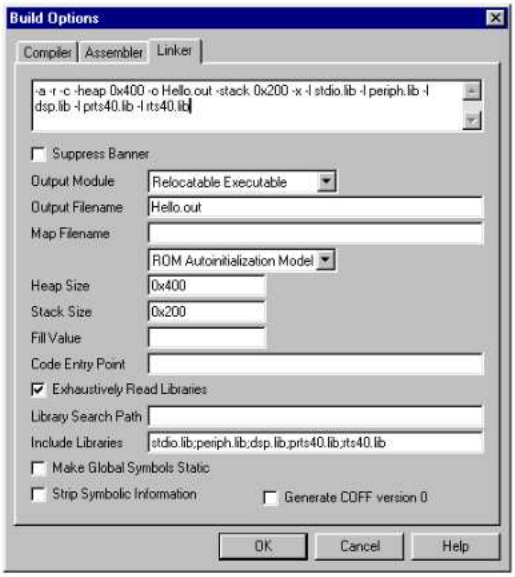


FIGURE 25. Code Composer Studio Linker Build Options

Once all the build options have been set, rebuild your project by clicking Project|Rebuild All in the Code Composer Studio menu bar. If errors are encountered in one or more source files, they are listed in the output window. You may visit and repair each error by either double clicking on each error in the Output window.

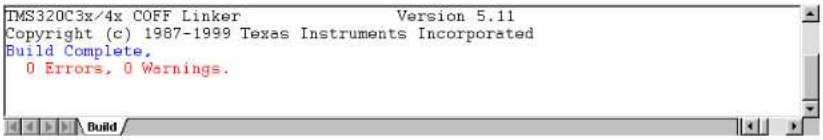


FIGURE 26. Code Composer Studio Build Results Window

Build Options (SBC32 Boards)

Click on `Project | Options` to open the Build Options dialog box, then click on the Compiler tab to show the current compiler options.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To configure the compiler options type the following command line in the compiler option box: "**`-gq -v32 -o2 -x2 -frC:\SBC32\Examples\Target`**" When finished, the compiler dialog screen should look exactly like one below.

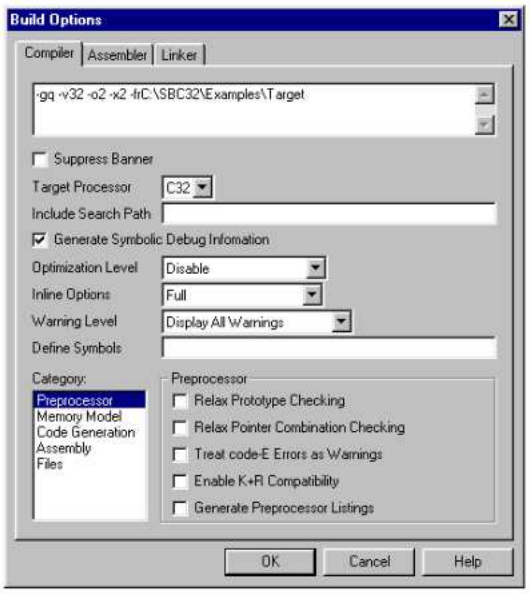


FIGURE 27. Code Composer Studio Compiler Build Options

Next, click on the Assembler tab and configure the assembler build options. In the Target Processor combination box, select "C3X". Then type the last assembler build option "**`-r`**" in the assembler option box. When finished, the assembler dialog screen should look like one below.

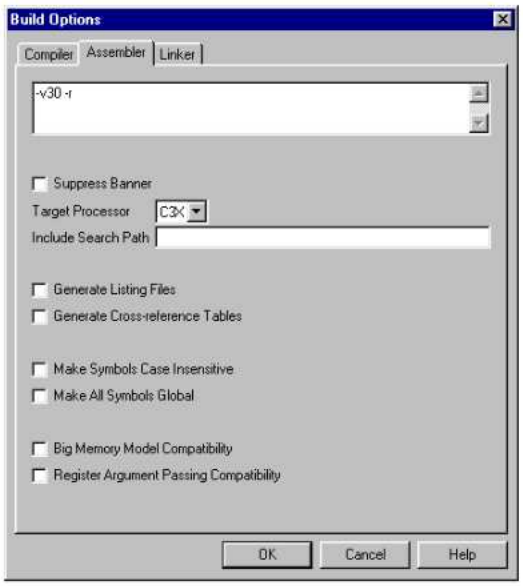


FIGURE 28. Code Composer Studio Assembler Build Options

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Finally, click on the Linker tab and configure the linker build options as follows. In the Output Module combination box select "Relocatable Executable". In the Output Filename box, type in the name of the output file. Then in the Map Filename combination box, select "ROM Autoinitialization Mode". Next, set the Heap Size to 0x400 bytes and the Stack Size to 0x200 bytes, the Heap & Stack size may change depending on your application's needs. Make sure the Exhaustively Read Libraries have been selected. Now, add stdio.lib; periph.lib; dsp.lib; and rts30.lib into the Include Libraries edit box (in that order). When finished, the linker dialog screen should look like one below.

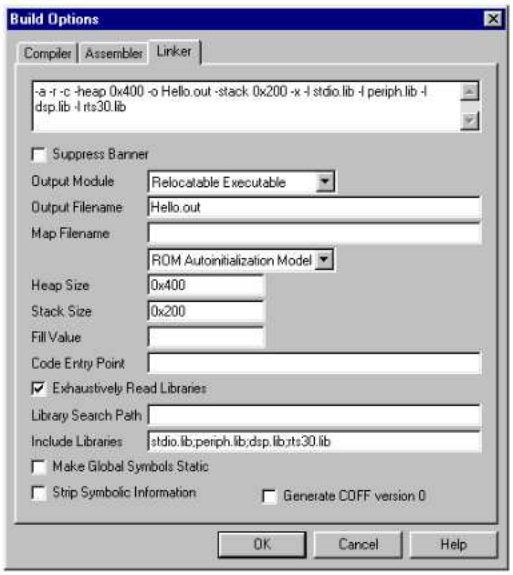


FIGURE 29. Code Composer Studio Linker Build Options

Once all the build options have been set, rebuild your project by clicking Project|Rebuild All in the Code Composer Studio menu bar. If errors are encountered in one or more source files, they are listed in the output window. You may visit and repair each error by either double clicking on each error in the Output window.

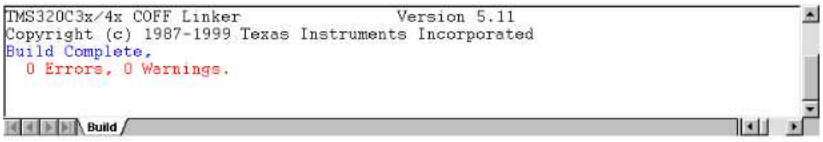


FIGURE 30. Code Composer Studio Build Results Window

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Automatic projectfile creation

When a project is created, opened, modified, built or rebuilt, the Code Composer Studio dependency generator automatically generates a project makefile (named <*project file*>.pjt, located in the project directory), which is capable of rebuilding the project's output file from its components.

This file is automatically submitted to the internal make facility whenever you click on build or rebuild within Code Composer Studio. The make facility automatically constructs the output file by recompiling the out-of-date source files including the dependencies contained within those source files.

Rebuilding a Project

It is sometimes necessary to force a complete rebuild of an output file manually, such as when you change optimization levels within a project. To force a project rebuild, select `Project | Rebuild All` from the Code Composer Studio menu bar.

Running the Target Executable

The `hello` program is very simple, only printing the single line "Hello, World" to the terminal emulator before waiting to echo any keystrokes and exiting. Bring up the "Hello, World" source file edit screen. Scroll down the source file by using cursor down button until you reach the call to `printf()`, which looks like the following:

```
printf("Hello, World\n");
```

Change the output string to read "`Hello, Brave New World\n`". You can now compile the new version by executing Build from the Project menu (or by clicking on its toolbar icon). This causes Code Composer Studio to start the compiler, which produces an assembly language output. The compiler then automatically starts the assembler, which produces a `.obj` output file (`hello.obj`). Code Composer Studio then invokes the TI Linker using the `generic.cmd` file, which is located in the root board directory. This rebuilds the executable file using the newly revised `hello.obj` . If no errors were encountered, this process creates the downloadable COFF file `hello.out`, which can be run on the target board. At this point, the program may be run using the terminal emulator applet, which may be invoked using the UniTerminal shortcut located within the target board program group created during the Zuma Libraries installation process. In the terminal emulator, download the `hello.out` file. The program runs and outputs the message "Hello, Brave New World" to the terminal emulator window.

If errors are encountered in the process, Code Composer Studio detects them and places them in the build output window. If the error occurred in the compiler or assembler (such as a C syntax error), the cursor may be moved to the offending line by simply double-clicking on the error line within the build output window, and the error message will be displayed in the Code Composer Studio status bar. If the linker returns a build error, the build output window shows the error file. From this information, the linker failure can be determined and corrected. For example, if a function name in a call is misspelled, the linker will fail to resolve the reference during link time and will error out. This error will be displayed on the screen in the build output window.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Note: Be sure to start the terminal emulator **BEFORE** starting Code Composer, to avoid resetting the DSP target in the midst of the debugging session. If the terminal emulator is not yet running and you wish to run the Hello object file, perform the following steps.

1. Execute Debug | Run Free to logically disconnect the DSP from the debugger software.
2. Terminate the Code Composer Studio application.
3. Invoke the terminal emulator "UniTerminal" application.
4. Restart the Code Composer Studio application.

This outlines the basics of how to recompile the existing sample programs within the Code Composer Studio environment.

Anatomy of a Target Program

While not providing much in the way of functionality, the `hello` program does demonstrate the code sequence necessary to properly initialization the target. The exact coding, however, is very specific to the I.I. C Development Environment, target boards, and is explained in this section in order to acquaint developers with the basic syntax of a typical application program.

Here we examine the M6x version of the `hello` program example. Although the source is not necessarily identical to that of `hello` for the other targets, it is typical of the overall structure of the typical application program designed under the development environment.

```
/*
*       HELLO.C
*       Test file/program for target board.
*/

#include "periph.h"
#include "stdio.h"

main()
{
        int key;
        enable_monitor();
        clrscr ();

        printf("Hello World!\n");
        printf("\nEchoing keystrokes...\n");
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
        do
                {
                key = getchar();
                putchar(key);
                }
        while(key != ESC);

        monitor();
}
```

The two lines of the program that being with a "#" are #include statements, which include the header files for the peripheral and standard I/O libraries. These include prototypes for all the library routines as well as variable definitions and #define statements for the peripheral memory-mapping addresses. These #defines are especially important for those who wish to perform direct peripheral access, rather than using the peripheral libraries.

The `enable_monitor()` function will setup the standard monitor I/O interface and reset the terminal window. The next two lines perform basic standard I/O functions; clearing the terminal emulation screen and printing "Hello World!" & "Echoing keystrokes...". These two lines are where custom code should be inserted.

The following `getchar()/putchar()` sequence simply echoes keys typed at the terminal emulator back to the terminal display, until the Esc key is pressed. When Esc is pressed, the `monitor()` function effectively terminates the program, except that interrupts are still active and interrupt handlers (if they had been installed) would still execute properly.

The `hello` program is very simple, but it contains the basic components of a typical DSP application, as well as the initialization needed to interact with the terminal emulator.

Use of Library Code

Library routines can be compiled and linked into your custom software simply by making the appropriate call in the source and adding the appropriate library to the linker command file. Refer to the library reference in this manual for library location information on each function.

In general, user software needs to #include the relevant library header file in source code. The header files define prototypes for all library functions as well as definitions for various data structures used by the library functions. The file **stdio.h** should be included by programs using the standard I/O library, and the file `periph.h` should be included if a program uses functions in the peripheral library. The function definitions in the peripheral library reference note which library a particular function resides in, as well as the header file, which that be included for the function.

Compiling/Assembling/Linking Outside Code Composer Studio

Under certain circumstances, it may not be possible to use Code Composer Studio macro definitions to compile inside the editor. `COMPILE.BAT`, `ASSEMBLE.BAT`, and `LINK.BAT` are provided in the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

%II_BOARD% directory and may be executed by typing their names followed by the source file on which they are to operate. For example, the file `mycode.c` can be compiled by typing:

```
compile mycode
```

at the DOS prompt. This causes the `COMPILE.BAT` script to start, which runs the compiler and generates the file `mycode.obj`, assuming no errors occurred. The `COMPILE.BAT` script also searches for the file `mycode.cmd` in the current directory. If the linker command file is found, then the linker is automatically run and the entire executable linked. If the command file is not found, processing stops with the generation of `mycode.obj`.

Assembly source (`mycode.asm`) may be assembled by typing:

```
assemble mycode
```

where the assembler is called and an object file generated.

Linking can also be performed. In this case the input file is not source code, but a linker command file (`mycode.cmd`):

```
link mycode
```

This line causes the linker to build the executable `mycode.out`, again assuming no errors have occurred during the process. Also, note that the `COMPILE.BAT` script will automatically link the executable if a linker command file of the same name exists.

In all the above cases, if any errors occur, an error file (`mycode.err`) is generated by the software tools. The `mycode.err` file contains the full console output of each of the tools. Any error that is generated by the tools will be recorded in this file.

The Next Step: Developing Custom Code

In building custom code for an application, Innovative Intergration recommends that you begin with one of the sample programs as an example and extend it to serve the exact needs of the particular job. Since each of the example programs illustrates a basic data acquisition or DSP task integrated into the target hardware, it should be fairly straightforward to find an example which roughly approximates the basic operation of the application. It is recommended that you familiarize yourself with the sample programs provided. The sample programs will provide a skeleton for the fully custom application, and ease a lot of the target integration work by providing hooks into the peripheral libraries and devices themselves.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 6

Developing Host Code

This section describes the Innovative Integration Windows host software development environment. The environment provides complete support for generating 32-bit Windows-compatible software, which is capable of controlling and communicating with Innovative Integration's DSP co-processor and data acquisition cards. Virtual device drivers (Windows 9x VxD or NT/2K Kernel Mode Driver) and dynamic link libraries (DLL) are included to provide an easy-to-use, portable low-level interface for the target hardware. Sample applications show how to call the DLL functionality and present basic interface examples with guidelines for on processor card control requirements and data movement.

Host software development is directly supported under either the Borland C++ Builder environment or the Microsoft MSVC environment for generating 32-bit Windows applications. Example application programs included in the development package are supplied with Borland project files, making program modification and regeneration as simple as possible. These programs utilize DspComponent, a VCL C++ component class provided in the Development package, to gain access to DLL features. Examples with similar functionality, utilizing an ActiveX wrapper derived from DspComponent are available for Microsoft Visual C++ users. See the online DspComponent.hlp file for a full description of the DspComponent.

Please Note: Only Windows application development is currently supported by the Developer's Package. Foreign operating systems, such as Unix and OS9 are not currently supported.

Dynamic Link Library

All target interactions takes place through calls to the supplied target DSP dynamic link library (DLL). This library supplies low-level functions for basic target board

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

control, including processor reset/run state, message passing via the board-specific mailbox registers, application downloading, and busmaster memory locking and access control.

The function calls available under the DLL are documented in the Host32.hlp Windows help file, provided in the Zuma Toolset. Sample applications (described below) provide working examples on how to interact with the card via host software.

Sample Host Programs

The DLL is capable of interacting with up to sixteen target DSP boards simultaneously by default (contact II if more targets are required). The DLL maintains a board-specific structure of information for each target, known as the `cardinfo` structure. An prototype of the cardinfo structure is located in the `\INCLUDE\HOST\` subdirectory in the `CARDINFO.H` file. An example is shown below.

```
//
//    cardinfo.h    --    definition of CARDINFO structure
//

#ifndef __CARDINFO_H__
#define __CARDINFO_H__

#include "ii_iostr.h"       // Common IO Driver/DLL Structures
#include "mailbox.h"        // Definition of MAILBOX structures

//
//  BoardInfo structure
//
typedef struct _BoardInfo
{
    ULONG        ProcessorCount;
    ULONG        DLL_Version;      // Version ID numbers
    ULONG        DrvVersion;
    ULONG        TalkerVersion;
    ULONG        CellSize;         // Target memory cell size, in bytes
    ULONG        CtlReg;           // Shadow of control register
    ULONG       FlashSectorSize; // Size of flash sectors, in bytes
    ULONG       FlashDeviceId;   // Flash device ID
    ULONG        QuietMode;        // Don't Display Messages if true
} BoardInfo;

//
//  InterruptInfo structure
//
typedef struct _InterruptInfo
{
    ULONG        IRQ;              // IRQ of attached interrupt
    HANDLE       Ring0Event;       // Ring 0 event handle
    HANDLE       Ring3Event;       // Ring 3 event handle
    void         (*Vector)(void *); // Virtual ISR function pointer
    void *      Context;          // Virtual ISR context pointer
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
 } InterruptInfo;

//
//  SerialInfo structure
//
typedef struct _SerialInfo
{
  LONG            In;         // Buffer for last character received
  LONG            ReadFlag;   // True when character received
  LONG            MbValue;    // Multi-byte value
  LONG            MbCtr;      // Multi-byte read state
  ULONG           RTS_state;  // Current state of the RTS output
  LONG           Bcr;    // Bus control register value for Flash access
  LONG            Reading;    // TRUE if currently reading a character
  OVERLAPPED      RxOverlap;  // Info used in asynch input
  OVERLAPPED      TxOverlap;  // Info used in asynch output
  COMMTIMEOUTS    Timeouts;   // Info for set/query time-out parameters
  DCB             Dcb;        // Device control block
} SerialInfo;

//
//  CARDINFO  structure
//
typedef struct _cardinfo

{
   ULONG           Target;         // Number of current target
   HANDLE          Device;         // Handle to Driver for device
   BoardInfo       Info;           // Board Info
   MAILBOX *       Mail;           // Talker Mailbox Array
   IoPortBlock     Port;           // Primary Port Block Information
   IoPortBlock     OpReg;          // Secondary Port Block Information
   MemoryBlock     DualPort;       // Shared Memory Area Information
   MemoryBlock     BusMaster;      // BusMaster Memory Information
   nterruptInfo    Interrupt;      // Interrupt Information
   SerialInfo      Serial;         // Serial Port I/O (SBC's)
} CARDINFO;

#endif
```

