GX3700/GX3700e
GX3701/GX3702/GX3788

User Configurable FPGA and Expansion Boards

GXFPGA Software

User's Guide

Last Updated: December 16, 2016

MARVIN TEST SOLUTIONS

Safety and Handling

Each product shipped by Marvin Test Solutions is carefully inspected and tested prior to shipping. The shipping box provides protection during shipment, and can be used for storage of both the hardware and the software when they are not in use.

The circuit boards are extremely delicate and require care in handling and installation. Do not remove the boards from their protective plastic coverings or from the shipping box until you are ready to install the boards into your computer.

If a board is removed from the computer for any reason, be sure to store it in its original shipping box. Do not store boards on top of workbenches or other areas where they might be susceptible to damage or exposure to strong electromagnetic or electrostatic fields. Store circuit boards in protective anti-electrostatic wrapping and away from electromagnetic fields.

Be sure to make a single copy of the software diskette for installation. Store the original diskette in a safe place away from electromagnetic or electrostatic fields. Return compact disks (CD) to their protective case or sleeve and store in the original shipping box or other suitable location.

Warranty

Marvin Test Solutions products are warranted against defects in materials and workmanship for a period of 12 months. Software products and accessories are warranted for 3 months. Unless covered by software support or maintenance agreement. Marvin Test Solutions shall repair or replace (at its discretion) any defective product during the stated warranty period. The software warranty includes any revisions or new versions released during the warranty period. Revisions and new versions may be covered by a software support agreement. If you need to return a board, please contact Marvin Test Solutions Customer Technical Services department via http://www.marvintest.com/magic the Marvin Test Solutions on-line support system.

If You Need Help

Visit our web site at http://www.marvintest.com for more information about Marvin Test Solutions products, services and support options. Our web site contains sections describing support options and application notes, as well as a download area for downloading patches, example, patches and new or revised instrument drivers. To submit a support issue including suggestion, bug report or question please use the following link: http://www.marvintest.com/magic

You can also use Marvin Test Solutions technical support phone line (949) 263-2222. This service is available between 7:30 AM and 5:30 PM Pacific Standard Time.

Disclaimer

In no event, shall Marvin Test Solutions or any of its representatives be liable for any consequential damages whatsoever (including unlimited damages for loss of business profits, business interruption, loss of business information, or any other losses) arising out of the use of or inability to use this product, even if Marvin Test Solutions has been advised of the possibility for such damages.

Copyright

Copyright © 2003-2016, by Marvin Test Solutions, Inc. All rights reserved. No part of this document can be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Marvin Test Solutions.

Trademarks

ATEasy®, CalEasy, DIOEasy®, DtifEasy, WaveEasy	Marvin Test Solutions (prior name is Geotest - Marvin Test Systems Inc.)
Quartus	Altera Corporation
C++ Builder, Delphi	Embarcadero Technologies Inc.
LabVIEW, LabWindows™/CVI	National Instruments
Microsoft Developer Studio, Microsoft Visual C++, Microsoft Visual Basic, .NET, and Windows	Microsoft Corporation

All other trademarks are the property of their respective owners.

Table of Contents

Safety and Handling....i
Warranty....i
If You Need Help....i
Disclaimer....i
Copyright....i
Trademarks....ii
Table of Contents....iii
Chapter 1 - Introduction....1
Manual Scope and Organization....1
Manual Scope....1
Manual Organization....1
Conventions Used in this Manual....2
Chapter 2 - Overview....3
Introduction....3
Features....3
Applications....4
Board Description....5
Architecture....7
Memory....8
PXI/PXIe and PC Connections....10
Inter-FPGA Bus Interface Timing....11
DMA FIFO Interface Timing....12
Specifications....13
Digital I/O Channel....13
Expansion Board Interface....13
Timing Source....13
User FPGA....14
Power....14
Environmental....14
Virtual Panel Description....15
Virtual Panel Initialize Dialog....16
Virtual Panel Setup Page....17
Virtual Panel I/O Page....18
Virtual Panel DAQ Page (GX3788)....19
Virtual Panel About Page....21

Chapter 3 - Installation and Connections 23
Getting Started 23
Interfaces and Accessories 23
Packing List 23
Unpacking and Inspection 23
System Requirements 24
Installation of the GXFPGA Software 24
Setup Maintenance Program 24
Overview of the GXFPGA Software 25
Installation Folders 25
Configuring Your PXI System using the PXI/PCI Explorer 26
Board Installation 27
Before you Begin 27
Electric Static Discharge (ESD) Precautions 27
Installing a Board 27
Plug & Play Driver Installation 29
Removing a Board 29
GX3701 Connectors 30
GX3701 J1 – Flex I/O Connector 31
GX3701 J2 – Flex I/O Connector 32
GX3701 J3 – Flex I/O Connector 33
GX3701 J4 – Flex I/O Connector 34
GX3788 Connectors 35
GX3788 J1 – Flex I/O Bank A Connector 35
GX3788 J2 – Flex I/O Bank D Connector 36
GX3788 J3 – Flex I/O Bank B Connector 37
GX3788 J4 – Flex I/O Bank C Connector 38
Jumpers 39
Chapter 4 - Programming the Board 43
The GXFPGA Driver 43
Programming Using C/C++ Tools 43
Programming Using Visual Basic and Visual Basic .NET 43
Programming Using Pascal/Delphi 43
Programming GXFPGA Boards Using ATEasy® 44
Programming Using LabVIEW and LabVIEW/Real Time 44
Using and Programming under Linux 44
Using the GXFPGA driver functions 45

Initialization, HW Slot Numbers and VISA Resource 45
Board Handle 46
Reset 46
Error Handling 46
Driver Version 46
Programming Examples 46
Distributing the Driver 46
Chapter 5 - GXFPGA Schematic Entry Tutorial 47
Introduction 47
Downloading Altera Design FPGA Design Tools 47
Create New Project 48
Device Selection 48
Pin Assignment Setup 49
Pin Assignments Table 49
Schematic entry project 51
Creating Design File with Schematic Entry 53
Phase 1: Creating the FPGA design - 32 bit Full Adder 53
Components Used 53
Schematic view 54
Design 55
Phase 2: Creating the FPGA Design - 2 to 1 Clock Mux 66
Components Used 66
Design 66
Phase 3: Creating the FPGA Design - 32 bit Dynamic Digital Pattern Sequencer 67
Components Used 67
Design 67
Configure Project to Output SVF and RPD Files 71
Compile an Example Project and Build RPD and SVF Files 73
Simulating the Design 75
Load Gx3700 with SVF File 78
Testing the Design 79
Adder Testing 79
Clock Mux Testing 79
Digital Sequencer Testing 80
Chapter 6 - GXFPGA Verilog Tutorial 81
Introduction 81
Downloading Altera Design FPGA Design Tools 81

Create New Project 82
Device Selection 82
Pin Assignment Setup 83
Pin Assignments Table 83
Verilog project 85
Creating Design File with Verilog 87
Phase 1: Creating the FPGA design - 32 bit Full Adder 87
Components Used 87
Top-level Verilog file 88
Top-level inputs and outputs 89
Phase 2: Creating the FPGA Design - 2 to 1 Clock Mux 95
Design 95
Configure Project to Output SVF and RPD Files 96
Compile an Example Project and Build RPD and SVF Files 98
Load Gx3700 with SVF File 100
Testing the Design 101
Adder Testing 101
Clock Mux Testing 102
Chapter 7 - GXFPGA VHDL Tutorial 103
Introduction 103
Downloading Altera Design FPGA Design Tools 103
Create New Project 104
Device Selection 104
Pin Assignment Setup 105
Pin Assignments Table 105
Schematic entry project 107
Creating Design File with VHDL 109
Phase 1: Creating the FPGA design - 32 bit Full Adder 109
Components Used 109
Top-level VHDL file 110
Top-level inputs and outputs 111
Phase 2: Creating the FPGA Design - 2 to 1 Clock Mux 121
Design 121
Configure Project to Output SVF and RPD Files 122
Compile an Example Project and Build RPD and SVF Files 124
Simulating the Design 126
Load Gx3700 with SVF File 129

Testing the Design 130
Adder Testing 130
Clock Mux Testing 131
Chapter 8 - GX3700 Expansion Boards 133
Expansion Board Design Guide 133
Mechanical Layout Guide 138
Expansion Board Connectors and Electrical Requirements 141
P1 Expansion Board Connector Pin Assignment 142
GX3701 Expansion Board 148
GX3701 Programming 148
GX3701 Expansion Board Specification 148
GX3702 Expansion Board 149
J1 – Flex I/O Bank A Connector 149
J2 – Flex I/O Bank B Connector 150
J3 – Flex I/O Bank C Connector 150
J4 – Flex I/O Bank D Connector 151
GX3702 Expansion Board Specification 151
GX3788 Expansion Board 152
GX3788 Programming 152
GX3788 Digital and Analog Multi-Function Expansion Board Specification 153
Chapter 9 - Function Reference 155
Introduction 155
GXFPGA Functions 156
GxFpgaDiscardEvents 159
GxFpgaDmaFreeMemory 160
GxFpgaDmaGetTransferStatus 161
GxFpgaDmaTransfer 162
GxFpgaGetBoardSummary 163
GxFpgaGetBoardType 164
GxFpgaGetEepromSummary 165
GxFpgaGetDriverSummary 166
GxFpgaGetErrorString 167
GxFpgaGetExpansionBoardID 170
GxFpgaInitialize 171
GxFpgaInitializeVisa 172
GxFpgaLoad 173
GxFpgaLoadFromEeprom 174

GxFpgaLoadStatus....175
GxFpgaLoadStatusMessage....176
GxFpgaPanel....177
GxFpgaRead....178
GxFpgaReadRegister....179
GxFpgaReset....180
GxFpgaSetEvent....181
GxFpgaUpgradeFirmware....182
GxFpgaUpgradeFirmwareStatus....183
GxFpgaWaitOnEvent....184
GxFpgaWrite....185
GxFpgaWriteRegister....186
Gx3788Initialize....187
Gx3788InitializeVisa....188
Gx3788Reset....189
Gx3788GetBoardSummary....190
Gx3788GetCalibrationInfo....191
Gx3788AnalogInGetGroundSource....192
Gx3788AnalogInMeasureChannel....193
Gx3788AnalogInScanGetChannelListIndex....195
Gx3788AnalogInScanGetCount....196
Gx3788AnalogInScanGetLastRunCount....197
Gx3788AnalogInScanGetSampleRate....198
Gx3788AnalogInScanIsRunning....199
Gx3788AnalogInScanReadMemoryRawData....200
Gx3788AnalogInScanReadMemoryVoltages....201
Gx3788AnalogInScanSetChannelListIndex....202
Gx3788AnalogInScanSetCount....203
Gx3788AnalogInScanSetSampleRate....204
Gx3788AnalogInScanStart....205
Gx3788AnalogInSetGroundSource....206
Gx3788AnalogOutGetOutputState....207
Gx3788AnalogOutGetVoltage....208
Gx3788AnalogOutReset....209
Gx3788AnalogOutSetOutputState....210
Gx3788AnalogOutSetVoltage....211
Gx3788PioGetPort....212

Gx3788PioGetPortChannel....213
Gx3788PioGetPortChannelDirection....214
Gx3788PioGetPortDirection....215
Gx3788PioReadPort....216
Gx3788PioReadPortChannel....217
Gx3788PioResetPort....218
Gx3788PioResetPortChannel....219
Gx3788PioSetPort....220
Gx3788PioSetPortChannel....221
Gx3788PioSetPortChannelDirection....222
Gx3788PioSetPortDirection....223
Gx3788TriggerGetOutputLevel....224
Gx3788TriggerReadInputLevel....225
Gx3788TriggerSetOutputLevel....226
Index....227

Chapter 1 - Introduction

Manual Scope and Organization

Manual Scope

The purpose of this manual is to provide all the necessary information to install, use, and maintain the GX3700 / GX3700e instruments. This manual assumes the reader has a general knowledge of PC based computers, Windows operating systems, and some understanding of digital I/O.

This manual also provides programming information using the GX3700 driver (referred in this manual **GXFPGA**). Therefore, good understanding of programming development tools and languages may be necessary. The **GXFPGA** function library supports both the GX3700 (PXI Hybrid slot compatible) and GX3700e (PXI Express) versions of the module. The terms GX3700 and GX3700e are used interchangeably throughout the manual, any differences are noted specifically.

Manual Organization

The GX3700 manual is organized in the following manner:

Chapter	Content
Chapter 1 - Introduction	Introduces the GX3700 manual. Lists all the supported board and shows warning conventions used in the manual.
Chapter 2 – Overview	Describes the GX3700 features, board description, its architecture, specifications and the panel description and operation.
Chapter 3 –Installation and Connections	Provides instructions on how to install a GX3700board and the GXFPGA software.
Chapter 4 – Programming the Board	Provides a list of the GXFPGA software driver files, general purpose and generic driver functions, and programming methods. Discusses supported application development tools and programming examples.
Chapter 5 – GXFPGA Schematic Entry Tutorial	Provides an example of how to use the Quartus II's Schematic Entry method to design and FPGA and then load and test the design using the GXFPGA panel.
Chapter 6 – GXFPGA Verilog Tutorial	Provides an example of how to use Quartus II and Verilog to design an FPGA and then load and test the design using the GXFPGA panel.
Chapter 7 – GXFPGA VHDL Tutorial	Provides an example of how to use Quartus II and VHDL to design an FPGA and then load and test the design using the GXFPGA panel.
Chapter 8 – Expansion Boards	Describes how to design a GX3700 expansion board and describes several standard expansion boards available from Marvin Test Solutions.
Chapter 9 – Functions Reference	Provides a list of the GX3700 driver functions. Each function description provides syntax, parameters, and any special programming comments.

Conventions Used in this Manual

Symbol Convention	Meaning
	Static Sensitive Electronic Devices. Handle Carefully.
	Warnings that may pose a personal danger to your health. For example, shock hazard.
	Cautions where computer components may be damaged if not handled carefully.
	Tips that aid you in your work.

Formatting Convention	Meaning
`Monospaced Text`	Examples of field syntax and programming samples.
Bold type	Words or characters you type as the manual instructs. For example: function or panel names.
Italic type	Specialized terms. Titles of other reference books. Placeholders for items you must supply, such as function parameters

Chapter 2 - Overview

Introduction

The GX3700 / GX3700e is a user configurable, FPGA based, 3U PXI / PXI Express card which offers 160 digital I/O signals which can be configured for single-ended or differential interfaces. The card employs the Altera Stratix III FPGA, which can support data rates up to 1.2 Gb/s (SerDes interface) and features over 65,000 logic elements and 2.636 Kb of memory. The GX3700 / GX3700e is supplied with an expansion board, GX3701 – Flex I/O Feed Through Module, providing access to the FPGA's 160 I/Os. Alternatively, users can design their own custom expansion cards for specific applications eliminating the need for additional external boards which are cumbersome and physically difficult to integrate into a test system. The design of the FPGA is done by using Altera's free Quartus II Web Edition tool set. Once the user has compiled the FPGA design, the configuration file can be loaded directly into the FPGA or via an on-board EEPROM.

Features

The GX3700 / GX3700e's digital I/O signals are 5 volt tolerant. Logic families supported by the I/O interface include LVTTL, LVDS and LVCMOS. The FPGA's I/Os includes 160 single ended I/O with support for 32 differential pairs, 4 dedicated global clock inputs (2 differential pairs), and various VCCIO voltages. At power-up, all I/Os will be isolated from the UUT. The FPGA device supports up to four phase lock loops (PLL) for clock synthesis, clock generation and for support of the I/O interface. An on-board 80 MHz oscillator is available for use with the FGPA device or alternatively, the PXI 10 MHz or 100 MHz clock can be used as a clock reference by the FPGA.

The FPGA has access to all of the PXI Express bus resources including the PXI 10 MHz clock, PXIe 100 MHz clock, PXIe Sync100, PXIe DStar triggers, the local bus, and the PXI triggers; allowing the user to create a custom instrument which incorporates all of the PXI Express bus resources. . The GX3700's FPGA has access to all of the PXI Hybrid slot compatible resources including PXI 10 MHz clock, the local bus, and the PXI triggers Control and access to the FPGA is provided via the GX3700 / GX3700e's driver which includes tools for downloading the compiled FPGA code as well as register read and write functionality.

The GX3700 / GX3700e include the provision to add a daughter board which will provide additional flexibility for those users who wish to design their own custom interfaces for specific applications.

Communication between the customer-programmable FPGA and the PXI/PXIe bus is implemented via a dedicated FPGA device (Interface FPGA). The Interface FPGA contains control and status registers for the board and provides in-system programmability of the customer-programmable FPGA. The Interface FPGA interfaces directly to the PXI/PXIe bus and will decode/encode the bus protocol.

The GX3700 has external SRAM, flash, and an external clock source that is accessible by the customer.

The GX3700 employs the Altera Stratix III 780 pin device. Key features for the Altera device includes:

- 47,500 logic elements (LEs) and 1.88Mbits of memory
- Supports up to four phase-locked loops (PLLs) for clock synthesis, clock generation and support of I/O interfaces
- Up to five outputs per PLL can be accessed
- Dynamically reconfigurable logic supports programmable phase shift, frequency multiplication/division, and in-system frequency re-programming without reconfiguring the device
- Support for high-speed external memory interfaces including DDR, DDR2, SDR, SDRAM, and QDRII SRAM at up to 400 megabits per second (Mbps)
- 327 I/O pins arranged in eight I/O banks that support a wide range of industry I/O standards
- Supports up to 875 Mbps receive and 840 Mbps transmit LVDS communications data rates

- Support for Bus LVDS (BLVDS), LVDS, RSDS®, mini-LVDS and PPDS® differential I/O standards
- Supported I/O standards include LVTTL, LVCMOS, SSTL, HSTL, PCI, PCI-X, LVPECL, LVDS, mini-LVDS, RSDS, and PPDS; PCI Express Base
- 160 single ended I/Os.
- 32 differential pairs.
- 4 dedicated global clock inputs (2 differential pairs).
- VCCIO can be preset using on-board jumpers to 1.2V, 2.5V, or 3.3V.
- Internal FPGA SRAM (memory size depends on internal FPGA model installed)
- 1MB external SRAM in addition to internal FPGA SRAM.
- 16MB flash
- User controlled LED.
- Integrated DMA engine.
- All of PXI/PXIe instrumentation signals such as differential Star Trigger, SYNC100, CLK100, CLK10, local bus, trigger bus, and single-ended Star Trigger are available to customer.

Applications

- Automatic Test Equipment (ATE) and Functional Test
- Data Acquisition
- Process Control
- Factory Automation

Board Description

The GX3700 is a 3U PXI hybrid slot compatible instrument card that consists of 160 TTL I/O Channels divided into groups of 40 channels. Each of these groups is connected to a 68 pin SCSI type connector on the front panel of the instrument (J1-J4) using a daughter board module (GX3701). A short on JP7 will force the user FPGA to be configured automatically on boot up using the contents of the EEPROM. For more information about the connectors and jumpers and their location on the board refer to Chapter 3 – Installation and Connections.

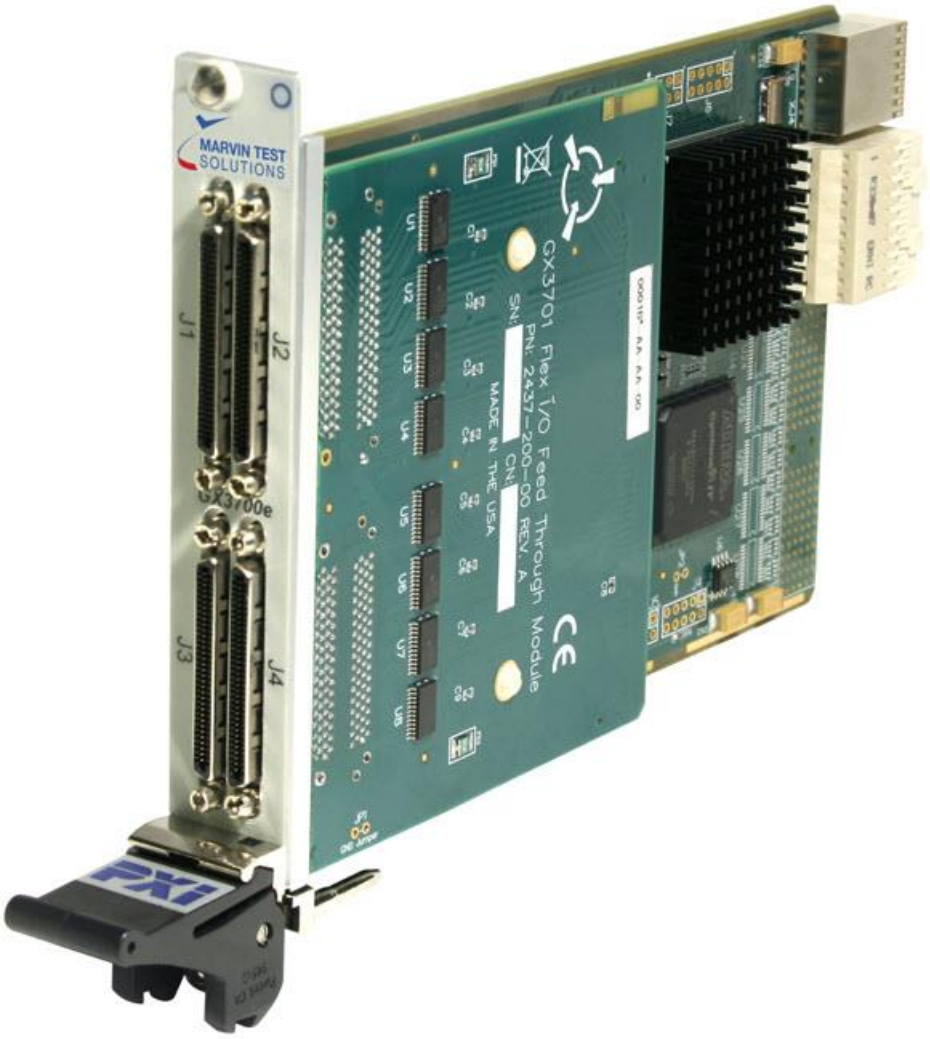


Figure 2-1: GX3700e Board with the GX3701 Module Mounted

Figure 2-2: GX3700 Board with the GX3701 Module Mounted

Architecture

The GX3700 consists of a user programmable FPGA that can access external resources and peripherals such as the PCI bus, SRAM and flash memories. The user FPGA is an Altera Stratix III that can be programmed directly through the software driver or indirectly by the onboard EEPROM that can store a FPGA bit stream for later use. An Expansion board connects to the User FPGA to provide external I/O. The standard expansion board provides 160 I/O channels that are brought out to the front panel. The user may design custom expansion boards based on documentation provided by Marvin Test Solutions.

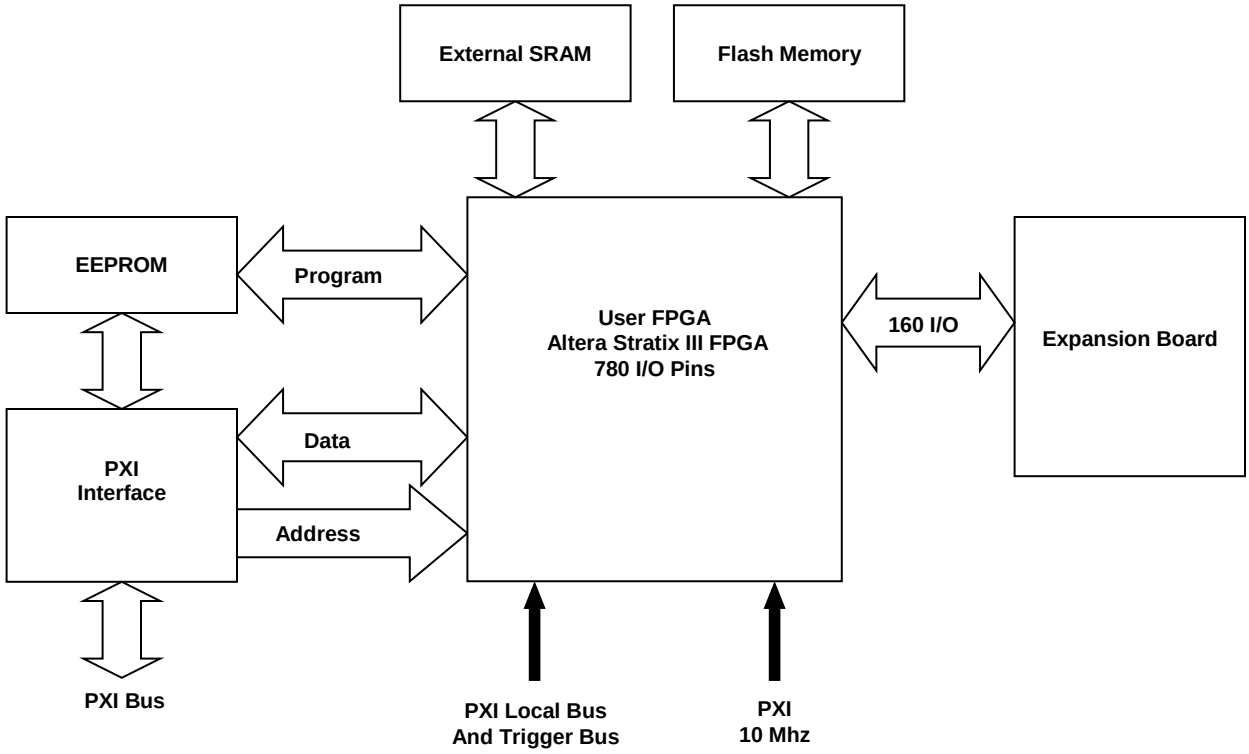
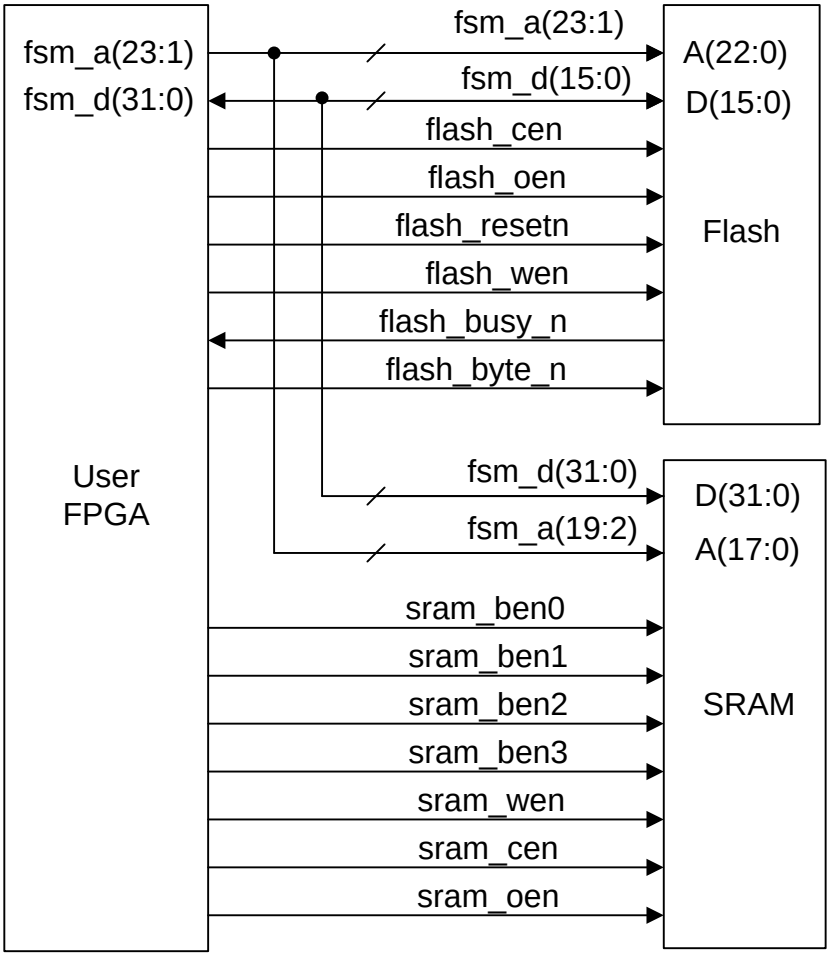


Figure 2-3: GX3700 Architecture

Memory

The Gx3700 has three types of memories, internal SRAM, external SRAM and Flash memory.