The `cardinfo` structure is accessed within Host application programs in order to gain access to board-specific parameters which are maintained by the DLL. For example, in order to ascertain the size of the shared memory area on a specific target card a host program could use:

```
/* send bus mastering physical address to target processor */

dsp = (CARDINFO*)target_cardinfo(target);

size = dsp->Dualport.Size;
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DspComponent

In the last few years, several different Rapid Application Development (RAD) environments have been developed to improve programmer productivity. Microsoft first produced Visual Basic, and Borland followed with Delphi (Object Pascal) and most recently, C++ Builder (which uses C++). What these tools have in common is the concept of a "component", a packaged piece of software than encapsulates the behavior of a button or other Windows control.

These can be arranged on forms with a visual editor provides a way to "drop" components into a window and add code to process events generated by components. For example, a button component generates a "Button Press" event, which can be set to automatically call a handler function by the RAD tool. The handler code will be called whenever the button is pressed.

DspComponent is a non-visual component provided by Innovative Integration, which provides a component interface to the board DLL that is included in each Zuma tool set. This allows the application programmer to avoid learning the details of connecting to the DLL, in favor of a drop-in package that is integrated into the language. This allows the developer to concentrate on the user interface and the DSP target programming, and not on the mechanics of Windows DLLs and drivers.

The DSPComponent Package consists of two components, an ActiveX and a Borland C++ Builder VCL component. Both perform the same function, and have essentially the same feature set, so that unless indicated a reference to the TIIDspComp component for C++ Builder applies also for the IIBoardX ActiveX control. The control was ported to ActiveX to allow its use in MSVC and Delphi.

The TIIDspComp component provides a high level interface to the Innovative Integration board DLL included as part of the Zuma tool set for a particular target board. This interface is common across all targets, and the same control can be used for all Innovative boards by picking the Target type at design time or even at run time. Mailbox access to the target is fully supported to the limits of the board selected, as simply as reading or writing to an array in the program. Interrupts in both directions, from host to target and target to host are supported. An interrupt from the target, if enabled, triggers a "software event" that can be handled just as any other event, such as a button click can. Resetting the board and downloading COFF files is simplified into a single call that returns a status code for failure conditions.

Features:

- Drop in VCL Component for C++ Builder and Active X version for Visual C++ and Delphi.
- A BoardType property selectable at design time or run time allowing one component to handle any Innovative target board.
- *Enabled* property to open and close target DLL
- Simplified board Booting and COFF downloading methods.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Interrupt support for both Target to Host and Host to Target interrupts.
- Target to Host interrupt handler by use of standard Event handler mechanism.
- Simplified mailbox property array (Mailbox[]) to access on-board mailboxes.
- Mailbox flag arrays to check for incoming mailbox data or free outgoing mailboxes without waiting for data.
- Shared memory properties to give access to board shared memory for high-speed data transfer to dualport or busmaster memory.
- Windows NT and Windows 95 support using Innovative's Zuma DLLs.

See the DspComponent subdirectory on your hard disk for detailed online help for DspComponent (Windows help: DspComponent.hlp) and for examples illustrating use of the component for streamlined Host application development.

Host Example Programs for the SBC671x Baseboard

Overview

The SBC671x Baseboard differs from other baseboards in that the normal means of communicating with the host is at a higher level than other cards. On other single-boards, the serial interface lends itself to low-level, character based communications well. The Dsp cards support some means of bulk data transfer and a separate low-level signalling and command interface (For example, Busmastering for bulk transfer and mailboxes for signalling and commands on the M62). For compatibility with other SBCs, the SBC671x also implements a simulation of mailboxes for transferring characters and low level data.

Message Packets. On this baseboard, using USB, the most natural way to transport information is by means of asynchronous packets of data that are transferred and decoded by the destination. These messages may be of varying sizes, allowing large messages to be efficient in sending bulk data while allowing small command messages to be mixed into the message stream. When delivered to the destination, the messages can be parsed and can result in any kind of processing desired.

By having the receipt of a message trigger the generation of additional messages, a message protocol can be developed to allow the transfer of data or the execution of control functions on demand from the other side of the link. This is even more natural if the applications are written in an event-driven style. The arrival of messages are the events to which the application responds.

Commands. The SBC671x also supports a second, more limited method of communication called *commands*. These are implemented via the USB VendorCommand mechanism. The host side can send a 32-bit word with a Write command, or send a 32-bit word and receive a 32-bit word reply with a Read command synchronously from the target. The requirement that the host initiate the transfer limits the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

use of commands, but the fact that the host will wait for the response before proceeding makes commands useful for synchronizing the target and host applications.

Interrupts. As with all baseboards, the SBC671x supports target-to-host and host-to-target interrupts.

On other baseboards, the standard host example was named "Scope" and demonstrated data transfer via the bulk interface signalled by interrupts. Because of the differences in the interface, the example programs for the SBC671x demonstrate the use of the forms of communication between the target and host and give an example of how to construct a command protocol between two applications, the target application and the host application. The two applications are very similar, the major difference being the platform used for development.

MessageTest

The MessageTest.exe example resides in the Examples\Host\Message directory. The target application matched with it is located in the Dsp subdirectory. The two applications share a header file, `Messages.h`, that contains the ID codes for the commands that are passed between the target and host. The IDs are entries in an enumeration, which assures that the IDs will be unique in each application. By using a single source file, we avoid the problem of two lists of IDs losing sync and producing difficult-to-detect errors during development.

The application uses the VCL DspComponent to provide the component interface to the application.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

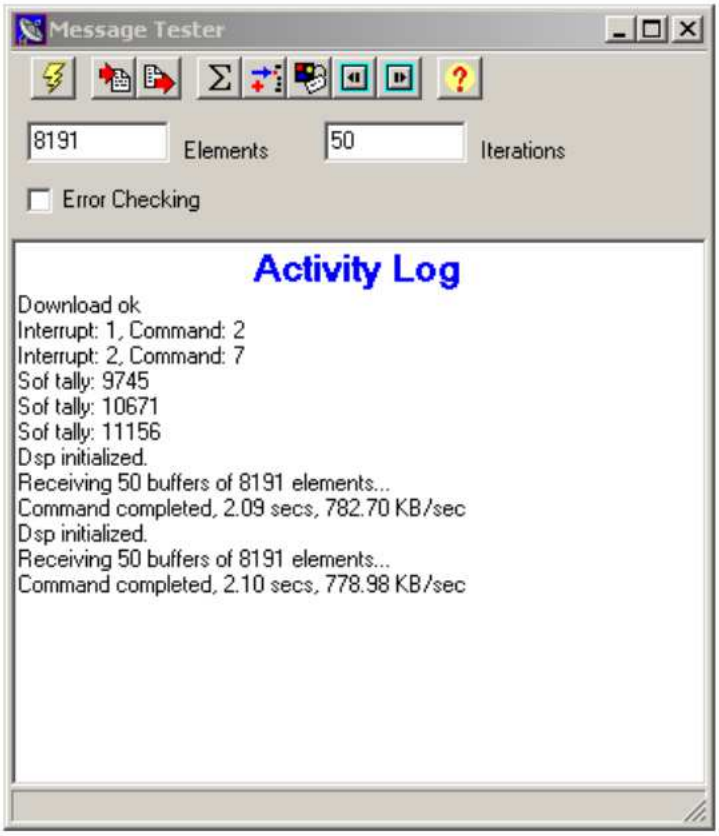


FIGURE 31. The MessageTest Example

Most of the area of the window is a visual log of the results of the functions performed when the buttons across the top of the screen are pressed. The remaining controls provide arguments to some of the other functions below. The code for this window can be found in `MessageTestMain.cpp`.

Download. The leftmost button is the only one active when the application begins. When pressed, it brings up the browser window, allowing the user to select which `.OUT` file will be downloaded to the target. There are two versions of `MsgTest.OUT` available, residing in the Examples\Host\Message\Dsp and Examples\Host\Message\Bios directories. Both have identical functionality, however one uses DSP/BIOS and the other does not. The browser defaults to the non-BIOS version. Once the file is selected, TForm1::DownloadButtonClick() is called. This function will attempt to prepare the target processor and download the target application onto it. It first attempts to load the baseboard DLL, printing a specific message if the DLL cannot be found. If the DLL is present, the DSP is booted and the target code is downloaded and executed. If the program downloads without error, the other buttons are enabled, allowing the communication tests to be run.

Message Receive Test. The second button performs a message receive rate test. (See TForm1::RcvButtonClick()). A single message to the target is used to trigger the sending of *Iterations* messages back, each containing data of size *Elements* in words in addition to the header information. These parameters are

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

entered from the edit controls on the main window. In addition, if the ErrorChecking box is checked, known data is inserted in the packets and checked for accuracy when it arrives at the host.

Messages are sent by calling the DspComponent function XmtMessage(). It takes a command code, and a pointer to the user-defined parameter region and its size in words. Note that since the message is asynchronously sent off, the host application cannot wait for the reply to arrive here. This portion of the processing must end with the posting of a request for action by the target application. When the target sends the reply messages, they will be processed by the message receiver code.

For this code, view the function TForm1::DspRcvMessage(). This function is an event handler for the Dsp Component's OnRcvMessage event. Whenever a message is received from the host, this handler is called to allow applications to process the message. The command ID of the message is used to select the processing function for the message. In this case, the TForm1::HandleTargetDataMessage() function is called to check the packet for errors, count the bytes transferred, and when all the messages have arrived, add to the log display the data rate for incoming data.

Message Transmission Test. The third button performs a similar test for message transmission. When the button is pressed, the TForm1::XmtButtonClick() function sends *Iterations* messages to the target, each containing data of size *Elements* in words in addition to the header information. If the ErrorChecking box is checked, known data is inserted in the packets for checking by the target on arrival.

SOF Counter Commands. The next two buttons demonstrate the use of commands. TForm1::ReadSofButtonClick() uses a Read Command to obtain a counter from the target that counts the number of SOF packets detected on the USB. These packets are sent about 1000 times a second. TForm1::InitDspButtonClick() uses a Write Command to clear the SOF counter to 0.

Revision Code Message. The next button uses a command message an reply message to create a simple protocol that in this case retrieves a revision code from the target. TForm1::RevisionButtonClick() sends a message that requests the revision code. The target program creates a reply message, which is processed by TForm1::HandleRevisionReply(). This processing function adds the revision code to the log. The non-BIOS example will report "Version: 1.01", while the DSP/BIOS example will return "Version: 1.02".

Target to Host Interrupts. The next button calls TForm1::HostInterruptButtonClick(), which sends a message to the target that requests an interrupt to the host. This interrupt results in the OnInterrupt event on the DspComponent. TForm1::DspInterrupt() is the handler for this event and prints a message in the log.

Host to Target Interrupts. The final test button demonstrates Host to Target interrupts. In TForm1::TargetInterruptButtonClick(), first the interrupt is generated by calling the Dsp Component Interrupt() method. After this, a message is posted to the target requesting the count of interrupts on the target. When this message arrives, TForm1::HandleMailboxTallyReply() posts the count to the log display.

The Target Application

The target application is involved in all of the transactions of the Message Test application. Each command or message must be processed and a reply might possibly be returned if required by the protocol. In this section we will run through how the target can perform these actions. See the file `MsgTest.c` to follow along with the description.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The main() Routine. The main routine is relatively simple, as most of the work involved in communication with the host is performed by the Zuma library. The interface to the library code is a number of callback function pointers that will be called when processing is required. These act very much like events do on the host side application. These handlers are:

Event Variable	Handler Function	Purpose
OnWriteCommand	WriteCommandHandler()	Process Write Commands from Host
OnReadCommand	ReadCommandHandler()	Process Read Commands from Host
OnProcessMessage	ProcessMessageHandler()	Process Messages from Host

After installing these handlers, the host communications are started with a call to init_communications(). Next, a handler for mailbox interrupts from the host is installed to process interrupts from the host. Finally, the main function remains in a loop calling the library function ProcessRcvMessage(). This function needs to be called to allow messages to be processed. It should be called in any long loop or idle polling process to process messages that may arrive in a timely manner.

ReadCommandHandler(). This handler is installed to process Read Commands from the host. The system passes in the command ID, along with other parameters. The data to be returned is passed back by the data argument. For example, the Get SOF command has the ID of *ccGetSofCnt*. The switch statement loads the data argument with get_sof_tally()'s return value. This sends the SOF count back as the response.

WriteCommandHandler(). This handler processes Write Commands from the host. Here the data argument is a parameter sent by the host. As an example, the *ccInit* command resets system parameters and the SOF counter.

ProcessMessageHandler(). This handler processes messages from the host. In a simple case, the *ccGetRevision* ID is sent to obtain the revision ID. The processing for this message is a simple call to XmtMessage(), which transmits the message to the host with the revision code inserted as part of the message. In a more complex version, the *ccGenerate* command results in the posting of a number of messages to the host to perform the receive message test. Each message coming from the host results in a call to a processing function.

Host to Target Interrupts. When an interrupt is sent by the host, the handler installed for mailbox interrupts is called. MailboxInterruptHandler() acknowledges the interrupt and increments a counter. This counter is retrieved by the host by a message request in order to verify the delivery of the interrupt.

Target to Host Interrupts. One message sent by the host, *ccHostXrpt*, generates an interrupt to the target by calling the library function mailbox_interrupt(). The parameter is delivered through the mailbox to the host for logging.

VcMessage Test

The second example is implemented using Microsoft Visual C++. It also demonstrates the means of communication between the target and host applications. It uses the same target application as the VCL example, and uses the same header file, `Messages.h`, to assure that the commands IDs remain consistent in the two applications.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

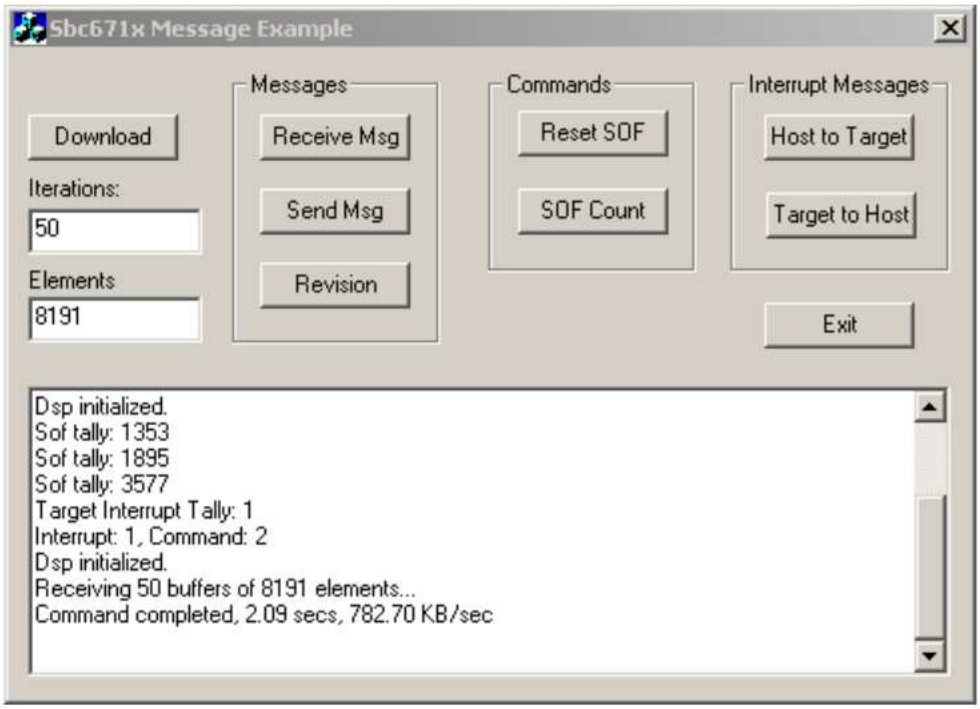


FIGURE 32. The VclMessage Example

To provide an object-oriented interface to the target, the VclMessage example uses the Dsp Component ActiveX control. This component allows the use of similar methods, properties, and events as the VCL component.

At the bottom of the window is a visual log of the results of the functions performed when the buttons in the top half of the window are pressed. The remaining controls provide arguments to some of the other functions.

The code for this application is located in Examples\Host\VcMessage directory. It is a dialog box based MFC application. The code for the window controls can be found in `VcMessageDlg.cpp`.

Download. The Download button is the only one active when the application begins. When pressed, it brings up the browser window, allowing the user to select which `.OUT` file will be downloaded to the target. There are two versions of `MsgTest.OUT` available, residing in the Examples\Host\VcMessage\Dsp and Examples\Host\VcMessage\Bios directories. Both have identical functionality, however one uses DSP/BIOS and the other does not. The browser defaults to the non-BIOS version. Once the file is selected, CVcMessageDlg::OnDownload() is called. This function will attempt to prepare the target processor and download the application. It first attempts to load the baseboard DLL, printing a specific message if the DLL can not be found. If the DLL is present, the DSP is booted and the target code is downloaded and executed. If the program downloads without error, the other buttons are enabled, allowing the communication tests to be run.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Message Receive Test. The first group of buttons collects tests involving Messages. The first button in this group performs a message receive rate test. (See CVcMessageDlg::OnRcvMessage()). A single message to the target is used to trigger the sending of *Iterations* messages back, each containing data of size *Elements* in words in addition to the header information. These parameters are entered from the edit controls on the main window.

Messages are sent by calling the DspComponent function XmtMessage(). It takes a command code, and a pointer to the user-defined parameter region and its size in words. Note that since the message is asynchronously sent off, the host application cannot wait for the reply to arrive here. This portion of the processing must end with the posting of a request for action by the target application. When the target sends the reply messages, they will be processed by the message receiver code.

For this code, view the function CVcMessageDlg::OnRcvMessageliboardx1(). This function is an event handler for the Dsp Component's OnRcvMessage event. Whenever a message is received from the host, this handler is called to allow applications to process the message. The command ID of the message is used to select the processing function for the message. In this case, the CVcMessageDlg::HandleTargetDataMessage() function is called to count the bytes transferred, and when all the messages have arrived, add to the log display the data rate for incoming data.

Message Transmission Test. The SendMsg button performs a similar test for message transmission. When the button is pressed, the CVcMessageDlg::OnSendMsg() function sends *Iterations* messages to the target, each containing data of size *Elements* in words in addition to the header information.

Revision Code Message. The Revision button uses a command message an reply message to create a simple protocol that in this case retrieves a revision code from the target. CVcMessageDlg::OnGetRevision() sends a message that requests the revision code. The target program creates a reply message, which is processed by CVcMessageDlg::HandleRevisionReply(). This processing function adds the revision code to the log. The non-BIOS example will report "Version: 1.01", while the DSP/BIOS example will return "Version: 1.02".

SOF Counter Commands. The central group contains two buttons that demonstrate the use of commands. CVcMessageDlg::OnGetSof() uses a Read Command to obtain a counter from the target that counts the number of SOF packets detected on the USB. These packets are sent about 1000 times a second. CVcMessageDlg::OnResetSof uses a Write Command to clear the SOF counter to 0.

Target to Host Interrupts. The right group of buttons contains buttons that test interrupts. The Target to Host button calls CVcMessageDlg::OnTargtoHost(), which sends a message to the target that requests an interrupt to the host. This interrupt results in the OnInterrupt event on the DspComponent. CVcMessageDlg::OnDspInterruptliboardx1() is the handler for this event and prints a message in the log.

Host to Target Interrupts. The final test button demonstrates Host to Target interrupts. In CVcMessageDlg::OnHostToTarget(), first the interrupt is generated by calling the Dsp Component Interrupt() method. After this, a message is posted to the target requesting the count of interrupts on the target. When this message arrives, CVcMessageDlg::HandleMailboxTallyReply() posts the count to the log display.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 7

Creating Target Software

Software is created for the target DSP by using one or more of the tools included in the Developer's Package. The tools can be used alone or in concert with each other to generate a downloadable executable COFF format file, which can be run on the target DSP board with the aid of the utilities included in the developer's package.

This section of the *Developer's Package Manual* details the use of the individual tools in the package to create executables for the target DSP. This section also gives step-by-step instructions on how to use the C compiler and Code Composer to write, compile, test, and debug custom C applications on the target. Sample C applications are also discussed

C Code Development

C Compiler

The Texas Instruments C compiler is an ANSI C compatible compiler, which produces optimized assembly code for the TMS320C4x family of processors. A complete set of manuals is included with the M6x Developers Package.

In addition to the excellent manuals from TI, refer to the Kernighan and Ritchie C Handbook (available at cost from I.I.) for generic C questions and syntax. The TI manuals primarily describe the use of the compiler with the TMS320C4x family and are not intended as C primers for the beginner.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C Library Reference

Complete source code to the entire suite of ANSI C libraries is provided with the C system to aid in code development. Refer to the *TMS320 Floating Point DSP Optimizing C Compiler Manual* for a complete list of TI C functions.

The I.I. M6x Developer's System also includes extensive high-level libraries useful in interacting with the various peripherals on the M6x board. The following sections describe by peripheral type the functions provided in the peripheral library. For a complete alphabetical listing of all peripheral functions, see Appendix.

M6x Zuma Toolset Libraries

The Zuma toolset provides both target peripheral libraries and Host DLLs along with numerous example programs to illustrate usage.

The peripheral libraries for the M6x provide support for the on-board peripherals and terminal I/O functions. The libraries are provided in three linkable `.LIB` files: `PERIPH.LIB`, `STDIO.LIB`, and `DSP.LIB`. `STDIO.LIB` holds all the console terminal emulation and communications routines listed in the following section, while `PERIPH.LIB` contains all other peripheral driver routines. DSP.LIB contains commonly requested C-callable digital signal processing functions, plus common math and queue management extensions. Source code for the routines is also provided, arranged by function in the `\PERIPH`, `\STDIO`, and `\DSP` subdirectories of the root II_BOARD directory, as follows:

Directory	Library Source
`\DSP`	Standard Digital Signal Processing Routines.
`\PERIPH\ANALOG`	Drivers for the M6x A4D4 instrumentation-grade analog I/O module and the complementary TERM mux module. Drivers for the SD high-performance audio module.
`\PERIPHERAL\BUS`	Drivers for V360 bus-mastering PCI interface.
`\PERIPH\DIGITAL`	Digital I/O, PIT Timer control, module FLASH ROMs, etc. Drivers for DIO module. Drivers for MOT motion control module.
`\PERIPHERAL\DIO`	DIO module DUART and digital I/O drivers.
`\PERIPH\MISC`	Miscellaneous processor control and data conversion functions.
`\PERIPH\RTS`	Modified boot-up routines for the M6x baseboard.
`\STDIO`	Console and terminal emulation functions.
`\TALKER`	Start-up umbilical 'C6x01 software.

TABLE 1. Zuma Toolset Source Directories

The toolset also contains various support files arranged as described below.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Directory	Library Source
\EXAMPLES\HOST	Example programs illustrating use of the DLL to control the DSP board from within MS Visual C programs.
\EXAMPLES\TARGET	Example programs illustrating use of the target peripheral libraries to perform common DSP tasks.
\INCLUDE\HOST	Header files used by Host Visual C programs.
\INCLUDE\TARGET	Header files used by target Texas Instruments C programs.
\LIB\HOST	Linkable library files for Host Visual C and C++ programs.
\LIB\TARGET	Linkable library files for target TI C and assembler programs.
\SRC	Useful public domain source files for the C6x01 processor.

TABLE 2. Zuma Toolset Support Subdirectories

STDIO Console Terminal Driver. The Developer's Package contains a full-featured terminal emulator application (uniterminal.exe), suitable for both user interface purposes as well as debugging use. The peripheral library provides a complete set of standard I/O routines, which can communicate directly with this terminal emulator. The source for the standard I/O routines is given in the `\STDIO` subdirectory under the installation directory. In general, the standard I/O library functionality is identical to that of the K&R standard I/O library. However, some M6x-specific functions are provided to allow higher level functionality such as cursor positioning, text attribute control, and graphical data plotting. The following target programming section gives details on how to use the standard I/O peripheral library to interact with the terminal emulator.

Digital Peripheral Drivers. The digital peripheral drivers control the 'C6x01 internal timers and the digital I/O lines. These drivers allow for high-level access to timebase control functions and digital I/O activity without doing direct hardware programming. The following target programming section gives details on how to use the digital peripheral library to program the digital peripherals. Source code for the functions is given in the `\PERIPH\DIGITAL` directory.

BUS Peripheral Drivers. The BUS peripheral drivers provide control functions for the onboard V360 PCI bus interface. The available routines support very-high speed bus-mastering transfers between the 512 Kbyte, external async SRAM of the M6x and host PC memory. This driver also includes hardware mailbox support routines, which are used extensively by the standard I/O library in order to support terminal emulation. Additionally, these mailbox routines provide a means of performing interrupt-driven communications with the Host PC. The target programming section gives details on how to use the bus peripheral library. Source code for the functions is given in the `\PERIPH\BUS` directory.

Miscellaneous Peripheral Drivers. The MISC directory contains code to support high-level access to the internal registers, byte packing and unpacking, interrupt vector support, and other functions. Source code for the functions is given in the `\PERIPH\MISC` directory.

RTS Peripheral Drivers. The RTS peripheral drivers provide board-specific versions of the functions called by the TI C Compiler during coldstart initialization of the C runtime engine. These files have been modified as necessary in order to provide a complete initialization of the M6x onboard hardware immediately prior to calling `main()` within application code. Additionally, the RTS functions include a modified version of the millisecond timer function required to support the TI C timekeeping functions (listed in `time.h`). Source code for the functions is given in the `\PERIPH\BUS` directory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Digital Peripheral Drivers. The digital drivers support access to all baseboard and add-on digital I/O functions.

The DIO peripheral drivers provide control functions for the optional DIO plug-in module. The functions provide high-level C access to the DIO module's 32, additional digital I/O lines, plus either interrupt-driven or polled use of the DIO's onboard DUART (Dual-channel Universal Asynchronous Receiver Transmitter). The target programming section gives details on how to use the digital peripheral library to program the digital peripherals. Source code for the functions is given in the `\PERIPH\DIGITAL\DIO` directory.

The MOT peripheral drivers provide control functions for the optional MOT plug-in module. The functions provide high-level C access to the MOT module's four, precision motion-control axes. Each of the axes features independent encoder inputs and either digital or 16-bit analog output. Digital output may be either pulse and direction positive or negative pulse to support stepper motor amplifier inputs. The target programming section gives details on how to use the MOT peripheral library to program these peripherals. Source code for the functions is given in the `\PERIPH\DIGITAL\MOT` directory.

Analog Peripheral Drivers. The Analog peripheral drivers provide control functions for the optional analog plug-in modules: A4D4, AIX, and SD modules. The functions provide high-level C access to the A4D4's analog input and output channels and their associated gain amplifiers. Additionally, the driver supports control of the optional TERM break-out panel, a companion to the A4D4 module. In order to support muxing of each of the A4D4 modules 8:1 to allow input from up to 32 simultaneous channels per A4D4 module. Source code for the functions is given in the `\PERIPH\ANALOG\A4D4` directory.

The AIX peripheral drivers provide control functions for the optional AIX plug-in module. The functions provide high-level C access to the AIX module's four, 2.5 MHz, 16-bit analog input channels. Source code for the functions is given in the `\PERIPH\ANALOG\AIX` directory.

The SD peripheral drivers provide control functions for the optional SD plug-in module. The functions provide high-level C access to the A4D4 module's four, audio-grade, 24-bit analog input and 20-bit output channels. Source code for the functions is given in the `\PERIPH\ANALOG\SD` directory.

The target programming section gives details on how to use the analog peripheral library to program these analog peripherals.

Digital Signal Processing Library. The DSP directory contains code to support high-level access to the common signal processing functions such as FFT's, filters and compression. Additional routines are provided for common functions such as matrix manipulation, curve fitting and general purpose queue management. Source code for the functions is given in the `\DSP` directory.

Texas Instruments C Libraries. Several libraries are included with the system that provide support for floating point and extended math functions, DSP oriented procedures and initialization examples. Chapter 5 in the *TMS320 Floating Point DSP Optimizing C Compiler User's Guide* describes the libraries.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following libraries are available:

Library	Operation
ASSERT.H	Defines the assert macro for runtime error message reporting.
CTYPE.H	Declares functions that test and convert characters.
LIMITS.H	Defines range limits for characters and variable types.
FLOAT.H	Defines floating point range limits.
MATH.H	Defines trigonometric, exponential and hyperbolic math functions.
ERRNO.H	Defines errno variable for catching range errors in function calls.
STDARG.H	Defines macros to aid in variable argument functions.
STDDEF.H	Defines two new types and macros used within runtime functions.
STDLIB.H	Declares many common library functions such as string conversion, sorting and searching functions, program exit functions and some integer-arithmetic that is not a standard part of C.
STRING.H	Declares functions for string manipulations.
TIME.H	Declares macros and types useful for time manipulations.

TABLE 3. Texas Instruments Standard Library Functions

M6x Hardware Interaction

All peripherals are memory mapped into the 'C6x01 address space, using the locations given in the following table. The table also lists the wait states applied to accesses to each peripheral.

The development system provides routines to access all integrated M6x peripherals. This section describes how to program the peripherals using the supplied library functions under C or via direct memory accesses to the supplied peripheral register map. In general, direct memory access delivers higher performance than using the C function library since it avoids the overhead of the function calls necessary to access the library. However, the libraries have been crafted to utilize inline code where possible to mitigate this effect. In the peripheral descriptions that follow, each device's access methods are called out for both high level and direct memory access. In the case of C functions, the function names and argument variables are called out. In the case of direct memory access operations, the relevant addresses are listed along with the functions they perform and accompanying `Periph` structure elements which may be used from C to simplify access. These elements are defined in the header file `periph.h`.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function	Address	C Language Mneumonic	Mem Space
FIFO Port	0x0400000	Periph->Fifo	CE0
V360 Registers	0x1400000	Periph->PciRegs	CE1
FIFO Port Reset	0x1410000	Periph->FifoReset	
AD9850 Reset	0x1470000	Periph->DDS.Reset	
AD9850 Frequency Update	0x1480000	Periph->DDS.Update	
AD9850 Write Clock	0x1490000	Periph->DDS.Clock	
Digital I/O Data Register	0x14A0000	Periph->Dio.Data	
Digital I/O Direction Control	0x14B0000	Periph->Dio.Direction	
Digital I/O Input Latch Clock Control Register	0x14C0000	Periph->Dio.LatchControl	
External Mux Control 0	0x14D0000	Periph->Mux[0]	
External Mux Control 1	0x14E0000	Periph->Mux[1]	
16 bit External Timer	0x14F0000	Periph->Timer	
External Interrupt Input 4 Select	0x1500000	Periph->EI[4]	
External Interrupt Input 5 Select	0x1510000	Periph->EI[5]	
External Interrupt Input 6 Select	0x1520000	Periph->EI[6]	
External Interrupt Input 7 Select	0x1530000	Periph->EI[7]	
I/O Module Strobe 0	0x1540000	Periph->Module[0]	
I/O Module Strobe 1	0x1550000	Periph->Module[1]	
I/O Module Strobe 2	0x1560000	Periph->Module[2]	
I/O Module Strobe 3	0x1570000	Periph->Module[3]	
I/O Module Strobe 4	0x1580000	Periph->Module[4]	
I/O Module Strobe 5	0x1590000	Periph->Module[5]	
I/O Module Strobe 6	0x15A0000	Periph->Module[6]	
I/O Module Strobe 7	0x15B0000	Periph->Module[7]	
I/O Module Strobe 8 (cM6x only)	0x15C0000	Periph->Module[8]	
I/O Module Strobe 9 (cM6x only)	0x15D0000	Periph->Module[9]	
I/O Module Strobe 10 (cM6x only)	0x15E0000	Periph->Module[10]	
I/O Module Strobe 11 (cM6x only)	0x15F0000	Periph->Module[11]	
Async SRAM (128Kx32)	0x1600000	Periph->ASRam[0..0x80000]	
SDRAM (16Mbyte) (optional)	0x2000000	Periph->SDRam[0..0x1000000]	CE2
SBSRAM (1Mbyte) (optional)	0x3000000	Periph->SBRam[0..0x100000]	CE3

TABLE 4. M6x External Peripheral Memory Map

This section does not describe peripheral hardware specifications and other hardware issues. Refer to the *M6x Hardware* section of this manual for additional hardware information.

Digital Input/Output

The digital input/output (I/O) buffers provide a means for generating 32 bits of direct digital input or output to and from external hardware. This I/O can be clocked from either the 'C6x01 processor or from external TTL sources, allowing external devices to automatically latch data into the I/O buffers for the 'C6x01 to read.

Input/output direction for either half of the 32-bit port may be programmed on the fly using on-board logic. The port may be configured in the software for input or output in groups of eight bits.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Memory Mapped Digital I/O Access. The following table shows the memory locations used to interact with the digital I/O buffers. Three C language routines are supplied to interact with the digital I/O port.

Function	C Language Mnemonic
Digital I/O Data Register	`Periph->Dio.Data`
Digital I/O Direction Control (4 bytes)	`Periph->Dio.Direction`
Digital I/O Latch Control	`Periph->Dio.LatchControl`

TABLE 5. Digital I/O Access Memory Location

The `Periph->Dio.Data` location is used to access the data lines of the digital I/O port. Results of read and write accesses depend on the I/O direction of the port (see below for information on setting the port direction). If the port is configured for input, a read access latches new read data from the external pins and the new data is read into the 'C6x01. If the port is configured for output, the most recently latched output data is read into the 'C6x01 (output data does not change). Write accesses to an input port cause no change to the port status, while write accesses to an output port cause the new data to be latched and output to the external I/O pins.

The `Periph->Dio.Direction` location controls the direction of each byte of the digital I/O port. The four least significant bits of this register are used to configure each of the bytes of the digital I/O port for either input or output, as follows:

Dio.Direction-Register Bit #	Value	Direction
0	0	DX[0..7] output (default)
	1	DX[0..7] input
1	0	DX[8..15] output (default)
	1	DX[8..15] input
2	0	DX[16..23] output (default)
	1	DX[16..23] input
3	0	DX[24..31] output (default)
	1	DX[24..31] input

TABLE 6. Digital I/O Direction Configuration

The `Periph->Dio.LatchControl` location controls the method of latching data into each byte of the digital I/O port. The four least significant bits of this register are used to configure the latch method as either internal (triggered by CPU accesses) or external (triggered by an external TTL pulse), as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Dio.LatchControl-Register Bit #	Value	Bits Affected	Clock Source
0	0	0..7	Internal (CPU-based)
	1		External
1	0	8..15	Internal (CPU-based)
	1		External
2	0	16..23	Internal (CPU-based)
	1		External
3	0	24..31	Internal (CPU-based)
	1		External

TABLE 7. Digital I/O Latch Configuration

C Language Digital I/O Functions. Data may be read or written to the digital I/O port using the following routines in the DIGITAL support library.

Function Name	Description
`DIO_dir()`	Sets the direction of all four bytes of the onboard 32-bit digital I/O port.
`DIO_read()`	Returns current state of all 32-bits of digital I/O port.
`DIO_write()`	Sets current state of all 32-bits of digital output port currently configured for output.
`DIO_latchcontrol()`	Sets the latch method of all four bytes of the onboard 32-bit digital I/O port.

TABLE 8. Digital I/O Library Functions

Timers

The timers provide the capability to generate hardware timebases, which can be used to trigger processor interrupts, analog signal conversions, or as direct outputs to external hardware. There are a total of six timebase sources built in to the M6x: two 32-bit timers internal to the 'C6x01 processor, and three 16-bit channels implemented with custom logic within the FPGA plus one AD9850 direct digital synthesizer. The supplied library functions initialize the timers to a free-running, pulse generation mode suitable for generating convert pulses to the analog hardware.

The timers are initialized by code in the `timebase()` routine each time it is called. Normally, no other function calls are necessary to use the timers. However, when supplying an external TTL signal to the 'C6x01 TCLK0/1 inputs in order to provide an external timebase to analog circuitry, it will be necessary to create and use a custom version of `timebase()`, which tristates the TCLK output driver to avoid contention with external sources. Please note that certain hardware setups might be required depending on the application. See the *M6x Hardware* section of this manual for more details on how to set up the M6x board.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C Language Timer Functions. The following functions give high-level access to the timer hardware. See the appendices for complete information on the functions.

Function Name	Operation
`timebase()`	Configures a specified timer channel (0..5) l for periodic counting at a specified frequency using a specified source clock rate.

TABLE 9. C Language Timer Functions

`timebase()` can be used to set a particular timebase to a particular frequency. For example, the following call sets PIT timer channel 1 to generate a 1000 Hz output pulse stream, assuming the hardware default 10 MHz input clock to the FPGA logic:

```
timer(1, 1000.0, 10.0);
```

Memory Mapped Timer Access. It is possible to directly access to the internal timer hardware controls via memory mapped registers at specific addresses. It may be necessary to use these addresses to set the timers to a custom mode. In general, unless custom functionality is required of the timers, it is recommended that the user exclusively access the timers via the `timer()` routine rather than programming the control and period registers manually.

For information about the 'C6x01 internal timers, please see the *TMS320C6x User's Guide*. For additional information about the custom PIT counter/timer device, contact Innovative Integration. For an example of direct timer channel control, refer to the source code for the `timebase()` function, located in the `PERIPH\DIGITAL` subdirectory.

STDIO Communication. C stdio terminal emulation is provided in the Peripheral Library. The stdio library communicates with the host UNITERMINAL.EXE program via the V360 PCI interface mailbox registers to provide stdio support to DSP applications running on the M6x. The stdio interface may be used for real-time, non-intrusive software debugging or to create a basic user interface for OEM applications.

The following list shows the available Peripheral Library calls and their operation. See the Appendix for complete information on the functions.

Function Name	Operation
`putchar()`	Emits an 8-bit character to the terminal emulator
`getchar()`	Gets an 8-bit character from the terminal emulator's keyboard buffer
`gets()`	Inputs a string into a target buffer
`puts()`	Displays a string from a target buffer
`sprintf()`	Formats a string into a memory buffer pointed to by buffer
`printf()`	Prints a formatted string to the terminal
`scanf()`	Inputs a formatted string from the terminal into a buffer
`sscanf()`	Converts a formatted string in memory into a buffer
`stdio_reset()`	Resets the terminal emulator display
`fopen()`	Opens a file on the Host PC, returning the file handle
`fclose()`	Closes a previously opened Host PC file.
`fread()`	Reads file contents into a target buffer

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

`fwrite()`	Writes a target buffer into a Host PC file
`fseek()`	Repositions the Host PC file pointer
`ferase()`	Erases the specified Host PC file
`kbd_hit()`	Returns a nonzero value if characters are currently available in the monitor keyboard buffer
`kbd_key()`	Returns 16-bit IBM scancode for pending keystroke from the terminal emulator's keyboard buffer.
`gotoxy()`	Moves the terminal cursor
`wherexy()`	Returns the terminal cursor position
`clreol()`	Clears to end of current line
`clrscr()`	Clears the terminal screen
`type()`	Types formatted, null terminated string to console
`bold()`	Enables bold text attribute in terminal emulator
`normal()`	Enables standard text attribute within terminal emulator
`get_attribute()`	Returns the current character display attributes
`set_attribute()`	Sets the current character display attributes
`cursor()`	Enables/disables the cursor
`get_busmaster_addr()`	Obtains the base of the host busmaster memory from the terminal emulator.
`plot()`	Plots a Host PC file as a graph.
`view()`	Plots a target buffer as a graph.

TABLE 10. STDIO Driver Functions

Using Interrupts. The M6x supports four external and numerous internal hardware interrupts. These include EI0, EI1, EI2, EI3, plus TINT0, TINT1 (internal timer/counters), internal com. port transmit and receive and DMA.

Interrupts on the TMS320C6x01 may be handled by writing either high-level C or assembly language procedures within your application files, which employ the following interrupt-specific function names:

`void c_intNN()` for C handlers or

`_c_intNN` for assembly language

where NN is numbered 0 through 99 for each of the interrupts. For each interrupt, a procedure must be coded, which will be executed upon acknowledgment of interrupt NN by the 'C6x01. This is described in more detail in the C Compiler Users Manual.

Consider the following code example:

```
/*

* EXAMPLE.C

*/

#define TINT0 14

main()

{
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

```
    enable_interrupts();       /* Enable unmasked xrpts */
    timebase(1, 1000.0, 1.0);  /* Internal timer 1 at 1kHz */
/* install interrupt handler on TINT1 */
    install_int_vector(c_int02, TINT0);
    enable_interrupt(TINT0);
            .
                /* Bulk of application */
            .
      disable_interrupt(TINT0);  /* Disable TINT0 xrpt */
}
/*
*  ISR for timer 0 - Tally a variable
*/
int milliseconds
void c_int14()
{
      milliseconds++;      /* Internal timer 0 is used to */
}                          /* synthesize a timebase */
```

In this code, the internal timer 0 is configured to output a pulse every millisecond, which drives TINT0 on the 'C6x01. The vector is installed into the jump table with a call to `install_int_vector()` and the bit associated with TINT0 in the interrupt enable register, is enabled. Finally, `main()` calls `enable_interrupts()` which, sets the global interrupt enable bit so that all unmasked interrupts can be processed.

Each time the counter expires, the routine `c_int14()` executes. In this example, the variable `milliseconds` is incremented during each interrupt service cycle.

Each DSP application should include a copy of the default interrupt vector table, which is defined in `vectors.asm`. This assembly file is located in the `PERIPH\RTS` directory. When it is compiled into a `.obj` file and linked into the application, it will cause all entries in the vector table to be initial-

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ized with a default handler. The one exception being the break interrupt vector, which is filled with the pointer to the talker program. If an application needs to make use of interrupts, those vectors which are affected need to be changed with `install_int_vector()` at run time.

See the target example programs provided on your distribution disks for further examples of the use of interrupts.

Example Target Programs for the M6x

The following section details the example target software included with the Developer's Package. These programs are provided as models for custom user software, and it is highly recommended that the user examine these examples before beginning a first development effort for the target DSP. Full source code is provided for user inspection and reuse in modified or custom applications.

These examples will run on a standard M6x card with no additional hardware required.

HELLO

`HELLO` is a very simple introduction to basic program components and use of the C stdio library for the target card. When run with the host terminal emulator active, the program simply initializes the target hardware and stdio interface and prints the message "Hello, World" via the stdio library to the terminal emulator screen. The program then drops into an infinite dwell loop.

`HELLO` may be rebuilt from with in the Code Composer Studio environment by loading the `HELLO.MAK` project from the \target\examples directory. Then, modifying the source file `HELLO.C`, and rebuilding the project (see the Code Composer Studio documentation for more information on the application's project management and make facilities).

For correct program functionality, it is necessary to run the `HELLO` application via the host terminal emulator program. If the terminal emulator is not active and communicating with the target M6x card on which `HELLO` is running, the application will appear to hang at the first instance of a stdio function call (usually a `getint()` or `putint()` call). This is due to the fact that all stdio calls use the M6x bus mailbox interface and are handshaken with the host terminal emulator application. Any such calls will hang if the terminal emulator is not active to complete the communication link.

TEST

`TEST` is board level hardware test program, which is capable of accessing the major peripherals on the M6x to double-check proper hardware functionality. As such, it contains routines for exercising each of the peripherals on the M6x, including:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1. Digital I/O
2. Internal timers
3. External timers
4. Communications Ports

Since the `TEST` program aims to be all-encompassing in that it tries to test as much of the board-level functionality as possible, it serves as a poor example for complicated operations such as A/D multi-channel sampling and display. However, since the code included for TEST is broken down into functional pieces, which are called separately for each subsystem to be tested, it is possible to factor out individual tests for use in other programs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 8

Target DSP Peripheral Libraries

Target Functions by Category

Category	Name	Description
Board Initialization & System Functions	baud	Set baud rate on current serial port.
	cpu	Set CPU number and mailbox.
	cpu_num	Get CPU number.
	cpu_number	Get CPU number (inline).
	detect_cpu_speed	Derive DSP clock speed.
	dma_done	Wait for DMA completion.
	dpram_addr	Return start address of Dualport RAM on PC31.
	dpram_type	Detects 16 or 32 bit Dualport RAM on PC31.
	init_serial	Initialize the serial I/O system.
	InitIP	Initialize Industry Pack access structure.
	mem_size	Detect size of memory space.
	test_mem	PC31 memory check.
Busmaster Transfer Functions	bm_init	Busmaster transfer initialization.
	bm_transfer	General busmaster transfer.
	fifo_init	Busmaster initialization.
	transfer_complete	Wait for Busmaster transfer to complete.
USB Bulk Transport Interface Functions	InitBulkTranport	Initialize the Bulk Transport Interface.
	StopBulkTransport	Shut down the Bulk Transport Interface.
	IsBulkTransportReady	Returns true if system can send data.
	OpenBulkTransport	Opens a channel of the Bulk Transport System.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