Figure 2-6 is a more detailed block diagram of the connections between the User's FPGA, the Flash and the SRAM



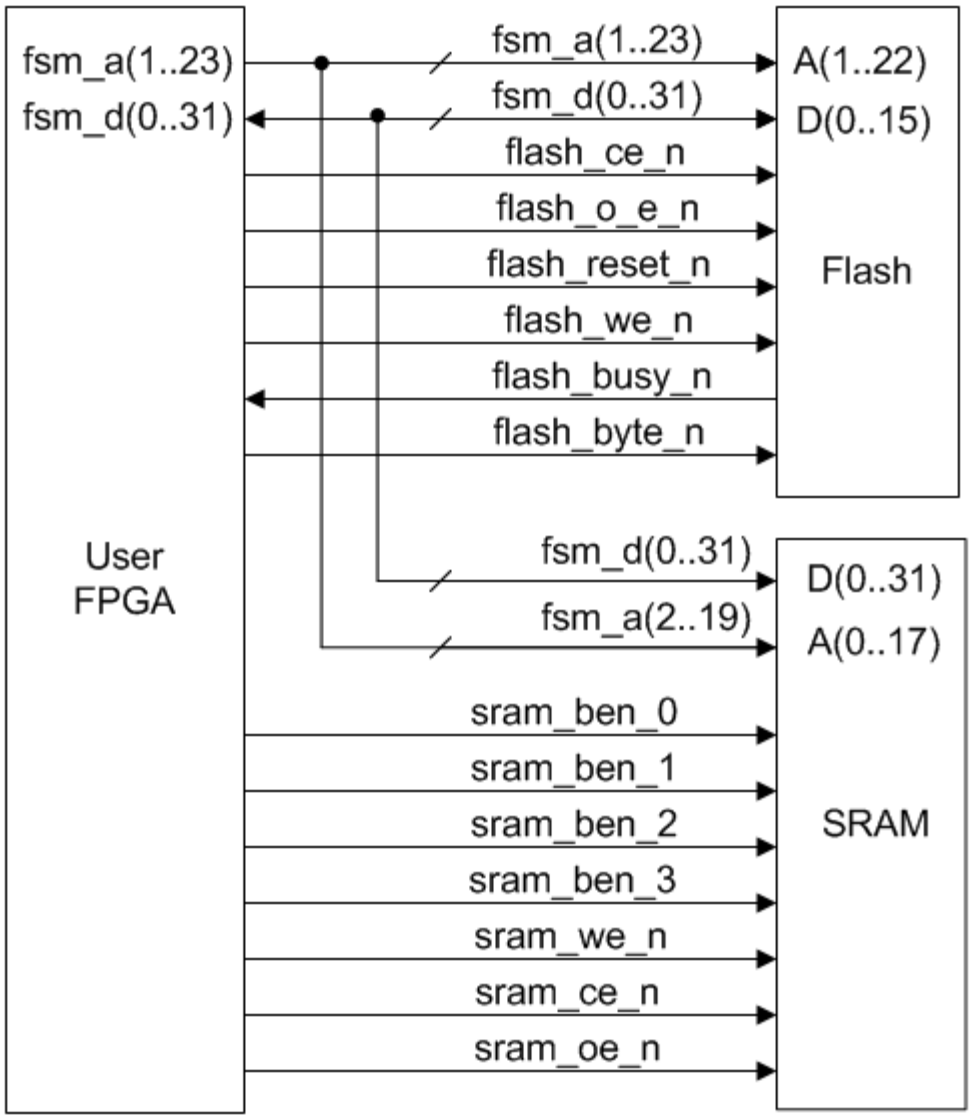


Figure 2-4: GX3700 / GX3700e Connections between User FPGA, Flash, and SRAM

PXI/PXIe and PC Connections

The User FPGA, Stratix III, can be configured either through the EEPROM or directly through the PXI Interface. It has access to PXI resources such as the local bus, trigger bus, and PXI 10 Mhz clock source and is also connected to the PXI Interface FPGA to give access to PCI resources and memory. This allows the User FPGA to communicate with the host system's operating system using the provided GXFPGA software library functions.

A more detailed diagram of the PXI/PXIe Signal Connections is shown below . It shows the different PXI/PXIe signals and how they are interfaced to the User FPGA.

The bi-directional bus switch with level shifting allows the PXI/PXIe signals to be interfaced to the User FPGA. The direction of the signals is controlled and determined by the signals from the User FPGA.

For example, to use the signal PXITrig(7), the user FPGA would be programmed as follows:

1. If this signal is only used as an input, define it inside the User FPGA as an input pin.
2. However, if the signal is used as an output only or a bidirectional I/O, define it as such in the User FPGA but make sure to drive the output to high impedance or tri-state level when the signal is not driving or is inactive.

In both of these cases the level translation and the direction of the signals are handled by the on-board bus switch.

Also shown are the buffers for the DSTAR_A, DSTAR_B and DSATR_C signals. These buffers conform to the standard as required by the PXI Systems Alliance's PXI Express Hardware Specification Rev 1.0. Note that the DSTAR signals are only available with the GX3700e module.

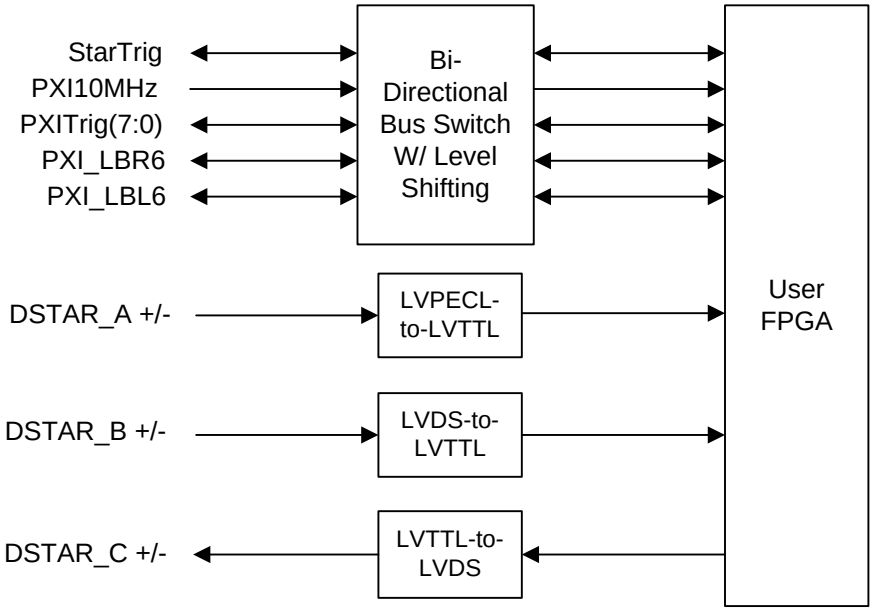


Figure 2-5: PXI/PXIe Signal Connections

Inter-FPGA Bus Interface Timing

The Flex FPGA communicates with the PCI/PCIe host via the PXI/PXIe Bridge FPGA. The following figure shows the inter-FPGA timing diagram for communication between the two FPGAs.

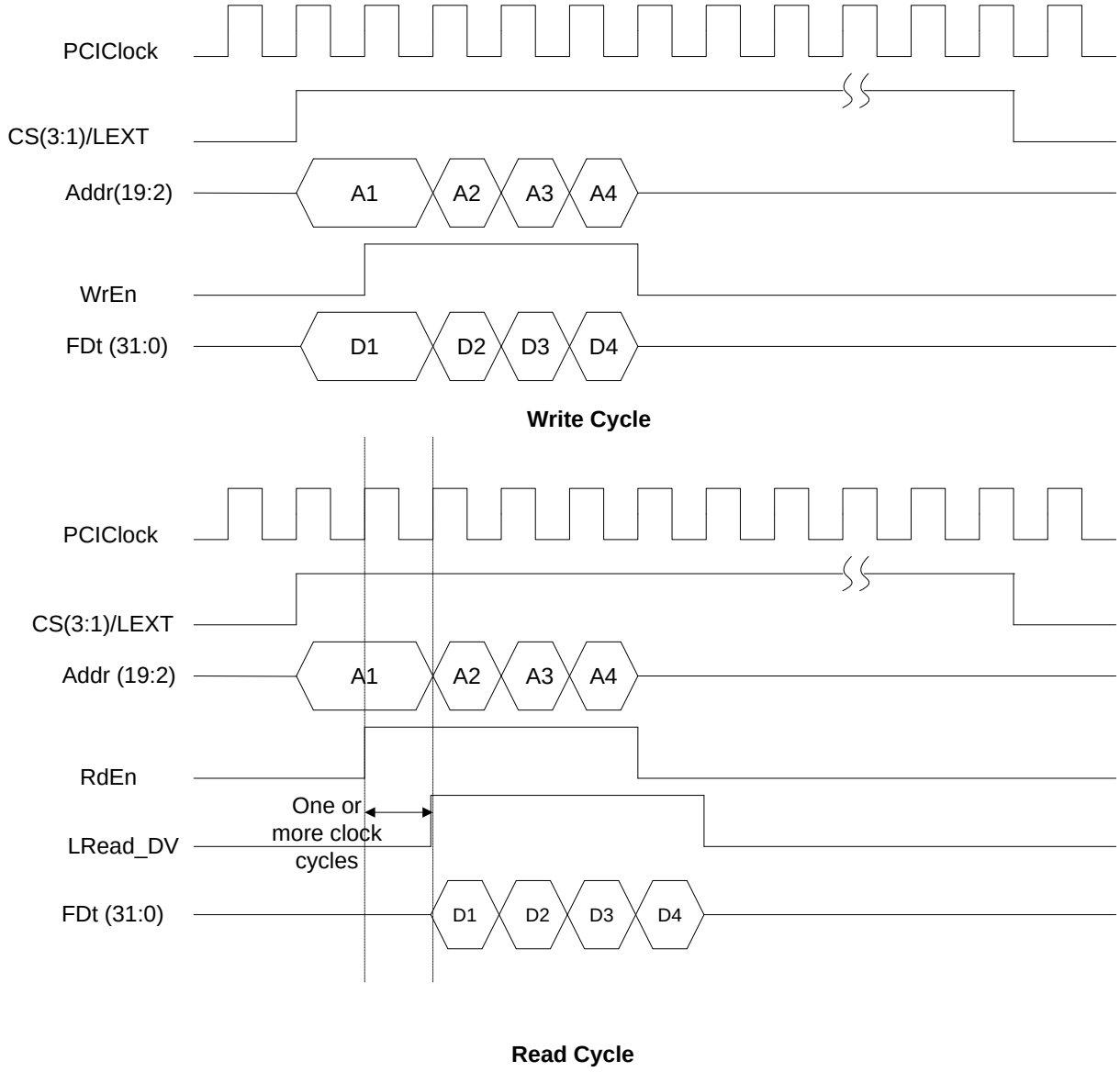


Figure 2-6 – Inter-FPGA Bus Interface Diagram

DMA FIFO Interface Timing

The PXI Bridge FPGA contains the DMA engine for transferring data between the Flex FPGA and the PCI/PCIe host. Unlike a Scatter-Gather DMA engine, this one will need a contiguous memory space.

There are two 32-bit buses between the PXI Bridge FPGA and the Flex FPGA for transmit and receive of DMA data.

For DMA write, the DMA controller will read data from the Flex FPGA and write this data to the host PC. The controller will only read data when it's in DMA write mode and will only read when the EMPTY signal is de-asserted. The controller will only read up to the number of byte count specified for the DMA transfer and will not read more even if the FIFO is still empty.

For DMA read, the DMA controller will read data from the PC host and will write this data to the Flex FPGA. When in DMA read mode, the Flex FPGA must expect data and must store it. Otherwise, this data will be lost.

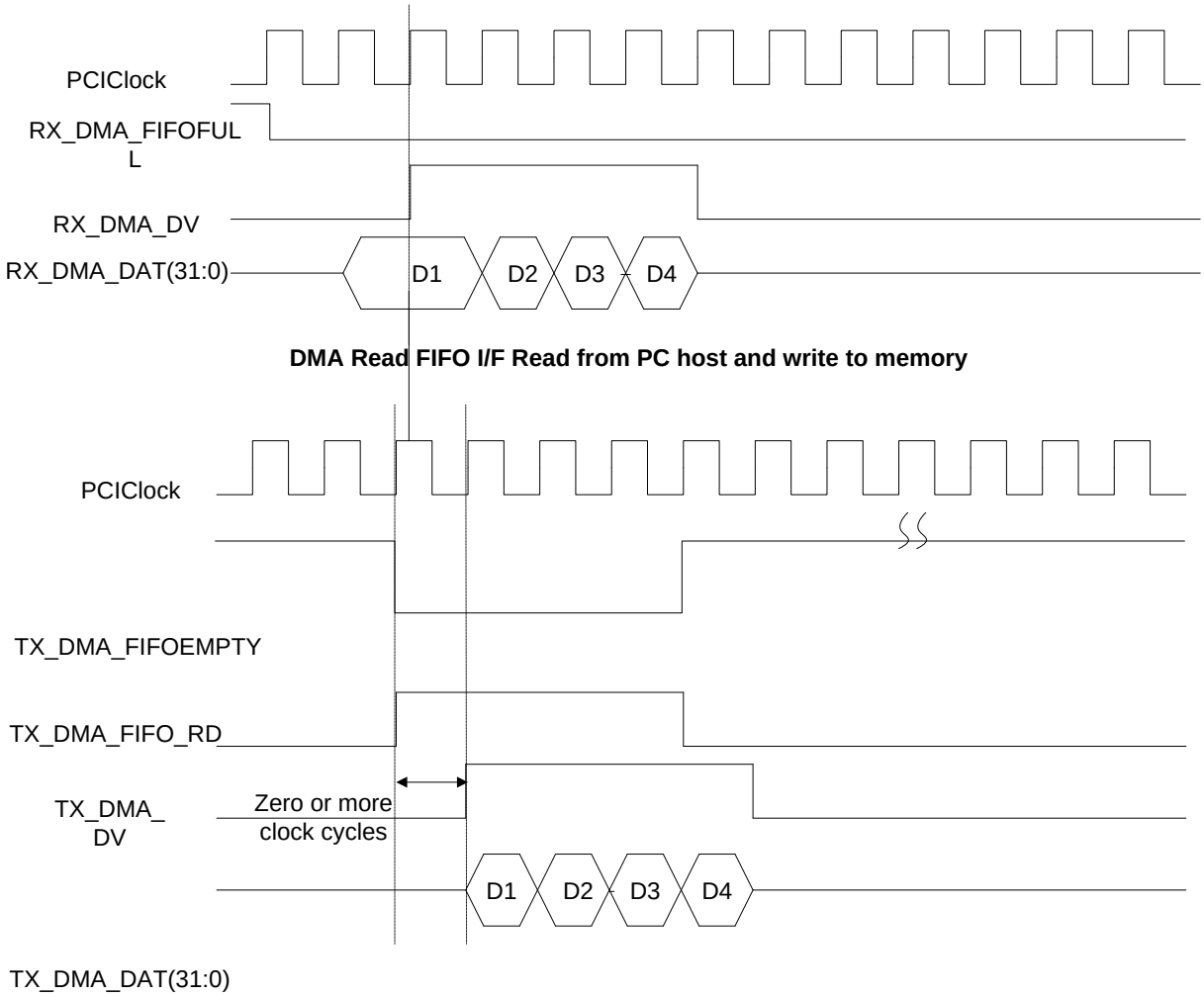


Figure 2-7: DMA FIFOs Timing Diagram

Specifications

The following table outlines the specifications of the GX3700 / GX3700e.

Digital I/O Channel

Logic Families	LVTTL, LVDS, configurable for 1.2 / 2.5 / 3.3 V logic; 5 volt compatible (programmable via the FPGA on a per pin basis)
Output Current	+/ 12.0 mA, max. (programmable via the FPGA on a per pin basis)
Input Leakage Current	+/- 10 uA
Power On State	Programmable by line, default is disconnect at power on
Number of Channels	4 banks of 40 I/O signals. Direction is configurable on a per pin basis Disconnect on a per bank basis
Protection	Overvoltage: -0.5V to 7.0V (input) Short circuit: up to 8 outputs may be shorted at a time
Daughter Board User Connectors	(4) SCSI III, VHDCI type, 68 pin female

Expansion Board Interface

Board ID	4 bits
Digital I/O	160, each bank of 40 can be configured to bypass or access the expansion board
FPGA Flex I/O	4 signals
Master Clear	From PXI interface
Power	+/- 12 volts, +5 volts, +3.3 volts, +2.5 volts, +1.2 volts

Timing Source

PXI 10 MHZ	PXI Bus
Internal	80 MHz oscillator, +/- 20 ppm

User FPGA

FPGA Type	Default: 3700: Stratix III, EP3SL50F780 3700e: Stratix III, EP3SL70F780 Check the instrument panel, About page for newer versions.
Number of PLLs	Four
Logic Elements	47,500
Internal Memory	FPGA dependent: EP3SL50: 2,133 Kb EP3SL70: 2,636 Kb EP3SL110: 4,875 Kb EP3SL150: 6,390 Kb EP3SL200: 10,646 Kb EP3SL340: 18,381 Kb EP3SE50: 5,625 Kb EP3SE80: 6,683 Kb EP3SE110: 8,727 Kb EP3SLE260: 16,282 Kb

Power

3.3 VDC	400 mA (typ.); 1 A (Max.)
5 VDC	300 mA (typ.); 1.2 A (Max.)
12 VDC (For Expansion Board)	Expansion Board Dependent

Environmental

Operating Temperature	0 to 50° C
Storage Temperature	-20° C to 70° C
Size	3U PXI
Weight	200 g

Virtual Panel Description

The GX3700 includes a virtual panel program, which enables full utilization of the various configurations and controlling modes. To fully understand the front panel operation, it is best to become familiar with the functionality of the board.

To open the virtual panel application, select **GX3700 Panel** from the **Marvin Test Solutions**, **GXFPGA** menu under the **Start** menu. The GX3700 virtual panel opens as shown here:

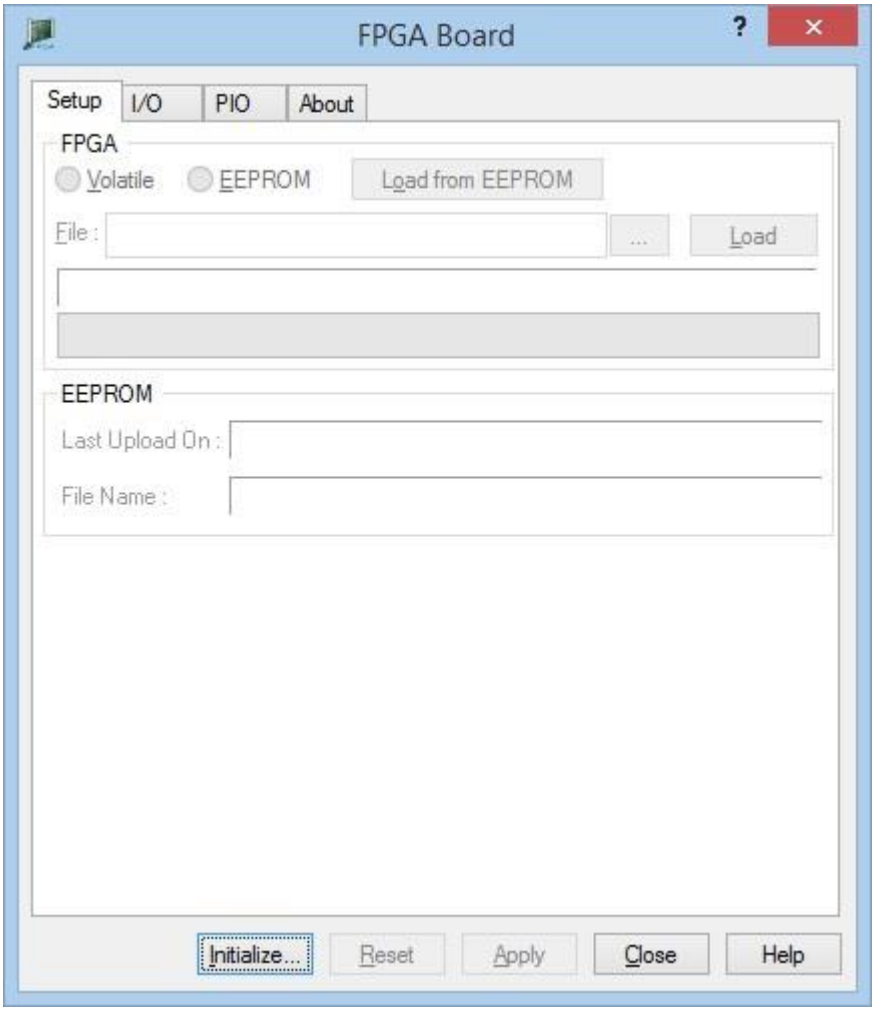


Figure 2-8: GX3700 Virtual Panel

Initialize – Opens the **Initialize Dialog** (see Initialize Dialog paragraph) in order to initialize the board driver. The current settings of the selected board will **not change after calling initialize**. The panel will reflect the current settings of the board after the Initialize dialog closes.

Reset – Resets the PXI board settings to their default state and clears the reading.

Apply – Applies changed settings to the board.

Close – Closes the panel. Closing the panel **does not affect** the board settings.

Help – Opens the on-line help window. In addition to the help menu, the caption shows a **What's This Help** button (?) button. This button can be used to obtain help on any control that is displayed in the panel window. To displays the What's This Help information click on the (?) button and then click on the control – a small window will displays the information regarding this control.

Virtual Panel Initialize Dialog

The Initialize dialog initializes the driver for the selected board. The board settings **will not change** after initialize is called. Once initialized, the panel will reflect the current settings of the board.

The Initialize dialog supports two different device drivers that can be used to access and control the board:

1. **Use Marvin Test Solutions' HW** – This is the device driver installed by the setup program and is the default driver. When selected, the **Slot Number** list displays the available **GX3700** boards installed in the system and their slots. The chassis, slots, devices and their resources are also displayed by the HW resource manager, **PXI/PCI Explorer** applet that can be opened from the Windows Control Panel. The **PXI/PCI Explorer** can be used to configure the system chassis, controllers, slots and devices. The configuration is saved to PXISYS.INI and PXIeSYS.INI located in the Windows folder. These configuration files are also used by VISA. The following figure shows the slot number 0x109 (chassis 1 Slot 9). This is the slot number argument (*nSlot*) passed by the panel when calling the driver **GxFpgaInitialize** function which is used to initialize the driver for the specified board.

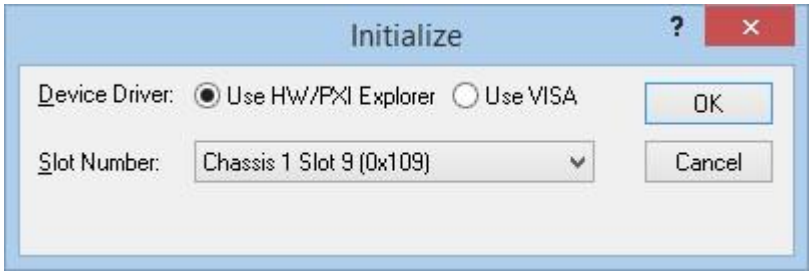


Figure 2-9: Initialize Dialog Box using Marvin Test Solutions' HW driver

2. **Use VISA** – This is a third-party device driver usually provided by National Instrument (NI-VISA). When selected, the **Resource** list displays the available boards installed in the system and their VISA resource address. The chassis, slots, devices and their resources are also displayed by the VISA resource manager, **Measurement & Automation** (NI-MAX) and by Marvin Test Solutions **PXI/PCI Explorer**. The following figure shows PXI9::13::INSTR as the VISA resource (PCI bus 9 and Device 13). This is a VISA resource string argument (*szVisaResource*) which is passed by the panel when calling the driver **GxFpgaInitializeVisa** function which initializes the driver for the specified board.

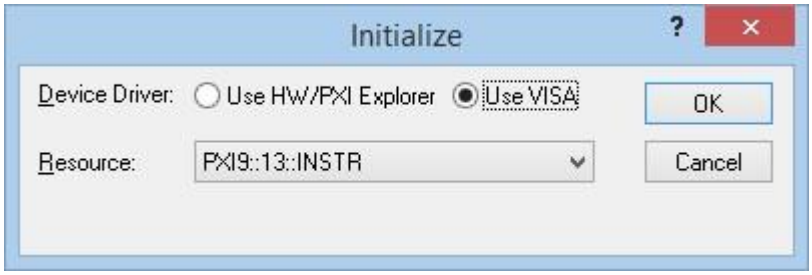


Figure 2-10: Initialize Dialog Box using VISA resources

Virtual Panel Setup Page

After the board is initialized, the panel is enabled and will display the current setting of the board. The following picture shows the **Setup page** settings:

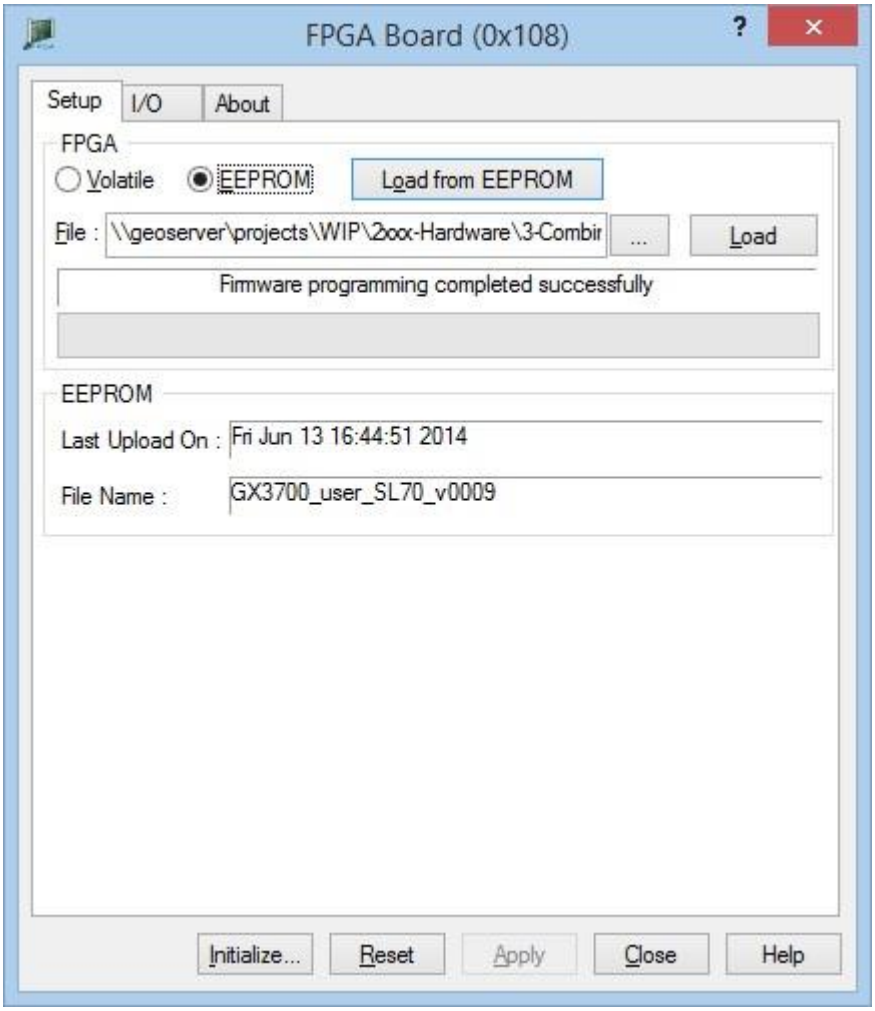


Figure 2-11: GX3700 Virtual Panel – Setup page

The following controls are shown in the Setup page:

Volatile radio button: Select this radio button to load the File to the Volatile (current) FPGA configuration.

EEPROM radio button: Select this radio button to load File to the EEPROM FPGA.

Load From EEPROM button: Loads the volatile (current FPGA) with the FPGA configuration that is stored in the EEPROM

File text box: File path to the programming file intended to load the volatile FPGA or EEPROM. The File type must be Serial Vector File (.svf) for Volatile loading or Raw Programming Data (.RPD) file for EEPROM.

Load Button: Starts the loading process, either to the volatile FPGA or to the EEPROM, depending on which radio button the user selects.

EEPROM Last Updated On Text: Indicates the last time the EEPROM was loaded.

EEPROM File Name Text: Indicates the last file name that was written to the EEPROM.

Expansion Board Bypass Checkboxes: These checkboxes control the routing of each of the FPGA's I/O Banks. When the box is checked, it indicates that the I/O Bank will be connected directly to the I/O front connectors. If the box is unchecked, it indicates that the I/O Bank will be connected to the expansion board.

Virtual Panel I/O Page

Clicking on the **I/O** tab will show the **I/O page** as shown in Figure 2-9: GX3700 Virtual Panel – I/O page

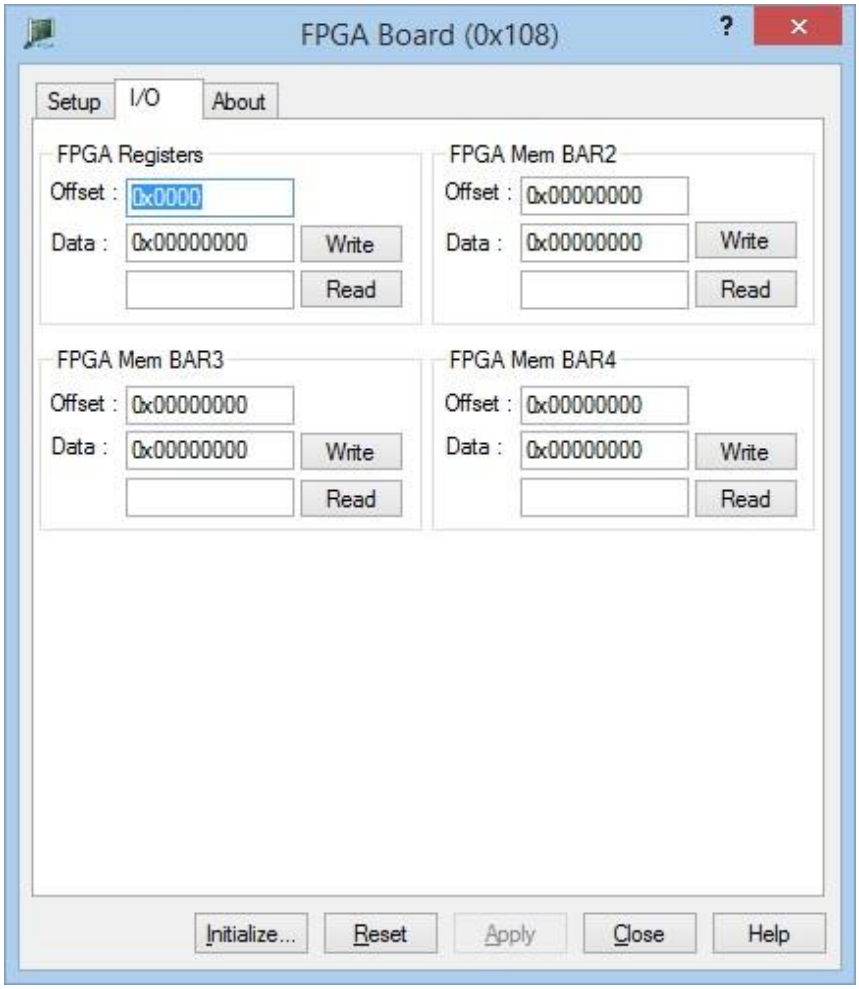


Figure 2-12: GX3700 Virtual Panel – I/O page

The following controls are shown in the I/O page:

Offset Text Field: The offset into the FPGA Register or Memory space (BAR2-4) in bytes. This field can be used with a decimal or hexadecimal value (prefix the value with 0x). The offset is limited to 0x400 bytes when reading the register space and 0x40000 bytes when reading the memory space. Offset must be specified on a 4-byte alignment.

Write Text Field: The 32-bit data (hexadecimal or decimal) to be written the specified offset in either FPGA Register or Memory space (BAR2-4).

Write Button: Write the 32-bit double word to either the FPGA Register or Memory space at the specified offset.

Read Text Field: The 32-bit data that has been read from the specified offset in either FGPA Register or Memory space. Value is specified in hexadecimal.

Read Button: Read the 32-bit double word from either the FPGA Register or Memory space at the specified offset.

Virtual Panel DAQ Page (GX3788)

Clicking on the **DAQ** tab will show the **DAQ page** as shown in Figure 2-13: GX3788 Virtual Panel – DAQ page

The **DAQ** tab only appears when the GX3788 daughter board is used.

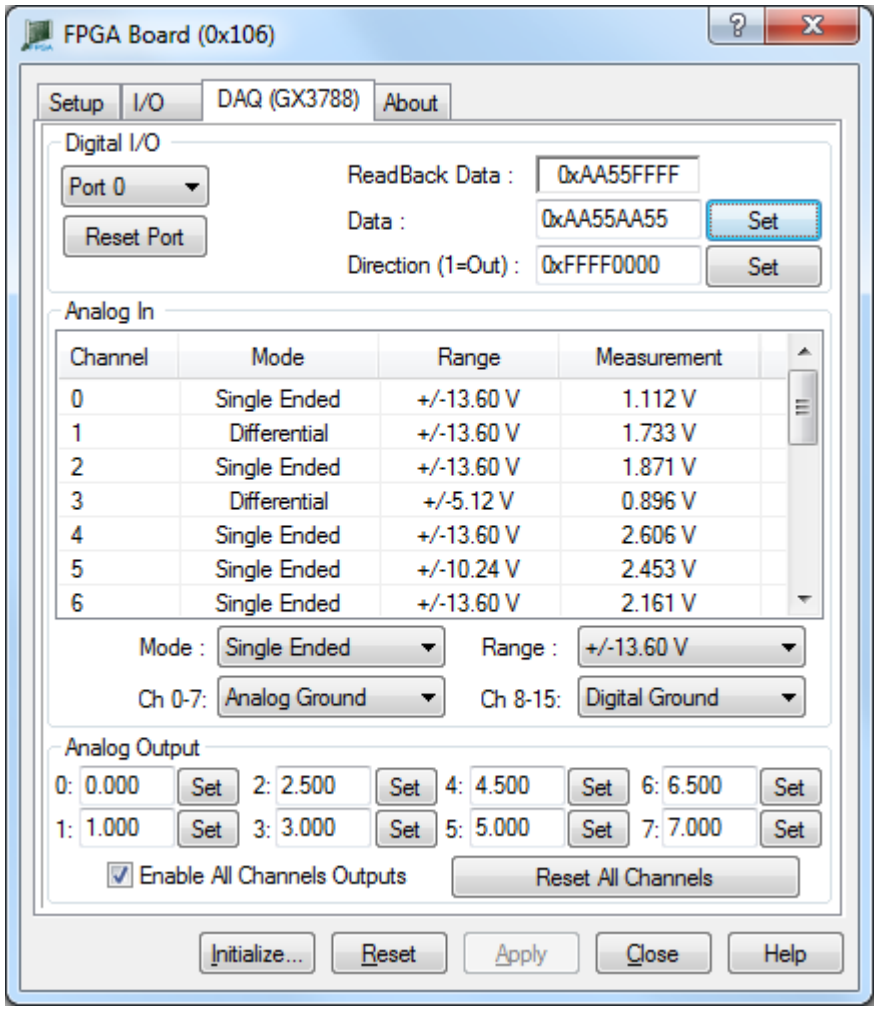


Figure 2-13: GX3788 Virtual Panel – DAQ page

The following controls are shown in the DAQ page:

Digital I/O Group Box

Port Combo Box: Select the digital I/O port (0-2) to configure. The output data and direction can be set and read for the selected digital port.

Readback Field: The 32-bit data that has been read from the specified digital port. This is an actual sampling of the digital line states at the selected digital port.

Data Text Field: The 32-bit output data (hexadecimal or decimal) to be written the specified digital port. A '1' bit signifies a logic high and a '0' bit signifies a logic low.

Set Button: Writes the Data field contents to the digital port selected.

Direction Text Field: The 32-bit direction data (hexadecimal or decimal) to be written to the specified digital port. A '1' bit signifies an output channel and a '0' bit signifies an input channel.

Set Button: Writes the Direction field contents to the digital port selected.

Analog In Group Box

Analog In List: Displays a continuously updating voltage measurement from each of the 16 analog input channels. In addition, each channels measurement mode and range are also shown.

Mode Combo Box: Sets the channel mode to use for a channel's measurement (single ended or differential)

Range Combo Box: Sets the channel range to sue for a channel's measurement

Ch 0-7 Combo Box: Sets the ground source for analog in channels 0 to 7

Ch 8-15 Combo Box: Sets the ground source for analog in channels 8 to 15

Analog Output Group Box

Enable All Channels Outputs Check Box: Sets all the analog output channels to enabled or disabled

Reset All Channels Button: Reset all the analog output channels to default settings

Voltage Edit Box Fields (Channel 0 to Channel 7): Enter output voltages for each of the analog output channels. Each channle's Voltage edit box has a set button to apply the new voltage settings.

Virtual Panel About Page

Clicking on the **About** tab will show the **About page** as shown in Figure 2-7

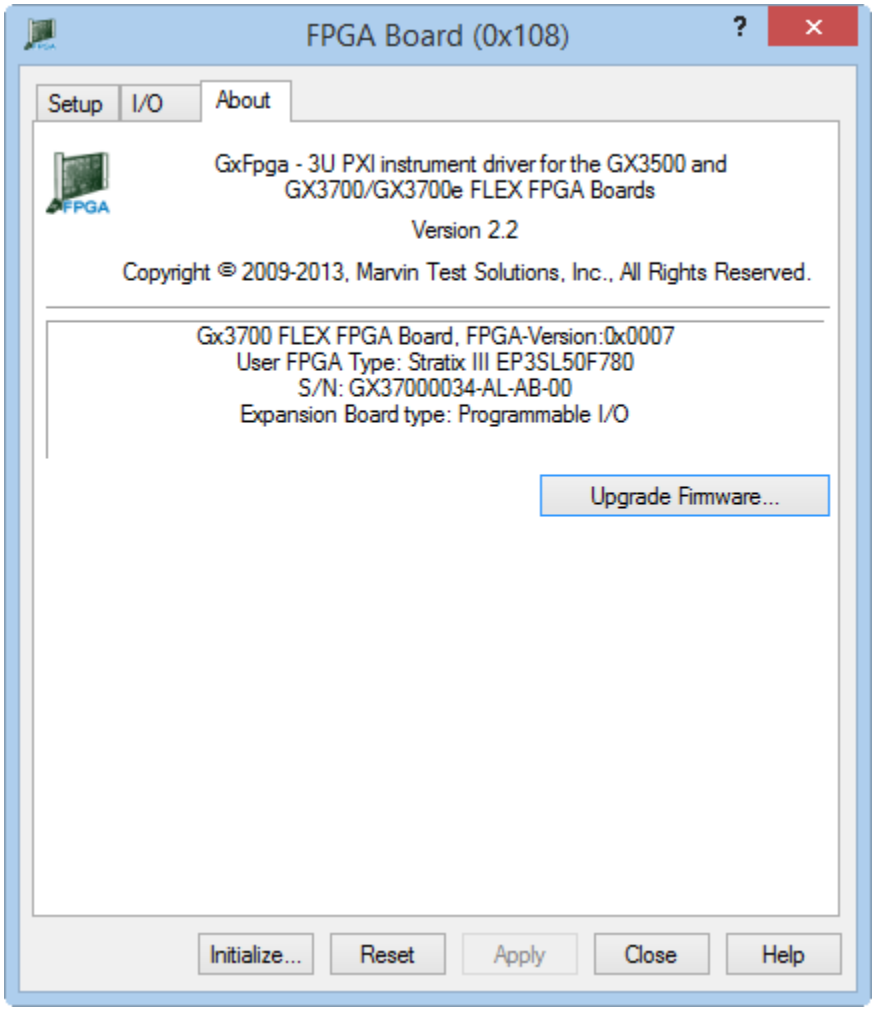


Figure 2-14: GX3700 Virtual Panel – About Page

The top part of the **About** page displays version and copyright of the GX3700 driver. The bottom part displays the board summary, including the main board FPGA version, user FPGA part number, serial number, and each installed I/O Module FPGA version. The **About** page also contains a button **Upgrade Firmware…** used to upgrade the board FPGA. This button maybe used only when the board requires upgrade as directed by Marvin Test Solutions support. The upgrade requires a firmware file (.jam) that is written to the board FPGA. After the upgrade is complete you must shut down the computer to recycle power to the board.

Chapter 3 - Installation and Connections

Getting Started

This section includes general hardware installation procedures for the GX3700 board and installation instructions for the GX3700 (GXFPGA) software. Before proceeding, please refer to the appropriate chapter to become familiar with the board being installed.

To Find Information on..	Refer to..
Hardware Installation	This Chapter
GX3700 Driver Installation	This Chapter
Programming	Chapter 4
GXFPGA Design Tools and Tutorial	Chapter 5, 6 and 7
Expansion Boards	Chapter 8
GX3700 Function Reference	Chapter 9

Interfaces and Accessories

The following accessories are available from Marvin Test Solutions for GX3700 switching board.

Part / Model Number	Description
GT95015	Connector Interface SCSI to 100 Mil Grid Differential
GT95021	2’ 68-Pin shielded cable
GT95022	3’ 68-Pin shielded cable
GT95028	10’ 68-Pin shielded cable
GT95031	6’ 68-Pin shielded cable

Packing List

All GX3700 boards have the same basic packing list, which includes:

1. GX3700 Board
2. GXFPGA Driver Disk

Unpacking and Inspection

After removing the board from the shipping carton:

Caution - Static sensitive devices are present. Ground yourself to discharge static.

1. Remove the board from the static bag by handling only the metal portions.
2. Be sure to check the contents of the shipping carton to verify that all of the items found in it match the packing list.
3. Inspect the board for possible damage. If there is any sign of damage, return the board immediately. Please refer to the warranty information at the beginning of the manual.

System Requirements

The GX3700 Instrument board is designed to run on PXI compatible computer running WindowsXP SP3-Windows 10 (32/64-bit).

The board requires one unoccupied 3U PXI bus slot.

Installation of the GXFPGA Software

Before installing the board, it is recommended that you install the GXFPGA software as described in this section. To install the GXFPGA software, follow the instruction described below:

1. Insert the Marvin Test Solutions CD-ROM and locate the **GXFPGA.EXE** setup program. If your computer's Auto Run is configured, when inserting the CD, a browser will show several options. Select the Marvin Test Solutions Files option and then locate the setup file. If Auto Run is not configured, you can open the Windows explorer and locate the setup files (usually located under \Files\Setup folder). You can also download the file from Marvin Test Solutions' web site (www.marvintest.com).
2. Run the GXFPGA setup and follow the instruction on the Setup screen to install the GXFPGA driver.

   Note: When installing under Windows, you may be required to restart the setup logging-in as a user with Administrator privileges. This is required in-order to upgrade your system with newer Windows components and to install kernel-mode device drivers (HW.SYS and HWDEVICE.SYS) which are required by the GXFPGA driver to access resources on your board.
3. The first setup screen to appear is the Welcome screen. Click **Next** to continue.
4. Enter the folder where GXFPGA is to be installed. Either click **Browse** to set up a new folder, or click **Next** to accept the default folder of `C:\Program Files\Marvin Test Solutions\`GXFPGA for 32-bit Windows or `C:\Program Files (x86)\Marvin Test Solutions\`GXFPGA for 64-bit Windows.
5. Select the type of Setup you wish and click **Next.** You can choose between **Typical**, **Run-Time** and **Custom** setups types. The **Typical** setup type installs all files. **Run-Time** setup type will install only the files required for controlling the board either from its driver or from its virtual panel. The **Custom** setup type lets you select from the available components.

The program will now start its installation. During the installation, Setup may upgrade some of the Windows shared components and files. The Setup may ask you to reboot after completion if some of the components it replaced were used by another application during the installation – do so before attempting to use the software.

You can now continue with the installation to install the board. After the board installation is complete you can test your installation by starting a panel program that lets you control the board interactively. The panel program can be started by selecting it from the Start, Programs, GXFPGA menu located in the Windows Taskbar.

Setup Maintenance Program

You can run the Setup again after GXFPGA has been installed from the original disk or from the Windows Control Panel – Add Remove Programs applet. Setup will be in the Maintenance mode when running for the second time. The Maintenance window show below allows you to modify the current GXFPGA installation. The following options are available in Maintenance mode:

- **Modify.** When you want to add, or remove GXFPGA components.
- **Repair.** When you have corrupted files and need to reinstall.
- **Remove.** When you want to completely remove GXFPGA.

Select one of the options and click **Next** and follow the instruction on the screen until Setup is complete.

Overview of the GXFPGA Software

Once the software is installed, the following tools and software components are available:

- **GXFPGA Panel** – Configures and controls the GXFPGA board various features via an interactive user interface.
- **GXFPGA driver** - A DLL based function library (GXFPGA.DLL, located in the Windows System folder) used to program and control the board. The driver uses Marvin Test Solutions' HW driver or VISA supplied by third party vendor to access and control the GXFPGA boards.
- **Programming files and examples** – Interface files and libraries for support of various programming tools. A complete list of files and development tools supported by the driver is included in subsequent sections of this manual.
- **Documentation** – On-Line help and User's Guide for the board, GXFPGA driver and panel.
- **HW driver and PXI/PCI Explorer applet** – HW driver allows the GXFPGA driver to access and program the supported boards. The explorer applet configures the PXI chassis, controllers and devices. This is required for accurate identification of your PXI instruments later on when installed in your system. The applet configuration is saved to PXISYS.ini and PXIeSYS.ini and is used by instruments HW driver and VISA. The applet can be used to assign chassis numbers, Legacy Slot numbers and instrument alias names. The HW driver is installed and shared with all Marvin Test Solutions products to support accessing the PC resources. Similar to HW driver, VISA provides a standard way for instrument manufacturers and users to write and use instruments drivers. VISA is a standard maintained by the VXI Plug & Play System Alliance and the PXI Systems Alliance organizations (http://www.pxisa.org/). The VISA resource manager such as National Instruments **Measurement & Automation** (NI-MAX) displays and configures instruments and their address (similar to Marvin Test Solutions' PXI/PCI Explorer). The GXFPGA driver can work with either HW or VISA to control an access the supported boards.

Installation Folders

The GX3700 driver files are installed in the default folder **C:\Program Files [(x86)]\Marvin Test Solutions\GXFPGA**. You can change the default GXFPGA folder to one of your choosing at the time of installation.

During the installation, GXFPGA Setup creates and copies files to the following folders:

Name	Purpose / Contents
…\Marvin Test Solutions\GXFPGA	The GXFPGA folder. Contains panel programs, programming libraries, interface files and examples, on-line help files and other documentation.
…\Marvin Test Solutions\HW	HW device driver. Provide access to your board hardware resources such as memory, IO ports and PCI board configuration. See the README.TXT located in this directory for more information.
…\ATEasy\Drivers	ATEasy drivers folder. GXFPGA Driver and example are copied to this directory only if ATEasy is installed to your machine.
…\Windows\System or System32	Windows System directory. Contains the GXFPGA DLL, GXFPGA64.DLL drivers, HW driver shared files and some upgraded system components, such as the HTML help viewer, etc.

Configuring Your PXI System using the PXI/PCI Explorer

To configure your PXI/PCI system using the **PXI/PCI Explorer** applet follow these steps:

1. **Start the PXI/PCI Explorer applet**. The applet can be start from the Windows Control Panel or from the Windows Start Menu, **Marvin Test Solutions**, **HW**, **PXI/PCI Explorer**.
2. **Identify Chassis and Controllers**. After the PXI/PCI Explorer is started, it will scan your system for changes and will display the current configuration. The PXI/PCI Explorer automatically detects systems that have Marvin Test Solutions controllers and chassis. In addition, the applet detects PXI-MXI-3/4 extenders in your system (manufactured by National Instruments). If your chassis is not shown in the explorer main window, use the Identify Chassis/Controller commands to identify your system. Chassis and Controller manufacturers should provide INI and driver files for their chassis and controllers which are used by these commands.
3. **Change chassis numbers, PXI devices Legacy Slot numbering and PXI devices Alias names.** These are optional steps and can be performed if you would like your chassis to have different numbers. Legacy slots numbers are used by older Marvin Test Solutions or VISA drivers. Alias names can provide a way to address a PXI device using a logical name (e.g. "FPGA1"). For more information regarding slot numbers and alias names, see the **GX3700Initialize** and **GxFpgaInitializeVisa** functions.
4. **Save your work**. PXI Explorer saves the configuration to the following files located in the Windows folder: PXISYS.ini, PXIeSYS.ini and GxPxiSys.ini. Click on the **Save** button to save your changes. The PXI/Explorer will prompt you to save the changes if changes were made or detected (an asterisk sign ' *' in the caption indicated changes).

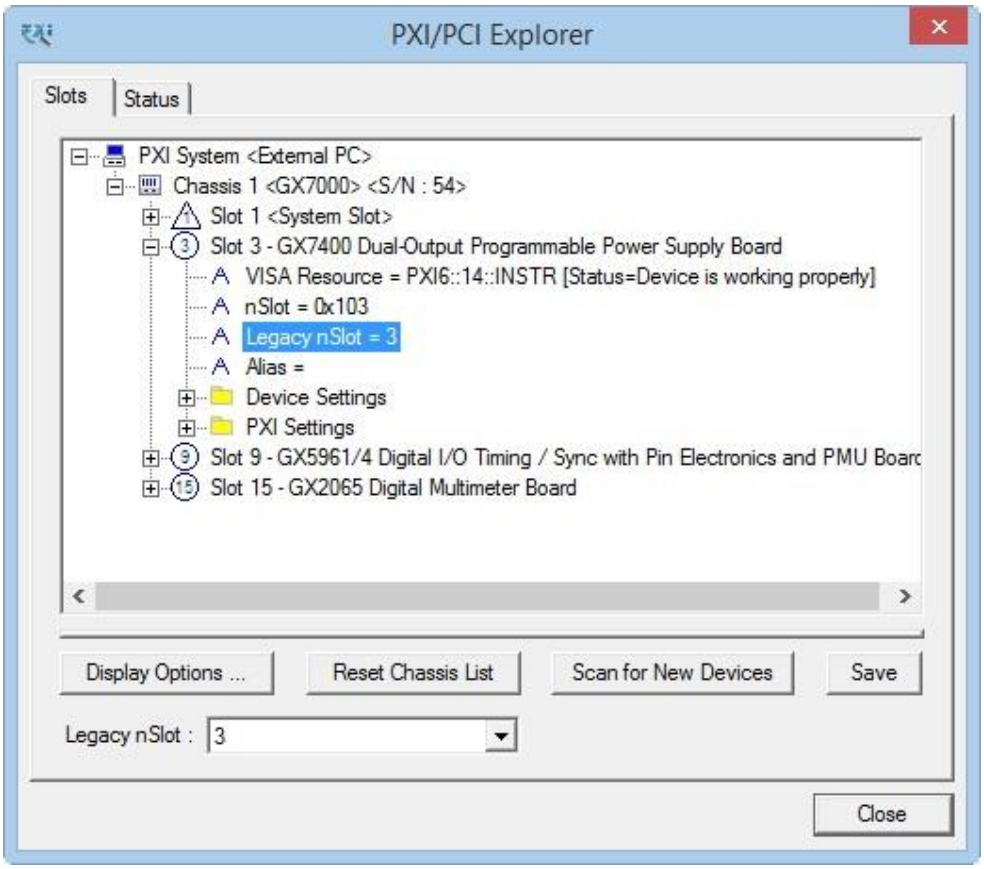


Figure 3-1: PXI/PCI Explorer

Board Installation

Before you Begin

- Install the GXFPGA driver as described in the prior section.
- Configure your PXI/PC system using **PXI/PCI Explorer** as described in the prior section.
- Verify that all the components listed in the packing list (see previous section in this chapter) are present.

Electric Static Discharge (ESD) Precautions

To reduce the risk of damage to the GX3700 board, the following precautions should be observed:

- Leave the board in the anti-static bags until installation requires removal. The anti-static bag protects the board from harmful static electricity.
- Save the anti-static bag in case the board is removed from the computer in the future.
- Carefully unpack and install the board. Do not drop or handle the board roughly.
- Handle the board by the edges. Avoid contact with any components on the circuit board.

Caution – Do not insert or remove any board while the computer is on. Turn off the power from the PXI chassis before installation.

Installing a Board

Install the board as follows:

1. Install first the GXFPGA Driver as described in the next section.
2. Turn off the PXI chassis and unplug the power cord.
3. Locate a PXI empty slot on the PXI chassis.
4. Place the module edges into the PXI chassis rails (top and bottom).
5. Carefully slide the PXI board to the rear of the chassis, make sure that the ejector handles are pushed **out** (as shown in Figure 3-2).

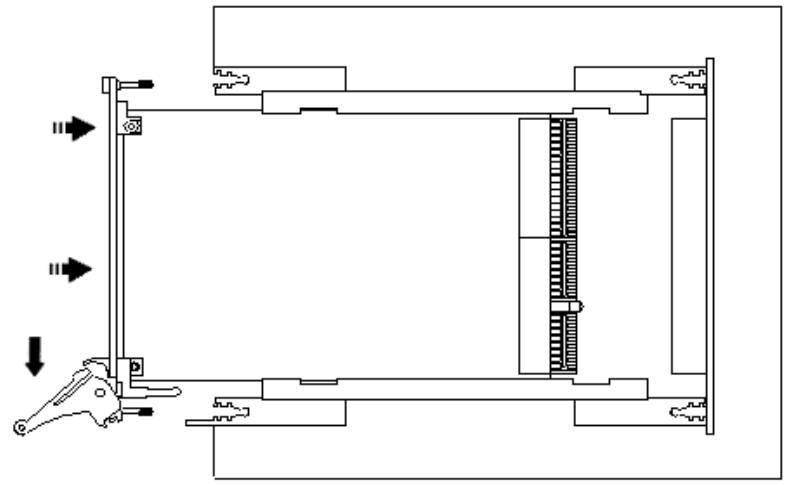


Figure 3-2: Ejector handles position during module insertion

6. After you feel resistance, push in the ejector handles as shown in Figure 3-3 to secure the module into the frame.

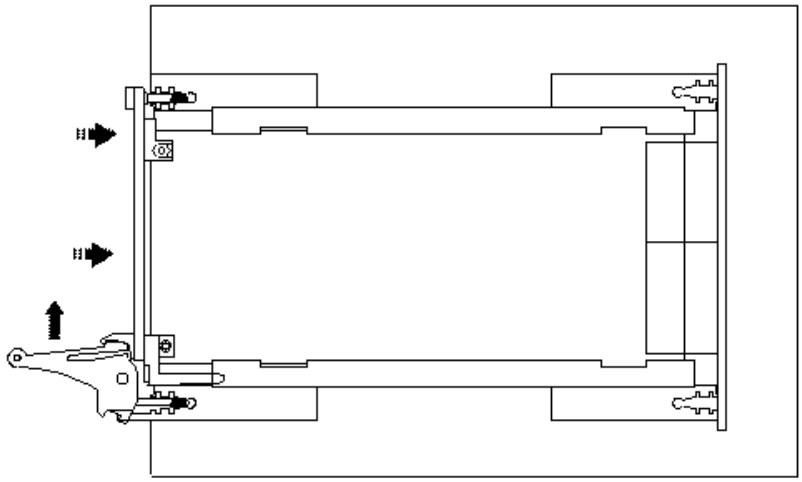


Figure 3-3: Ejector handles position after module insertion

7. Tighten the module's front panel to the chassis to secure the module in.
8. Connect any necessary cables to the board.
9. Plug the power cord in and turn on the PXI chassis.

Plug & Play Driver Installation

Plug & Play operating systems such as Windows notifies the user that a new board was found using the **New Hardware Found** wizard after restarting the system with the new board.

If another Marvin Test Solutions board software package was already installed, Windows will suggest using the driver information file: HW.INF. The file is located in your Program Files\Marvin Test Solutions\HW folder. Click **Next** to confirm and follow the instructions on the screen to complete the driver installation.

If the operating system was unable to find the driver (since the GXFPGA driver was not installed prior to the board installation), you may install the GXFPGA driver as described in the prior section, then click on the **Have Disk** button and browse to select the HW.INF file located in **C:\Program Files [(x86)]\Marvin Test Solutions\HW**.

If you are unable to locate the driver click **Cancel** to the found New Hardware wizard and exit the New Hardware Found Wizard, install the GXFPGA driver, reboot your computer and repeat this procedure.

The Windows Device Manager (open from the System applet from the Windows Control Panel) must display the proper board name before continuing to use the board software (no Yellow warning icon shown next to device). If the device is displayed with an error, you can select it and press delete and then press F5 to rescan the system again and to start the New Hardware Found wizard.