	CloseBulkTransport	Shuts down an open channel of the Bulk Transport System.
	ReadBulk	Read a block from a Bulk Transport channel.
	WriteBulk	Writes a block to a Bulk Transport channel.
	BulkDataAvailable	Returns the amount of data available for reading on a channel.
	BulkSpaceAvailable	Returns the room for new data available on a channel.
	FlushBulk	Forces the transmission of all data in a channel.
Digital I/O Functions	C31_dig_dir	Program the direction of PC31/SBC31 PIA Digital I/O bytes.
	C31_read_dig	Read PC31/SBC31 PIA Digital I/O lines.
	C31_write_dig	Write to PC31/SBC31 PIA Digital I/O lines.
	C31_write_dig_bit	Update a single bit on PC31/SBC31 PIA Digital I/O.
	dig_dir	Program the direction of Digital I/O bytes.
	read_abits	Read state of ABITS output lines.
	read_abits_bit	Read state of a single ABITS output bit.
	read_dig	Read Digital I/O lines.
	read_dig_bit	Read state of a single digital bit.
	write_abits	Write to ABITS digital output.
	write_abits_bit	Update a single ABITS digital output bit.
	write_dig	Write to digital output.
	write_dig_bit	Update a single digital output bit.
Analog I/O Control Functions	enable_analog	Initialize analog subsystem.
	trigger_adc	Set triggering mode for an ADC.
	trigger_adc_pair	Set triggering mode for an ADC pair.
	trigger_dac	Set triggering mode for an DAC.
	trigger_dac_pair	Set triggering mode for an DAC pair.
	write_analog_interrupt_mask	Set which analog conversions fire interrupts.
Analog Input Functions	correct_adc	Adjust ADC reading to proper range.
	correct_adc_pair	Adjust a pair of ADC readings to proper range.
	convert_adc	Manually trigger an ADC conversion.
	convert_adc_pair	Manually trigger an ADC conversion on an ADC pair.
	read_adc	Read data from ADC.
	read_adc_pair	Read data from a pair of ADCs.
	read_adc_automux	Read data from ADC, and switch multiplexer.
	read_adc_pair_automux	Read data from a pair of ADCs, and switch mux.
Analog Output Functions	correct_dac	Adjust DAC reading to proper range.
	correct_dac_pair	Adjust a pair of DAC readings to proper range.
	convert_dac	Manually trigger a DAC conversion.
	convert_dac_pair	Manually trigger a DAC conversion on a DAC pair.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

	convert_dacs	Manually trigger DAC conversions using a bit mask.
	read_dac	Read last value loaded into a DAC.
	read_dac_pair	Read last value loaded into a DAC pair.
	update_dac	Write DAC value and automatically trigger conversion.
	update_dac_pair	Write DAC pair and automatically trigger conversion.
	write_dac	Write value to DAC.
	write_dac_pair	Write value pair to a DAC pair.
Programmable Gain Functions	gain_to_mode	Convert Gain into equivalent Gain Mode number.
	mode_to_gain	Convert gain mode to actual gain value.
	read_gain	Read last Gain setting.
	write_gain	Update gain setting for a channel.
	write_gains	Update gain setting for all channels.
Mux Control Functions	auto_mux	Configure automatic multiplexing feature.
	read_mux	Read last setting of a particular mux.
	write_mux	Update multiplexer setting for a channel.
	write_muxes	Update multiplexer setting for all channels.
Mailbox and Semaphore Functions	check_inbox	Check incoming mailbox for new data.
	check_outbox	Check outgoing mailbox for new data.
	clear_mailboxes	Clear mailboxes.
	get_semaphore	Get hardware semaphore.
	read_mailbox	Read from incoming mailbox.
	read_mb_terminate	Read from incoming mailbox if data available.
	release_semaphore	Release hardware semaphore.
	write_mailbox	Write to outgoing mailbox.
	write_mb_terminate	Write to outgoing mailbox if box is ready.
Interrupt Support Functions	deinstall_int_vector	Remove vector from vector table.
	disable_interrupt	Disable specific interrupt.
	enable_interrupt	Enable specific interrupt.
	host_interrupt	Target to host interrupt.
	install_int_vector	Install vector into vector table.
	mailbox_interrupt	Post a mailbox interrupt to the host.
	mailbox_interrupt_ack	Acknowledge a mailbox interrupt.
	mailbox_interrupt_deinstall	Unload the handler for mailbox interrupts.
	mailbox_interrupt_disable	Disable mailbox interrupts.
	mailbox_interrupt_enable	Enable mailbox interrupts.
	mailbox_interrupt_install	Load a handler for mailbox interrupts.
	suspend	Idle until interrupts arrive.
	interrupt_cpu	Interrupt specified multiprocessor target CPU.
	cpu_int_src	Return source code # for specified multiprocessor CPU.
	cpu_xrpt_bit	Return register index to specified multiprocessor CPU.
Timer Functions	disable_clock	Disable system millisecond timebase.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

	enable_clock	Initialize system millisecond time-base.
	ms	Dwell milliseconds.
	read_timer	Read value from a hardware timer.
	timebase	Set hardware timer frequency.
	timer	Set hardware timer frequency.
	uclock	Get system millisecond timer value.
	us	Dwell microseconds.
Memory Movement Functions	copy_mem	Fast on-chip memory copy.
	fill_mem	Fast on-chip memory fill.
	mem_to_port	Fast on-chip transfer of data to a port.
	port_to_mem	Fast on-chip transfer of data to a port.
	dma_copy_mem	Fast DMA memory copy.
	dma_fill_mem	Fast DMA memory fill.
	dma_mem_to_port	Fast DMA transfer of data to a port.
	dma_port_to_mem	Fast DMA transfer of data to a port.
Conversion Functions	from_ieee	Convert from IEEE-754 floating point format.
	packb	Pack byte value into int.
	packh	Pack half word value into int.
	to_ieee	Convert to IEEE-754 floating point format.
	unpackb	Unpack byte values from int.
	unpackh	Unpack half word values from int.
Flash Memory Programming	fast	Restore PBCR to original value after Flash access.
	flash_erase	Erase entire Flash memory.
	flash_init	Initialize Flash for programming.
	flash_rd	Read Flash byte.
	flash_read	Read 32-bit word from Flash.
	flash_sector_erase	Erase a Flash sector.
	flash_wr	Write a byte to Flash memory.
	flash_write	Write 32-bit word to Flash.
	slow	Reduce speed of I/O accesses to access Flash memory.
CPU Register I/O	clear_interrupt_flag	Disable interrupt enable bit.
	get_DIE	Retrieve 320C4x DIE register.
	get_IE	Retrieve 320C3x IE register.
	get_IIE	Retrieve 320C4x IIE register.
	get_IF	Retrieve 320C3x IF register.
	get_IIF	Retrieve 320C4x IIF register.
	get_IOF	Retrieve 320C3x IOF register.
	get_ST	Retrieve 320C3x/4x Status register.
	set_DIE	Set 320C4x DIE register.
	set_IE	Set 320C3x IE register.
	set_IF	Set 320C3x IF register.
	set_IIE	Set 320C4x IIE register.
	set_IIF	Set 320C4x IIF register.
	set_IOF	Set 320C3x IOF register.
	set_interrupt_flag	Set 'C3x Interrupt Flag Bit.
	set_PC	Set processor program counter.
	set_ST	Set processor status register.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FIFO Library Functions

Catagory	Name	Description
FIFO Link Support	set_fifo_link_AF_levels	Set almost-full threshold levels.
	fifo_link_emit	Send a character to link using handshake.
	fifo_link_key	Get a character from link using hand-shake.
	fifo_link_spit	Send a character to link without using handshake.
	fifo_link_eat	Get a character from link without using handshake.
	bleed_fifo_link	Drain FIFO into memory buffer.
	fill_fifo_link	Fill FIFO from memory buffer.
	reset_fifo_link	Initialize a link to empty state.
	get_fifo_link_status	Obtain fullness state information.
	login()	Query subordinate processors for login sequence.
	sub_login	Send login sequence to master processor.
	fifo_link	Return register index to FIFO link for specified CPU.
FIFO Port Support	set_fifo_port_AF_levels	Set almost-full threshold levels.
	fifo_port_emit	Send a character to link using handshake.
	fifo_port_key	Get a character from link using hand-shake.
	fifo_port_spit	Send a character to link without using handshake.
	fifo_port_eat	Get a character from link without using handshake.
	bleed_fifo_port	Drain FIFO into memory buffer.
	fill_fifo_port	Fill FIFO from memory buffer.
	reset_fifo_port	Initialize a link to empty state.
	get_fifo_port_status	Obtain fullness state information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Standard I/O Library Functions

Category	Name	Description
Console Terminal Control Functions	bold	Set console text bold attribute.
	clreol	Clear console to end of line.
	clrscr	Clear console screen.
	cursor	Enable/disable console cursor.
	get_attibute	Get current console text attribute type.
	gotoxy	Set cursor position.
	normal	Set console text normal attribute.
	set_attibute	Set current console text attribute type.
	wherexy	Get cursor position.
Low Level I/O	emit	Send a character to the terminal emulator.
	getchar	ANSI get character from console.
	kbd_hit	Install vector into vector table.
	kbd_key	Get a key from the terminal emulator.
	key	Get a character from the standard mail-box.
	putchar	ANSI put character to console.
C Standard I/O Library Emulation Functions	fclose	Close a host disk file.
	ferase	Delete a host disk file by name.
	fflush	Commits an open file I/O stream to disk.
	fopen	Open a host disk file for read.
	fread	Read from host disk file into target mem-ory.
	fseek	Moves the file pointer to a specified loca-tion.
	fwrite	Write to host disk file from target mem-ory.
	gets	ANSI gets from console.
	printf	ANSI printf to console.
	puts	ANSI puts to console.
	scanf	ANSI scanf from console.
	sprintf	ANSI sprintf.
	sscanf	ANSI sscanf.
	type	Send a character string to the terminal emulator.
Terminal Applet Extensions	get_busmaster_addr	Retrieve host busmaster address from the terminal emulator.
	plot	Transfer data buffer to host for plotting.
	stdio_reset	Reset the terminal emulator program.
	stdio_terminate	Send the termination code to the terminal emulator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

DSP Library Functions

Category	Name	Description
Signal Processing Functions	bartlett	Bartlett window generation.
	bitrev	Bit reversal function.
	blackman	Blackman window generation.
	buffer_statistics	Calculate statistics on a data buffer.
	ffft_r1	Forward Fast Fourier Transform - Real.
	ffft_r2	Forward Fast Fourier Transform - Complex.
	fir	Finite Impulse Response Filter.
	hamming	Hamming window generation.
	hanning	Hanning window generation.
	harris	Harris window generation.
	ifft_r1	Inverse Fast Fourier Transform - Real.
	ifft_r2	Inverse Fast Fourier Transform - Complex.
	vmul	Multiply two vectors into a third vector.
Matrix Functions	matrix_add	Add two matrices and return a sum MATRIX.
	matrix_allocate	Allocate a matrix and return its MATRIX pointer.
	matrix_crop	Form sub-matrix from a larger matrix.
	matrix_det	Return the determinant of a square matrix.
	matrix_free	Free matrix area and MATRIX structure.
	matrix_invert	Invert a square matrix, return inverse MATRIX.
	matrix_mult	Multiply two matrices, return new MATRIX.
	matrix_mult_pwise	Multiply two matrices element by element.
	matrix_print	Print the elements of a matrix to stdout.
	matrix_scale	Scale all of a matrix by a constant.
	matrix_sub	Subtract two matrices and return a difference MATRIX.
	matrix_transpose	Transpose a matrix, return pointer to new MATRIX.
Queue Support Functions	dequeue_ptr	Remove data from a queue and adjust pointer.
	enqueue_ptr	Load data into Queue and update pointers.
	enqueued	Return count of data elements in a Queue.
	queue_init	Initialize memory Queue structure.
BERR Sequence Generation Functions	berr_decode	Tests a value in a BERR sequence.
	berr_encode	Generate the next value in a BERR sequence.
	berr_initialize	Set up a BERR sequence generator.
Data Compression Functions	a_compress	A-Law data compression.
	a_expand	A-Law data expansion.
	mu_compress	Mu-Law data compression.
	mu_expand	Mu-Law data expansion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 9

Host DLL Reference

DLL Functions Grouped by Function

The functions tabularized below may be used in any Host program written in a language, which supports access to a Dynamic Link Library. The prototypes for these functions are listed in the `BASEBOARD\INCLUDE\HOST\TARGET.H` file. The names of these functions are aliai of the actual board-specific library function names, which are proto-typed in `BASEBOARD\INCLUDE\HOST\ALIAS.H`. For examples on using the DLL, install DspComponents from the installation CD.

TABLE 11. Generic DLL Function List

Category	Function Prototype	Function Description
General	BOOL target_open(int target)	Opens driver for specified target DSP board. Returns boolean.
	BOOL target_close(int target)	Closes driver for specified target DSP board. Returns boolean.
	LPVOID target_cardinfo(int target);	Returns address of cardinfo structure for target.
	int iicoffld(char *, int target, HWND hParent);	Loads a COFF executable file onto target DSP.
Interrupt Functions	BOOL host_interrupt_enable(int target);	Enables a previously installed virtual interrupt handler.
	BOOL host_interrupt_disable(int target);	Disables a previously enabled virtual interrupt handler.
	void host_interrupt_install(int target, void (*virtual_isr)(void *), void * context);	Installs a virtual interrupt handler.
	void target_interrupt(int target);	Interrupts target DSP board.
	void host_interrupt_deinstall(int target);	Removes a virtual interrupt handler.
	void mailbox_interrupt(int target, unsigned int value);	Interrupts the target DSP after writing value to special mailbox.
	unsigned int mailbox_interrupt_ack(int target);	Acknowledges target to Host interrupt, returns special mailbox contents.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Control Functions	void target_reset(int target);	Physically asserts reset on the target DSP board.
	void target_run(int target);	Deasserts reset on the target DSP board.
	void target_outport(int target, int port, int value);	Outputs a value to specified DSP board I/O port address.
	int target_inport(int target, int port);	Inputs a value from specified DSP board I/O port.
	void target_opreg_outport(int target, int port, int value);	Outputs a value to specified DSP board operation port address.
	int target_opreg_inport(int target, int port);	Inputs a value from specified DSP board operation port.
	void target_control(int target, int bit, int state);	Modifies a bit in the control register of the target DSP board.
Mailbox Functions	int read_mailbox(int target, int);	Reads the specified mailbox of the target DSP board.
	void write_mailbox(int target, int, int);	Writes to the specified mailbox of the target DSP board.
	BOOL check_outbox(int target, int);	Interrogates the specified output mail-box status.
	BOOL check_inbox(int target, int);	Interrogates the specified input mail-box status.
	int read_mb_terminate(int target, int, int *, int wide);	Reads the specified input mailbox, if full.
	int write_mb_terminate(int target, int box_number, int value, int wide);	Writes to the specified output mail-box, if empty.
	void clear_mailboxes(int target);	Clears all mailboxes to empty state.
	int target_key(int target);	Reads terminal mailbox, returns an 8-bit contents.
	void target_emit(int target, int value);	Writes 8-bit value to terminal mail-box.
	void target_Tx(int target, int value);	Writes 32-bit value to terminal mail-box.
	int target_Rx(int target);	Reads 32-bit value from terminal mailbox.
Bulk Transport Interface Functions	int BULK_GetNumDevices();	Returns the number of SBC62 USB devices detected.
	BOOL BULK_OpenDevice(int iDevice, HANDLE *phDevice)	Opens a device for BULK transport access.
	BOOL BULK_CloseDevice(IN HAN-DLE hDevice)	Closes a device for BULK transport access.
	BOOL BULK_OpenChannel(int iDevice, WORD wChannel, BOOL fOverlapped, BULK_HANDLE *pHandle);	Opens a data channel in BULK mode.
	BOOL BULK_CloseChannel(BULK_HANDLE Handle)	Closes a data channel opened with BULK_OpenChannel().
	BOOL BULK_Read(BULK_HANDLE Handle, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead, LPDWORD lpNumberOfBytesRead, LPOVER-LAPPED lpOverlapped);	Reads a block of data in BULK mode.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

	BOOL BULK_Write(BULK_HANDLE Handle, LPCVOID lpBuffer, DWORD dwNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVER-LAPPED lpOverlapped);	Writes a block of data in BULK mode.
	BOOL BULK_GetOverlappedReadResult(BULK_HANDLE Handle, LPOVERLAPPED lpOverlapped, LPDWORD lpNumberOf-BytesTransferred, BOOL bWait)	Gets the WIN32 Overlapped Result for the Read portion of the data channel.
	BOOL BULK_GetOverlappedWriteResult(BULK_HANDLE Handle, LPOVERLAPPED lpOverlapped, LPDWORD lpNumberOf-BytesTransferred, BOOL bWait);	Gets the WIN32 Overlapped Result for the Write portion of the data channel.
	BOOL BULK_CancelIo(BULK_HANDLE Handle)	Cancels all pending I/O on the device.
	BOOL EXPORT STREAM_Open(int iDevice, WORD wChannel, WORD wBuffer-Size,WORD wBlockSize, BULK_HANDLE *pHandle)	Opens s data channel in STREAM node.
	BOOL STREAM_Close(BULK_HANDLE handle)	Closes a STREAM data channel.
	WORD STREAM_WriteAvailable(BULK_HANDLE handle)	Returns the amount of space available for Write data. (in integers)
	WORD STREAM_ReadAvailable(BULK_HANDLE handle)	Returns the amount of data available on the STREAM channel. (in integers)
	WORD STREAM_Write(BULK_HANDLE handle, INT32 *pBuffer, WORD wElement-Count)	Writes a block of data to the STREAM channel.
	void STREAM_Read(BULK_HANDLE handle, INT32 *pBuffer, WORD wEle-mentCount)	Reads a block of data from the STREAM channel.
	void STREAM_Flush(BULK_HANDLE handle)	Writes all the output data to the target.
Sempahore Functions	void get_semaphore(int target, int sema-phore);	Gains ownership of specified target semaphore.
	void request_semaphore(int target, int semaphore);	Requests ownership of specified target semaphore.
	BOOL own_semaphore(int target, int semaphore);	Interrogates ownership status of specified semaphore.
	void release_semaphore(int target, int semaphore);	Relinquishes control of specified semaphore.
Talker Functions	int target_check(int target);	Interrogates for Talker running on target.
	void start_app(int target);	Starts a previously downloaded target application program.
	int start_talker(int target);	Starts the target Talker executing.
	int target_revision(int target);	Returns the revision of the target Talker.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 10

DOS Environment Requirements

Innovative Integration Developers Packages, including the TI C Compiler, make use of environment variables in order to locate header files monitor script files, etc. Be sure to set the following environment variables when installing either the C compiler or I.I. libraries. Note that several of these environment variables may be automatically set when running the SETUP program on the distribution disks. However, when upgrading from previous versions or when mixing development components from I.I. or other sources, problems can arise.

Use the table below to insure that you specify all needed environment variables.

Environment Variable Name	Products Affected	Suggested settings
II_BOARD	Dev. Pkg. Applets	**set II_BOARD**=<*board dir*> ie set II_BOARD=c:\M6x
C_DIR	All TI C Compilers All I.I. peripheral libraries	**set C_DIR**=%ii_board%;%ii_board%\include\target;%ii_board%\lib\target;%compiler dir%\rtdx\include;%compiler dir%\rtdx\lib;%compiler dir%\cgtools\include;%compiler dir%\cgtools\lib;%compiler dir%\bios\include;%compiler dir%\bios\lib; ie set C_DIR=C:\M6x;C:\M6x\include\target;C:\m6x\lib\target;C:\ti\c6000\rtdx\include;C:\ti\c6000\rtdx\lib;C:\ti\c6000\cgtools\include;C:\ti\c6000\cgtools\lib;C:\ti\c6000\bios\include;C:\ti\c6000\bios\lib; Specified order is critical!
A_DIR	All TI Assemblers	Same as C_DIR above

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D_DIR	TI Debuggers	**set D_DIR**=<*debugger dir*> ie set D_DIR=c:\ti\c600\evm6x\lib
D_SRC	All Debuggers	**set D_SRC**=<*source code dir1*>;<*dir2*>;...;<*dir n*> ie set D_SRC=c:\M6x\stdio;c:\M6x\dsp; c:\M6x\periph\analog;c:\M6x\periph\digital;… ;c:\M6x\periph\bus
PATH	All II products All TI Tools	**set path**=%path%;%ii_board%;%ii_board%\lib\host;%compiler dir%\cgtools\bin;%compiler root%\bin ie set path=%PATH%;C:\M6x;C:\M6x\lib\host;C:\ti\c6000\cgtools\bin;C:\ti\bin

TABLE 12. Required Disk Directory Structure for I.I. Development Tools

The Innovation Integration, TI, and C Development System for the M6x requires the following environment variables be set properly for correct operation:

set ii_board=C:\M6x

set C_DIR=C:\M6x;C:\M6x\include\target;C:\m6x\lib\target;C:\ti\c6000\rtdx\include;
C:\ti\c6000\rtdx\lib;C:\ti\c6000\cgtools\include;C:\ti\c6000\cgtools\lib;C:\ti\c6000\bios\include;C:\ti\c6000\bios\lib;

set A_DIR=%C_DIR% (same as c_dir)

set d_src=C:\M6x;C:\M6x\stdio;C:\M6x\periph\bus;C:\M6x\periph\rts;C:\M6x\periph\digital;
C:\M6x\periph\misc;C:\M6x\dsp;C:\M6x\periph\analog

set path=%PATH%;C:\M6x;C:\M6x\lib\host;C:\ti\c6000\cgtools\bin;C:\ti\bin

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 11

M6x/cM6x Hardware

M6x/cM6x Hardware Functions

The M6x is a PCI bus compatible digital signal processor (DSP) card based around the Texas Instruments TMS320C6x01 processor. Implementing a modular I/O expansion system, the M6x is particularly suited to data acquisition and control tasks, and is supported by a collection of I/O bus function cards, which provide hardware interfacing to real-world equipment.

The M6x's features include:

- TMS320C6x01 processor.
- Optional external zero wait-state SBSRAM and one wait-state SDRAM memory pools.
- Two inter-board communications ports (up to 80 Mbytes/sec transfer rate).
- Six channels of on-board timing (two on-chip timers, three custom 16-bit timers in FPGA logic and the 9850 DDS timebase).
- OMNIBUS module compatible (two available slots on M6x, three on cM6x).
- 32 bits of digital I/O.
- Two serial port connectors.
- External mux board control connectors (compatible with external TERM multiplexer and signal conditioner boards).
- JTAG hardware emulation support.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following figure gives a block diagram of the M6x/cM6x.

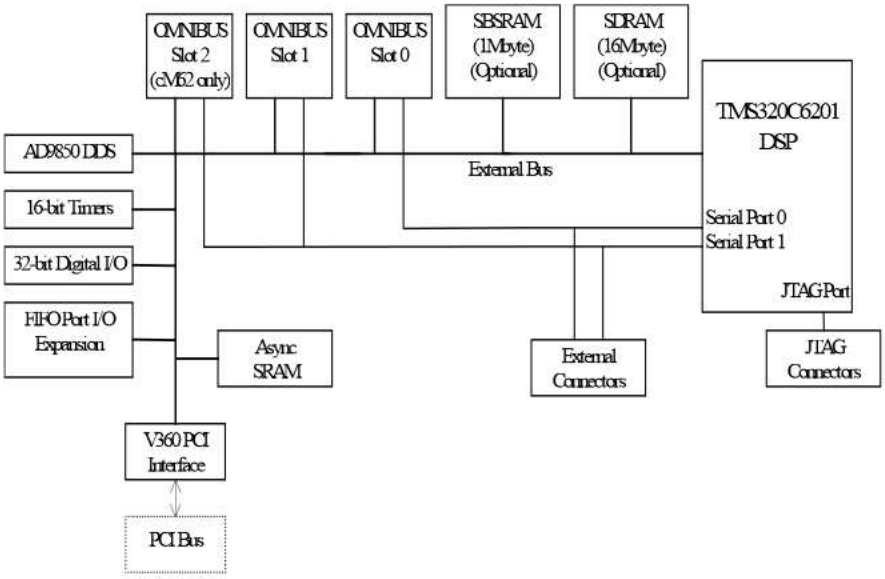


FIGURE 33. M6x/cM6x Block Diagram

The cM6x is a Compact PCI compatible version of the M6x board. The cM6x retains all of the features of the M6x but is intended for use in Compact PCI host systems. In addition to the M6x feature set, the cM6x includes an additional OMNIBUS slot (allowing up to three OMNIBUS modules to be installed on a single cM6x board).

For brevity's sake, this section will refer to both cards as the M6x. Any differences in functionality between the two boards (support of the third I/O bus site, different types of connectors, etc.) will be noted as required. In addition, the PCI and Compact PCI buses are collectively referred to as the PCI bus.

Memory Map

The M6x processor operates in 'C6x01 HPI boot mode with memory map of type 1. In this mode, the processor's memory is available to the PCI host computer via the processor's host port interface (HPI). The on-chip memory is mapped starting at address 0. Applications programs are loaded via the HPI by the host while the card is in reset mode. Once the program is loaded, reset is deasserted by the host and the processor boots from on-chip RAM starting at address 0.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following figure gives the processor memory map of the M6x for external peripherals and memory. Please note that this table ignores any on-chip resources.

Function	Address	Memory Space
FIFOPort	0x400000	CE0
V360 Registers	0x1400000	CE1
FIFOPort Reset	0x1410000	
FIFOPort Enable	0x1420000	
OMNIBUS Control (reserved)	0x1430000	
External Mux Control 0	0x1440000	
External Mux Control 1	0x1450000	
AD9850 Reset	0x1470000	
AD9850 Frequency Update	0x1480000	
AD9850 Write Clock	0x1490000	
Digital I/O Data Register	0x14A0000	
Digital I/O Direction Control	0x14B0000	
Digital I/O Input Latch Clock Control Register	0x14C0000	
Transmit FIFOPort PEN* Mode	0x14D0000	
Receive FIFOPort Level Status	0x14D4000	
Transmit FIFOPort Level Status	0x14D8000	
16 bit PIT Timers	0x14F0000	
Interrupt Polarity Control (cM6x Rev. C+ only)	0x14F4000	
External Interrupt Input 4 Select	0x1500000	
External Interrupt Input 5 Select	0x1510000	
External Interrupt Input 6 Select	0x1520000	
External Interrupt Input 7 Select	0x1530000	
OMNIBUS Strobe 0	0x1540000	
OMNIBUS Strobe 1	0x1550000	
OMNIBUS Strobe 2	0x1560000	
OMNIBUS Strobe 3	0x1570000	
OMNIBUS Strobe 4	0x1580000	
OMNIBUS Strobe 5	0x1590000	
OMNIBUS Strobe 6	0x15A0000	
OMNIBUS Strobe 7	0x15B0000	
OMNIBUS Strobe 8 (cM6x only)	0x15C0000	
OMNIBUS Strobe 9 (cM6x only)	0x15D0000	
OMNIBUS Strobe 10 (cM6x only)	0x15E0000	
OMNIBUS Strobe 11 (cM6x only)	0x15F0000	
Async SRAM (128Kx32)	0x1600000	
SDRAM (16Mbyte) (optional)	0x2000000	CE2
SBSRAM (1Mbyte) (optional)	0x3000000	CE3

TABLE 13. M6x External Memory Map

M6x Hardware Initialization Requirements

The M6x design requires the following values to be written to its hardware control registers in order to provide access to on-board hardware:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Register	Address	Value
EMIF Global Control	0x01800000	0x00003069
CE1 Control	0x01800004	0x73E70F22
CE0 Control	0x01800008	0x30800520
CE2 Control	0x01800010	0x00000030
CE3 Control	0x01800014	0x00000040
SDRAM Control	0x01800018	0x07117000
SDRAM Refresh	0x0180001C	0x00000618
Interrupt Polarity	0x019C0008	0x0000000F

TABLE 14. M6x Bus Control Register Initialization Values

These values are initialized automatically by C programs compiled under the M6x Development Package software libraries. Be sure to include initialization of these values whenever software is developed outside the Development Package or when a JTAG hardware assisted debugger is employed for code downloading to the M6x. (i.e. when using Code Composer or any other JTAG debugger package)

External Memory

The M6x offers three types of external memory: asynchronous SRAM (ASRAM), synchronous DRAM (SDRAM), and synchronous burst SRAM (SBSRAM). The 128Kx32 ASRAM memory comes standard with the M6x, while the SBSRAM and SDRAM are optional.

ASRAM is used by the M6x as a buffer for bus master and slave data movement on the PCI bus. The ASRAM is accessible by the V360 PCI bus interface device, allowing the processor to setup bus master data transfers, which are handled as a DMA-style transfer by the V360. The M6x utilizes the 'C6x01's HOLD/HOLDA bus grant feature to provide ASRAM access to the V360. In addition, the ASRAM memory acts as a target for slave accesses by other PCI bus masters (either the host processor or other adapter cards).

The optional SBSRAM and SDRAM memories provide large, fast areas to store copious amounts of data or program information. The SBSRAM and SDRAM memories are not accessible by the PCI interface.

The 'C6x01 processor operates in big endian addressing mode, allowing 8, 16, and 32 bit wide data movement to and from external SBSRAM and SDRAM memory. Async SRAM supports 32-bit accesses only, as does the V360 PCI bus interface.

M6x OMNIBUS

The M6x I/O bus provides a modular, high-speed expansion area which is directly tied to the processor's bus and which is ideally suited for I/O hardware expansion. Direct memory-mapped accesses allow the processor to transfer data to and from I/O bus peripherals constructed as plug-in modules, which can be mixed and matched to suit the particular user's functional requirements.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The OMNIBUS slots are accessed as memory-mapped peripherals with the M6x providing four decoded chip select signals per slot. The following figure gives the memory map for the OMNIBUS slots, and shows the decoded signal to slot mapping.

Function	Starting Address	Module Slot
OMNIBUS Strobe 0	0x1540000	0
OMNIBUS Strobe 1	0x1550000	0
OMNIBUS Strobe 2	0x1560000	0
OMNIBUS Strobe 3	0x1570000	0
OMNIBUS Strobe 4	0x1580000	1
OMNIBUS Strobe 5	0x1590000	1
OMNIBUS Strobe 6	0x15A0000	1
OMNIBUS Strobe 7	0x15B0000	1
OMNIBUS Strobe 8 (cM6x only)	0x15C0000	2
OMNIBUS Strobe 9 (cM6x only)	0x15D0000	2
OMNIBUS Strobe 10 (cM6x only)	0x15E0000	2
OMNIBUS Strobe 11 (cM6x only)	0x15F0000	2

TABLE 15. M6x I/O Bus Memory Mapping

Each module site provides a 32-bit wide data bus connection to the processor's data bus, with 12 bits of low-order address signals for additional decoding beyond the four chip select signals available per slot. Each module also connects to a 'C6x01 serial port (serial port zero for slot zero, and serial port 1 for slots 1 and 2) to allow serial port driven I/O. Bus reset, RDY, R/W, and processor clock signals are available, as are power connections for digital 5V and analog +-5V and +-15V. Timebase connections include timer channels from both the custom 16-bit timers and the 9850 direct-digital synthesizer.

Each OMNIBUS slot has a 50 pin undedicated connector (JP17 on slot 0, JP21 on slot 1, and JP32 on slot 2) for use in providing external I/O to/from a module installed in the slot. The slot's I/O connector is in turn pinned out to a 50 pin .100" square double row header (JP18 for slot 0, JP22 for slot 1) on the M6x and to 50 pin mini SCSI style connectors on the cM6x (JP18 for slot 0, JP22 for slot 1, and JP33 for slot 2). The M6x also provides 15 pin external connectors for each slot which allow the highest numbered 15 signals on the header connectors to be pinned out external to the host computers chassis.

Connector pinouts for the module sites are provided in the appendices. Individual pin functions are noted in the tables, and in general the OMNIBUS pinout represents a direct connection to the 'C6x01 local bus.

M6x OMNIBUS Memory Mapping

Since the 'C6x01 processor is a byte addressable machine, which implements its address bus based on a 32-bit transfer width (i.e. the address bus starts at A2 and separate byte enable pins are supplied to control accesses to individual bytes within the 32-bit wide location denoted by the address bus), users must take care when writing software which performs OMNIBUS accesses.

The OMNIBUS specification requires 32-bit accesses and does not support byte or half-word (16-bit) accesses. No support is included in the specification for the 'C6x01's byte enable pins. This means that software performing accesses must always perform 32-bit transactions with the OMNIBUS modules.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When writing C code for the M6x, programmers should use only variables of type int or unsigned int (or their derived types). All accesses should be word justified (the least significant nibble of the address must always be a multiple of four). Accesses generated using pointers to variables of type char, short, or long will cause erroneous non-32-bit accesses. Correct OMNIBUS module operation under these situations is not guaranteed.

It should be noted that memory decoding within the OMNIBUS decode regions uses 32-bit addressing and that the memory map tables given in the *OMNIBUS Hardware Manual* should be treated appropriately. For example, the description of the OMNIBUS DIG module notes that the byte 3 direction control register for a module installed in site 0 is mapped to address IOMOD2 + 3. This address should be literally interpreted as 0x156000C, where IOMOD2 is equal to 0x1560000 and the offset adds decimal 12 (three 32-bit words of offset). IOMOD2 + 3 should NOT be interpreted as 0x1560003, since the offset is 3 32-bit words, not 3 bytes.

This addressing is most easily handled in C by using integer pointers and integer pointer arithmetic, which will always result in the required address alignment. For example, the following code defines a pointer and accesses the byte 3 direction control register with the documented offset:

```
unsigned int *pointer = 0x1560000;

*(pointer + 3) = 0x0;       /* set byte 3 to output mode */
```

The actual accessed memory location is 0x156000C, due to the way pointer math is handled in C.

OMNIBUS Power

The OMNIBUS interface provides five separate power supplies for use by modules along with two separate ground return connections. The following table lists the supplies and their power ratings. A separate digital 5V supply is provided along with separate digital grounds to minimize the digital noise present on the analog power supplies.

Pin Name	Voltage	Current Rating (max)
DVCC	5V (digital)	(System dependent)
+12	12V	(System dependent)
-12	-12V	(System dependent)
AVCC	5V (analog)	500 mA
-AVCC	-5V	500 mA
+AV	+15V	330 mA
-AV	-15V	330 mA

TABLE 16. I/O Bus Power Ratings

Please note that the AGND and DGND busses are separated on the M6x and for proper ground referencing they must be tied together on modules which use the analog power supplies (any supply other than digital 5V, 12V, or –12V). Innovative Integration recommends that a ferrite bead (Panasonic EXC-ELSA35V or equivalent) be used on custom modules to connect the two ground busses in order to prevent high frequency digital noise on the DGND bus from polluting the clean AGND return.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

FIFOPort I/O Expansion

The FIFOPort feature provides a buffered bidirectional 16-bit interface which allows external hardware or other M6x boards to communicate with the M6x at high data rates. A single input FIFO is provided to buffer incoming strobed parallel data, while a FIFOPort compatible output supports clocking data to external hardware or other FIFOPorts. Access to the 'C6x01 timer I/O pins is provided to support simple bit I/O requirements.

The following diagram illustrates the FIFOPort's operation. The FIFO buffer memory serves to clock incoming data and store it for use by the 'C6x01. Data is formatted as a 16-bit wide data bus synchronous with an rising edge strobe signal, which acts as the FIFO load clock. The output portion consists of the same two signals: output data plus the strobe signal for the receiving end of the port.

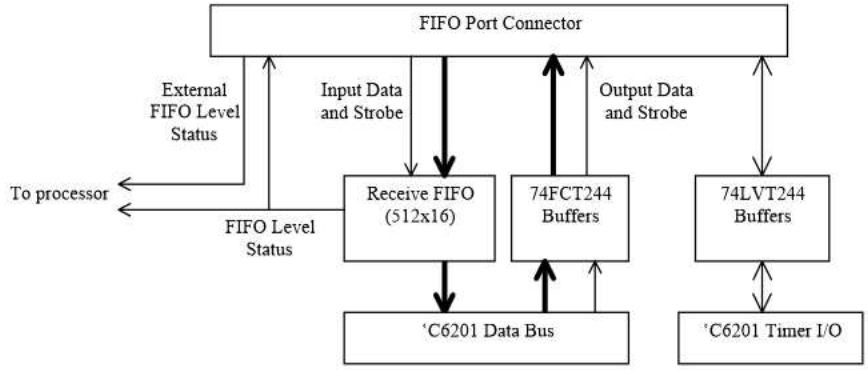


FIGURE 34. FIFOPort Block Diagram

The FIFOPort also provides external access to receive the FIFO's empty, full, and programmable almost full flags to allow hardware to monitor the FIFO's level status. The port can also receive FIFO level status from external hardware to allow the 'C6x01 processor to monitor level status of FIFOs located off the M6x card. Both the onboard receive FIFO level status and the off board FIFO status lines may be polled or may generate interrupts to the 'C6x01 processor.

In addition to the FIFO data management functions, access to the 'C6x01 timer I/O pins is provided to support simple bit I/O requirements. The timer I/O pins are buffered through LVT family logic buffers and driven on or off the card for use where individual bit I/O control is needed for the external hardware.

Also available on the FIFOPort connector is an external interrupt input, which is connected to the processor's interrupt switch matrix. This external interrupt input allows the 'C6x01 to receive an active low interrupt from external hardware.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Transmitting and Receiving FIFOPort Data

Data is transmitted and received on the FIFOPort by means of processor address location 0x400000. EMIF read and write accesses (due to either CPU or DMA activity) cause read and write strobes to be generated to the FIFOPort circuitry only when this address is accessed.

In the case of a write access, an active high output strobe is generated on the external connector and 16-bit bus data is driven out to the output bits. This data should be latched by external hardware on the rising edge of the FIFOPort output strobe. Write accesses do not affect the current state of the receive FIFO.

In the case of a read, an input read strobe is generated to the receive FIFO and its output data latched by the processor. If the data item being read in the current cycle is not the last item stored in the buffer, the next data item is clocked out by the FIFO and held ready for the next read access by the processor. Read accesses do not generate output strobes to the external connector.

Please note that the data returned by the FIFO on a read access is present on the least significant 16-bits of the processor's data bus. The most significant 16 bits are not driven and are not defined. If 32-bit CPU accesses are being used to read data from the FIFO, then the upper 16 bits of the result should be masked off before use. The DMA controller may be programmed for 16-bit access width and will automatically perform 16-bit to 32-bit data translation. Each stored 32-bit wide data item retrieved will be the concatenation of two 16-bit values read from the FIFO.

If the receive FIFO grows empty, the last data item's value will be output on any subsequent read accesses.

Monitoring FIFO Status

The FIFOPort provides a FIFO level monitoring feature, which allows software to read the receive FIFO's level indicators as well as FIFO level data from external hardware (if connected). The receive FIFO's empty, full, and programmable almost full flags can be read at any time by the CPU. The interrupt selection matrix may also be programmed to notify the CPU of level events via an interrupt (see Interrupts section for more information). The same functionality is provided for the external FIFO, allowing the CPU to read back or be interrupted by any of the six different level state conditions.

The FIFO level status is monitored using two registers, one for the receive FIFO and one for the transmit FIFO (if connected). The register bit definitions are given below.

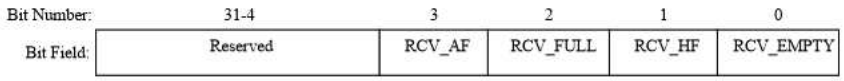


FIGURE 35. Receive FIFOPort Level Status Register

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Bit Field Name	Function
RCV_EMPTY	Receive FIFO Empty Flag (0 = empty, 1 = not empty)
RCV_HF	Receive FIFO Half-full Flag (0 = not half full, 1 = at least half full)
RCV_FULL	Receive FIFO Full Flag (1 = not full, 0 = full)
RCV_AF	Receive FIFO Almost-full Flag (1 = not almost-full, 0 = almost-full)

TABLE 17. Receive FIFOPort Level Status Register Definition

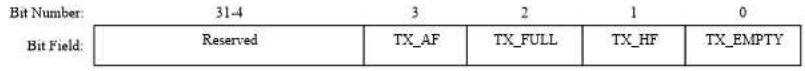


FIGURE 36. Transmit FIFOPort Level Status Register

Bit Field Name	Function
TX_EMPTY	Transmit FIFO Empty Flag (0 = empty, 1 = not empty)
TX_HF	Transmit FIFO Half-full Flag (0 = not half full, 1 = at least half full)
TX_FULL	Transmit FIFO Full Flag (1 = not full, 0 = full)
TX_AF	Transmit FIFO Almost-full Flag (1 = almost-full, 0= not almost-full)

TABLE 18. Transmit FIFOPort Level Status Register Definition

The receive FIFO level bits are read directly from the FIFO hardware on the corresponding FIFOPort, while the transmit FIFO bits are read from the level input pins on the FIFOPort connector. If no external status is being reported by the hardware connected to the FIFOPort, then these bits will read as ones (onboard 10K pullup resistors hold the transmit input pins high). If external FIFO level reporting is not desired, the level inputs may be used for application specific bit inputs to report other hardware status conditions or trigger interrupts on the M6x processor. Note that this is in addition to the dedicated timer I/O pins and the processor interrupt input pin on the FIFOPort connector, which remain available regardless of the use of the FIFO status inputs. The digital return levels given for the transmit FIFO assume connection to another 'C6x card manufactured by Innovative Integration, or to hardware emulating similar FIFO level reporting

Note: The receive FIFOPort must be reset after power-up initialization and before the first read of the status registers is attempted. The status register contents are indeterminate until the FIFO reset operation is performed. See the *FIFOPort Reset* section below for details on resetting the FIFOPort receive FIFO.

With appropriate programming, the FIFO levels may also be monitored using processor interrupts. The three status bits for each FIFO in each direction are available as sources to the interrupt selection matrix for each processor. This technique is typically used to drive DMA transfers to and from the FIFOPort, where one FIFO status interrupt triggers one or more transfers using DMA synchronization. Alternatively for CPU interrupts where the target CPU in a transfer wants to be interrupted when data (or space)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

is available in the FIFO. This would be typical of "one-shot" FIFO transfers, where a single full FIFO's worth of data is transferred at once. The receiving processor needs to be notified when the FIFO reached the full state so that a read operation on the other side of the FIFO may commence. For more information on using the FIFO levels to trigger interrupts to the 'C6x01 processors, see the Interrupts section.

FIFOPort Reset

The receive FIFO may be cleared and its condition reset at any time by accessing the FIFOPort reset register at address 0x1410000. The data written to the register is not critical: a write access of any data to this address will reset the FIFO. Upon reset, the FIFO levels are cleared, the flags change to reflect the FIFO empty status, and the programmable almost full control variables are reset to default values (see below for more information).

FIFOPort Enable

After a board reset or power up and prior to reading data from the receive side of the FIFOPort, software must enable data output by accessing the FIFOPort enable register at address 0x1420000. Either a read or write access to the register may be used to enable the FIFOPort. Data reads issued to the FIFOPort prior to enabling the port will clock buffered data out of the port (if any data is stored in the FIFO) but the data will not be read correctly by the processor.

Controlling the FIFOPort Programmable Almost-full Flag

In addition to the fixed function empty and full flags, the FIFOPort provides a programmable almost-full flag, which can be used to enable notification on partial FIFO transfer lengths. This feature is particularly suitable to DMA block transfers on the FIFOPort because it maximizes the transfer rates on both sides of the FIFO by keeping the buffer partially filled.

The almost-full flag operates as follows: given two initialization bytes (X and Y), the FIFO outputs an almost-full/almost-empty flag function, which is active whenever the FIFO contains X or less words of data or 512-Y or more words of data. By programming the X value equal to the almost-full level and the Y value to zero (0), the FIFO's programmable flag effectively becomes a variable partial full indicator. For example, programming the X variable to 128 and the Y variable to zero (0) yields a quarter-full output function.

The programmable almost-full flag control variables for the transmit half of the FIFOPort are initialized by enabling PEN mode, then writing the variables to the FIFOPort. PEN mode is enabled by writing a zero to the transmit FIFOPort PEN mode register at address 0x14D0000. The X variable is then written to the FIFOPort, followed by the Y variable, with both data values being 8-bits wide and right justified on the bus. The default values for the X and Y variables are both 64 (the FIFO reverts back to these values on a reset). Following the completion of the Y variable write, PEN mode should be disabled by writing a one to the PEN mode register. Please note that the almost-full flag variables may only be written immediately after a FIFO reset has been issued to the transmit side FIFO and before any data is written to the transmit FIFO.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Note: the above description of the PEN mode register operation was a change to the M6x control logic made in April 1999. Boards purchased earlier than this date should be returned to Innovative for an update. Please contact Innovative with questions concerning this feature.

Please note that this initialization operation only affects the transmit FIFO (i.e. the FIFO on the external hardware or other M6x or Quatro6x card). The FIFOPort architecture does not allow the onboard processors to initialize the programmable levels of the FIFOPort receive FIFOs. This initialization is always performed by the external hardware prior to writing data to the receive FIFO.

Timer I/O and the FIFOPort

The FIFOPort also provides a connection to the processor's timer I/O pins. This allows designers of hardware connecting to the FIFOPort easy access to four bits of unidirectional I/O for control purposes and status reporting. The on-chip timers of the 'C6x01 may be programmed to generate or receive clock and count events on the pins, or the pins may be used for general purposes bit I/O.