Removing a Board

Remove the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI slot on the PXI chassis.
3. Disconnect and remove any cables/connectors connected to the board.
4. Un-tighten the module's front panel screws to the chassis.
5. Push out the ejector handles and slide the PXI board away from the chassis.
6. Optionally – uninstall the GXFPGA driver.

GX3701 Connectors

These connectors exist only with the GX3701 daughter board card mounted on the GX3700/GX3700e.

Connector	Description
J1	FLEX I/O differential channels 1-32 or single ended 1-64
J2	FLEX I/O channels 33-64
J3	FLEX I/O channels 65-96
J4	FLEX I/O channels 97-128

Table 3-1: GX3701 Connectors

Figure 3-4: GX3701e Connectors J1-J4

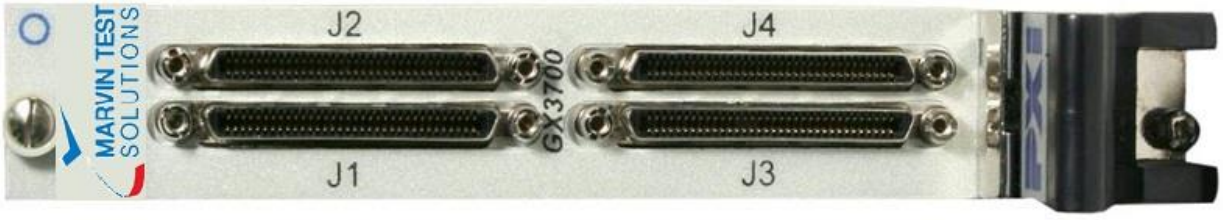


Figure 3-5: GX3701 Connectors J1-J4

Connections to the GX3700/GX3701 may be made with 68-pin VHDCI male plug connector. Shielded cables with matching connectors are available from Marvin Test Solutions.

The following section describes J1-J4 connectors.

GX3701 J1 – Flex I/O Connector

Pin	Function	Pin	Function	Pin	Function	Pin	Function
1	Flex I/O 1P	18	Flex I/O 18P	35	Flex I/O 1N	52	Flex I/O 18N
2	Flex I/O 2P	19	Flex I/O 19P	36	Flex I/O 2N	53	Flex I/O 19N
3	Flex I/O 3P	20	Flex I/O 20P	37	Flex I/O 3N	54	Flex I/O 20N
4	Flex I/O 4P	21	Flex I/O 21P	38	Flex I/O 4N	55	Flex I/O 21N
5	Flex I/O 5P	22	Flex I/O 22P	39	Flex I/O 5N	56	Flex I/O 22N
6	Flex I/O 6P	23	Flex I/O 23P	40	Flex I/O 6N	57	Flex I/O 23N
7	Flex I/O 7P	24	Flex I/O 24P	41	Flex I/O 7N	58	Flex I/O 24N
8	Flex I/O 8P	25	Flex I/O 25P	42	Flex I/O 8N	59	Flex I/O 25N
9	Flex I/O 9P	26	Flex I/O 26P	43	Flex I/O 9N	60	Flex I/O 26N
10	Flex I/O 10P	27	Flex I/O 27P	44	Flex I/O 10N	61	Flex I/O 27N
11	Diff Clock Input P	28	Flex I/O 28P	45	Diff Clock Input N	62	Flex I/O 28N
12	Flex I/O 12P	29	Flex I/O 29P	46	Flex I/O 12N	63	Flex I/O 29N
13	Diff Clock Input P	30	Flex I/O 30P	47	Diff Clock Input N	64	Flex I/O 30N
14	Flex I/O 14P	31	Flex I/O 31P	48	Flex I/O 14N GND	65	Flex I/O 31N
15	Flex I/O 15P	32	Flex I/O 32P	49	Flex I/O 15N	66	Flex I/O 32N
16	Flex I/O 16P	33	User 3.3V	50	Flex I/O 16N	67	User 3.3V
17	Flex I/O 17P	34	GND	51	Flex I/O 17N GND	68	GND

Table 3-2: GX3701 J1 Flex IO Pin Out

P: positive differential I/O signal (e.g. Flex I/O 1P)

N: negative differential I/O signal (e.g. Flex I/O 1N)

Diff Clock Input: Dedicated differential clock inputs.

GX3701 J2 – Flex I/O Connector

Pin #	Function	Pin #	Function	Pin #	Function	Pin #	Function
1	Flex I/O 33P	18	Flex I/O 42N	35	GND	52	GND
2	Flex I/O 34N	19	Flex I/O 41N	36	GND	53	GND
3	Flex I/O 33N	20	Flex I/O 42P	37	GND	54	GND
4	Flex I/O 34P	21	Flex I/O 53	38	GND	55	GND
5	Flex I/O 35P	22	Flex I/O 54	39	GND	56	GND
6	Flex I/O 36N	23	Flex I/O 55	40	GND	57	GND
7	Flex I/O 35N	24	Flex I/O 56	41	GND	58	GND
8	Flex I/O 36P	25	Flex I/O 57	42	GND	59	GND
9	Flex I/O 37P	26	Flex I/O 58	43	GND	60	GND
10	Flex I/O 38N	27	Flex I/O 59	44	GND	61	GND
11	Flex I/O 37N	28	Flex I/O 60	45	GND	62	GND
12	Flex I/O 38P	29	Flex I/O 61	46	GND	63	GND
13	Flex I/O 39P	30	Flex I/O 62	47	GND	64	GND
14	Flex I/O 40N	31	Flex I/O 63	48	GND	65	GND
15	Flex I/O 39N	32	Flex I/O 64	49	GND	66	GND
16	Flex I/O 40P	33	User 3.3V	50	GND	67	User 3.3V
17	Flex I/O 41P	34	GND	51	GND	68	GND

Table 3-3: GX3701 J2 Flex IO Pin Out

P: positive differential I/O signal (e.g. Flex I/O 1P)

N: negative differential I/O signal (e.g. Flex I/O 1N)

GX3701 J3 – Flex I/O Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	Flex I/O 65	18	Flex I/O 82	35	GND	52	GND
2	Flex I/O 66	19	Flex I/O 83	36	GND	53	GND
3	Flex I/O 67	20	Flex I/O 84	37	GND	54	GND
4	Flex I/O 68	21	Flex I/O 85	38	GND	55	GND
5	Flex I/O 69	22	Flex I/O 86	39	GND	56	GND
6	Flex I/O 70	23	Flex I/O 87	40	GND	57	GND
7	Flex I/O 71	24	Flex I/O 88	41	GND	58	GND
8	Flex I/O 72	25	Flex I/O 89	42	GND	59	GND
9	Flex I/O 73	26	Flex I/O 90	43	GND	60	GND
10	Flex I/O 74	27	Flex I/O 91	44	GND	61	GND
11	Flex I/O 75	28	Flex I/O 92	45	GND	62	GND
12	Flex I/O 76	29	Flex I/O 93	46	GND	63	GND
13	Flex I/O 77	30	Flex I/O 94	47	GND	64	GND
14	Flex I/O 78	31	Flex I/O 95	48	GND	65	GND
15	Flex I/O 79	32	Flex I/O 96	49	GND	66	GND
16	Flex I/O 80	33	User 5V	50	GND	67	User 5V
17	Flex I/O 81	34	GND	51	GND	68	GND

Table 3-4: GX3701 J3 Flex IO Pin Out

GX3701 J4 – Flex I/O Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	Flex I/O 97	18	Flex I/O 114	35	GND	52	GND
2	Flex I/O 98	19	Flex I/O 115	36	GND	53	GND
3	Flex I/O 99	20	Flex I/O 116	37	GND	54	GND
4	Flex I/O 100	21	Flex I/O 117	38	GND	55	GND
5	Flex I/O 101	22	Flex I/O 118	39	GND	56	GND
6	Flex I/O 102	23	Flex I/O 119	40	GND	57	GND
7	Flex I/O 103	24	Flex I/O 120	41	GND	58	GND
8	Flex I/O 104	25	Flex I/O 121	42	GND	59	GND
9	Flex I/O 105	26	Flex I/O 122	43	GND	60	GND
10	Flex I/O 106	27	Flex I/O 123	44	GND	61	GND
11	Flex I/O 107	28	Flex I/O 124	45	GND	62	GND
12	Flex I/O 108	29	Flex I/O 125	46	GND	63	GND
13	Flex I/O 109	30	Flex I/O 126	47	GND	64	GND
14	Flex I/O 110	31	Flex I/O 127	48	GND	65	GND
15	Flex I/O 111	32	Flex I/O 128	49	GND	66	GND
16	Flex I/O 112	33	User 5V	50	GND	67	User 5V
17	Flex I/O 113	34	GND	51	GND	68	GND

Table 3-5: GX3701 J4 Flex IO Pin Out

The GX3701 J7 connector is for internal use only and is not user accessible.

GX3788 Connectors

These connectors exist only with the GX3788 daughter board card mounted on the GX3700/GX3700e.

Connector	Description
J1	Digital Port 0 Channels 0-31 Differential
J2	Digital Port 2 Channels 0-31
J3	Analog Input Channels 0-15 and Analog Output Channels 0-7
J4	Miscellaneous.

Table 3-6: GX3788 Connectors

Connections to the GX3700/GX3788 may be made with 68-pin VHDCI male plug connector. Shielded cables with matching connectors are available from Marvin Test Solutions.

The following section describes J1-J4 connectors:

GX3788 J1 – Flex I/O Bank A Connector

This connector has 31 differential channels (P used for Positive, N for Negative pins).

Pin	Function	Pi	Function	Pi	Function	P	Function
1	DIO Port 0 Ch 0P	18	DIO Port 0 Ch 17P	35	DIO Port 0 Ch 0N	5	DIO Port 0 Ch 17N
2	DIO Port 0 Ch 1P	19	DIO Port 0 Ch 18P	36	DIO Port 0 Ch 1N	5	DIO Port 0 Ch 18N
3	DIO Port 0 Ch 2P	20	DIO Port 0 Ch 19P	37	DIO Port 0 Ch 2N	5	DIO Port 0 Ch 19N
4	DIO Port 0 Ch 3P	21	DIO Port 0 Ch 20P	38	DIO Port 0 Ch 3N	5	DIO Port 0 Ch 20N
5	DIO Port 0 Ch 4P	22	DIO Port 0 Ch 21P	39	DIO Port 0 Ch 4N	5	DIO Port 0 Ch 21N
6	DIO Port 0 Ch 5P	23	DIO Port 0 Ch 22P	40	DIO Port 0 Ch 5N	5	DIO Port 0 Ch 22N
7	DIO Port 0 Ch 6P	24	DIO Port 0 Ch 23P	41	DIO Port 0 Ch 6N	5	DIO Port 0 Ch 23N
8	DIO Port 0 Ch 7P	25	DIO Port 0 Ch 24P	42	DIO Port 0 Ch 7N	5	DIO Port 0 Ch 24N
9	DIO Port 0 Ch 8P	26	DIO Port 0 Ch 25P	43	DIO Port 0 Ch 8N	6	DIO Port 0 Ch 25N
10	DIO Port 0 Ch 9P	27	DIO Port 0 Ch 26P	44	DIO Port 0 Ch 9N	6	DIO Port 0 Ch 26N
11	DIO Port 0 Ch 10P	28	DIO Port 0 Ch 27P	45	DIO Port 0 Ch 10N	6	DIO Port 0 Ch 27N
12	DIO Port 0 Ch 11P	29	DIO Port 0 Ch 28P	46	DIO Port 0 Ch 11N	6	DIO Port 0 Ch 28N
13	DIO Port 0 Ch 12P	30	DIO Port 0 Ch 29P	47	DIO Port 0 Ch 12N	6	DIO Port 0 Ch 29N
14	DIO Port 0 Ch 13P	31	DIO Port 0 Ch 30P	48	DIO Port 0 Ch 13N	6	DIO Port 0 Ch 30N
15	DIO Port 0 Ch 14P	32	DIO Port 0 Ch 31P	49	DIO Port 0 Ch 14N	6	DIO Port 0 Ch 31N
16	DIO Port 0 Ch 15P	33	User 3.3V	50	DIO Port 0 Ch 15N	6	User 3.3V
17	DIO Port 0 Ch 16P	34	GND	51	DIO Port 0 Ch 16N	6	GND

Table 3-7: J1 Flex IO Bank A Pin Out

GX3788 J2 – Flex I/O Bank D Connector

Pin	Function	Pin	Function	Pin #	Function	Pin #	Function
1	DIO Port 2 Ch 0	18	DIO Port 2 Ch 17	35	GND	52	GND
2	DIO Port 2 Ch 1	19	DIO Port 2 Ch 18	36	GND	53	GND
3	DIO Port 2 Ch 2	20	DIO Port 2 Ch 19	37	GND	54	GND
4	DIO Port 2 Ch 3	21	DIO Port 2 Ch 20	38	GND	55	GND
5	DIO Port 2 Ch 4	22	DIO Port 2 Ch 21	39	GND	56	GND
6	DIO Port 2 Ch 5	23	DIO Port 2 Ch 22	40	GND	57	GND
7	DIO Port 2 Ch 6	24	DIO Port 2 Ch 23	41	GND	58	GND
8	DIO Port 2 Ch 7	25	DIO Port 2 Ch 24	42	GND	59	GND
9	DIO Port 2 Ch 8	26	DIO Port 2 Ch 25	43	GND	60	GND
10	DIO Port 2 Ch 9	27	DIO Port 2 Ch 26	44	GND	61	GND
11	DIO Port 2 Ch 10	28	DIO Port 2 Ch 27	45	GND	62	GND
12	DIO Port 2 Ch 11	29	DIO Port 2 Ch 28	46	GND	63	GND
13	DIO Port 2 Ch 12	30	DIO Port 2 Ch 29	47	GND	64	GND
14	DIO Port 2 Ch 13	31	DIO Port 2 Ch 30	48	GND	65	GND
15	DIO Port 2 Ch 14	32	DIO Port 2 Ch 31	49	GND	66	GND
16	DIO Port 2 Ch 15	33	User 3.3V	50	GND	67	User 3.3V
17	DIO Port 2 Ch 16	34	GND	51	GND	68	GND

Table 3-8: J2 Flex IO Bank D Pin Out

GX3788 J3 – Flex I/O Bank B Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	Analog In 0	18	Analog In 15	35	COM1	52	COM2
2	Analog In 1	19	NC	36	COM1	53	AGND
3	Analog In 2	20	NC	37	COM1	54	AGND
4	Analog In 3	21	CAL_V	38	COM1	55	CAL_GND
5	Analog In 4	22	NC	39	COM1	56	NC
6	Analog In 5	23	NC	40	COM1	57	GND
7	Analog In 6	24	NC	41	COM1	58	GND
8	Analog In 7	25	Analog Out 0	42	COM1	59	AGND
9	NC	26	Analog Out 1	43	AGND	60	AGND
10	NC	27	Analog Out 2	44	AGND	61	AGND
11	Analog In 8	28	Analog Out 3	45	COM2	62	AGND
12	Analog In 9	29	Analog Out 4	46	COM2	63	AGND
13	Analog In 10	30	Analog Out 5	47	COM2	64	AGND
14	Analog In 11	31	Analog Out 6	48	COM2	65	AGND
15	Analog In 12	32	Analog Out 7	49	COM2	66	AGND
16	Analog In 13	33	User 5V	50	COM2	67	User 5V
17	Analog In 14	34	AGND	51	COM2	68	AGND

Table 3-9: J3 Flex IO Bank B Pin Out

GX3788 J4 – Flex I/O Bank C Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	NC	18	GND	35	GND	52	GND
2	NC	19	NC	36	GND	53	GND
3	NC	20	NC	37	GND	54	GND
4	NC	21	NC	38	GND	55	GND
5	NC	22	NC	39	GND	56	GND
6	NC	23	NC	40	GND	57	GND
7	NC	24	NC	41	GND	58	GND
8	NC	25	NC	42	GND	59	GND
9	NC	26	NC	43	GND	60	GND
10	NC	27	NC	44	GND	61	GND
11	NC	28	NC	45	GND	62	GND
12	NC	29	NC	46	GND	63	GND
13	NC	30	NC	47	GND	64	GND
14	NC	31	NC	48	GND	65	GND
15	NC	32	NC	49	GND	66	GND
16	NC	33	User 5V	50	GND	67	User 5V
17	NC	34	GND	51	GND	68	GND

Table 3-10: J4 Flex IO Bank C Pin Out

Jumpers

Jumpers	Description
JP2	Reserved. Normally disconnected.
JP3	Connect 3.3V to VCCIO for customer programmable FPGA. Normally connected.
JP4	Connect 2.5V to VCCIO for customer programmable FPGA. Normally disconnected.
JP5	Connect 1.2V to VCCIO for customer programmable FPGA. Normally disconnected.

Table 3-11: GX3700 Jumpers

Figure 3-6 shows GX3700 board JP2, JP3, JP4 and JP5 jumpers (in red rectangular):

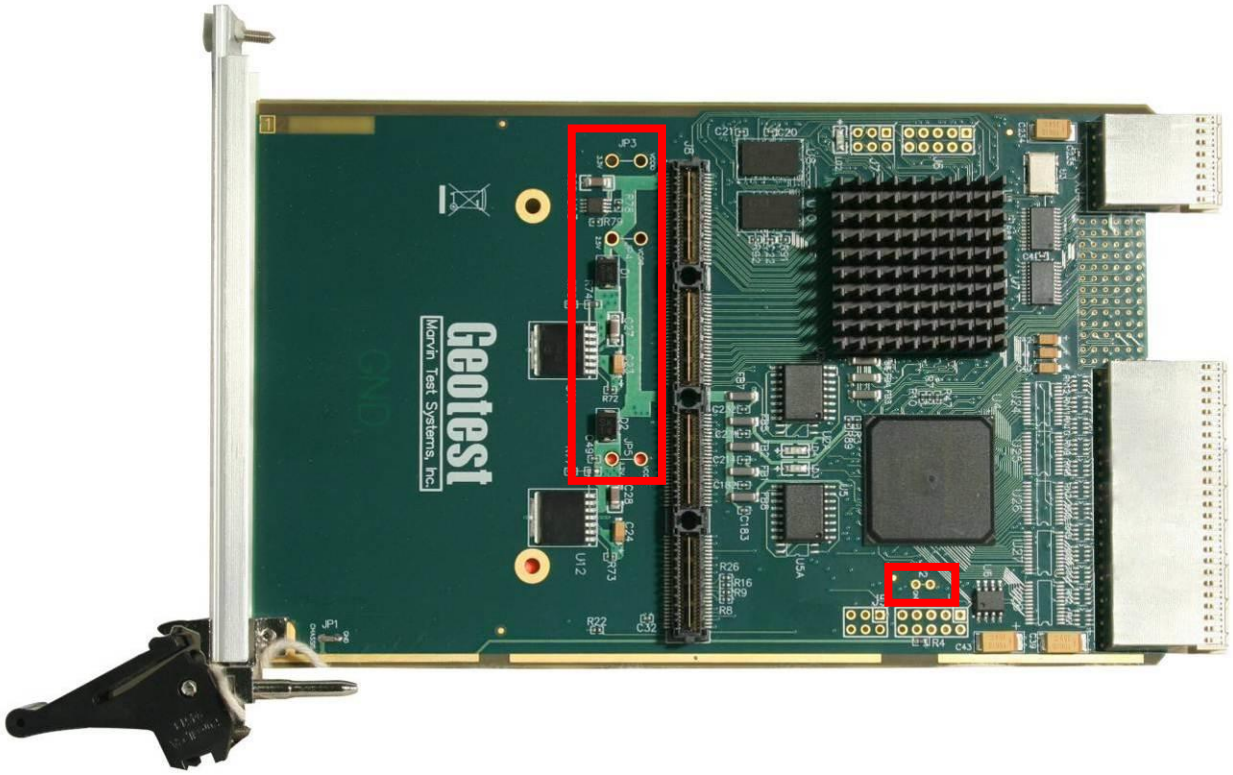


Figure 3-6: GX3700 – Front View Jumpers JP3-JP5 and JP2

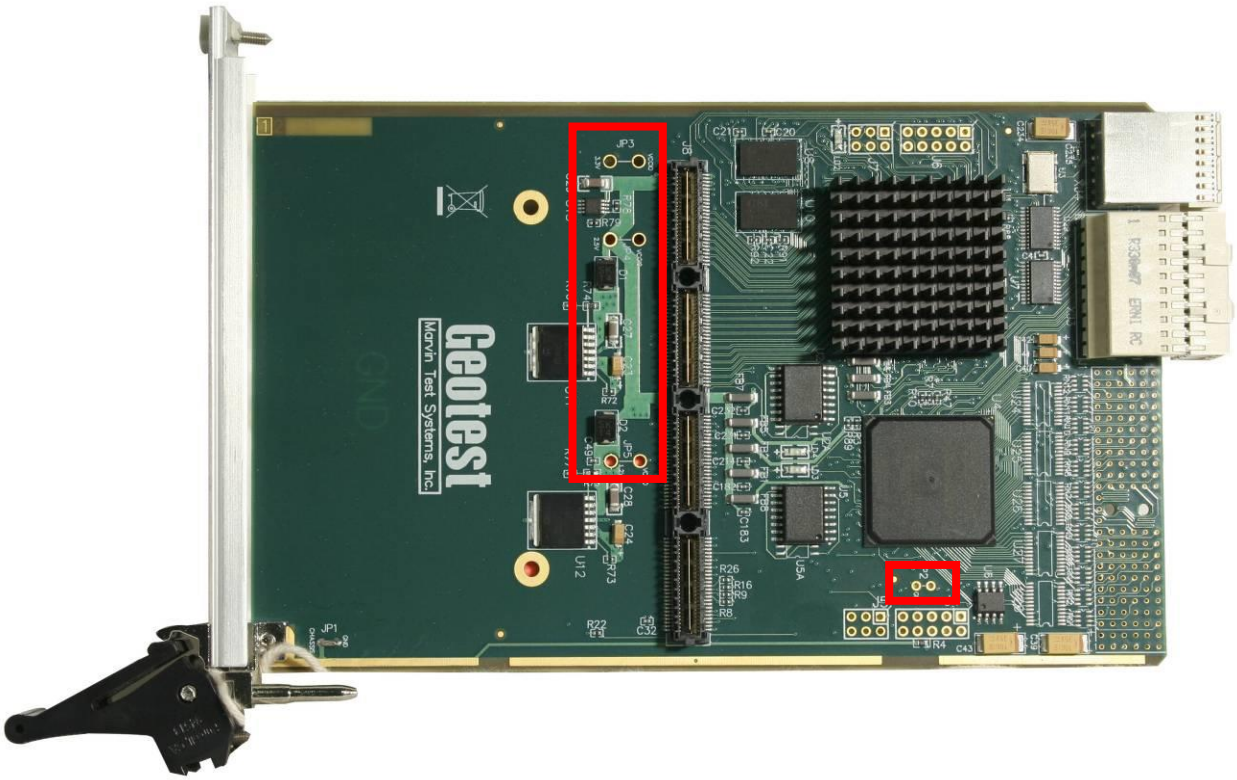


Figure 3-7: GX3700e – Front View Jumpers JP3-JP5 and JP2

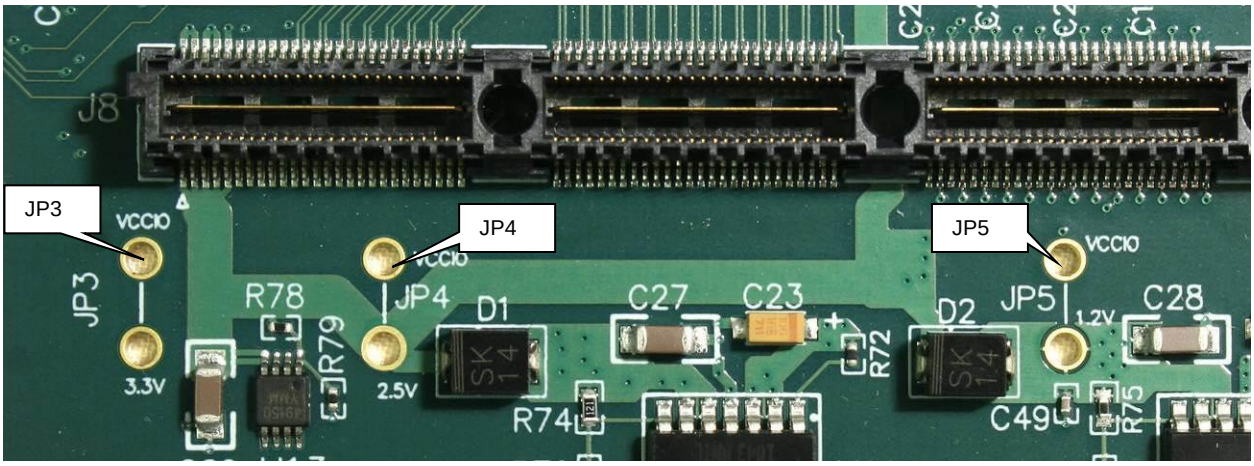


Figure 3-8: GX3700/GX3700e Jumpers JP3-JP5

Figure 3-9 shows GX3700 board JP2 Jumper:

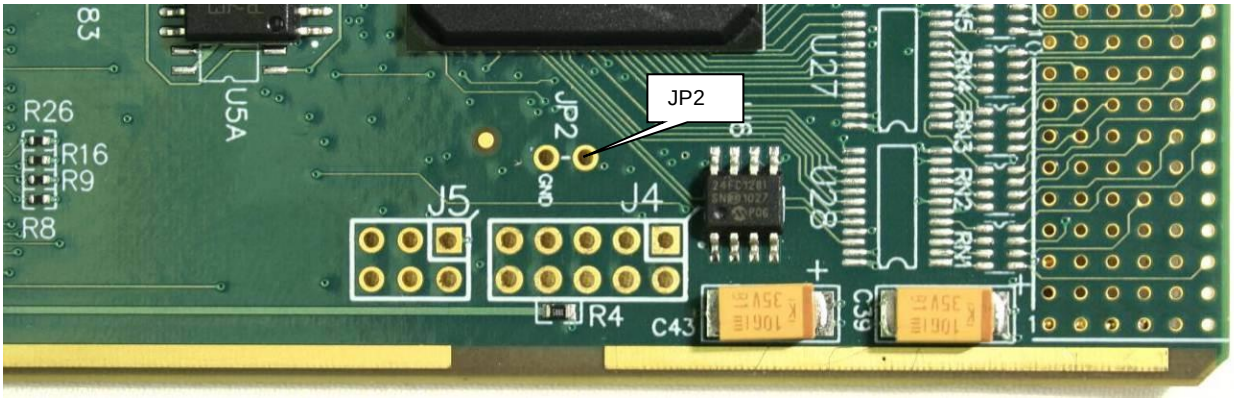


Figure 3-9: GX3700/GX3700e Jumper JP2

Chapter 4 - Programming the Board

This chapter contains information about how to program the GX3700 board using the GXFPGA driver.

The GXFPGA driver contains functions to initialize, reset, and control the board. A brief description of the functions, as well as how and when to use them, is included in this chapter.

The GXFPGA driver supports many development tools. Using these tools with the driver is described in this chapter. In addition, the GXFPGA directory contains examples written for these development tools.

The GXFPGA Driver

The GXFPGA DLL driver is provided with support for 32 bit Windows (GXFPGA.DLL) and 64 bit Windows (GXFPGA64.DLL). Additional drivers are provided for other operating systems such as Linux and LabVIEW Real-Time, see the readme file for more information regarding these drivers. The 32-bit DLL is used with 32 bit applications running under Windows 32 or 64 bit and the 64-bit DLL runs on Windows 64 bit editions. The DLLs uses device driver (HW provided by Marvin Test Solutions or VISA provided by a third-party vendor) to access the board resources. The device driver HW includes HW.SYS and HW64.SYS is installed by the GXFPGA setup program and is shared by other Marvin Test Solutions products (ATEasy, GTDIO, etc.).

The DLLs can be used with various development tools such as Microsoft Visual C++, Borland C++ Builder, Microsoft Visual Basic, Borland Pascal or Delphi, ATEasy and more. The following paragraphs describe how to create an application that uses the driver with various development tools. Refer to the paragraph describing the specific development tool for more information.

Programming Using C/C++ Tools

The following steps are required to use the GXFPGA driver with C/C++ development tools:

- Include the GXFPGA.h header file in the C/C++ source file that uses the GXFPGA function. This header file is used for all driver types. The file contains function prototypes and constant declarations to be used by the compiler for the application.
- Add the required .LIB file to the projects. This can be import library GXFPGA.lib and GXFPGA64.lib (for 64 bit applications) for Microsoft Visual C++ and GXFPGABC.lib for Borland C++. Windows based applications that explicitly load the DLL by calling the Windows **LoadLibrary()** API should not include the .LIB file in the project.
- Add code to call the GXFPGA as required by the application.
- Build the project.
- Run, test, and debug the application.