The M6x implements LVT family buffering between the timer I/O pins and the FIFOPort connector. Output and input levels are TTL compatible, but the outputs will not drive beyond 3.3V on the high side, and are tolerant of input voltages of up to 5V. This feature makes the FIFOPort timer I/O pins suitable for direct interfacing to 3.3V or 5V TTL compatible logic. Such logic families as HCT, LSTTL, FCT, ABT, and ACT may be directly connected to the FIFOPort timer I/O pins.

Designing External Hardware for use with the FIFOPort

Use caution when designing external hardware, which is to be connected to the FIFOPort. The signals present on the interface connector are extremely high speed and failure to handle them appropriately can cause functional problems with the FIFOPort as well as the M6x's onboard components. Innovative does not recommend driving cables directly as capacitive load and ringing issues can cause corruption of the transmitted data. FIFOPort connector pinouts have been provided in the appendices.

The M6x provides mechanical mount holes suitable for use in attaching daughterboard style printed circuit boards to the FIFOPort connector. The combination of the FIFOPort connector retention and the mount hole positioning allow designers to easily create interface modules for use in adapting the FIFOPort connector to external hardware.

The following diagram gives mechanical dimensions for a FIFOPort compatible daughter PC board.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

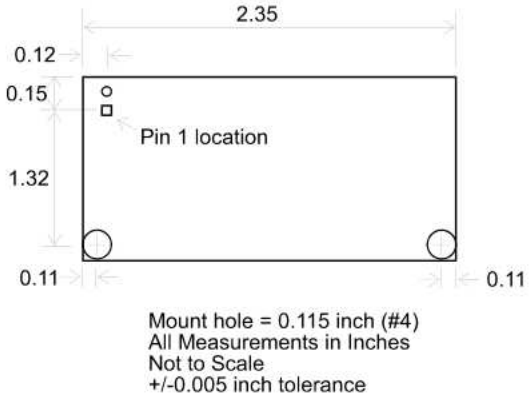


FIGURE 37. FIFOPort Daughterboard Mechanical Dimensions

FIFOPort Timing

The following diagrams give timing information for the FIFOPort circuitry. This data is derived from device specifications and is not factory tested.

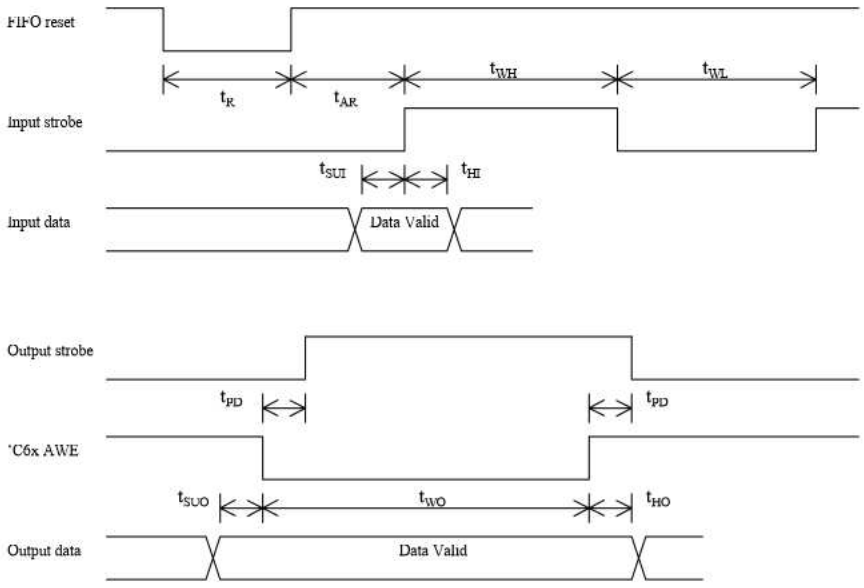


FIGURE 38. FIFOPort Timing

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Parameter	min (ns)	max (ns)
t_{WH}	7	
t_{WL}	7	
t_{SUI}	5	
t_{HI}	0	
t_{R}	10	
t_{AR}	5	
t_{PD}		7
t_{SUO}	10^{1}	
t_{HO}	0^{1}	
t_{WO}	10^{1}	

TABLE 19. FIFOPort Timing Parameters

Notes: Dependent on EMIF programming for CE0 space as well as processor cycle frequency. These values are determined from recommended EMIF register values.

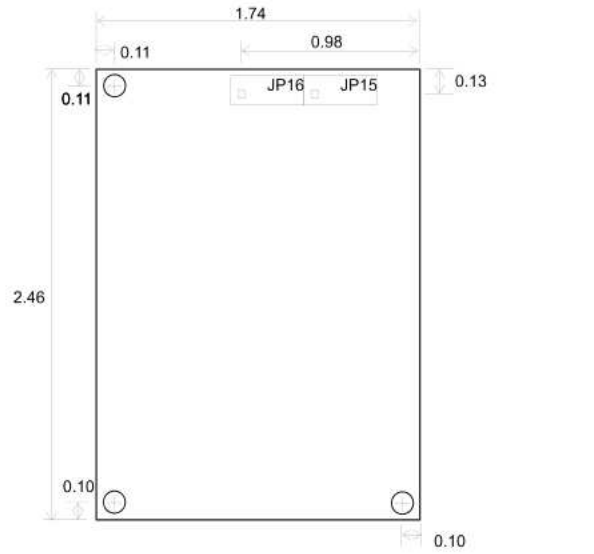
Serial Ports

The 'C6x01's on-chip serial ports are pinned out to connectors JP15 (port 0) and JP16 (port 1) for use with external hardware. The serial ports are also connected to the OMNIBUS slots for use with modules designed to interface to the processor serially.

Pinouts for the serial port connectors are given in the appendices. Innovative recommends buffering these ports with off board hardware in order to preserve signal integrity.

The following diagram shows the mechanical dimensions for a suggested printed circuit board outline for use in providing buffering of serial bus signals to external hardware.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Top view, serial port connectors facing down (mounted on opposite face of board)
Mount hole = 0.115 inch (#4)
Connectors arrayed at 2mm spacing (pin 1 locations shown)
All Measurements in Inches
Not to Scale
+/-0.005 inch tolerance

FIGURE 39. Serial Port Daughterboard Mechanical Dimensions

Timers

The M6x provides a total of six channels of independent timebase generation on board for use in timing data acquisition, servo controls, real-time counters, and other applications. The functionality is divided into three devices: two 32-bit timer channels on the 'C6x01 processor, three custom 16-bit timer channels in external logic, and a 32-bit direct digital synthesizer (DDS) channel in the AD9850 device. This section discusses the AD9850 and external timers: for more information on the on-chip timers, see the *TMS320C6x01 Peripherals Reference Guide*.

On-chip Timers

The on-chip timers are available for use as software timebases and interrupt generators. They are also pinned out to connector JP31 for use with external hardware. All four processor timer pins are available on JP31, allowing applications to use each timer as a time base output, an event counter input, or as bit I/O. The timer pins are also available on the FIFOPort connector for use in controlling external hardware attached to the FIFOPort. Refer to the appendix for pinout descriptions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

16-bit Timers

The M6x implements three, custom, 16-bit timers onboard, external logic which are capable of triggering processor interrupts and acting as clock sources for the I/O modules and external hardware. The timers provide readback capability for the current count register, which allows them to be used as digital event counters. Each channel may be driven either by an onboard 10 MHz clock source or by external clock input. In addition, external gating signal are available for each timer channel which allow an external TTL signal to selectively enable or disable the timer's clock input to control counting.

All three timers consist of 16-bit decrementing free-running counters with matching 16-bit period registers driven by a source timebase. The timers decrement once per input clock until they reach zero, whereupon they automatically reload from the period register and continue counting down. The timer output is normally high and falls low for one source clock cycle upon expiration of the counter value. The source clock may be selected from either a 10 MHz onboard timebase or an external input pin. The source clock is optionally gateable via a second set of gate inputs. The gating and clock input options allow the external timers to act as event counters for external hardware.

The following table shows the memory map for the timer control registers.

Function	Address
Clock Mode	0x14F0000
Channel 0 Period	0x14F0008
Channel 1 Period	0x14F000C
Channel 2 Period	0x14F0010
Channel 0 Count	0x14F0014
Channel 1 Count	0x14F0018
Channel 2 Count	0x14F001C

TABLE 20. External Timer Control Registers

The clock mode register controls the source of the clock used to drive each channel. Data bus bit D0 controls channel zero, D1 controls channel one, and D2 controls channel two. Writing a zero to a bit selects the onboard 10 MHz clock source as the timers clock input, while writing a one selects the external INCLKx inputs available on the digital I/O connector. The INCLKx inputs allow for each of the three channels to be driven by independent clocks from external hardware. For example, writing the hex value 0x2 to the clock mode register selects the 10 MHz clock as the source timebase for channels zero and two, while channel one is driven by the INCLK1 signal.

The timer period registers are used to store the period values for each timer channel. Each timer's output pulse period is equal to the period register value plus two source clock cycles. For example, if the clock mode register for channel zero was programmed to select the 10 MHz clock as the source clock for channel zero, and the period register were programmed with the value 98. Then timer channel zero's output pulse would occur with a period of (98 + 2) source clock cycles, or a frequency of

$$10\ \text{MHz}/(98 + 2) = 100\ \text{kHz}$$

The highest legal value for the period register is 65534 (yielding a lowest possible output frequency of 152 Hz when using the 10 MHz onboard source clock). Please note that a period register write causes

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

an immediate counter reload (i.e. the counter immediately starts counting down from the new period value).

The timer gate inputs allow external signals to control when the counter will decrement. Pulling the gate line low will disable clocking of the appropriate timer channel. The gate inputs are individually pulled up to 5V via a 10K resistor.

The timer output signals (TMR0, TMR1, and TMR2 for channels zero, one, and two respectively), input clock gating signals (GATE0, GATE1, and GATE2), and input clocks (INCLK0, INCLK1, and INCLK2) are all available for external connections on the digital I/O connector. Refer to the appendices for details on the pinouts.

Note: The INCLKx signals are buffered on the board using LVT or ALVC family logic buffers. These buffers have an input rise/fall timing requirement which must be met for proper operation. Specifically, signals must transition at a rate of 10 ns/V or faster to avoid inducing edge-related errors in the received signal. Slower edge rates may result in spurious extra clock edges received by the clock logic, which in turn will cause errors in the output frequency of the timers.

Driving the inputs with similar logic types (i.e. LVT or ACL family logic) is sufficient to meet this requirement. Comparators with adequate output slew rate may also be used externally to square up slow slew rate signals.

Using the External Timers as Digital Signal Counters

As described above, the external clock inputs on the 16-bit timers may be used to connect a signal to be used as a timebase by the timers. This feature, in concert with the counter value readback registers, allows the timers to be used as counters to automatically monitor external digital events. Each rising edge of a digital signal on the external input will cause the corresponding timer to advance, allowing the timer to count the external events on the input signal. The number of events counted is monitored by reading the current count value for the corresponding timer channel.

Setting up and operating the timers as counters is straightforward. The clock source for the timer(s) to be used as counters must be set to select the external input clock (i.e. set the clock select bit in the clock mode register to one for the corresponding timer channel). The counter period register must next be written with a value appropriate to the number of counts expected to be received. This value should be set larger than the number of events to be counted to avoid a rollover condition. This requirement is due to the fact that the timer channels use downward counting logic, i.e. the timer starts from the value in the period register and decrements the counter value once per rising edge of the input clock. The counter counts down through the value 0, rolls over to the value 65535 (the largest count value the 16-bit counter register can hold), then reloads from the period register to reset itself to the initial count value as it continues to count lock edges.

For example, if the period register is set to the value 100, the count sequence used by the counter would be as follows:

100

99

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

98

...

2

1

0

65535 -- counter rollover

100

99

...

The count value advances once per rising edge of the input signal. (The count values of 0 and 65535 are the reason for the period register set conditions detailed above, i.e. the actual timer period is always two greater than the period register value due to these "extra" two states).

To determine the number of events which have occurred, software must keep track of the starting value of the counter prior to the reception of the events. Once the events are received, the application then subtracts the new counter value from the starting value to determine the number of edges received.

In situations where large numbers of events must be monitored, or where the events are continuous over long time periods, software can compensate for rollovers of the external timers. This can be done by correcting in the count math for the rollover state. Note that in such a case software must poll the counter register at a rate of at least once per rollover to ensure that more than one rollover does not occur between reads of the count register. No provision is made in the hardware to count rollovers, so the true count value will be lost if more than one rollover occurs between counter reads (software will not be able to determine how many rollovers have occurred, so the number of received clock events is indeterminate).

Several additional details must be noted for correct counter operation:

1. Always observe the rise time restrictions on the external clock input pins. Presenting very slow clocks on the clock input pins may result in counting errors.

2. The counters have an initial startup delay of two clock edges each time a period register is loaded. Software will not read the period register value back from the counter register until two rising edges have occurred on the input clock after the period register write. Indeterminate data is returned for reads of the count register which occur before the two clock delay.

3. Count register values are not latched: reading the counter at the same time as a clock edge is advancing the value may result in invalid reads. In cases where the external events are asynchronous to the read process, it is recommended that two reads of the counter be performed to verify the counter value.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

AD9850 Direct Digital Synthesizer

The AD9850 direct digital synthesizer (DDS) is a precision programmable clock source, which is capable of generating frequencies in the range of 0 to 25 MHz with a resolution of 0.019 Hz/step. Unlike a digital counter-timer chip, which uses a digital counter to divide down a high input clock rate, the DDS uses phase-locked-loop synthesizer technology to tune a sine wave oscillator based on a 32-bit digital word. This method realizes a linear output frequency over input range rather than the nonlinear one associated with counter-timer chips, whose resolution drops dramatically as the period register used to program them falls. The counter-timer device has a nonlinear frequency step change over its input code range, as opposed to the DDS device, which maintains a linear frequency step for each input code increment. This results in the counter-timer's increased resolution at the high end of its input code range, with a correspondingly low resolution at the low end. The AD9850 timebase should be selected for use when a fairly fast but very precise and accurate clock is required by the application.

The AD9850 is mapped into I/O space as shown in the table below. The device is interfaced using the parallel I/O method, with an address to write data, one to trigger frequency/phase updates, and one to control the reset pin of the device.

Function	I/O Space Address
AD9850 Reset	0x1470000
AD9850 Frequency Update	0x1480000
AD9850 Write Clock	0x1490000

TABLE 21. AD9850 Control Registers

The write clock address latches frequency/phase data into the AD9850 one byte at a time. The least significant eight bits of the processor bus carry the bytewide data. The frequency update address causes the output frequency and phase of the DDS clock to update to the values contained in its input latches. The reset address causes an active high reset pulse to be generated to the AD9850. All three registers are write-only.

The M6x Development Package includes a routine (`timebase()`) which makes it easy to set the AD9850's output frequency.

Digital I/O

The M6x includes 32 bits of software programmable digital I/O for use in controlling digital instruments or acquiring digital inputs. The digital I/O port controls are mapped into memory space using three addresses: one to read/write the digital I/O data as a single 32-bit word, one for direction control for each byte of the port, and one for controlling the source of the clock edge used to latch input data into the digital I/O port register. The following table lists the addresses and their functions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Function	Address
Digital I/O Data Register	0x14A0000
Digital I/O Direction Control	0x14B0000
Digital I/O Input Latch Clock Control Register	0x14C0000

TABLE 22. Digital I/O Control Registers

The direction control register provides for software control of the drive direction of the port. The least significant four bits of the register control the four bytes available on the I/O port. Bit D0 sets the direction for the least significant eight bits if the port (port bits 0-7), D1 the next least significant bits (8-15), D2 the next least significant (16-23) and D3 the most significant (24-31). Each byte is individually controllable by writing a zero (to select output) or a one (to select input) to the respective bit in the direction control register. For example, if the value 0xC were written to the direction control register, bits 0-15 would act as inputs while bits 16-31 would act as outputs. All bytes default to input mode upon board powerup or reset.

The data register allows software to directly read data from port pins programmed for input, or write data to pins programmed for output. Read operations performed from the data register on port bytes programmed for output will return the current value of the digital I/O latch (i.e. the last value written to that portion of the port). For example, suppose that the direction control was programmed to 0xC and the data register written with the data word 0x12340000. Since the most significant 16 bits are setup as outputs, those pins on the port connector would assume the value 0x1234. A subsequent read of the port would yield the value 0x1234xxxx, where xxxx is the value of the signals present on the digital I/O connector.

The input latch clock register allows the user to select either software read clocking or external hardware clocking. Writing a zero to the register selects software clocking, while writing a one selects external hardware clocking. If software clocking is selected, then the port latches programmed for input will clock in the digital data present on the external pins at the beginning of a read cycle executed on the port data register (30-50 ns before the data is returned to the processor, depending on processor clock speed). If external clocking is selected, then the port will latch data on the falling edge of the TTL signal EXT_DIG_RD_CLK* on the digital I/O connector. The data will be held for the processor to read until the next low-going edge of the EXT_DIG_RD_CLK* signal. In the external hardware clocking mode, read operations by the processor do not affect the contents of the digital I/O latch. The latched data may be reread as many times as is required, and only another EXT_DIG_RD_CLK* pulse will cause new data to be latched into the port.

The 'FCT16952 devices used to implement the digital I/O port are capable of sourcing 32 mA and sinking 64 mA per pin.

Digital I/O Timing

The following diagram gives timing information for the digital I/O port when used in external readback clock mode (see above for details). This data is derived from device specifications and is not factory tested.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

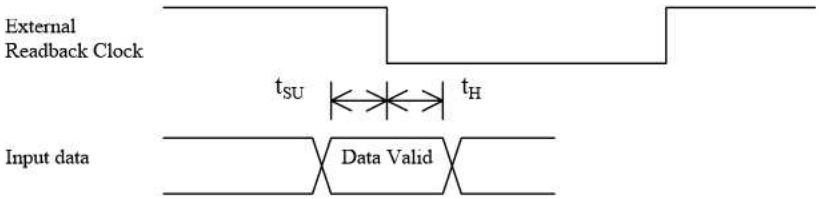


FIGURE 40. Digital I/O Port Timing

Parameter	min (ns)
t_{SU}	0
t_H	10

TABLE 23. Digital I/O Port Timing Parameters

External Mux Control

The M6x provides two external multiplexer control bus connectors for use with the TERM line of external multiplexer boards. Control for the multiplexer connectors is provided at the addresses listed in the following table.

TERM Module	Function	I/O Space Address
0	Mux #0 Channel Select	0x14D0000
	Mux #1 Channel Select	0x14D0004
	Mux #2 Channel Select	0x14D0008
	Mux #3 Channel Select	0x14D000C
	All Muxes Channel Select	0x14D0010
	Reset	0x14D001C
1	Mux #0 Channel Select	0x14E0000
	Mux #1 Channel Select	0x14E0004
	Mux #2 Channel Select	0x14E0008
	Mux #3 Channel Select	0x14E000C
	All Muxes Channel Select	0x14E0010
	Reset	0x14E001C

TABLE 24. TERM Function Memory Map

The control connectors (JP25 for TERM module 0 and JP26 for TERM module 1) select multiplexer channel numbers. The first four addresses from the start of each mux control address map allow the selection of incoming signals on each multiplexer device on the TERM. The fifth address location allows the simultaneous selection of the same channel on all multiplexer devices. The remaining address performs a global reset of the TERM hardware.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to the *OMNIBUS Manual* for additional information regarding the use of Innovative's TERM modules with the M6x.

Interrupts

The 'C6x01 processor implements four interrupt input pins, which allow external hardware events too directly trigger software activity. Processor interrupt inputs are supported on the M6x through a set of control registers and multiplexers, which allows application software to dynamically select the source of the signal which will drive each particular interrupt input.

The available interrupt source signals are as follows:

1. External interrupt input pins 0-5 (from the OMNIBUS modules).
2. External timer channels 0-2.
3. 9850 direct digital synthesizer clock.
4. PCI bus interrupt.
5. Various receive and transmit FIFOPort level status.

The following table shows the addresses of the control registers for each processor interrupt input. A value written to the appropriate control register causes the interrupt mux to select the interrupt source given in the next two table (see below). Note that the selections vary depending on which interrupt input is being programmed.

Function	Address
External Interrupt Input 4 Select	0x1500000
External Interrupt Input 5 Select	0x1510000
External Interrupt Input 6 Select	0x1520000
External Interrupt Input 7 Select	0x1530000

TABLE 25. External Interrupt Input Control Registers

Interrupt Control Register Value	Interrupt Source
0	External Interrupt Input 0
1	External Interrupt Input 1
2	External Interrupt Input 2
3	External Interrupt Input 3
4	External Timer 0
5	External Timer 1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6	External Timer 2
7	9850 DDS Clock
8	PCI bus
9	Receive FIFOPort empty
10	Receive FIFOPort half full
11	Receive FIFOPort full
12	Receive FIFOPort almost-full
15	Deactivated (interrupt held high)

TABLE 26. Interrupt Source 4 and 5 Select Register Values, M6x

Interrupt Control Register Value	Interrupt Source
0	External Interrupt Input 0
1	External Interrupt Input 1
2	External Interrupt Input 2
3	External Interrupt Input 3
4	External Timer 0
5	External Timer 1
6	External Timer 2
7	9850 DDS Clock
8	PCI bus
9	Transmit FIFOPort empty
10	Transmit FIFOPort half full
11	Transmit FIFOPort full
12	Transmit FIFOPort almost-full
15	Deactivated (interrupt held high)

TABLE 27. Interrupt Source 6 and 7 Select Register Values, M6x

Interrupt Control Register Value	Interrupt Source
0	External Interrupt Input 0
1	External Interrupt Input 1
2	External Interrupt Input 2
3	External Interrupt Input 3
4	External Timer 0
5	External Timer 1
6	External Timer 2
7	9850 DDS Clock
8	PCI bus
9	Receive FIFOPort empty
10	Receive FIFOPort half full
11	Receive FIFOPort full
12	Receive FIFOPort almost-full
13	External Interrupt Input 4
14	External Interrupt Input 5
15	Deactivated (interrupt held high)

TABLE 28. Interrupt Source 4 and 5 Select Register Values, cM6x

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Interrupt Control Register Value	Interrupt Source
0	External Interrupt Input 0
1	External Interrupt Input 1
2	External Interrupt Input 2
3	External Interrupt Input 3
4	External Timer 0
5	External Timer 1
6	External Timer 2
7	9850 DDS Clock
8	PCI bus
9	Transmit FIFOPort empty
10	Transmit FIFOPort half full
11	Transmit FIFOPort full
12	Transmit FIFOPort almost-full
13	External Interrupt Input 4
14	External Interrupt Input 5
15	Deactivated (interrupt held high)

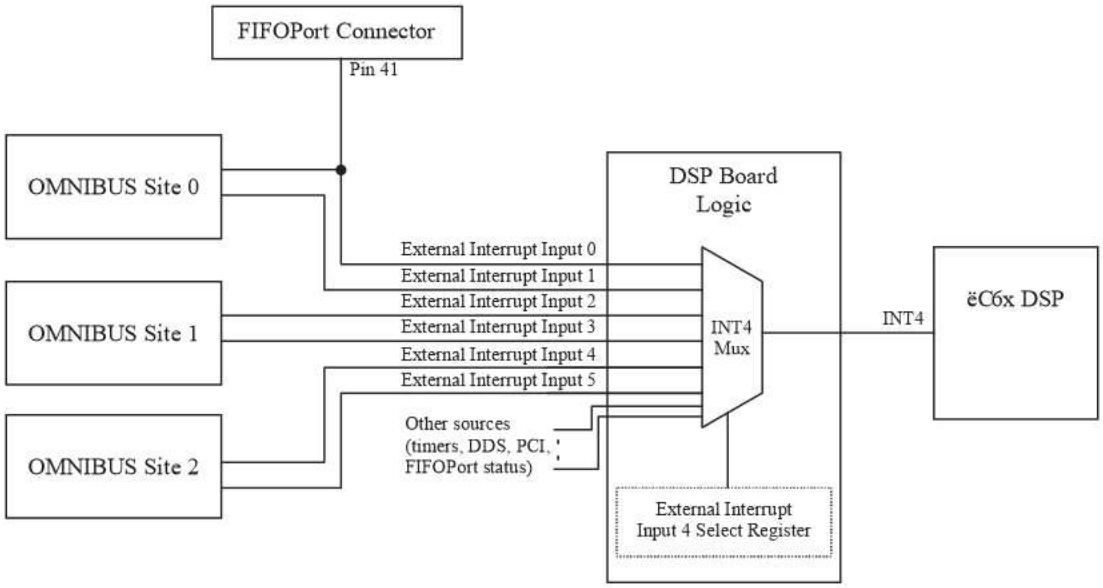
TABLE 29. Interrupt Source 6 and 7 Select Register Values, cM6x

For example, if the application requires the output from external timer channel two to drive processor interrupt input five, the value six should be written to memory location 0x1510000. All interrupt control registers default to setting 15 (disabled) on powerup or board reset. The processor interrupt signals generated by the logic are active low (falling edge trigger), and the `C6x01 External Interrupt Polarity Register must be set to the value 0xF to correctly receive interrupts. The four bits (0 - 3) of the Polarity Control Register correspond to the four external interrupts (0 - 3).

Note: Two of the Receive FIFOPort flags require special attention. The Empty flag is low when the receive FIFO is empty and becomes high when the first data point is received. Likewise, the Half Full flag is low when the FIFO is less than half full and high when the FIFO becomes half full. (See "Receive FIFOPort Level Status Register Definition" table in FIFOPORT I/O Expansion section.) This polarity is opposite to that described in the previous paragraph. In order to receive an interrupt for either of these events, the bit for the assigned external interrupt must be inverted (set to 0) in the Polarity Register. For more information on the Polarity Register refer to the TI Peripherals Manual.

The following diagram illustrates the flow of interrupt signals from the various source locations through the logic to the processor. The diagram shows the handling circuitry for the INT4 signal only: the other three interrupt inputs are handled in similar fashion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



The diagram shows the routing of the external interrupt input signals from the OMNIBUS and FIFOPort connectors through the INT4 multiplexer to the INT4 signal pin on the 'C6x DSP. The External Interrupt Input 4 Select Register controls operation of the INT4 mux and causes it to select one of the possible input source signals and drive the signal to the INT4 pin on the DSP. For clarity, the other possible source inputs to the INT4 mux have been abbreviated.

Note that the diagram shows the connection of the External Interrupt Input 0 signal to the FIFOPort at pin 41. This connection allows the signal to be driven by offboard hardware, allowing users to send a single interrupt signal to the DSP via the INT4 mux. Note also that since the connection is a direct short to the OMNIBUS site 0 interrupt signal pin, the FIFOPort may not have a signal connected to it when there is also a module installed in OMNIBUS site 0 which is actively driving the External Interrupt Input 0 node. Since most modules make use of this interrupt signal to notify DSP software of real-time events, it is generally not possible to install an OMNIBUS module in site 0 and connect a signal to FIFOPort pin 41. In the case where a customer needs to install a module in OMNIBUS site 0 and also needs to connect an external interrupt signal to the DSP board, Innovative recommends the use of the FIFOPort transmit status signal pins on the FIFOPort, since these may be programmed as interrupt sources to the DSP (see the FIFOPort section above for a discussion of the transmit status pins). Use care when attempting to connect an external signal to pin 41 of the FIFOPort connector, as an electrical conflict between the external signal driver and an installed OMNIBUS module may cause damage to the external hardware, the OMNIBUS module, or both.

The polarity of the signal driven to the processor (on rev C+ cM6x cards only) may be controlled through the use of the interrupt polarity control register. The figure below shows the control register bit definition.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Bit Number:	31-4	3	2	1	0
Bit Field:	Reserved	POL7	POL6	POL5	POL4

FIGURE 41. Interrupt Polarity Control Register

Bit Field Name	Function
POL4	INT4 polarity normal = 0 (default), inverted =1
POL5	INT5 polarity normal = 0 (default), inverted =1
POL6	INT6 polarity normal = 0 (default), inverted =1
POL7	INT7 polarity normal = 0 (default), inverted =1

TABLE 30. Interrupt Polarity Control Register Definition

The register defaults to drive active low (falling edge sensitive) interrupts to the processor. If a bit in the register is set high, the corresponding interrupt signal is inverted, effectively making it active high (rising edge sensitive). The four external interrupt input signals are controlled independently. Polarity control is convenient for those applications which are attempting to synchronize DMA transfers to external interrupt signals (the 'C6x01 processor provides polarity control on the INT4-7 pins only for CPU interrupts, while DMA synchronization is always rising edge sensitive).

JTAG Test Bus

The M6x implements a JTAG 1149.1-compatible scan path loop through the onboard 'C6x01, with connector compatible with the specification provided in the *TMS320C6x01 User's Guide*. When connecting a JTAG controller card cable (from an Innovative Integration Code Hammer debugger card, Texas Instruments XDS-510, or other vendor's JTAG hardware), the JP11 connector is used. A shunt should always be installed on jumper JP13 when the JTAG debugger is in use.

Note: the M6x design boots the 'C6x01 processor using the HPI boot mode. On device power up or reset, it is not possible to start JTAG debugger software until after the HPI boot process has been completed. The software will return with "cannot init target" error message if it is started after the processor has been released from reset, but before the processor has finished the boot process. Innovative Integration includes a small bootstrapping utility (BOOT.EXE) in the M6x Zuma Toolset which will bootload a small test application onto the M6x and which should be used prior to starting the debugger after a reset.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

M6x PCI Bus Features

The M6x uses the V360 PCI bus bridge chip, along with external glue logic and asynchronous SRAM, to implement its interface to the PCI bus. The V360 acts as a bridge chip to translate accesses from the PCI bus into accesses on the 'C6x01 bus. It also performs DMA style data transfers between PCI address space and the M6x's asynchronous SRAM. Access to the 'C6x01's HPI port through the V360 is used by host applications to bootload software into the 'C6x01.

PCI Bus I/O and Memory Map

The M6x uses the V360 base address registers and address apertures to map three sets of functionality into PCI bus I/O and memory space: the V360 internal registers, the async SRAM memory, and the processor's host port interface. Address assignments are made to the board via PCI configuration cycles on system powerup, or by the host operating system. The following descriptions of the addressed features assume a working knowledge of PCI plug and play technology as well as any host operating system support provided by the system in use. The M6x Development Package provides host drivers and access support, which is highly recommended to shorten software development time.

The V360's internal register set is mapped into I/O space on the PCI bus using base address register zero (PCI_IO_BASE in the V360 data sheet), and allows host access to all of the features of the bridge chip. Host accesses to the I/O space in which the registers are mapped result in slave responses from the V360 device. The lowest part of the register set also provides a convenient access point to the PCI configuration space registers of the device.

The async SRAM is mapped into host PCI memory using base address register one (PCI_BASE0 in the V3 literature). This allows the host processor to gain slave mode access to the SRAM memory on the M6x for data transfers, and allows other expansion boards to act as bus masters to the M6x for direct data transfers. Accesses made to the PCI mapping address by either the host processor or another bus master result in slave responses from the V360. An async SRAM access results in a HOLD/HOLDA arbitration request by the V360 device to the 'C6x01. The slave access will be held not ready until the 'C6x01 has dropped into hold mode and released access to the M6x's processor bus. The V360 then completes the required transfer between the PCI bus and the async SRAM and releases the HOLD request t the 'C6x01.

Please note that the 'C6x01 must be in a state where it is capable of releasing bus ownership to the V360 for the PCI bus access to complete normally. Do not set the NOHOLD bit in the EMIF Global Control Register prior to attempting slave accesses from the PCI bus. Also note that the base address register used to map the async SRAM into PCI bus space requests 32-bit address mapping, which means that 32-bit capable host software is required to access the async SRAM memory.

The 'C6x01's HPI feature is also mapped into PCI bus memory, using the V360's base address register 2 (PCI_BASE1 in the V360 data sheet). The following table gives the mapping of the various HPI registers within the PCI bus address space.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PCI Bus BAR1 Offset	Function
0x0	HPIC
0x4	HPIC
0x8	HPIA low half word
0xC	HPIA high half word
0x10	HPID low half word, with address autoincrement
0x14	HPID high half word, with address autoincrement
0x18	HPID low half word, without address autoincrement
0x1C	HPID high half word, without address autoincrement

TABLE 31. HPI Port PCI Bus Mapping

Accesses within the PCI mapped HPI interface result in slave responses from the M6x. The various registers of the HPI interface are mapped as shown in the above table, and have the read/write limitations noted in the TMS320C6xxx Peripherals Reference Guide. The HPI interface allows access to both the standard HPID interface (without address auto-incrementing) and to the HPID mapping which causes the current address register to be incremented automatically with each data access.

The HPI port interface uses software ready monitoring to poll the current status of the interface. Host software must poll the status of the HRDY bit in the HPIC register to determine if a current access is finished and a new access may be started.

M6x Bootstrapping

The M6x processor operates in HPI boot mode and supports direct host access to the processor's HPI port via memory mapped registers on the PCI bus. This feature allows the host to access any 'C6x01 memory location and is intended for processor bootstrapping.

The 'C6x01 boot process involves the following steps:

1. Toggle the processor reset active, then inactive.

2. Via the HPI interface, place a bootstrap compatible code image in the processor's internal memory starting at address zero.

3. Once the code has been placed in processor memory, write a one to the DSPINT bit in the HPIC register to wake the CPU from the reset state. The processor will then begin software execution starting at address zero in internal memory.

Although the HPI MAP1 boot mode begins running software from onchip memory, it is possible to load code anywhere in offchip memory. This is provided that the bus control registers for the memory area in question are initialized prior to any writes via the HPI interface.

There is a complete host COFF compatible M6x bootload routine included in the M6x Development Package which facilitates 'C6x01 processor bootloading.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Multi-Card Timing Synchronization

The M6x has several features to support multi-card synchronization and clock sharing. Synclink is a simple TTL bus that allows the M6x to send or receive clock and trigger signals to other cards. Clock-Link is an LVDS input and output pair allowing the system to share high-speed clock and trigger signals. A variety of DSP and data acquisition products from Innovative feature SyncLink and ClockLink to allow the construction of multi-card synchronized sys-tems for channel expansion and common triggering.

The SyncLink bus has one master who is the originator of all clock and trigger signals, referred to as the master, while all other cards are referred to as slave devices. The slave cards can only receive clock/ trigger signals from the master. The following table below shows the signals that may be transmitted over the SyncLink bus when it is the master. The SyncLink signal selection register resides at 0x014E8000.

The ClockLink connections for input and output do not use the concept for a master and slave card. These signals are always on and must be wired in the cable to connect the inputs to the outputs.

Caution - connecting two ClockLink outputs together may cause damage to one or both.

If you are receiving the clocklink signal on a card other than an Innovative product, you must terminate the signal with 100 ohms, and use and LVDS receiver chip such as the Texas Instruments SN75LVDS179 (the 'LVDS179 also has one driver section for other uses). Twisted pair wire is recommended and the wire length should not exceed 2 feet.

Bit	Description
0	0 - Sync0 = timer0 - default at power on 1 - Sync0 = timer1
1	0 - Sync1 = timer0 - default at power on 1 - Sync1 = timer1
2,3	00 - ClockLink Out = DDS - default at power on 01 - not used 10 - ClockLink Out = Timer0 11 - ClockLink Out = Timer1

TABLE 32. SyncLink Signal Selection Register Bit Definitions

In the case of the SyncLink/ClockLink signals being received on a slave card, these signals may be used on the slave card in place of timer 0, timer1 and the DDS so that the timebase signals are synchronous with the master card. These selections are made as follows in the SyncLink selection register at 0x014E4000.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Bit	Destination of Synclink Signals and Master Control
0	0 - synclink is master (sends signals) - default at power on 1 - synclink is slave (receives signals)
1	0 - timer0 = timer0 input - default at power on 1 - timer0 = synclink0 input
2	0 - timer1 = timer1 input - default at power on 1 - timer1 = synclink1 input
3	0 - DDS = DDS input - default at power on 1 - DDS = Clocklink in

TABLE 33. SyncLink Destination Selection Register Bit Definitions

The SyncLink master bit is controlled by the DSP at address 0x014E4000. Write a '1' to bit 0 of this register to allow the M6x to act as the SyncLink master. Only one master should be enabled between all the cards on the SyncLink bus to prevent conflicts.

The SyncLink signals are recommend for use below 1 MHz to a maximum of 8 TTL loads. Termination may be required depending on the cable and impedance of the loads to prevent signal ringing. More than eight loads may require additional buffering.

ClockLink is intended for higher speed signals that are usually card to card. The ClockLink allows the M6x to share clocks up to 80 MHz over short distances, or farther for slower signals. Flat ribbon cable performs well, but twisted pair cable is recommended for the best signal integrity. Note that the transmit/receive signal wire pairs must be crossed to connect two cards together.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

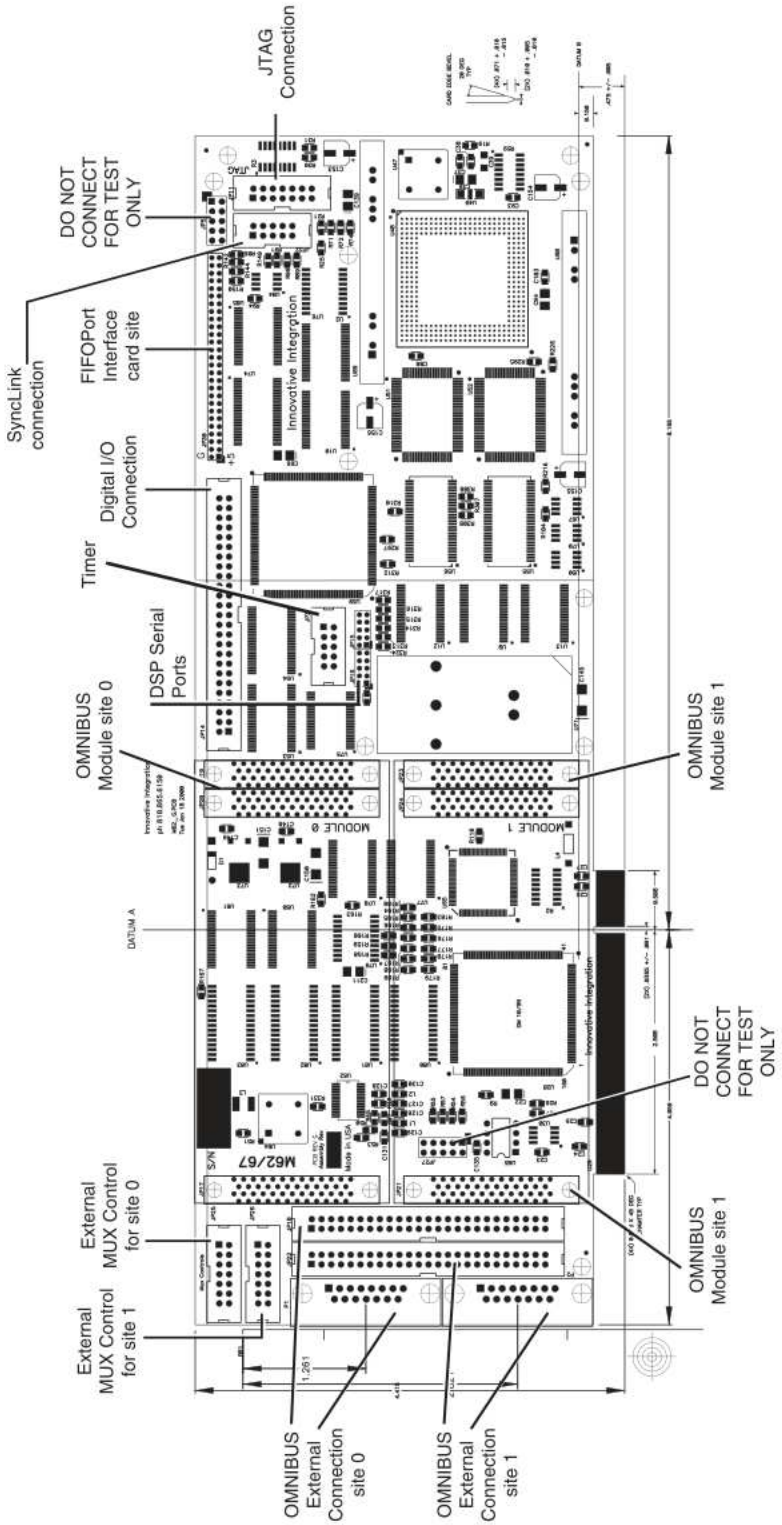
CHAPTER 12

Appendices

Board Layout

A schematic of the board layout is displayed on the following page. Please review this schematic to familiarize yourself with the circuit board's configuration.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Connector pinouts

JP17, JP18, JP21, JP22, P1, P2 - OMNIBUS I/O Connectors (M6x only)

Connector types: JP17, JP21: AMP .05 Subminiature D male

JP18, JP22: .100" header

P1, P2: Male DB15 connector

Number of pins: JP17, JP21: 50

JP18, JP22: 50

P1, P2: 15

Mating connector: JP17, JP21: AMP 173279-3

JP18, JP22: AMP 1-746285-0

P1, P2: AMP 747909-2

The following table shows the interconnections between the JP17 (OMNIBUS slot 0) and JP21 (OMNIBUS slot 1) module I/O connectors and their respective external I/O connectors, JP18 and P1 (OMNIBUS slot 0) and JP22 and P2 (OMNIBUS slot 1).

JP17, JP21 Pin Numbers	JP18, JP22 Pin Numbers	P1, P2 Pin Numbers
1-35	1-35	NA
36	36	1
37	37	9
38	38	2
39	39	10
40	40	3
41	41	11
42	42	4
43	43	12
44	44	5
45	45	13
46	46	6
47	47	14
48	48	7
49	49	15
50	50	8

TABLE 34. OMNIBUS I/O Connector Pinouts (M6x)

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

JP17, JP18, JP21, JP22, JP32, JP33 - OMNIBUS I/O Connectors (cM6x only)

Connector types: JP17, JP21, JP32: AMP .05 Subminiature D male

JP18, JP22, JP33: AMP Amplimite Series III

Number of pins: JP17, JP21, JP32: 50

JP18, JP22, JP33: 50

Mating connector: JP17, JP21, JP32: AMP 173279-3

JP18, JP22, JP33: AMP 750913-5

The following table shows the interconnections between the JP17 (OMNIBUS slot 0), JP21 (OMNIBUS slot 1), and JP22 (OMNIBUS slot 2) I/O connectors and their respective external I/O connectors, JP18 (OMNIBUS slot 0), JP22 (OMNIBUS slot 1), and JP33 (OMNIBUS slot 2).