Programming Using Visual Basic and Visual Basic .NET

To use the driver with Visual Basic 4.0 or above (for 32-bit applications), the user must include the GXFPGA.bas to the project. The file can be loaded using *Add File* from the Visual Basic *File menu*. The GXFPGA.bas contains function declarations for the DLL driver. If you are using Visual Basic .NET – use the GXFPGA.vb.

Programming Using Pascal/Delphi

To use the driver with Borland Pascal or Delphi, the user must include the GXFPGA.pas to the project. The GXFPGA.pas file contains a **unit** with function prototypes for the DLL functions. Include the GXFPGA unit in the **uses** statement before making calls to the GXFPGA functions.

Programming GXFPGA Boards Using ATEasy®

The GXFPGA package is supplied with a separate ATEasy driver for each board types. For example, the GX3700 is supplied with GXFPGA.drv ATEasy driver. The ATEasy driver uses the GXFPGA.dll to program the board. In addition, each driver is supplied with an example that contains a program and a system file pre-configured with the ATEasy driver. Use the driver shortcut property page from the System Drivers sub-module to change the PXI HW slot number or VISA resource string before attempting to run the example.

Using commands declared in the ATEasy driver are easier to use than using the DLL functions directly. The driver commands will also generate exceptions that allow the ATEasy application to trap errors without checking the status code returned by the DLL function after each function call.

The ATEasy driver contains commands that are similar to the DLL functions in name and parameters, with the following exceptions:

- The *nHandle* parameter is omitted. The driver handles this parameter automatically. ATEasy uses driver logical names instead i.e. FPGA1 for GX3700.
- The *nStatus* parameter was omitted. Use the Get Status commands instead of checking the status. After calling a DLL function the ATEasy driver will check the returned status and will call the error statement (in case of an error status) to generate exception that can be easily trapped by the application using the **OnError** module event or using the **try-catch** statement.

Some ATEasy drivers contain additional commands to permit easier access to the board features. For example, parameters for a function may be omitted by using a command item instead of typing the parameter value. The commands are self-documented. Their syntax is similar to English. In addition, you may generate the commands from the code editor context menu or by using the ATEasy's code completion feature instead of typing them directly.

Programming Using LabVIEW and LabVIEW/Real Time

To use the driver with LabVIEW use the provided lab view library GXFPGA.llb. The library is located in the GXFPGA folder. An example for LabView is also provided in the Examples folder. A DLL located in the LabViewRT folder can be used for deployment with LabVIEW/Real-Time.

Using and Programming under Linux

Marvin Test Solutions provides a separate software package **GtLinux** with Linux driver (Marvin Test Solutions Drivers Pack for Linux). The software package can be downloaded from the Marvin Test Solutions website. See the ReadMe.txt in that package for more information regarding using and programming the driver under Linux.

Using the GXFPGA driver functions

The following paragraphs describe the steps required to program the boards.

Initialization, HW Slot Numbers and VISA Resource

The GXFPGA driver supports two device drivers HW and VISA which are used to initialize, identify and control the board. The user can use the **GxFpgaInitialize** to initialize the board 's driver using HW and **GxFpgaInitializeVisa** to initialize using VISA. The following describes the two different methods used to initialize:

1. **Marvin Test Solutions's HW** – This is the default device driver that is installed by the GXFPGA driver. To initialize and control the board using the HW use the **GxFpgaInitialize**(*nSlot, pnHandle, pnStatus*) function. The function initializes the driver for the board at the specified PXI slot number (*nSlot*) and returns boards handle. The **PXI/PCI Explorer** applet in the Windows Control Panel displays the PXI slot assignments. You can specify the *nSlot* parameter in the following way:
   - A combination of chassis number (chassis # x 256) with the chassis slot number, e.g. 0x105 for chassis 1 and slot 5. The chassis number can be set by the **PXI/PCI Explorer** applet.
   - Legacy nSlot is used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the PXI/PCI Explorer applet: 23 in this example.

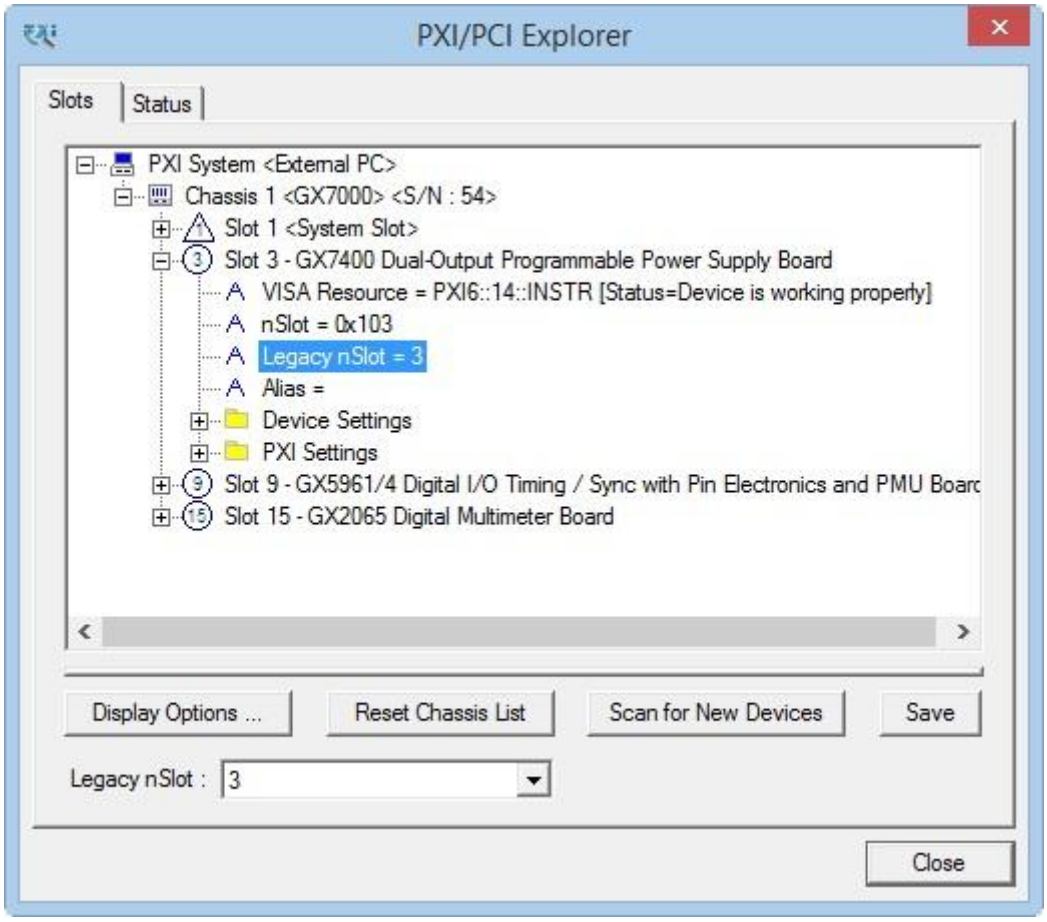


Figure 4-1: PXI/PCI Explorer

2. **VISA** – This is a third-party library usually supplied by National Instruments (NI-VISA). You must ensure that the VISA installed supports PXI and PCI devices (not all VISA providers supports PXI/PCI). GXFPGA setup

installs a VISA compatible driver for the GXFPGA board in-order to be recognized by the VISA provider. Use the GXFPGA function **GxFpgaInitializeVisa** (*szVisaResource, pnHandle, pnStatus*) to initialize the driver's board using VISA. The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI **Measurement and Automation** (NI_MAX). It is also displayed by Marvin Test Solutions **PXI/PCI Explorer** as shown in the prior figure. The VISA resource string can be specified in several ways as the following examples demonstrate:

- Using chassis, slot: "PXI0::CHASSIS1::SLOT5"
- Using the PCI Bus/Device combination: "PXI9::13::INSTR" (bus 9, device 9).
- Using the alias: for example, "COUNTER1". Use the PXI/PCI Explorer to set the device alias.

Information about VISA is available at http://www.pxisa.org.

Board Handle

The **GxFpgaInitialize** and the **GxFpgaInitializeVisa** functions return a handle that is required by other driver functions in order to program the board. This handle is usually saved in the program as a global variable for later use when calling other functions. The initialize functions do not change the state of the board or its settings.

Reset

The Reset function sets the board to a known default state. A reset is usually performed after the board is initialized. See the Function Reference for more information regarding the reset function.

Error Handling

All the GXFPGA functions returns status - *pnStatus* - in the last parameter. This parameter can be later used for error handling. The status is zero for success status or less than zero for errors. When the status is error, the program can call the **GxFpgaGetErrorString** function to return a string representing the error. The **GxFpgaGetErrorString** reference contains possible error numbers and their associated error strings.

Driver Version

The **GxFpgaGetDriverSummary** function can be used to return the current GXFPGA driver version. It can be used to differentiate between the driver versions. See the Function Reference for more information.

Programming Examples

The README.txt located on the GXFPGA folder contains a list of the GXFPGA programming examples provided with the GXFPGA software. Examples are provided for various programming languages including C, VB.NET, VB (6.0). ATEasy and more.

Distributing the Driver

Once the application is developed, the driver files (GXFPGA.dll, GXFPGA64.dll and the HW device driver files) can be shipped with the application. Typically, the GXFPGA.dll should be copied to the Windows System directory. The HW device driver files should be installed using a special setup program HWSETUP.EXE that is provided with GXFPGA driver files (see Marvin Test Solutions\HW folder) or a standalone setup HW.exe. Alternatively, you can provide the GXFPGA.exe setup to be installed along with the board.

Chapter 5 - GXFPGA Schematic Entry Tutorial

Introduction

This tutorial will go over the basic workflow to start designing and loading a FPGA configuration for the Gx3700. The example provides creation of a project using the schematic entry design method. The "Tutorial design top reg.doc" contains the design register map.

The tutorial contents will entail:

- Downloading and installing the FPGA design tool
- Creating a new FPGA Design project with the Stratix III as the target device
- Setup the pin assignment to work with the GX3700 and Stratix III FPGA
- Use the design tool to create an example FPGA configuration
- Compile the project and generate the SVF and RPD programming files
- Loading the board with the generated programming files
- Testing the design using the Gx3700 Front Panel software and ATEasy
- The example configuration is broken down into three phases, each with a distinct function:
    - **Phase 1:** Take two values located in PCI Registers and generate a Sum (Adder) which can then be read through a third PCI Register.
    - **Phase 2:** 2 to 1 multiplexer to choose between the 10 MHz Clock and the PCI Clock which will be output on one of the FlexIO pins. The clock will be selected through a PCI Register.
    - **Phase 3:** A simple dynamic digital sequencer with a memory depth of 32 double words (written to through the PCI bus) driven by a PLL that continuously outputs digital patterns to the 32 FlexIO pins on J2 connector.
- The source code for the examples in this chapter is provided in the Examples\Quartus\Gx3700 folder.

Downloading Altera Design FPGA Design Tools

The Marvin Test Solutions Gx3700 User programmable FPGA board can be designed using the free Altera Quartus II Web Edition or Subscription Edition design tool. This FPGA design tool allows end users to generate fully featured FPGA designs that can be downloaded to the Gx3700 board using the Marvin Test Solutions GXFPGA software API or software front panel. Other 3rd party tools can also be used to design the FPGA. Before proceeding with this tutorial, you must have Altera Quartus II **v11.0 SP1** installed on your PC. More information about this tool and how to download it can be found at http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html.

Create New Project

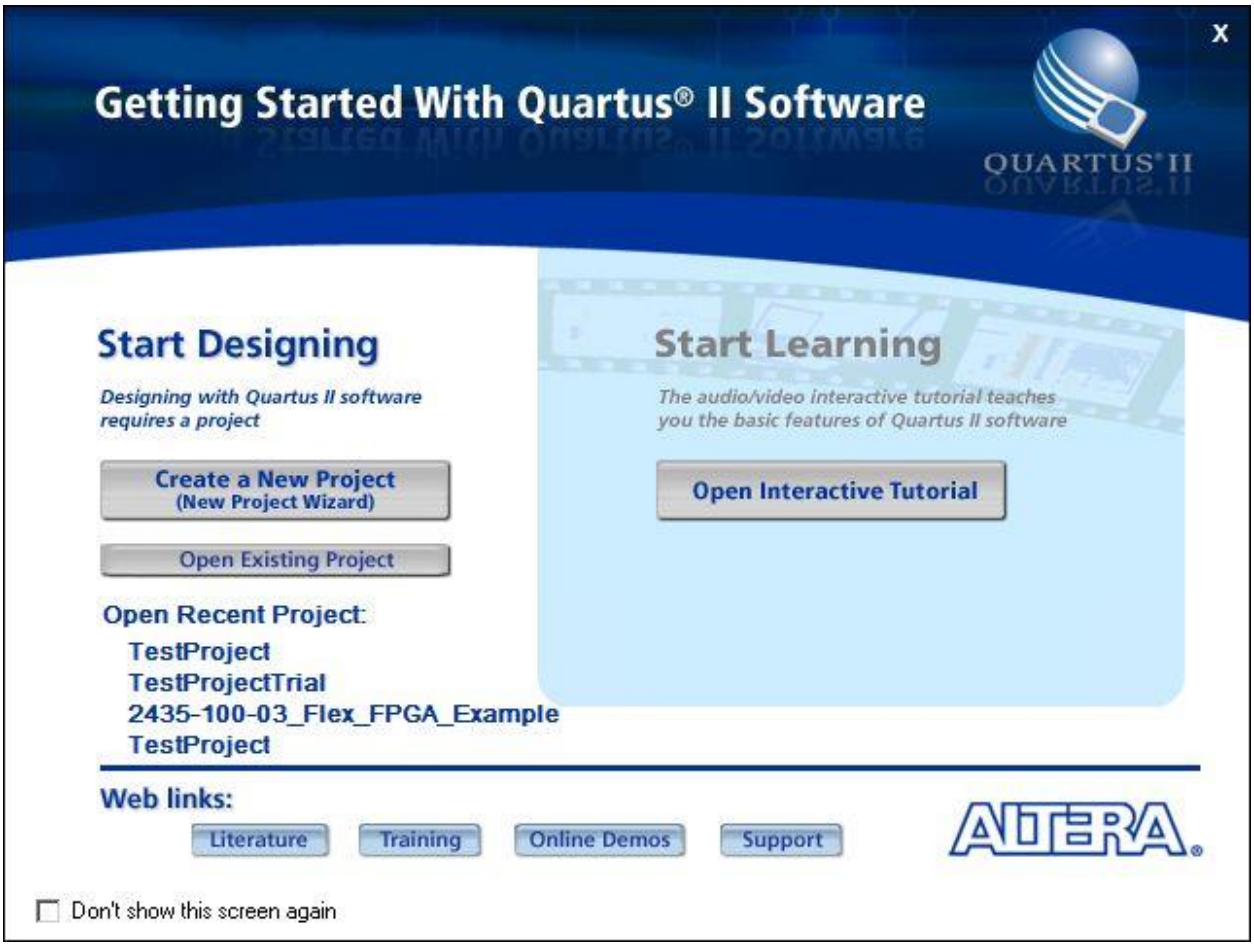


Figure 5-1: Quartus II Start Dialog

After installing Quartus II Web Edition, start the application and select C**reate a new Project** to start the New Project Wizard or select **File, New**, **New Quartus Project**.

Click on **Next** and then select the Project Folder and enter **tutorial_design_top** as the project name.

Click on **Next** twice (skip the adding files window).

Device Selection

The next window will allow you to select the FPGA target device. Select **Stratix III** as the Family and **EP3SL50F780C3** when using the GX3700 or **EP3SL70F780C3** for the GX3700e. For newer GXFPGA boards, the device ID will be displayed in the instrument front panel **About** page.

Click on **Next** twice (skip the Specify Tools window).

A window summarizing all the choices made for the creation of this project is shown. Click on **Finish**.

Pin Assignment Setup

You should now have an empty skeleton project loaded in Quartus II. Before you can get started on the FPGA design, you must assign the FPGA pins distinct names so that you can reference them in your design. This can be accomplished by running a **TCL script** which contains all the information necessary to configure the pin assignments as well as settings the project to e either schematic entry or Verilog entry. These pin assignments are unique to this Stratix III FPGA and the GX3700 in particular. The following table lists all the pin assignments and their respective descriptions. The Pin Alias's listed in the table are the pin names you will be using in your design to reference the actual hardware pins on the FPGA.

Pin Assignments Table

Pin Alias (Node Name)	Description
Clocks	
10Mhz	Input. 10 MHz Reference Clock Signal from the PXI Backplane
PCIClock	Input. 33 MHz PCI Bus clock or 125MHz PCI Express application clock.
RefClk	Input. 80 Mhz Reference Clock onboard the GX3700
PCI Bus	
Addr[2..19]	Input. The PCI Address lines from the PCI bus
FDt[0..31]	Bidir. PCI Data lines from the PCI bus
CS[1..3]	Input. Chip Select lines from the PCI bus. CS[1] is for FPGA registers, CS[2] is for internal SRAM, CS[3] is currently not used.
LEXT	Input. External SRAM chip select. This is chip select for external SRAM on PCB.
RdEn	Input. PCI Read Enable line from the PCI bus
WrEn	Input. PCI Write Enable line from the PCI bus
LREAD_DV	Output. Read data valid. This is data valid for FDt(31:0) data bus.
LUW	Input. Currently not used. Upper Word.
LLW	Input. Currently not used. Lower Word.
LRESET	Input. Currently not used. Reset coming from PXI bridge FPGA.
PXI Bus	
PxiTrig[0..7]	Bidir. PXI Bus trigger signals
StarTrig	Output. PXI Star Trigger signal. This signal can be re-defined by the user as bi-directional.
PXI_LBL6	Bidir. PXI Local Bus Left 6. This is local bus according to PXIe spec.
PXI_LBR6	Bidir. PXI Local Bus Right 6. This is local bus according to PXIe spec.
PXIe_DSTARA	Input. PXIe DSTAR trigger A. This is DSTAR trigger according to PXIe spec.
PXIe_DSTARB	Input. PXIe DSTAR trigger B. This is DSTAR trigger according to PXIe spec.
PXIe_DSTARC	Output. PXIe DSTAR trigger C. This is DSTAR trigger according to PXIe spec.
PXIE_100M	Input. PXIe 100MHz clock. This is 100MHz clock according to PXIe spec.
PXIE_SYNC100	Input. PXIe Sync100. This is Sync100 signal according to PXIe spec.
I/O	
FlexIO[1..160]	Bidir. The physical IO Channels including 4 global clock inputs (2 differential pairs).

External Flash	
Fsm_a[1..23]	Output. Address bus shared by external SRAM and flash.
Fsd[0..31]	Bidir. Data bus shared by external SRAM and flash.
Flash_ce_n	Output. Flash chip enable.
Flash_oe_n	Output. Flash output enable.
Flash_we_n	Output. Flash write enable.
Flash_reset_n	Output. Flash chip reset
Flash_byte_n	Output. Flash byte/word select.
Flash_busy_n	Input. Flash busy
External SRAM	
Sram_be_n[0..3]	Output. External SRAM byte enable.
Sram_ce_n	Output. External SRAM chip select.
Sram_oe_n	Output. External SRAM output enable.
Sram_we_n	Output. External SRAM write enable.
RX DMA FIFO I/F	
RX_DMA_DAT[0..31]	Input. Receive DMA data coming from PC host.
RX_DMA_DV	Input. Receive DMA data valid.
RX_DMA_FIFOFULL	Output. Receive DMA FIFO full. This will throttle data from PC host.
RX_DMA_SP1	Output. Spare. Currently not used.
RX_DMA_SP2	Output. Spare. Currently not used.
TX DMA FIFO I/F	
TX_DMA_DAT[0..31]	Output. Transmit DMA data from memory going to PC host.
TX_DMA_DV	Output. Transmit DMA data valid.
TX_DMA_FIFOEMPTY	Output. Transmit DMA FIFO empty. When empty and is sending data to PC host, the DMA engine in PXI bridge FPGA will assert FIFO read enable TX_DMA_FIFO_RD.
TX_DMA_FIFO_RD	Input. Transmit DMA FIFO read enable.
Misc	
Spare[0..7]	Bidir. Do Not Use. Spares connected to PXI bridge FPGA.
IRQ	Output. Interrupt output pin going to PXI bridge FPGA IRQ = 1 means interrupt will be generated to PC host. IRQ = 0 means no interrupt.
FSpr[0..3]	Bidir. Spare Signals connected to Expansion Board
MClr	Input. FPGA Master Clear, Active High
TP[0..5]	Bidir. Connected to test header J7 on the GX3700 PCB
ACTIVE_LED_N	Output. Active LED. Connect to LD1 LED on board. '0' = LED on, '1' = LED off.

Table 5-1: Pin Assignments Table

Schematic entry project

In order to configure the project as schematic entry and configure the pin assignment the TCL configuration script should be added to the project. To add the script to the project, click on **Project | Add/Remove Files in Project…** In the dialog box, click on the **…** button and browse for GX3700Schem.tcl file in the "C:\Program Files\Marvin Test Solutions\GxFpga\" folder. On some systems, it is recommended to move the desired TCL file to your project's source location prior to adding it to the project. Click Open and then the **Add** button.

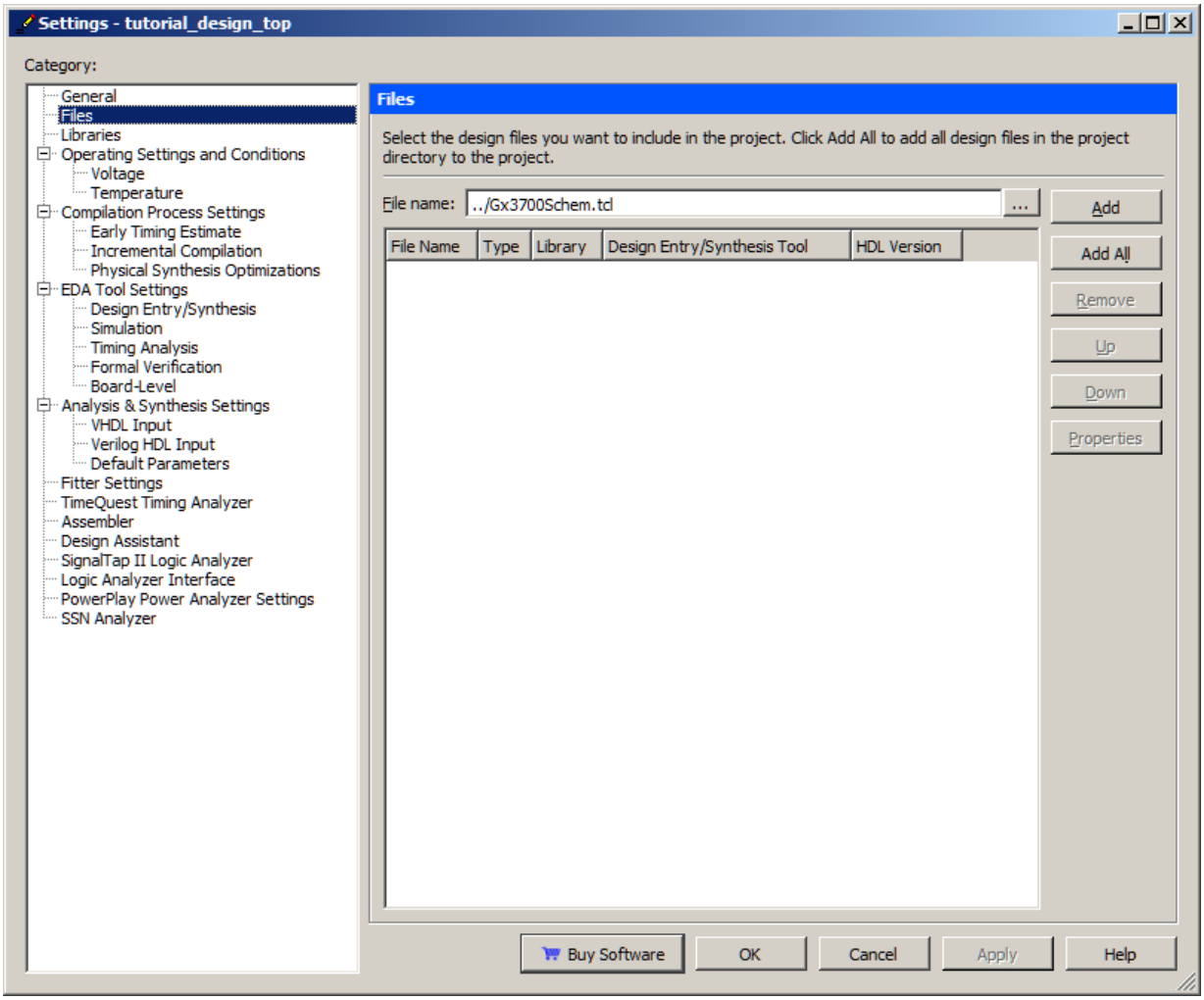


Figure 5-2: Add Tcl Script to Project

Then click on Tools | TCL Scripts … Select the configuration script file, **GX3700Schem.tcl** and click on **Run**. This will configure your FPGA pin assignments.

Note: The TCL file will automatically add all the source files needed for the tutorial design to the Quartus II project.

You can view the pin assignments by running the Pin Planner application which is found in the Tasks list as highlighted below:

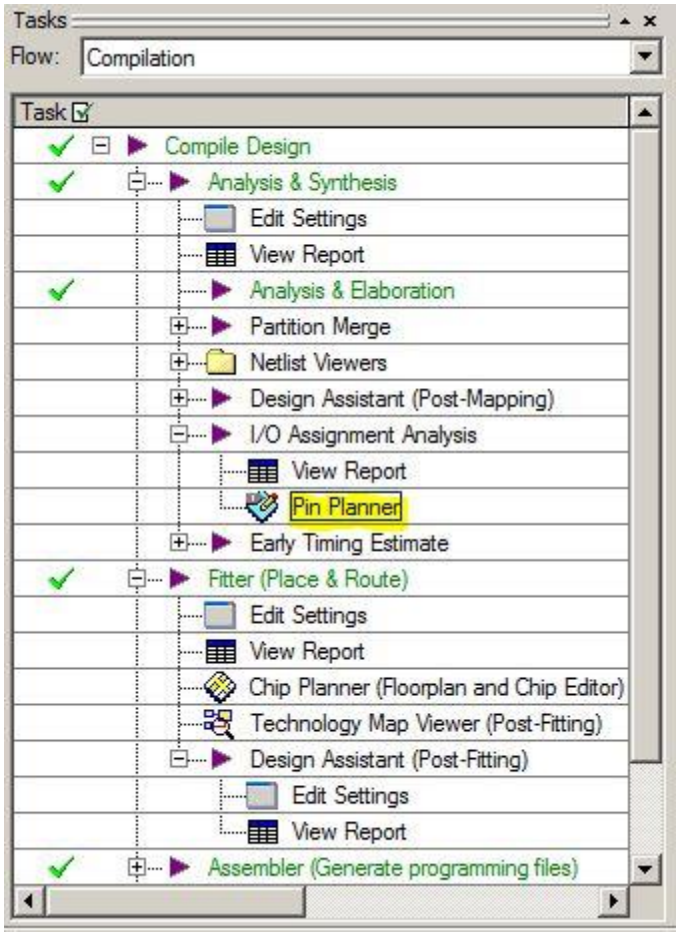


Figure 5-3: Task Flow

The Pin Planner will display a matrix of the physical FPGA pins and their mapped names as well as the I/O standard supported by the pin. These mapped names are used in the FPGA design, as wire names and I/O pins, to connect to the physical connections of the FPGA.

Creating Design File with Schematic Entry

At this point you will have successfully created an FPGA design based on the source codes provided. This section will walk you through the steps of creating your own source file using schematic entry.

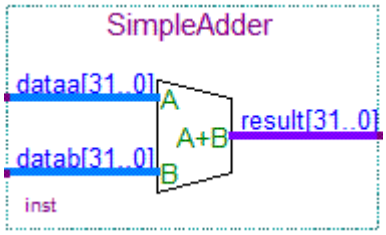
Note: There is more than one way to accomplish the following designs.

Phase 1: Creating the FPGA design - 32 bit Full Adder

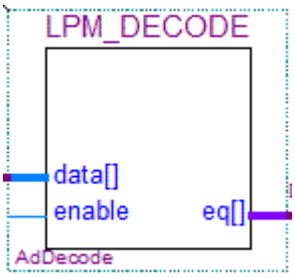
This design will take two double word (32 bit) values, located in the first two double words in the Register space (byte offset 0x0 and 0x4), and add them together. The sum of the two values will be immediately output to the third double word in the Register space (byte offset 0x8).

Components Used

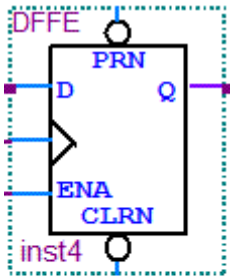
1x 32 bit Full Adder

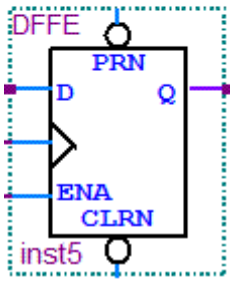


1x Decoder

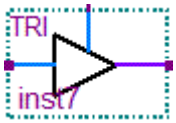


2x D Flip Flops

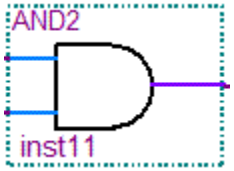




1x Tristate Buffer



2x AND Gate



1x Constant

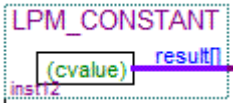


Figure 5-4: Phase 1 Adder Components

Schematic view

In order to open the schematic view, click on **File** menu, and then **New** the following dialog appears.

Select Block Diagram/Schematic File:

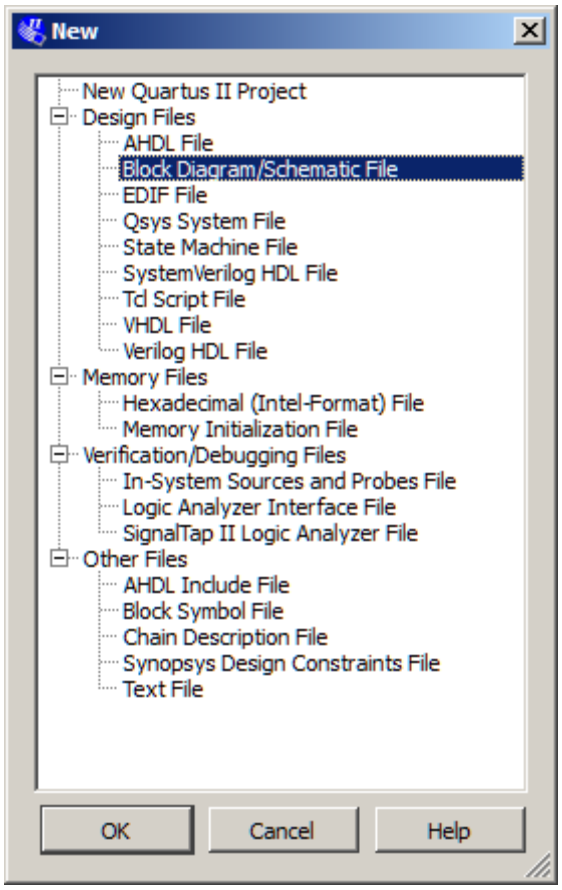


Figure 5-5: Open Schematic view Dialog Box

Design

First start with creating the circuitry required to decode the PCI Address when data is to be written from the PC to the FPGA. This circuit will be used in all three functions of this example project. The signals required for PCI Write access will be the **PCI Clock**, **Write Enable**, **Chip Select 1**, and some **PCI Address lines**. The PCI Address lines 5 to 2 will be fed to a decoder which will generate a 32-bit value, and the result will be ANDed with the Chip Select 1 bit. Each Chip Select bit represents a certain PCI BAR access (GX3700 has two bars, memory and register memories). Bit 1 represents BAR1 of the PCI memory space (bit 2 for BAR2). BAR1 is the general-purpose Control Register BAR for the GX3700. The results of the AND operation will be once again ANDed to the Write Enable PCI signal.

Double click on the blank space in the schematic view and select **lpm_decode** from the **Megafunction, Gates** directory.

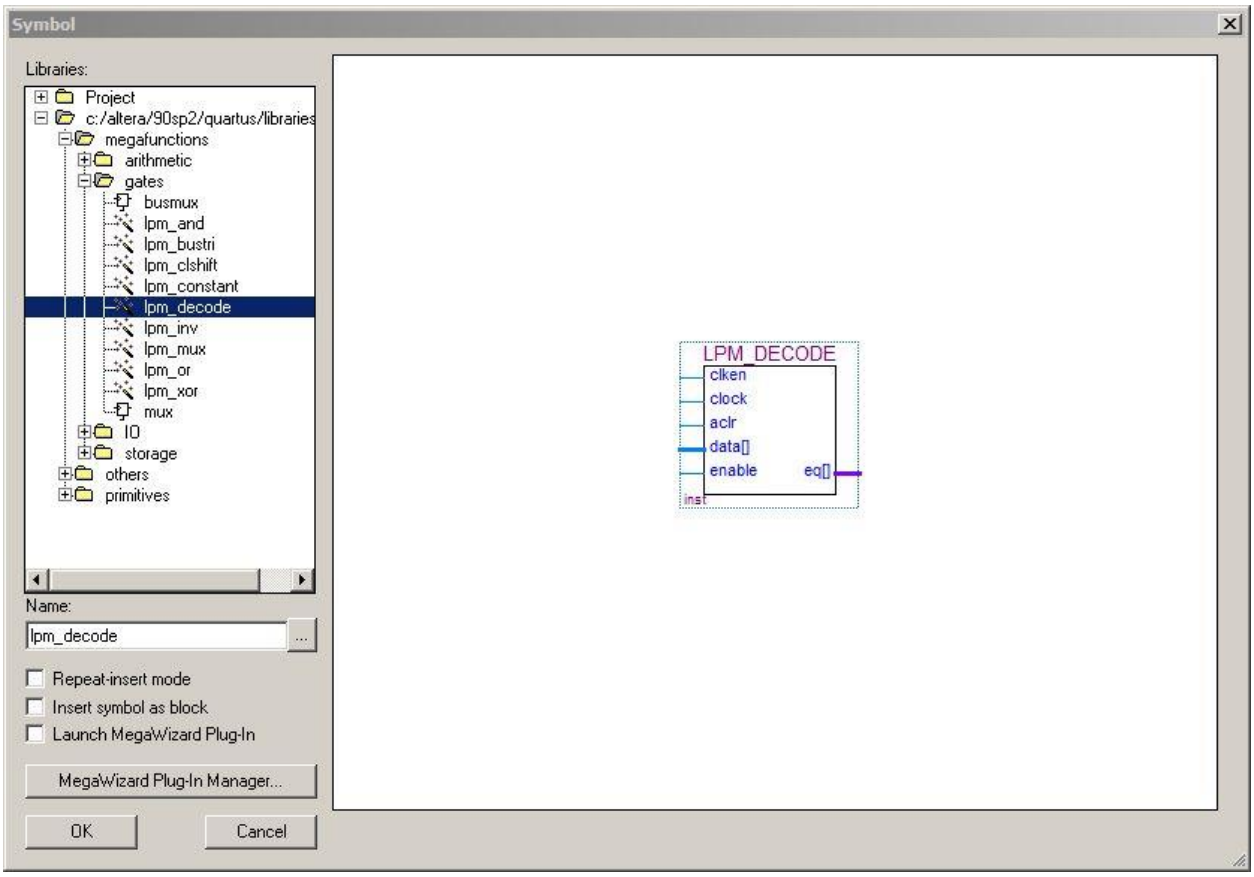


Figure 5-6: Symbol Insert Dialog Box

Make sure the Launch **MegaWizard Plug-In** checkbox is unchecked.

Click **OK** and place the symbol on the blank design document.

Now that the Decoder has been placed, some of its parameters have to be set. Right click on the Decoder symbol and select **Properties**. Click on the **Parameters** tab. Set the **Width** and **Decodes** properties as shown below:

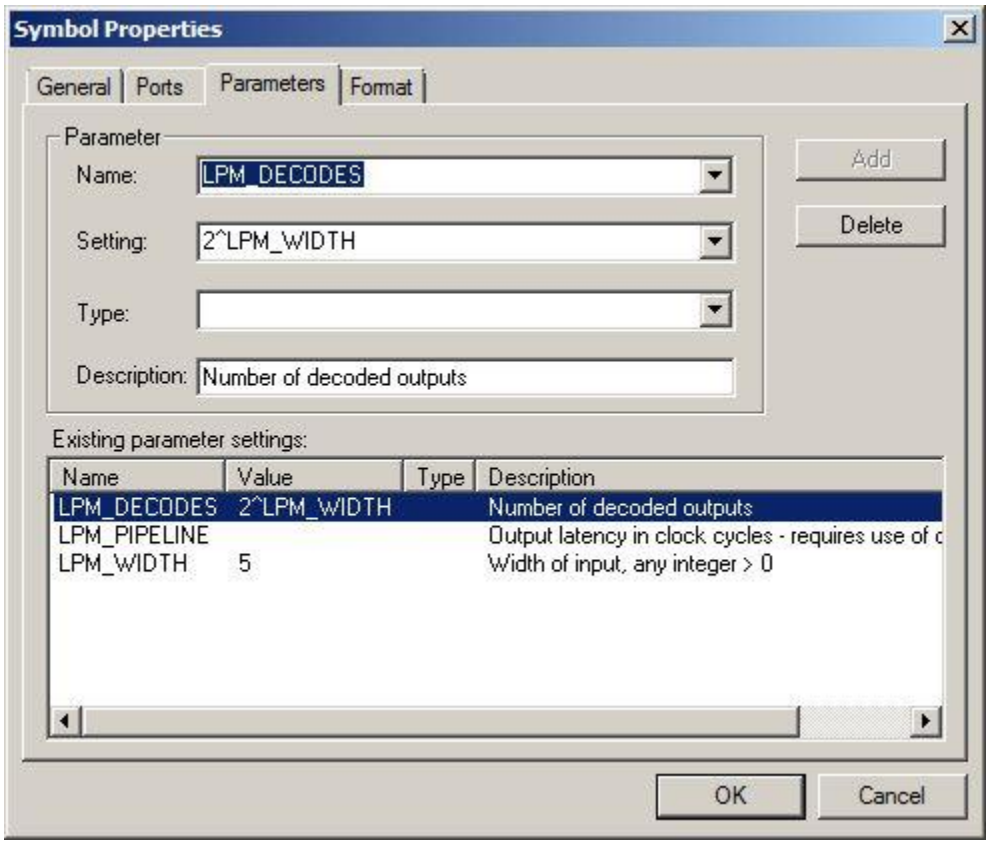


Figure 5-7: Decoder Properties

Click **OK** when done. Place another symbol on the design by double clicking on the design document, and selecting **Input Pin** from **Primitives**, **Pin**, **Input**. After placing the input pin symbol, rename it to **Addr[6..2]**. The symbol will now represent 5 PCI address lines that will be used to communicate with the PC.

Also place 2 AND gates after the Decoder and a few more input pins with the appropriate names **DecAddr, Sel and WE** as the following figure shows:

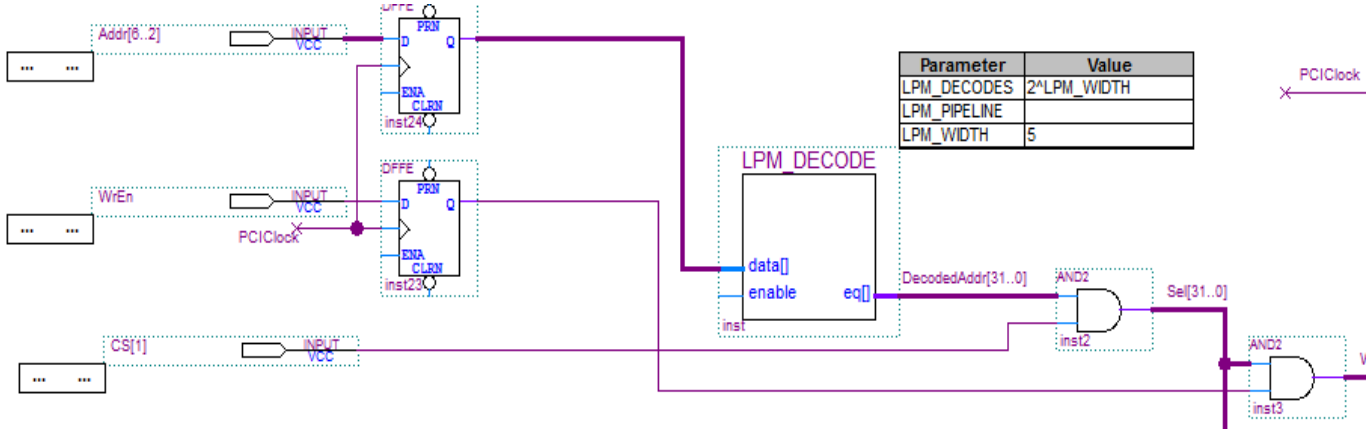


Figure 5-8: PCI Address Decoder Circuit

Note: To wire several signals together (as a bus), such as Addr[6..2] or Sel[31..0], use the **Bus Wiring Tool** highlighted in red below. We use two D-Flip-Flops to clean up the signal going into our design before we use it.

Figure 5-9: Bus Wiring Tool

Now that the PCI address decoder circuit is complete, we can feed the appropriate bits from the WE bus to D Flip Flops that will store data clocked in from the PCI data lines. For example, the first double word in PCI memory (representing the first number to be summed) will be written to a D Flip Flop with its enable line tied to WE[0] (the first bit in the WE bus). The second double word to be added will be written to another D Flip Flop with its enable line tied to WE[1]. Finally, the PCI Clock signal (33Mhz) will be used as the clock source of the D Flip Flops. Note that each bit of the Sel and WE busses represent a consecutive double word address (bit 0 corresponds with byte 0, bit 1 corresponds with byte 4, bit 2 corresponds with byte 8 etc.)

Place two D Flips Flops (located at **primitives**, **storage**, **dffe**) and an input pin named **PCIClock**. We will leave the D Flip Flops input lines (D) disconnected for now. Eventually the PCI data lines will drive these inputs.

Wire the output of the AND gate to D Flips Flops as shown below.

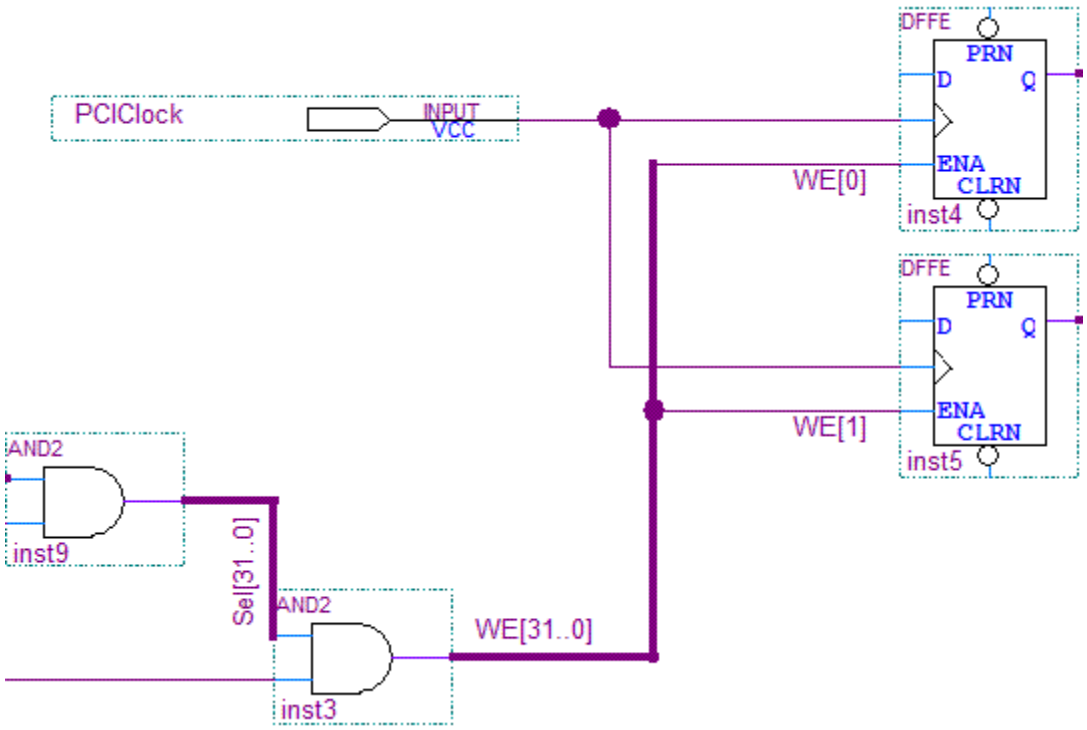


Figure 5-10: D Flip Flops

The D Flips Flops will feed a 32-bit adder and the resulting summation will be wired to the PCI data lines so that the PC can read the result.

The 32-bit adder will be placed onto the design using the MegaFunction wizard tool. This tool will customize a component by allowing you to make selections through a wizard.

Double click on the design window and navigate to **megafunctions, arithmetic, lpm_add_sub**. Make sure the **Launch Megafunctions Wizard** checkbox is selected and click **OK**. You will see a dialog box like the following:

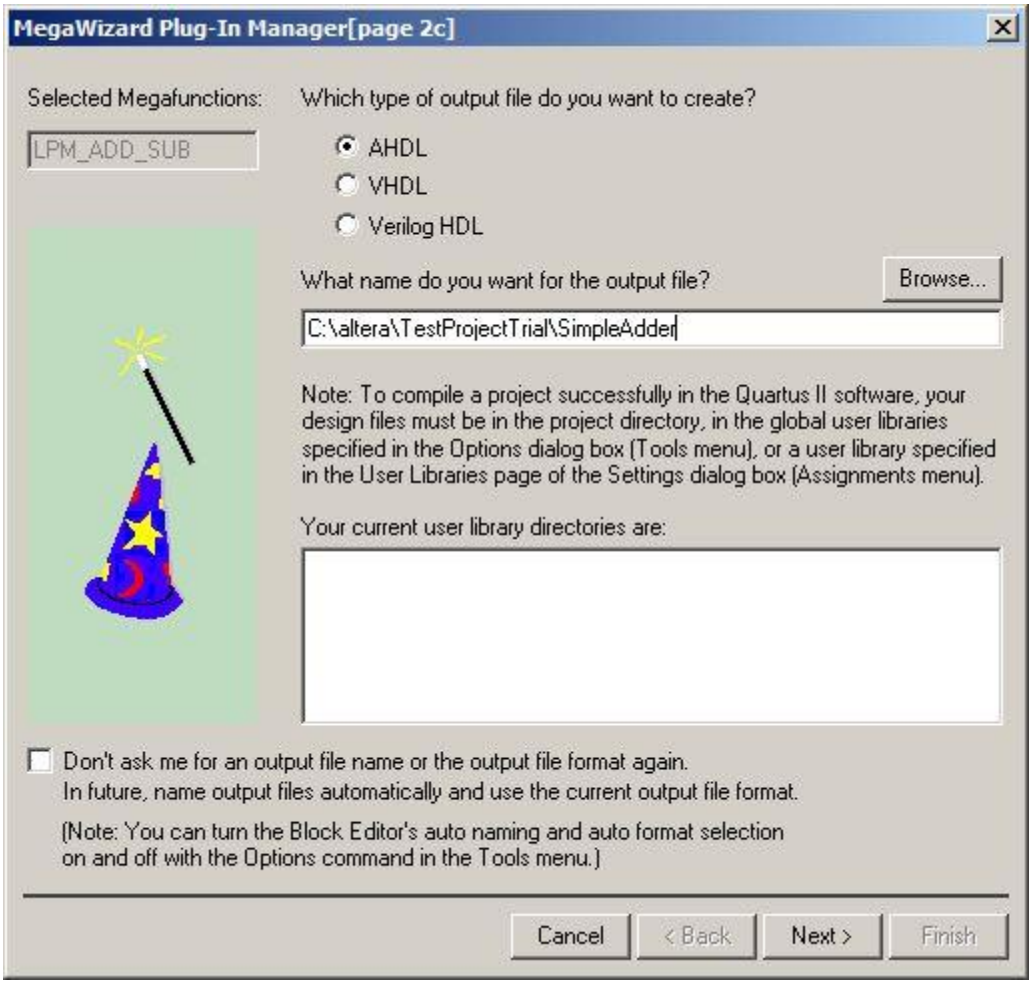


Figure 5-11: Adder Wizard

Name the output file **SimpleAdder** and make sure the path is the same as your project. Click **Next** and enter **32** as the data width.

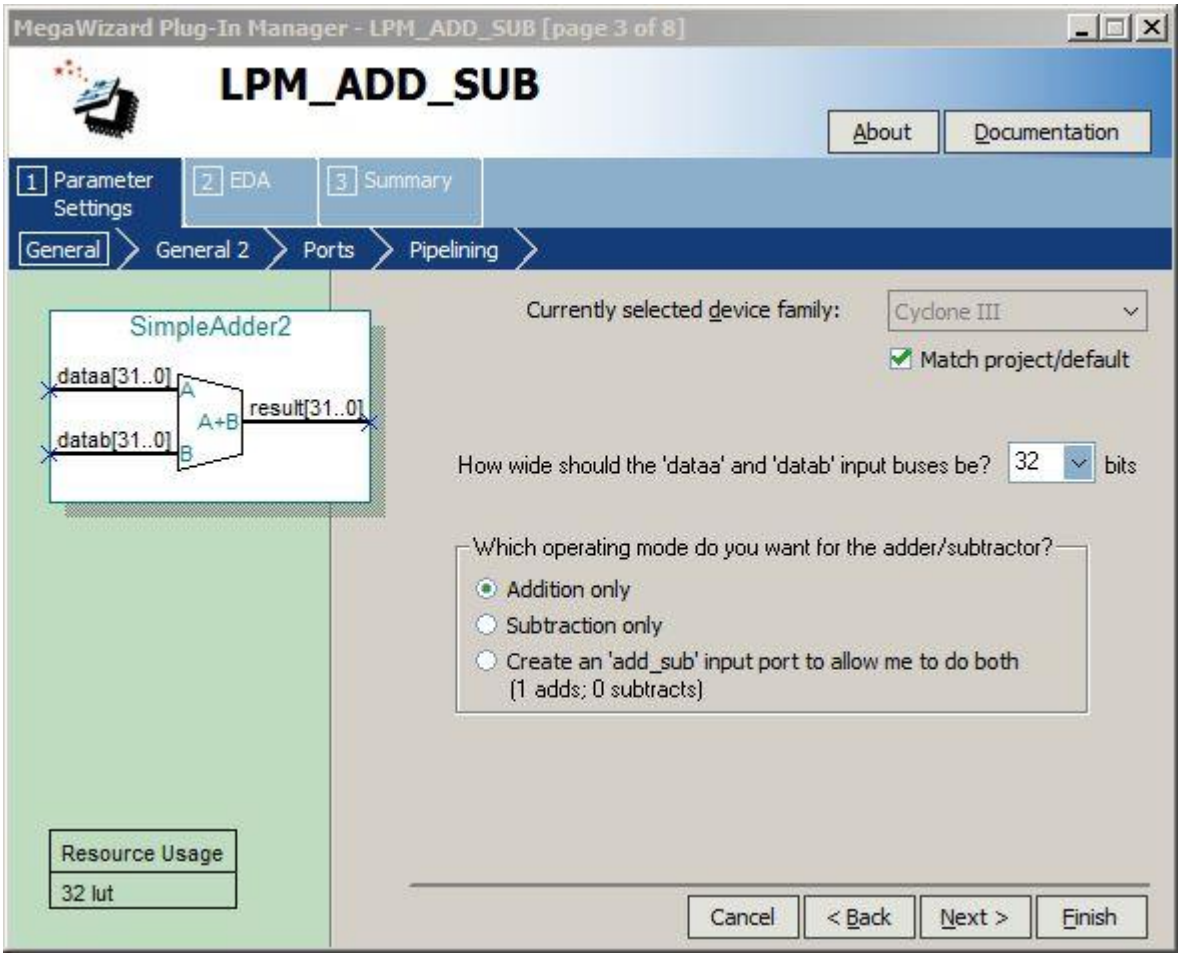


Figure 5-12: Adder Wizard 2

Click **Next** through the rest of the wizard and keep the default choices. Finally, the dialog box will show the newly created design files that will be included in your project. Click **Finish** and place the newly created Adder in your design.

Wire the adder to the flip flops and add an AND gate, Read Enable pin, and tristate buffer as the following shows:

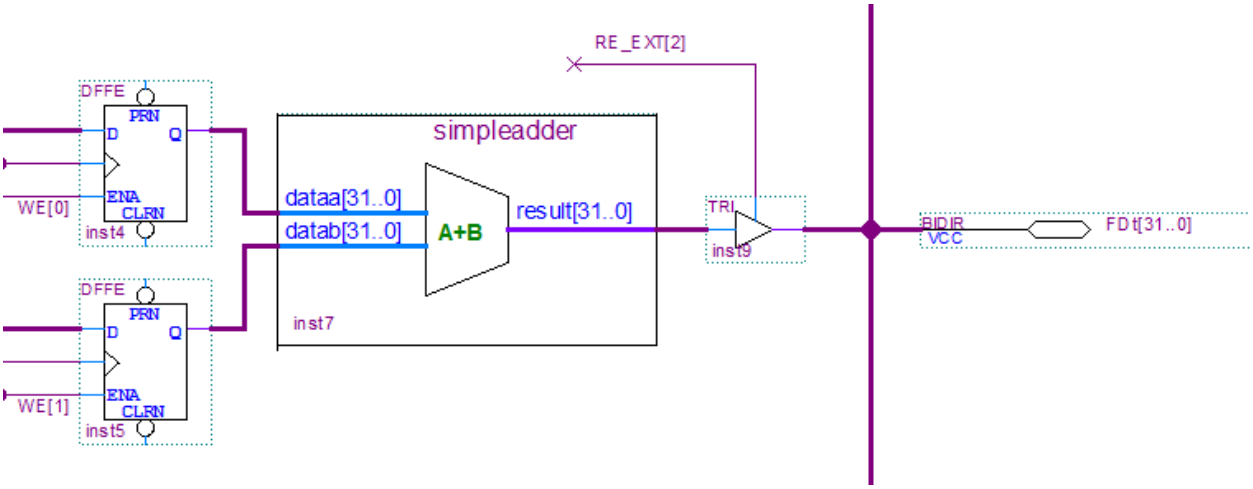


Figure 5-13: Adder Circuit

Note that we are using the FDt[31..0] PCI data lines as bidirectional pins since we will be reading and writing to the PCI bus. The Tristate buffer is used to select whether the Adder will be driving the PCI Data lines or not. The Tristate buffer is controlled by the 3rd bit of the decoded PCI Address ANDed with the Read Enable line. When both signals are high (Sel[2] and RdEn) it indicates that the PCI Bus is expecting the 3rd double word to be written to the PCI bus. In our case, this means the 32-bit result from the Adder.

Before moving on we must first extend the RdEn to 2 PCI clock cycles by adding a small circuit as demonstrated below:

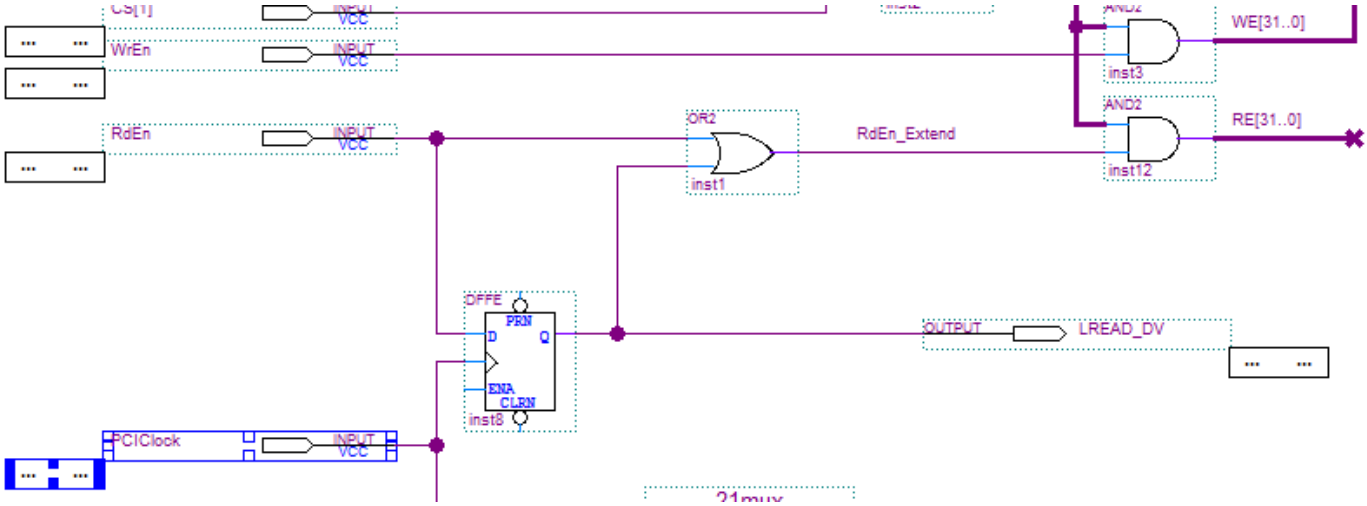


Figure 5-14: RdEn to 2 PCI Circuit

We also create a Read Data Valid output pin, **LREAD_DV**. This comes from a D-Flipflop with the PCIClock as an input clock and the RdEn as the input data. The D-Flipflop also creates our extender for our ReadEnable.