JP17, JP21, JP32 Pin Numbers	JP18, JP22, JP33 Pin Numbers
1-35	1-35
36	36
37	37
38	38
39	39
40	40
41	41
42	42
43	43
44	44
45	45
46	46
47	47
48	48
49	49
50	50

TABLE 35. OMNIBUS I/O Connector Pinouts (cM6x)

The following diagram gives the physical pin locations for JP18, JP22, and JP33 connectors on the cM6x board. Please note that these physical pin positions do not use the same numbering scheme as standard SCSI 50 pin connectors.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

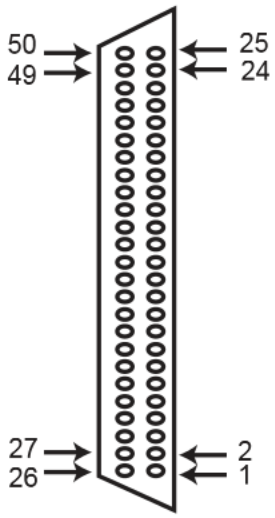


FIGURE 42. OMNIBUS I/O Connector Pin Configuration

JP19, 20, 23, 24, 34, 35 - OMNIBUS Bus Connectors

Connector types:	AMP .05 Subminiature D male
Number of pins:	50
Mating connector:	AMP 173279-3

The following table gives the pin numbers and functions for the JP19 (OMNIBUS slot 0), JP23 (OMNIBUS slot 1), and JP34 (OMNIBUS slot 2) (available only on the cM6x) connectors. The functions for JP23 and JP34 are identical to those of JP19, except where noted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Pin Number	JP19 Function	JP23 Function	JP34 Function (cM6x only)	Direction (from M6x)
1, 19	Digital +5V			O, power
2, 20	Digital ground			O, power
3-18	Data bus 0-15			I/O
21, 43, 40, 45, 39, 26, 27	Address bus 2-8			O
28	Reset (active low)			O
29	External interrupt 0	External interrupt 2	External interrupt 4	I
30	Bus ready (active low)			I (open-collector)
31	Processor CLKOUT2 / 4			O
32	PIT timebase channel 0			O
33	R/W*			O
34	9850 timebase			O
35-38	IOMOD0-3 decoded selects (active low)	IOMOD4-7 decoded selects (active low)	IOMOD8-11 decoded selects (active low)	O
25	-12V			O, power
23	+12V			O, power
41,42	Analog ground			O, power
22,24	Analog -15V			O, power
44,46	Analog +15V			O, power
47,49	Analog +5V			O, power
48,50	Analog -5V			O, power

TABLE 36. I/O Module Bus Connectors (cM6x)

The following table gives the pin numbers and functions for the JP20 (OMNIBUS slot 0), JP24 (OMNIBUS slot 1), and JP35 (OMNIBUS slot 2) (available only on cM6x) connectors. The functions for JP24 and JP35 are identical to those of JP20, except where noted.

Pin Number	JP20 Function	JP24 Function	JP35 Function (cM6x only)	Direction (from M6x)
1, 3-6	Address bus 9-13			O
2, 19, 20, 49, 50	Digital ground			O, power
7-18	Reserved	Reserved	Reserved	NA
21	PIT timebase channel 1			O
22	External trigger 0	External trigger 1	External trigger 1	O
23,25	+12V (from PCI bus)			O, power
24	CLKS0	CLKS1	CLKS1	I
26	CLKR0	CLKR1	CLKR1	I/O
27	FSR0	FSR1	FSR1	I/O
28	CLKX0	CLKX1	CLKX1	I/O
29	External interrupt 1	External interrupt 3	External interrupt 5	I

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

30	DR0	DR1	DR1	I
31	FSX0	FSX1	FSX1	I/O
32	DX0	DX1	DX1	O
33-48	Data bus 16-31			I/O

TABLE 37. I/O Module Bus Connectors (cM6x)

JP14 – Digital I/O Connector

Connector type: 0.100" square double row shrouded header

Number of pins: 50

Mating connector: AMP 1-746285-0

The following table gives the pin numbers and functions for the JP14 connector.

Pin Number	JP14 Function	Direction (from M6x)
1-32	Digital I/O bit 0..31	I/O
33	External Trigger 0 Input (active low)	I
34	9850 DDS Clock Output	O
35	External Trigger 1 Input (active low)	I
36, 38, 40	External Timer Ch. 0, 1, 2 Clock Outputs	O
37, 39, 41	External Timer Ch. 0, 1, 2 Gate Inputs	I
42, 44, 46	External Timer Ch. 0, 1, 2 Timebase Inputs	I
45	External Digital Readback Clock (active low)	I
47	Digital +5V	Power
49	Digital Ground	Power
43, 48, 50	Reserved	NA

TABLE 38. Digital I/O Connector

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

JP31 – Miscellaneous Digital I/O Connector

Connector type: 0.1” square header

Number of pins: 10

Mating connector: AMP 111811-1

The following table gives the pin numbers and functions for the JP31 connector.

Pin Number	JP31 Function	Direction (from M6x)
1	External Trigger 0 Input (active low)	I
2	External Timer 0 Clock Output	O
3	On-chip Timer 1 Out	O
5	On-chip Timer 1 In	I
7	On-chip Timer 0 Out	O
9	On-chip Timer 0 In	I
4,6,8	Reserved	NA
10	Digital Ground	Power

TABLE 39. Miscellaneous Digital I/O Connector

JP15, JP16 – Processor Serial Port Connectors

Connector type: 10 pin shrouded header

Number of pins: 10

Mating connector: AMP 746285-1 (for ribbon cable termination) or Samtec SQT style (for board-board applications)

The following table gives the pin numbers and functions for the JP15 and JP16 connectors. Pin functions of JP16 are identical to those of JP15 except where noted.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Pin Number	JP15 Function	JP16 Function	Direction (from M6x)
1	CLKS0	CLKS1	I
2	FSR0	FSR1	I/O
3	CLKR0	CLKR1	I/O
4	FSX0	FSX1	I/O
5	CLKX0	CLKX1	I/O
6	Digital 3.3V		Power
7	DR0	DR0	I
8	Digital 5V		Power
9	DX0	DX0	O
10	Digital Ground		Power

TABLE 40. Processor Serial Port Connector

JP11 – JTAG Debugger Connector

Connector type: 14 pin shrouded header

Number of pins: 14

Mating connector: AMP 746285-2

The following table gives the pin numbers and functions for the JP11 connector. This connector follows the recommendations given in section 13 of the *TMS320C62xx Peripherals Reference Guide*.

Pin Number	JP11 Function	Direction (from M6x)
1	TMS	I
2	TRST*	I
3	TDI	I
5	Digital +3V	Power
7	TDO	O
9,11	TCK	I
13	EMU0	I/O
14	EMU1	I/O
4, 6, 8, 10, 12	Digital ground	Power

TABLE 41. JTAG Debugger Connector

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

JP30 – FIFOPort Connector

Connector type: 2mm header

Number of pins: 54

Mating connector: Samtec SQW style

The following table gives the pin numbers and functions for the JP30 connector.

Pin Number	JP30 Function	Direction (from M6x)
1	Digital 5V	Power
2, 44	Ground	Power
3-18	Input Data Bits 15-0	I
19	Reserved	NA
20	Input Strobe	I
21	On-chip Timer 1 Out	O
22	On-chip Timer 1 In	I
23	On-chip Timer 0 Out	O
24	On-chip Timer 0 In	I
25-40	Output Data Bits 0-15	O
41	External Interrupt Input	I
42	Output Strobe	O
43	Digital 3.3V	Power
45	Half-full Flag Out	O
46	Almost-full Flag Out	O
47	Almost-full Level Control In	I
48	Full Flag Out	O
49	Almost-full Level Control Out	O
50	Empty Flag Out	O
51	Half-full Flag In	I
52	Almost-full Flag In	I
53	Empty Flag In	I
54	Full Flag In	I

TABLE 42. **FIFOPort Connector**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

JP32 - SyncLink

The SyncLink connector allows M6x to synchronize to external hardware or other Innovative cards.

Connector type: 0.1" double-row shrouded header

Number of pins: 10

Mating connector: AMP 111810-1

The following table gives the pin numbers and functions for the JP30 connector.

Pin Number	JP30 Function	Direction (from M6x)
1	Clocklink Out +	O
2	Clocklink Out -	O
3	Clocklink In +	I
4	Clocklink In -	I
5	Synclink bus pin 2	I/O
6	Digital ground	Power
7	Synclink bus pin 1	I/O
8	Digital ground	Power
9	Synclink bus pin 0	I/O
10	Digital ground	Power

TABLE 43. SyncLink Connector

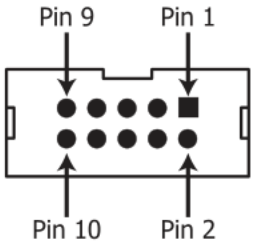


FIGURE 43. SyncLink Connector Pin Orientation

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TMS320C6x01 Limitations and Errata

As of this writing, the TMS320C6x01 processor has several limitations and errata that can affect the maximum clock rate at which the processor can successfully run and which may impede the proper operation of certain software applications. This section discusses limitations discovered in Innovative Intergration's testing of early M6x prototype cards, as well as errata announced by Texas Instruments regarding the current 2.0 revision silicon.

This information is being supplied to current customers and potential users of the M6x in an effort to keep you informed of the state of 'C6x01 processor development and any performance limitations imposed on the M6x hardware design. Innovative Intergration will continuously update this information as new data becomes available and particularly when new silicon revisions are released by Texas Instruments to us for testing.

Processor Speed Limitations and External Memory

The current revision 2.0 silicon 'C6x01 devices have bus timing issues which prohibit the use of full speed external synchronous burst SRAM (SBSRAM) and synchronous DRAM (SDRAM) devices. These limitations affect the maximum processor speed Innovative will be able to ship with current processors, dependent on the external memory configuration of the M6x hardware as ordered by the customer.

Specifically, Texas Instruments has announced that SDRAM support is limited to approximately 90MHz operation (180 MHz processor speed), while SBSRAM operation is limited to 133 MHz (133 MHz processor speed). These figures were determined through board level testing at the Texas Instruments facility.

Testing of hardware at Innovative Integration shows that SDRAM is supported at least through 80 MHz (160 MHz processor speed), while SBSRAM has been tested up to a rate of 80 MHz (160 MHz processor speed with SBSRAM in half-rate mode). Testing at a processor speed of 133 MHz with SBSRAMs running in full rate mode has shown a read/write failure rate of about 10-ppm. Texas Instruments has specified that not all SBSRAMs are reliable in their tests, so it is possible that the devices Innovative is currently using may not be up to spec. Innovative is in the process of procuring SBSRAM samples of the manufacturer and type used by Texas Instruments and will continue testing of the external memory interface to determine which devices are more reliable. In addition, Innovative will be procuring additional clock source devices, which will allow the testing of intermediate processor speeds (180 MHz, for example).

Current processors are capable of 200 MHz operation and have been tested at this rate on the M6x design. These tests involve running software strictly from on-chip memory. The external peripheral interfaces (I/O bus sites, serial ports, FIFOPorts, onboard peripherals, async SRAM, PCI interface) are unaffected by the memory interface issue as they use less aggressive bus timing.

Innovative Integration's policy on processor speed is to deliver the fastest possible speed consistent with the requested external memory configuration on the board. In situations where customers order external memory, Innovative must downgrade the processor speed to match the memory interface limitations. Current processor speeds available versus memory requirements are as follows:

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

External Synchronous Memory Type	Delivered Processor Speed
None	200 MHz
SDRAM	160 MHz
SBSRAM	160 MHz (SBSRAM operating in half-rate mode)

As Innovative continues testing the memory subsystems of the M6x, these rates may change to improve the memory access and processor speeds.

Texas Instruments Device Errata

The current Texas Instruments device errata for the most current revision silicon TMS32C6x01 devices is attached below. At this time Innovative Integration does not consider these errata to be significant to the overall operation of the card design.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Insert Texas Instrument errata files here:

6201SIERAT4

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Insert Texas Instrument errata files here:

6701SIERAT1_18_00.PDF

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

CHAPTER 13

Troubleshooting

Initialization Problems

The system does not recognize my board(s).

Follow the following steps:

1. Make sure each board is seated in a PCI slot correctly.
2. The device driver must be installed properly. Insure that the proper inf file (see the table for each board's inf file name) is located in the Windows INF folder and iixwdm.sys is located in the Windows System32\Drivers folder.
3. Each board must have an IRQ. Therefore, after booting up, verify that each board does have an IRQ.
   - To do this, bring up the Control Panel | System function. Click on the **Device Manager** tab/button and find the boards. Check the Properties | Resources tab for each board.
   - If you have conflicting IRQs, you will have to go into your Bios Setup at start-up and change the IRQ of your board(s).

TABLE 44.

Board	INF File name	Driver
ChicoPlus	ChicoDrvx2k.inf, ChicoDrvx9x.inf	iixwdm.sys, IIdrvX.vxd
Conejo	MatadorDrvx2k.inf, MatadorDrvx9x.inf	iixwdm.sys, IIdrvX.vxd
Toro	MatadorDrvx2k.inf, MatadorDrvx9x.inf	iixwdm.sys, IIdrvX.vxd

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 44.

Board	INF File name	Driver
Vista	MatadorDrvx2k.inf, MatadorDrvx9x.inf	iixwdm.sys, IIdrvX.vxd
M6x	M62Drvx2k.inf, M62Drvx9x.inf	iixwdm.sys, IIdrvX.vxd
Quatto6x	Q62Drvx2k.inf, Q62Drvx9x.inf	iixwdm.sys, IIdrvX.vxd
JTAG	JtagDrvx2k.inf, JtagDrvx9x.inf	iixwdm.sys, IIdrvX.vxd

Borland C++Builder Problems

The Builder Help Index system is blank.

Borland uses the standard Microsoft help system for its online help. Normally this works perfectly, however, Windows 9x Help system has a limitation on the amount of references it can use simultaneously. If you have too many links in your help index, you will not see any help at all on the index. However, if you are in Builder and you use the F1 functionality for your help, it will work perfectly.

To get your Help index working again, you will need to slim down the contents of the Open Help Index list found. Open Builder.

- Bring up Open Help by selecting *Help* | *Customize*.
- Click the **Index** tab. Remove files until your Index Help is working.
- Remember to Save the project before checking the Index Help.

To add Innovative help files if they did not automatically get added, follow the instructions above, except add help files. All help files can be found at *Program Files* | *Innovative* | *Documents* | *manuals*. The files to be added should **not** include files with 'Pismo' or 'Zuma' in their names as these are target side help files.

I created an EXE file and when I try to run it, the system requires a DLL which I don't have.

Depending on the settings your application has for building, it may require certain DLLs, such as **borlndmm.dll**. When you try to run your newly created executable, you may get an error such as "Dynamic link library borlndmm.dll can't be found". One cause of this is when you have the project set for dynamic rather than static linking of the Borland VCL packages. While dynamic linking can result in smaller EXEs, dynamic linking can result in dynamic link error messages, such as the one mentioned above, when a DLL is unavailable or not find-able by the application program at invokation.

We recommend static-binding of all executables. To do this:

- Bring up the Project Options menu by clicking the Project | Options menu in Builder. Select the **Linker** tab.
- Deselect the "Use dynamic RTL" option. This will force the inclusion of the DLLs that are required by your application.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Another possibility is that you have built your application with runtime packages. By building in this fashion, the executable requires certain BPLs to be in the system directory. To build your application without this dependency, click the **Packages** tab on your Project Options menu. Deselect "Build with runtime packages" option.

What DLLs do I have to deploy with my newly created executable?

The following DLLs must reside on the path for any deployed Vista application: Typically, these files are placed in the Windows system directory.

For Windows 9x, the system directory is the **C:\windows\system** directory. For Windows NT, it is **C:\Winnt\system32** directory.

Function	Required DLLs
Intel native signal processing libraries	nsp.dll, nspa6.dll, nspm5.dll, nspm6.dll, nspp6.dll, nsppx.dll, nspw7.dll
Intel native image processing libraries	ipl.dll, iplm5.dll, ipla6.dll, iplm6.dll, iplp5.dll, iplp6.dll, iplpx.dll, iplw7.dll
Innovative device driver DLL	iidrvx40.dll
Innovative Cali-ente DLL	CalienteDll6.dll (MSVC users runtime only)
Innovative Regis-tration DLL	UserRegister6.dll (BCB users design-time only)

How do I know what DLLs my executable is dependent on?

The Vista Applets\Third Party folder contains an archive call depends20_x86.zip which contains a utility called Depends.exe the provides a utility that allows you to determine the external dependencies of and Windows executable file. Clock File | Open from the menu bar and browse to the name of your application executable. The utility will list all DLLs on which your application is dependent in order to run. Note, however, that Windows programs are always dependent on Windows system DLLs, such as **Advapi32.dll, Kernel32.dll, Version.dll, Comctl32.dll, Gdi32.dll, User32.dll, Ole32.dll and Oleaut32.dll**. These dependencies cannot be eliminated, but will not cause a runtime error, since Windows systems allways provide these DLLs.

If the utility exposes DLL dependencies that you would like to remove, follow the steps in the question above "I created an EXE file and when I try to run it, the system requires a DLL which I don't have."

When I run my newly created application, the graphed data within my TIIGraph object stays red and doesn't graph.

Check the following in your application:

1. Verify that the graph input pins are connected to their data source. To verify, check the **Inputs** property array on the **TIIPlot** object has the correct source connected.

2. Make sure the **StreamActive** property on the is set to **true**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3. Make sure the Input and Output streams are linked to their appropriate endpoints. To do this, double click the IICaliente component on your form. Remember to link input and output if you have both.

4. Build and run the application.

DSP Hardware Problems

The I/O seems like it is not connected or doesn't work.

Double-check the connections to the I/O connector. The most common error is to connect to the wrong pins on the I/O connector.

How can I tell what version of logic I am using?

The logic version is reported by the VSProm applet. When the applet is opened, the current logic version number is displayed.

ChicoPlus Hardware Problems

Updating the Logic without the VSProm Loader Applet

Note: VSProm is also known as the Chico Logic Programmer Applet.

It may be necessary to update the with new logic should a bug be found, or new feature added. This is easily accomplished in minutes under normal circumstances using the VSProm application described in the applet chapter of this manual. This program can in-system reprogram the FLASH memory that holds the logic file. The PC must be powered off then on to for the logic to take effect. You will be prompted for this at the end of the reprogramming.

However, VSProm may only be used on cards that are behaving normally in the PC. This is because the reburning of the logic FLASH memory uses the card's own PCI interface logic to control the card. If the PCI interface is not operational, VSProm cannot be used to update the logic. This could occur if the PC crashes or if power is lost during a FLASH burn operation. Use caution when reburning the flash for the short 2-3 minutes it is being reburnt to help avoid these situations.

In the case where the is not working in the PC, the FLASH may be pulled from its socket and reprogrammed using any FLASH programmer. The standard FLASH memory used is the AMD 29F040 or compatible. All logic updates are distributed as text files, in Motorola S-Record format, with an EXO extension. The logic is burnt into the flash starting at address 0 and consumes the entire flash.

Caution: The FLASH memory is static sensitive. Use proper precautions against static electricity damage.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Caution: The FLASH memory socket may be damaged by undue force used for extraction. *Carefully and gently,* pry under alternate corners of the chip (not the socket) to remove the chip from the socket. A unbent large paper clip wire works well if you do not have a proper tool.

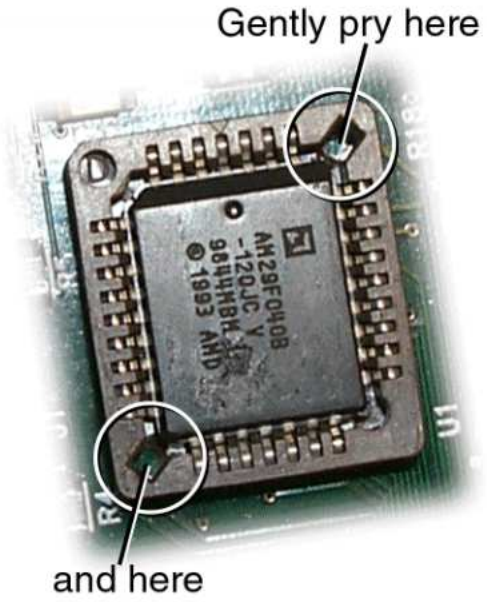


FIGURE 44. Removing the FLASH

If you do not have a FLASH programmer, or are otherwise unable to burn the FLASH, contact Innovative Integration Technical Support for assistance.

The I/O seems like it is not connected or doesn't work.

Double-check the connections to the module I/O connector. The most common error is to connect to the wrong pins on module connector. As described in the paragraph "OMNIBUS I/O Connector (JP4)", the pins on the module have a mapping to the end connector which means the pin numbers do NOT correspond one-to-one. The module pin numbers in the Omnibus hardware manual are the pin numbers on the mating connectors on the card itself. These internal pins are mapped to the SCSI-2 type connector on the card end for user connections according to the Omnibus SCSI-2 mapping table in the "Hardware" chapter.

The timer frequencies I can generate do not allow me to sample at the rates I need.

The timers are a simple count-down from the timer source selected. For many of the examples provided this means that the available clock rates are rather coarse at high sample rates. For example, if you divide the standard timer crystal, which is 49.152 MHz, the first few frequencies available are as shown here.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

TABLE 45. Possible Timer Frequencies for Input = 49.152 MHz

N	Output frequency
2	12.288 MHz
3	8.192 MHz
4	6.144 MHz
5	4.9152 MHz
6	4.096 MHz

As can be observed, the first few frequencies are particularly widely spread. The best way to solve this is to either replace the crystal so that it is an integer multiple of the desired timer rate, or use the external clock input (J1) to the card and drive it using a bench top signal generator.

If you use a benchtop signal generator, remember that the input clock should be made TTL signal levels (zero < 0.7 V, one >2.4 V) and must drive a 50 ohm input impedance. The signal source may the easiest way during the experimental phase of the project, then a crystal may be procured for the exact frequency desired when the application is working and ready to field.

On ChicoPlus and Hombre, there is an additional DDS clock source that does not suffer form this defect, allowing precise frequency generation up to 25MHz. On these baseboards, the DDS should be preferred for high rate time bases instead of using Timer 0 or Timer 1.

How can I tell what version of logic I am using?

The logic version is reported by the VSProm applet. When the applet is opened, the current logic version number is displayed.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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