Since this design is created to be able to be implemented in both the 3700 and the 3700e, we need to extend our write enable pins, WE[31..0], and read enable pins, RE[31..0], for 3 more clock cycles. Below is the circuit to do that.

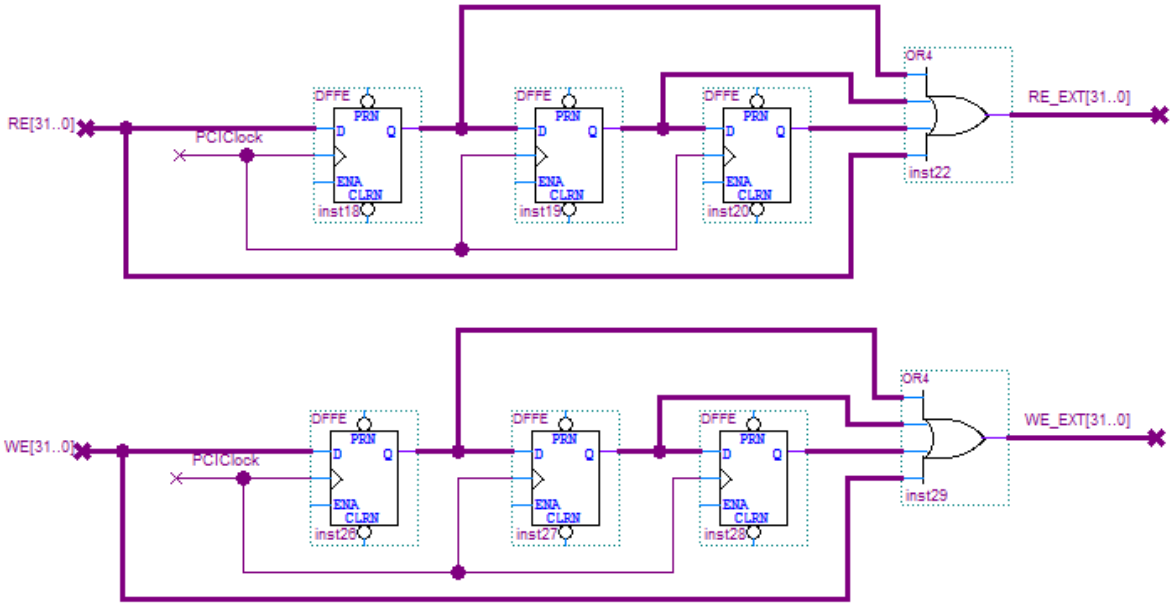


Figure 5-15: RE[31..0] and WE[31..0] extend Circuit

The inputs to the D Flips Flops can now be wired to the PCI data lines (FDt). We need to clean up the FDt signal as is comes back into our circuit by adding the D-FlipFlop.

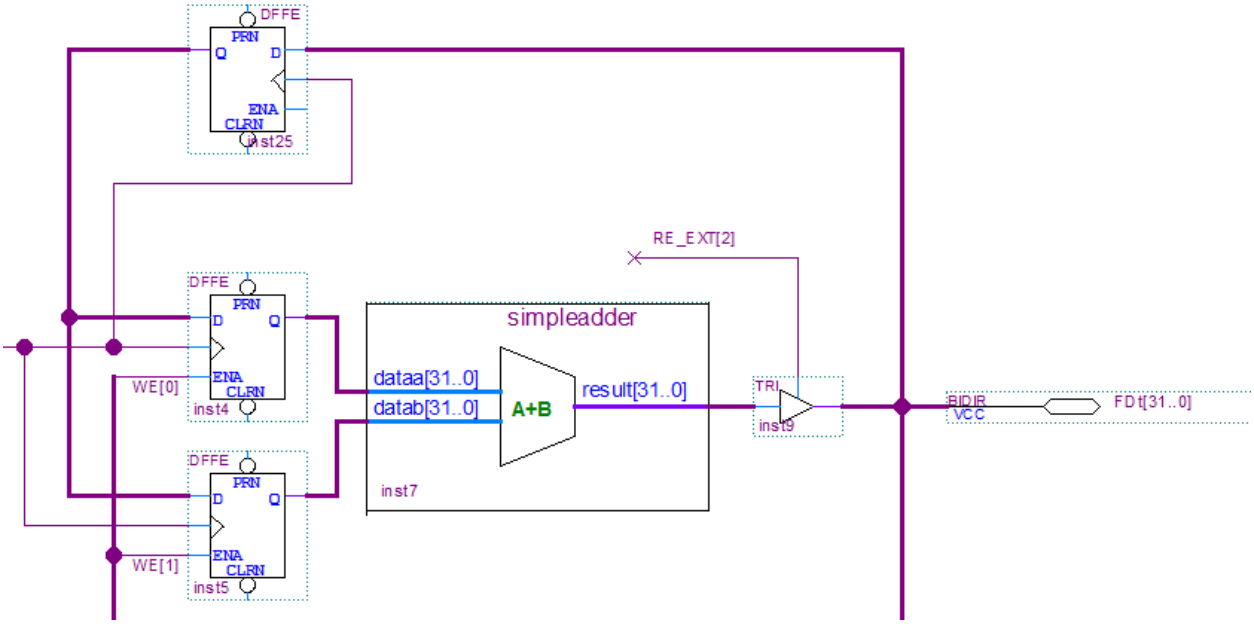


Figure 5-16: Adder Circuit with PCI Bus Connection

Now that the design has been completed, a revision number should be added so that the end user can read it back from the PCI bus at the 32nd register double word location (byte address 0x7C).

Including a revision number constant to the design is a Marvin Test Solutions standard practice that we recommend end users to follow. The revision constant is 32 bits long and is read as a hexadecimal number such as 0x3564A000. The first two digits of the hexadecimal number represent the company, in this case 35 is for Marvin Test Solutions designs. The next two digits are the design specific code, 64 in this case. And the last 4 digits, A000, is the revision of the design.

A constant component needs to be placed in the design (LPM_CONSTANT). When placing this component make sure that the “Launch MegaWizard Plug-In” selection is unchecked. After placing the component, right click on it and select properties to set the value and width of the constant as the following figures show:

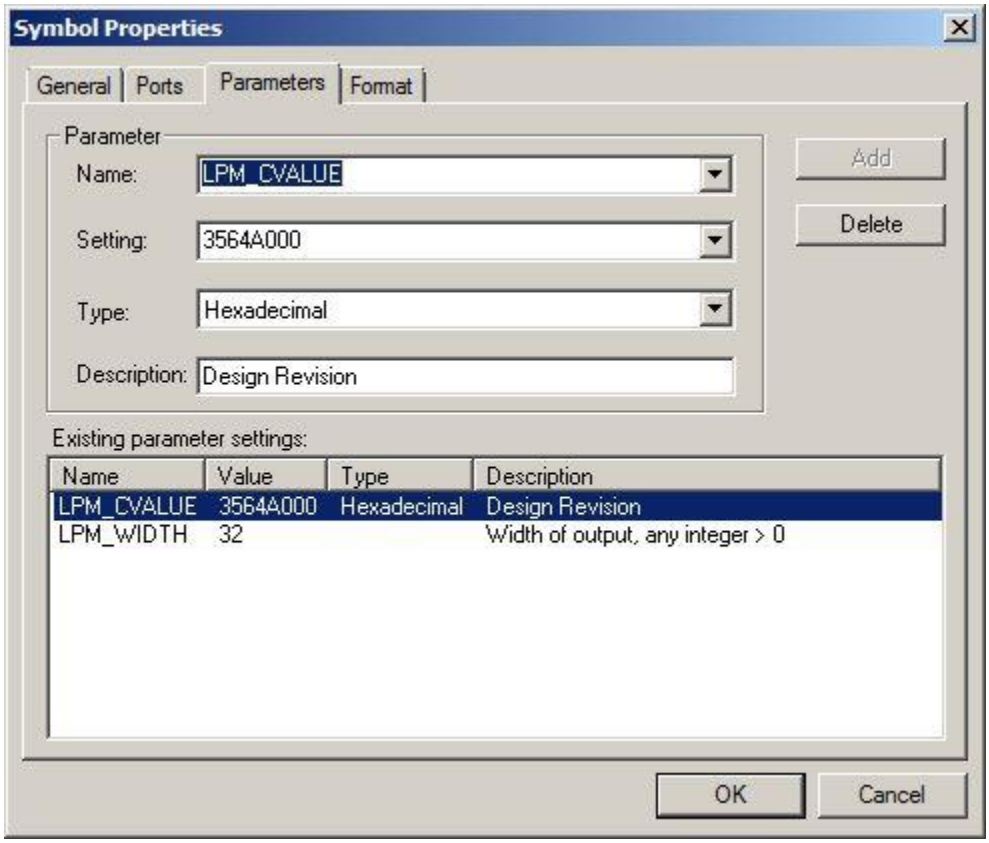


Figure 5-17: Symbol Properties

Now place the 2 port AND gate and the tri-state buffer. You can rotate it, as shown in Figure 5-17: Symbol Properties, by right clicking on the symbol (after placing it) and select “Rotate By Degrees | 90”.

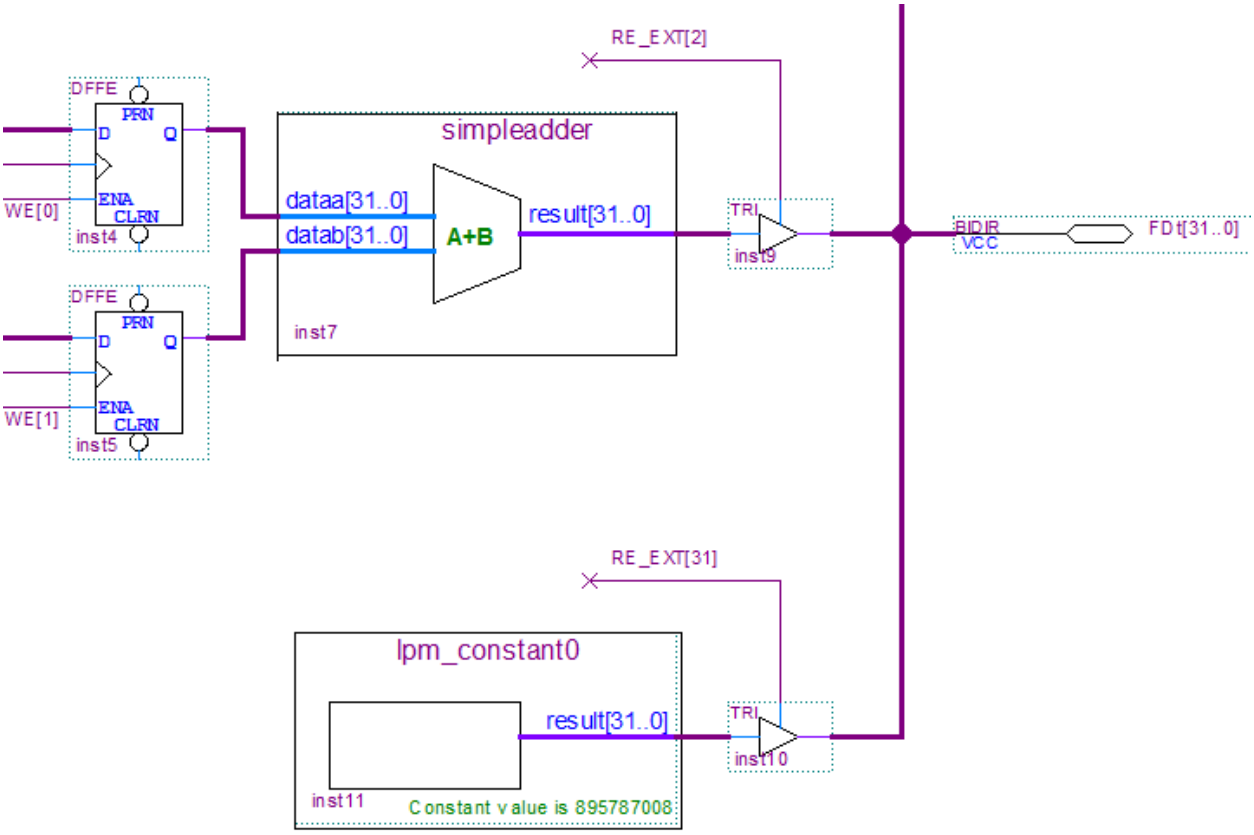


Figure 5-18: Adder Circuit with Revision Constant

Finally, you can change the inputs to the adder from write-only to read/write by connecting the output of the D-Flipflops to the FDt inout pin via a tristate buffer. After adding this buffer, the complete adder circuit should appear as below:

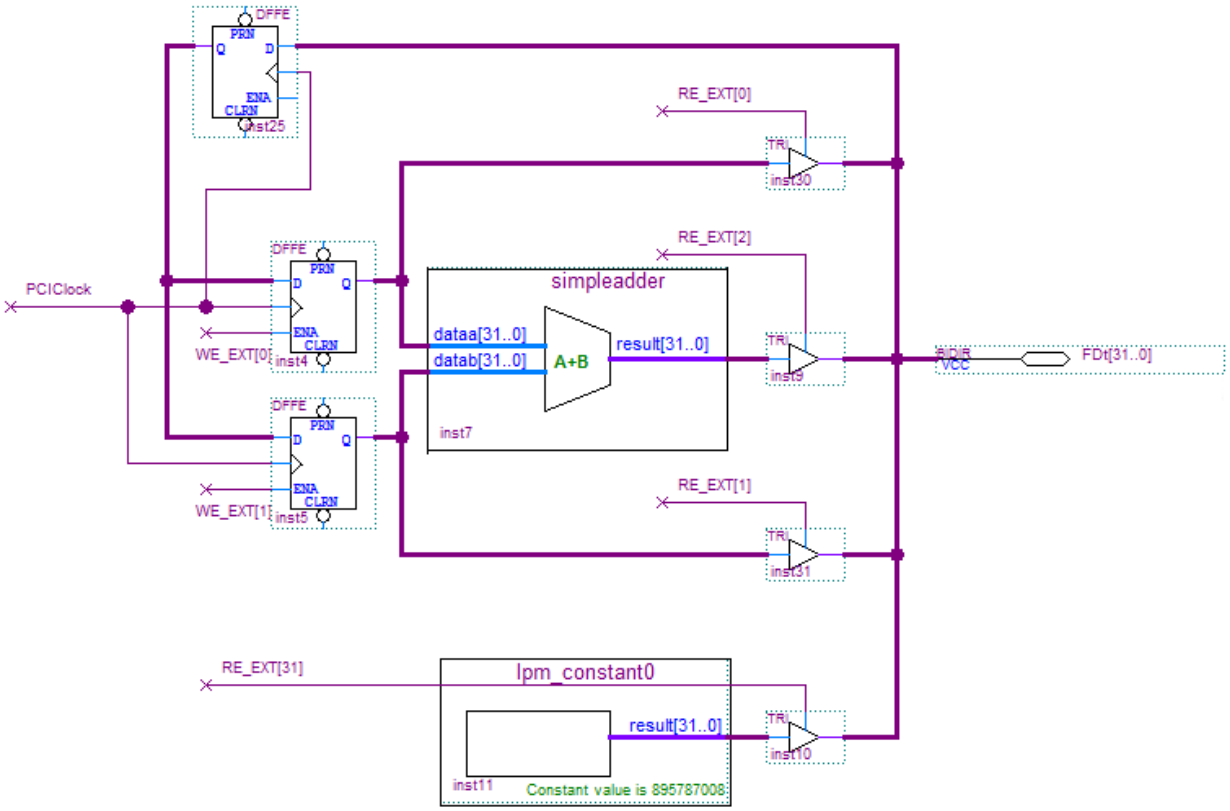


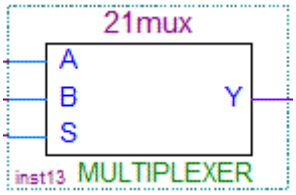
Figure 5-19: Completed Adder Circuit

Phase 2: Creating the FPGA Design - 2 to 1 Clock Mux

This design will output either the PCI Clock (33Mhz) or the 10Mhz clock to Flex I/O Channel 65 (check the connector tables to find the pin number) depending on what was written to the 4th double word in the PCI register space (byte offset 0xC). A 1 will select the 10Mhz clock signal, and a 0 will select the PCI clock signal.

Components Used

1x 2 to 1 Mux



1x D Flip Flops

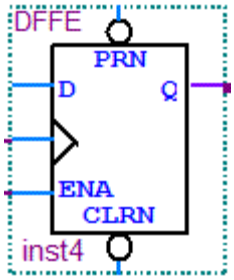


Figure 5-20: Phase 2 Mux Components

Design

You will now build upon the tutorial project to add the functionality of a 2 to 1 Clock Mux. The 10Mhz clock will be brought into the design by an input pin. The PCI Clock signal input pin is already present in the Phase 1 circuit, so this will be reused. FlexIO[65] (IO Channel 65) will be used to output the selected clock to the outside world.

Place the 2 to 1 Mux symbol by double clicking on the design area and selecting megafuncitons **others, maxplus2, mux21.**

Create three wires attached to the D, ENA(enable) and B inputs of the D Flip Flop. Name the wires **FDt[0], Sel[3]**, and **PCIClock** respectively. Note that you did not have to place new input pins to access these signals. This is due to the fact that input pins were already created for these signals in the Phase 1 design. Therefore, you can just use named wires to tap into the same input pins.

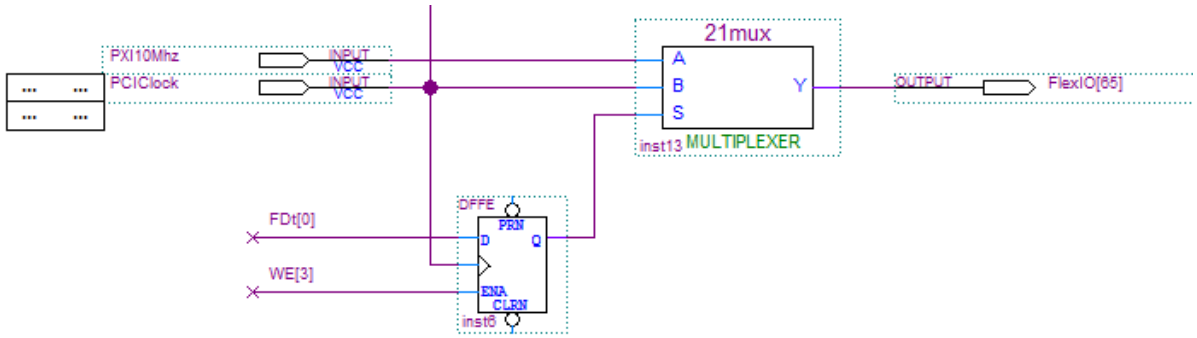


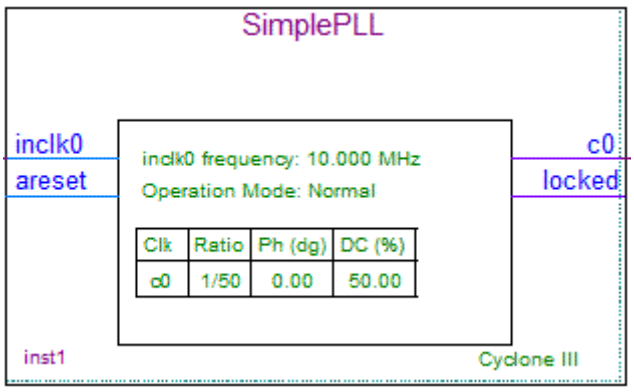
Figure 5-21: Clock Mux Circuit

FDt[0] is the first bit of the PCI data bus. This bit can either be 0 or 1, to indicate which clock source to choose. Sel[3] is the 4th bit from the decoded PCI Address. When this bit is high, it indicates that the PCI Bus is addressing the 4th double word (byte offset 0xC) of the Register space for the GX3700. In our case, the value of this double word is used to select which clock is selected by our Mux.

Phase 3: Creating the FPGA Design - 32 bit Dynamic Digital Pattern Sequencer

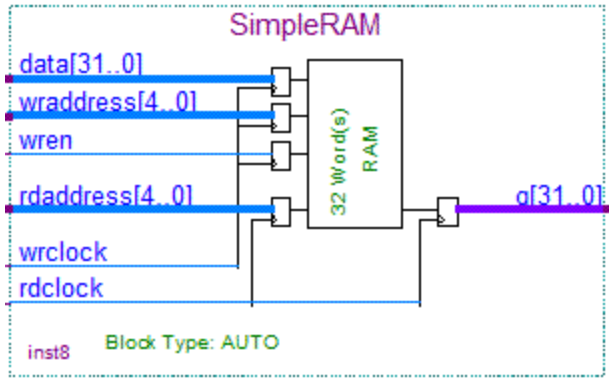
Components Used

1x PLL



1x 5 bit Counter

1x 32 by 32 bit RAM



1x AND gate

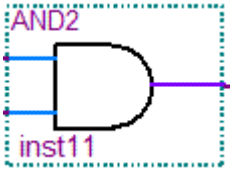


Figure 5-22: Phase 3 Dynamic Digital Sequencer Components

Design

This design functions as a simple dynamic digital pattern generator. A PLL drives a Counter which iterates through a 32-double word memory that outputs 32 bit wide digital patterns to the I/O Pins. The memory is loaded through the PCI bus, allowing users to program the device with vectors through the software front panel or the DLL API.

This phase will require the use of the **MegaFunction Wizard** to generate all three components, **PLL**, **RAM**, and **counter**. The wizard will allow you to customize the component for this particular application. The generated component will be stored in a file (.qip) that will automatically be included in the project.

First insert the PLL component by double clicking on an empty space in the design and clicking on **MegaFunction Plug-In Manager**. Choose to create a new **MegaFunction** variation and click **Next**. Then select the symbol called **ALTPLL** under the **I/O folder**. Name the new variation **SimplePLL** and click **Next**. The next dialog box will prompt you for the input clock frequency. We will be using a 10Mhz reference clock source so enter **10Mhz** into this field.

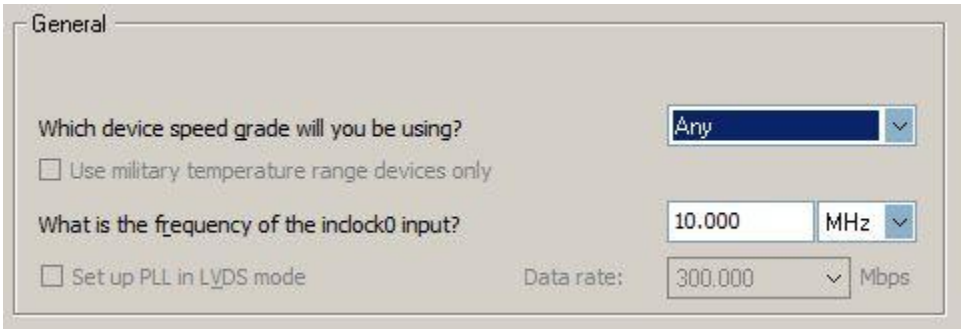


Figure 5-23: PLL Wizard Dialog Box 1

Proceed through the next few screens, with the default choices until you get to step 3 in the wizard entitled **Output Clocks**. Select **50** as the division factor as shown in the following figure:

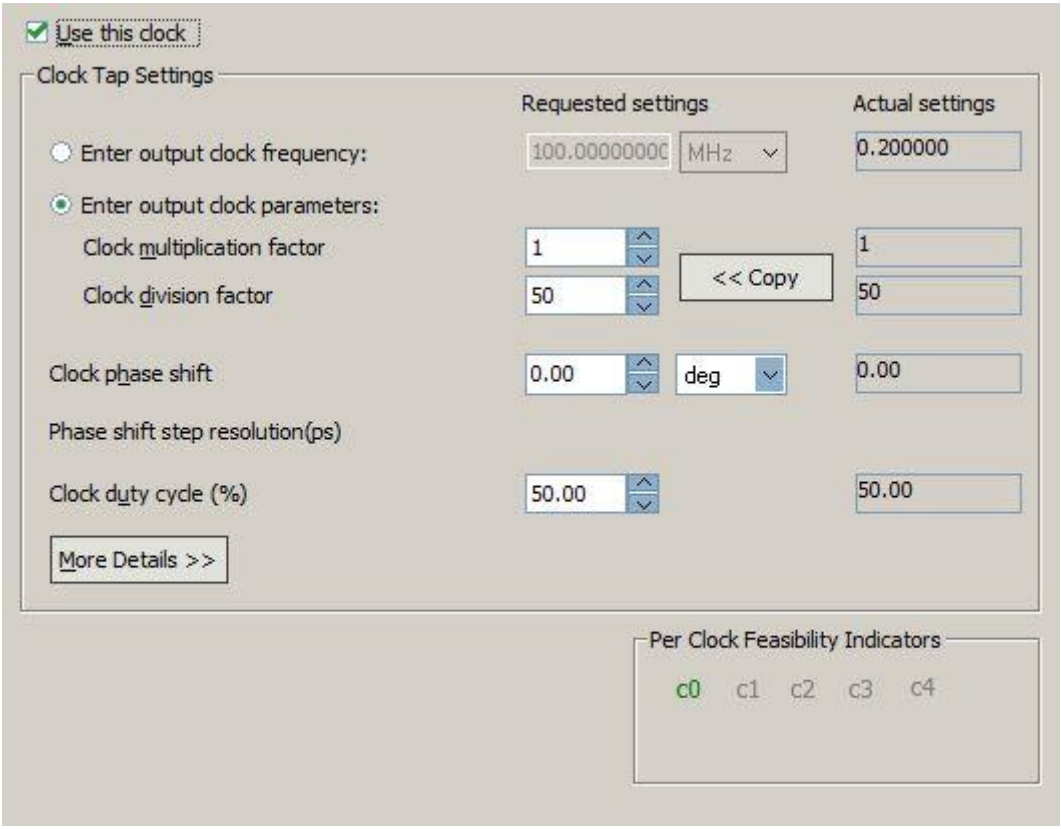


Figure 5-24: PLL Wizard Dialog Box 2

Click **Next** for the rest of the windows until you get to the last window showing you the files that will be created and then click **Finish**. The customized component will now be included in your project automatically so that you can start using it. Click **OK** to return to the design view, and then place the newly created symbol on your design.

Attach a wire to the inclk0 terminal of the PLL symbol, and name the wire **10Mhz**. This will connect the wire to the 10Mhz input pin that has already been created in the phase 2 design.

Repeat the previous steps to create a new custom component using the **MegaFunction Wizard** and select **LPM_COUNTE**R from the arithmetic folder. Name the custom component **SimpleCounter** and click next. Select 5 bits for the output bus width. We have chosen 5 bits for the width because we need to count from 0 to31 which requires 5 bits. You can now click next for the rest of the windows and finally click finish to place the symbol on your design.

Wire the c0 output terminal from the PLL to the clock input on the counter.

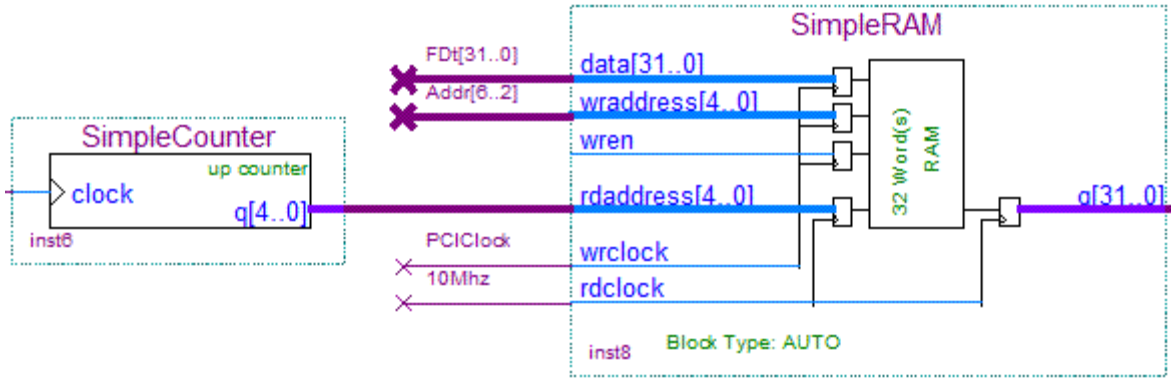


Figure 5-25: PLL and Counter Circuit

The last component needed is a 32 double word RAM. You will need to deploy the **MegaFunction Wizard** once again, and select the **2 port RAM** component from the **Memory Compiler** folder. Call the new component file **SimpleRAM** and click **Next**. Make sure to select 32 as the word length and 32 as the input width as the following figure shows:

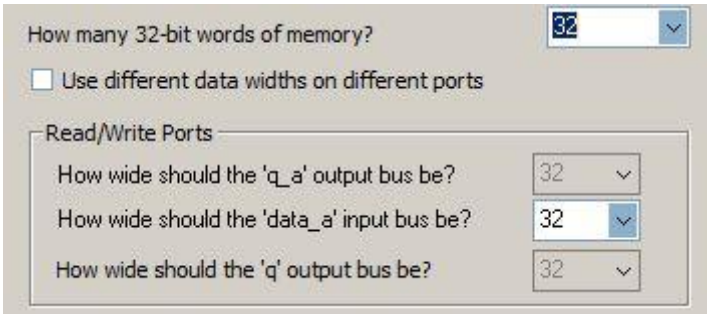


Figure 5-26: RAM Wizard Dialog Box 1

In the next window make sure to select a dual clock for reading and writing so that data can be written to the RAM from the PCI bus and read out to the IO pins concurrently.

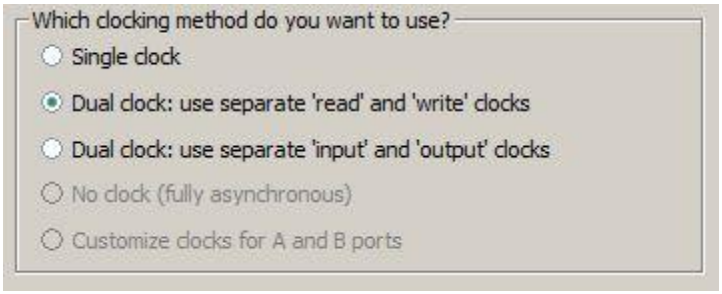


Figure 5-27: RAM Wizard Dialog Box 2

Click **Next** on the rest of the windows and click **Finish** placing the RAM component on your design. Wire the output bus, q[4..0], from the counter to the read address, rdaddress[31..0], of the RAM component.

Connect a bus to data[31..0] and wraddress[4..0]. Name these busses **FDt[31..0]** and **Addr[6..2]** respectively. Then connect wires to wrclock and rdclock and name the wires PCIClock, and 10Mhz respectively.

You will need to place an AND gate next to the RAM component and wire a new input pin called CS[2] and a wire named WrEn to it. The output of the AND gate should be connected to the wren input of the RAM. This AND logic ensures that only BAR2 PCI accesses are able to write to the RAM. This will allow us to use the FGPA Memory

space to write out digital patterns to the sequencer instead of the FPGA Register space (which is being used for control). Note that when CS[2] is high, it signifies an access from BAR2.

Finally create a bus connected to the q[31..0] output from the RAM and name it **FlexIO[64..33]**. This connects the RAM output to the 32 physical IO pins.

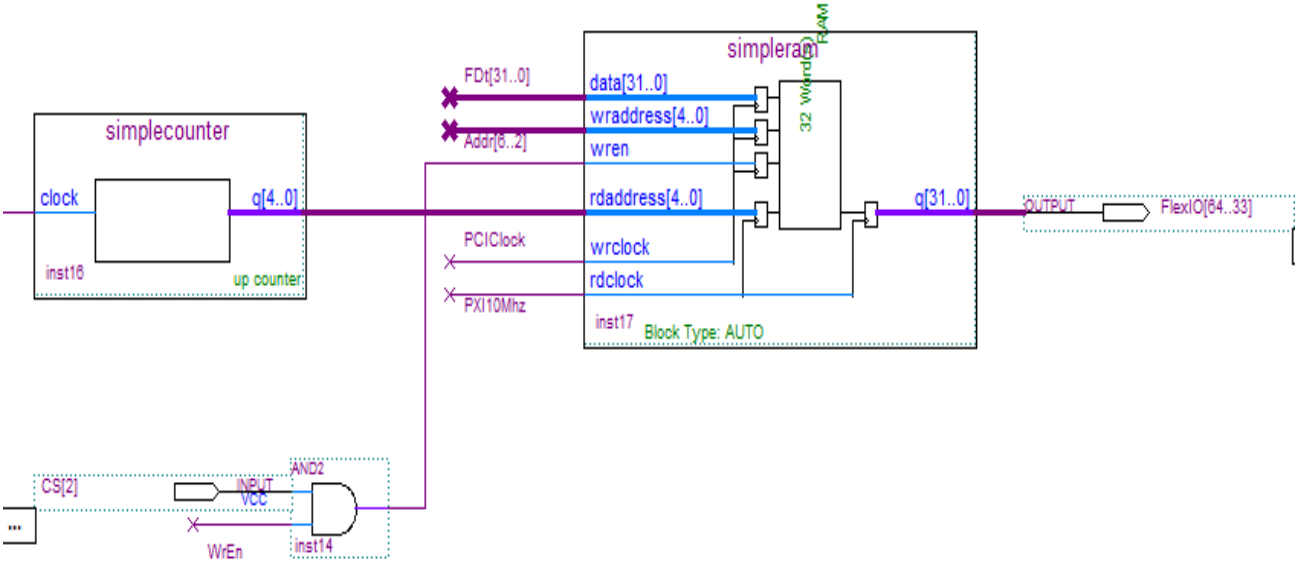


Figure 5-28: Dynamic Digital Sequencer Circuit

At this point the design is complete, continue with the next sections to generate SVF or RPD files and load your design to the GX3700.

Configure Project to Output SVF and RPD Files

To ensure that a SVF file is generated upon project compilation, go to the **Assignments**, **Device ...** and click on the **Device and Pin Options** button. Then click on the **Programming Files** and verify that the **Serial Vector Format File** checkbox has been selected.

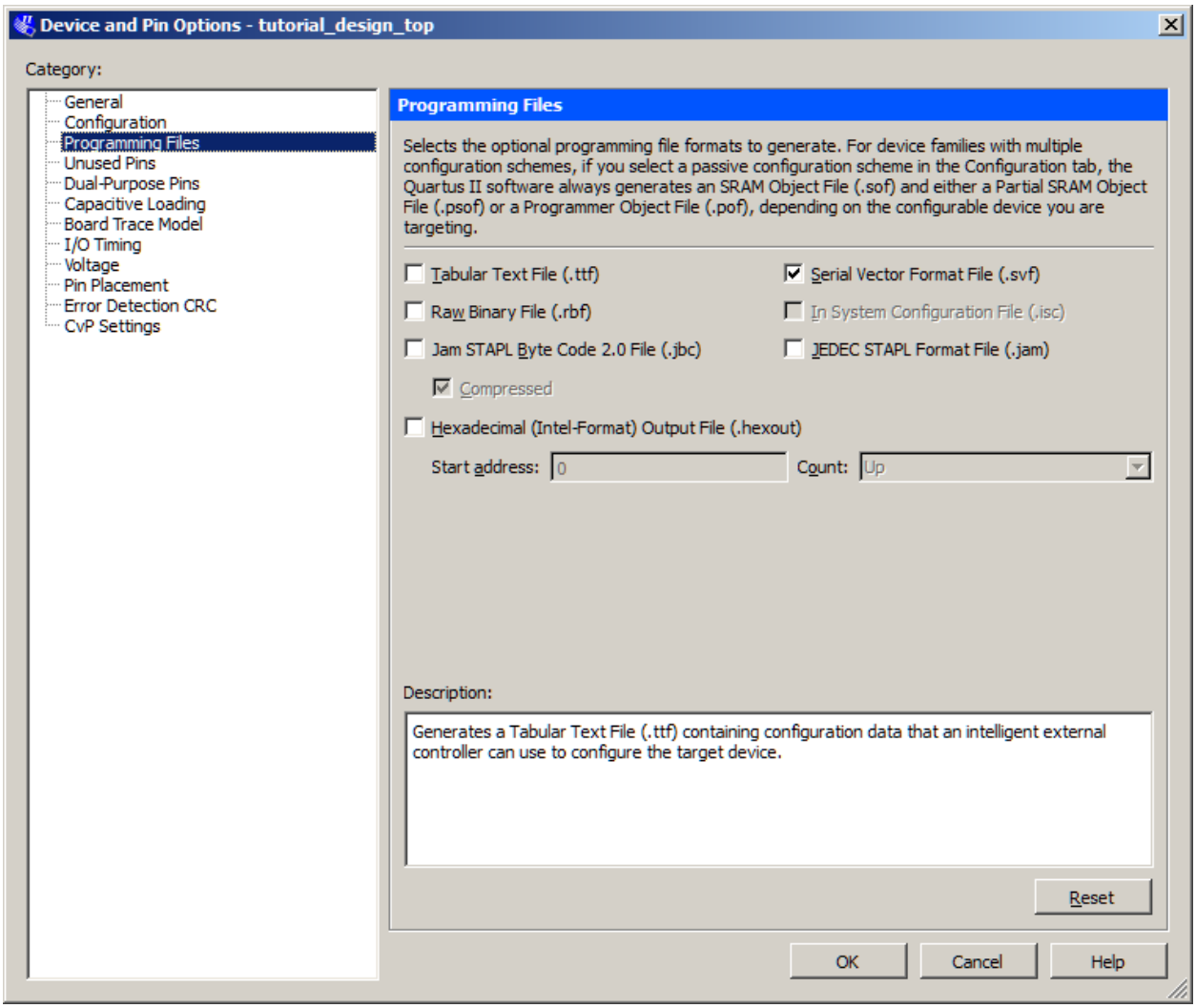


Figure 5-29: Select SVF as output file

Now click on the **Configuration** choose **Active Serial Configuration Scheme**, check **Use Configuration Device** checkbox and select **EPCS64** as the configuration device from the drop down selection. Finally click on **OK** twice to exit the settings dialog boxes.

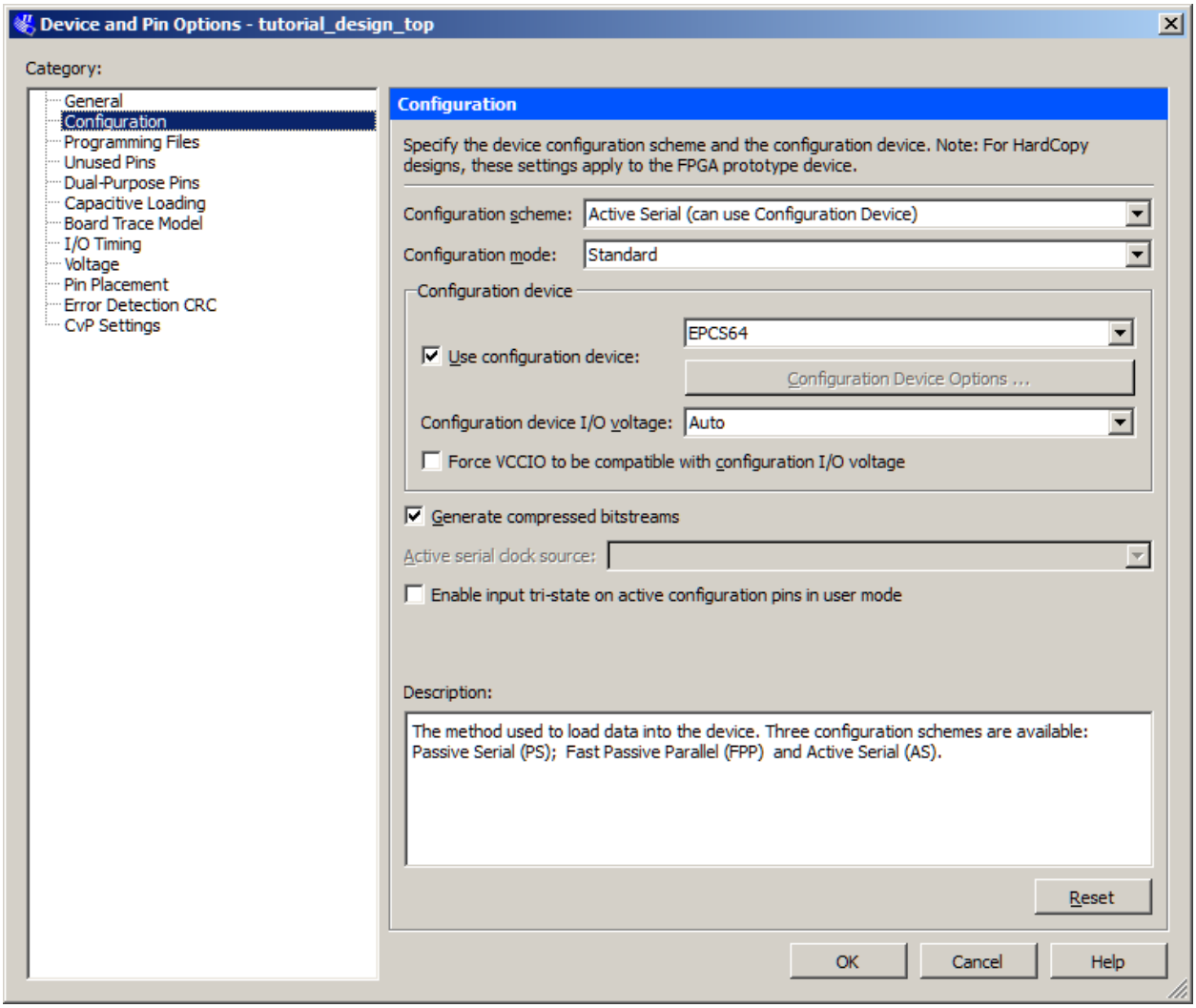


Figure 5-30: Select Configuration Device

Compile an Example Project and Build RPD and SVF Files

Click on **Processing** menu tab and choose **Start Compilation** to start the compilation process for the example project. After the process has ran successfully, you should now see in Quartus II something similar to the figure below. The green check marks indicate success and the red X indicates failure. The process will succeed only when there is no error. There may or may not be any warning. If there is any warning, make sure that it is OK for the design before moving forward. For this tutorial design, ignore all warnings.

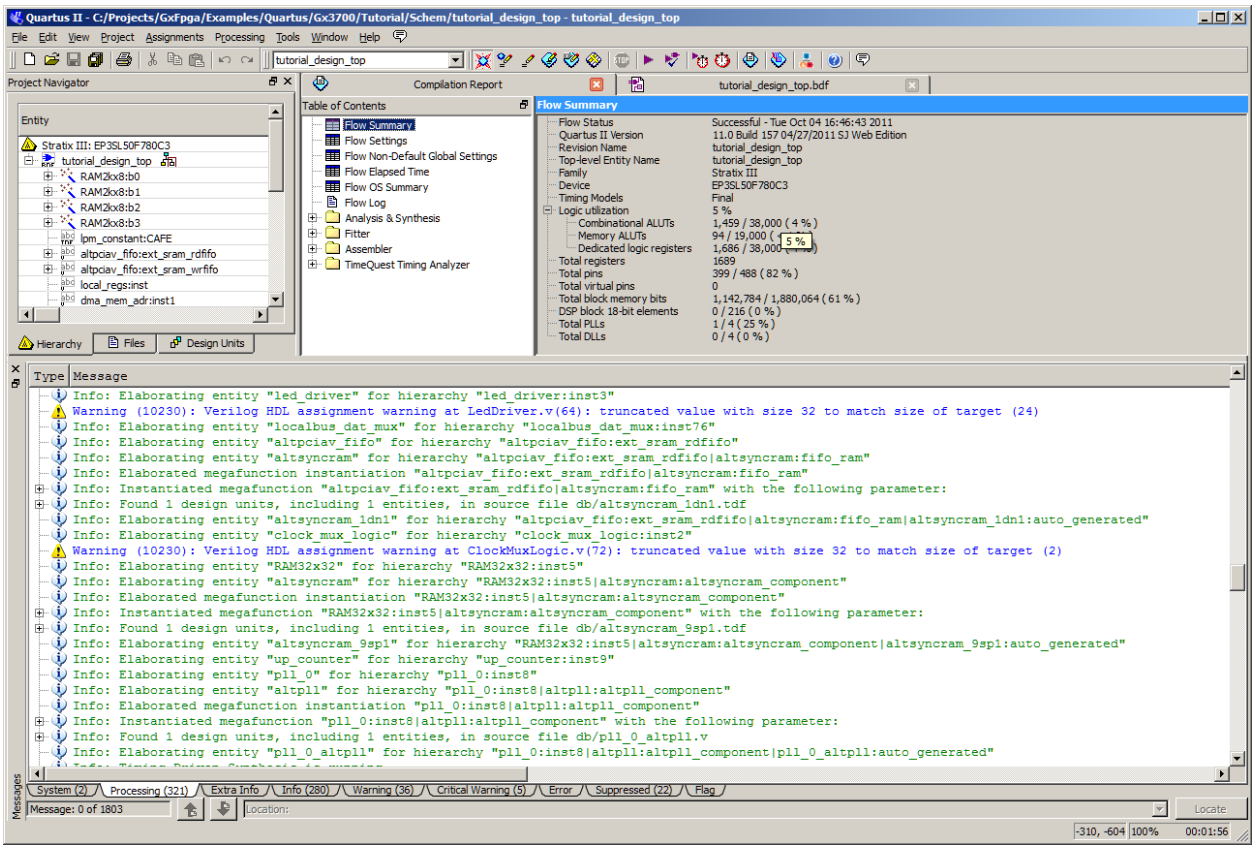


Figure 5-31: Compilation Tools and Status

The SVF file will be generated after the project compilation has finished. The **Compilation Task** window will show green check marks next to each major task to indicate completion.

In order to generate RPD file go to **File**, **Convert Programming Files …**

Select **Raw Programming Data File (.rpd)** as the **Programming file type** and **tutorial_design_top.rpd** as the **File Name**. Click on the **Add File** button and select **tutorial_design_top.pof**. The .pof file should now appear below the **POF Data** node as shown below. Finally, click the **Generate** button to create the RPD file.

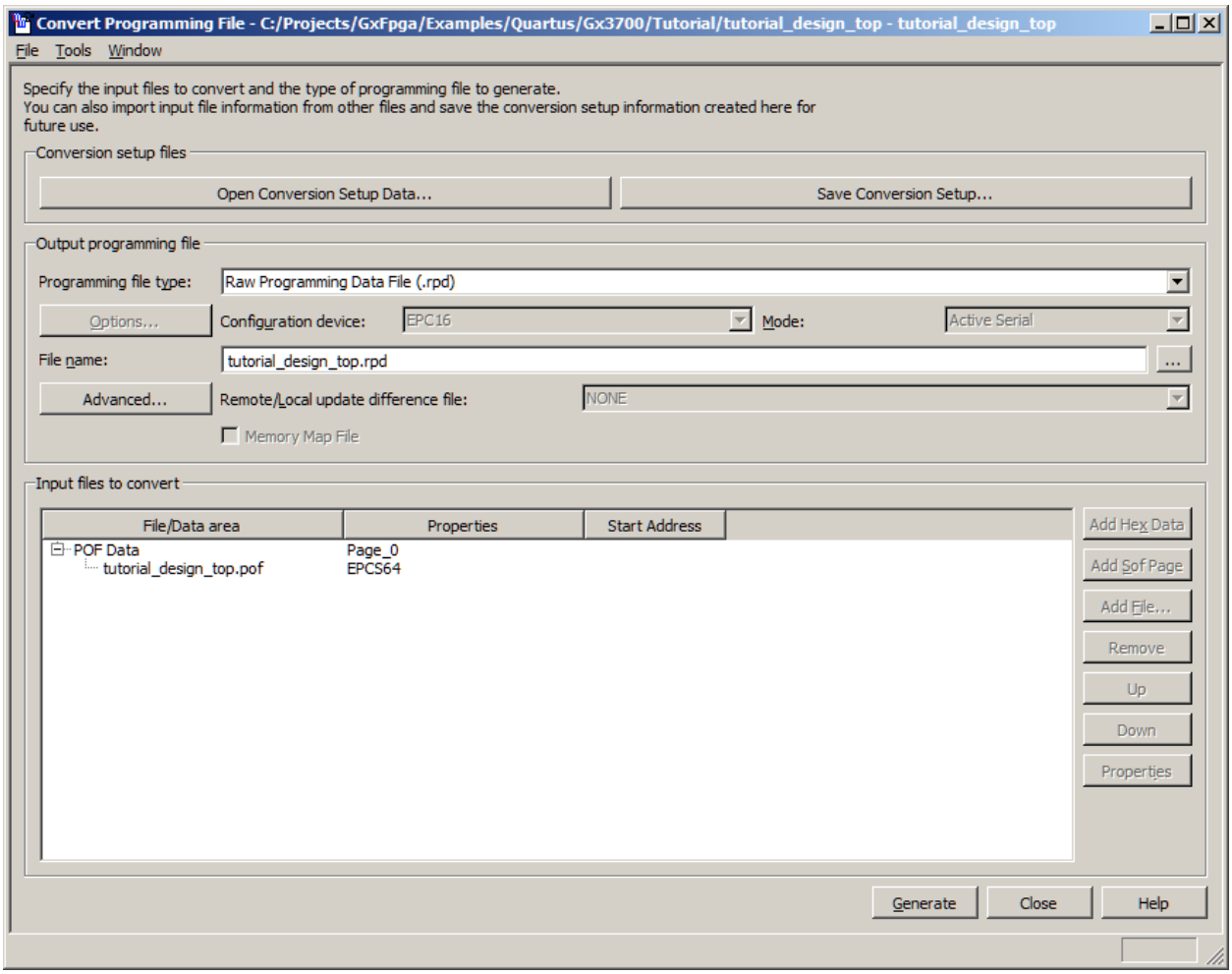


Figure 5-32: Convert Programming Files Dialog Box

Simulating the Design

To simulate the design, we will use **ModelSim** application from Altera. You can download the software for free form the Altera website. There is a test bench for this tutorial that is already created for you inside the **GXFPGA\Examples\Quartus\GX3700\Tutorial.**

Follow these steps to simulate the design:

1. Open the ModelSim application:

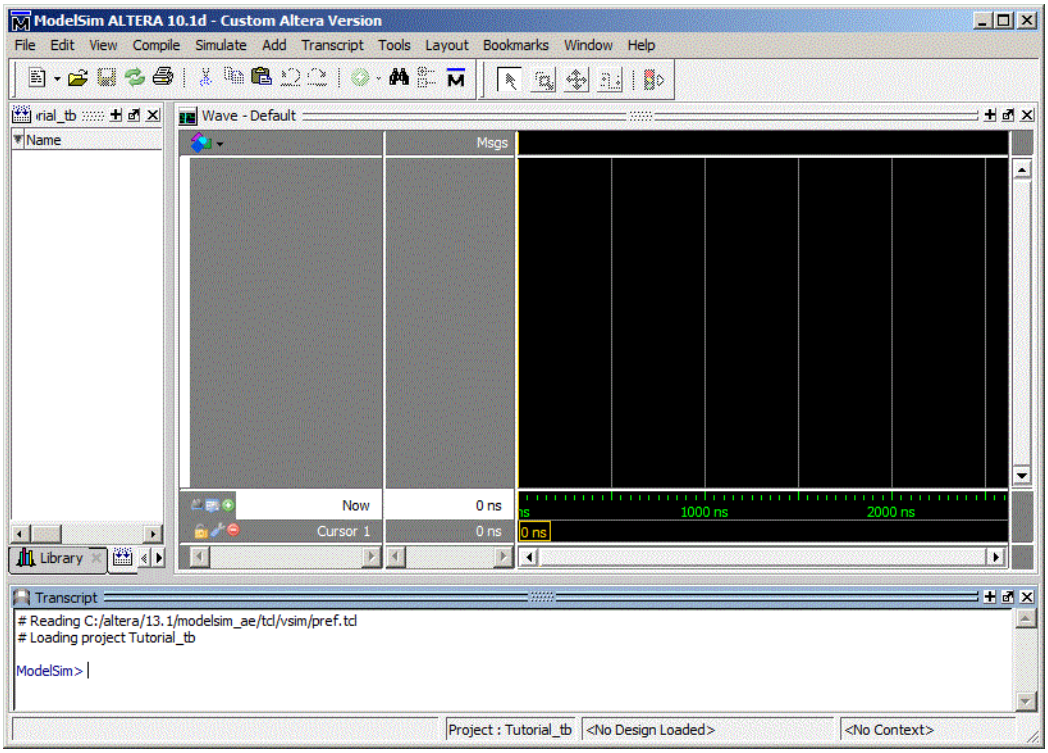


Figure 5-33: ModelSim Main Window

2. Click **File→ Change Directory** and choose the **sim** folder under the Tutorial folder. At this point the ModelSim should display the simulation pins:

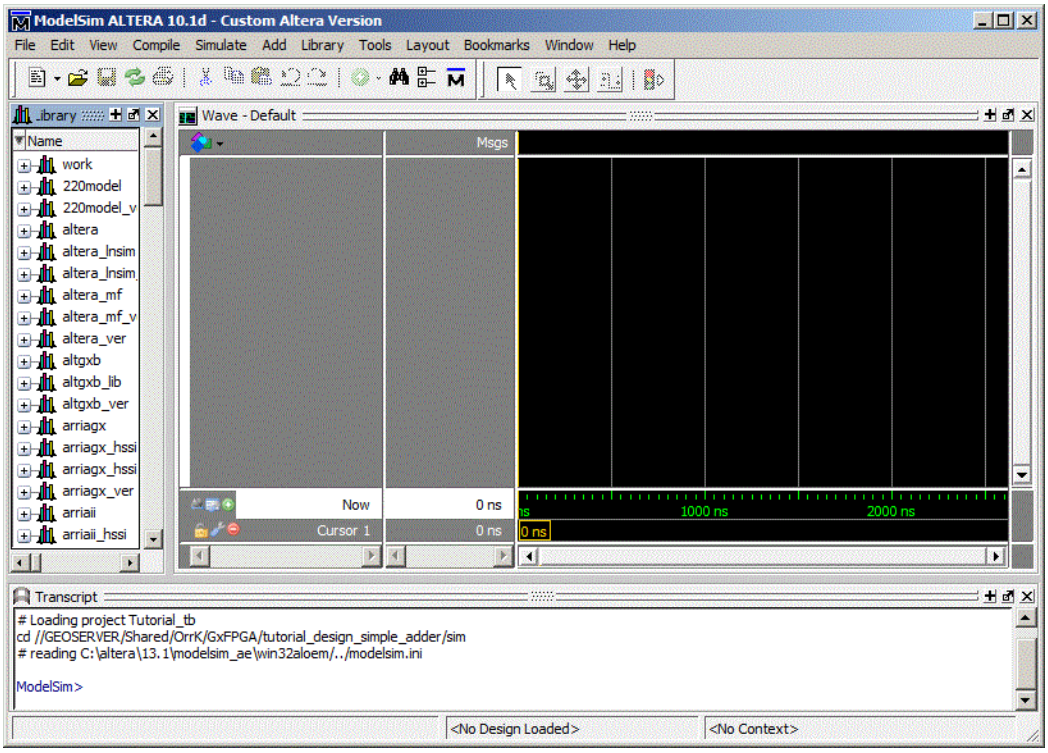


Figure 5-34: ModelSim Tutorial Simulation

3. Click **Tools→ Tcl→Execute Macro…** and choose **sim.tcl,** and click **Open.** When ModelSim asks to close the current project, click **Yes**. The screen should appear like the screen below:

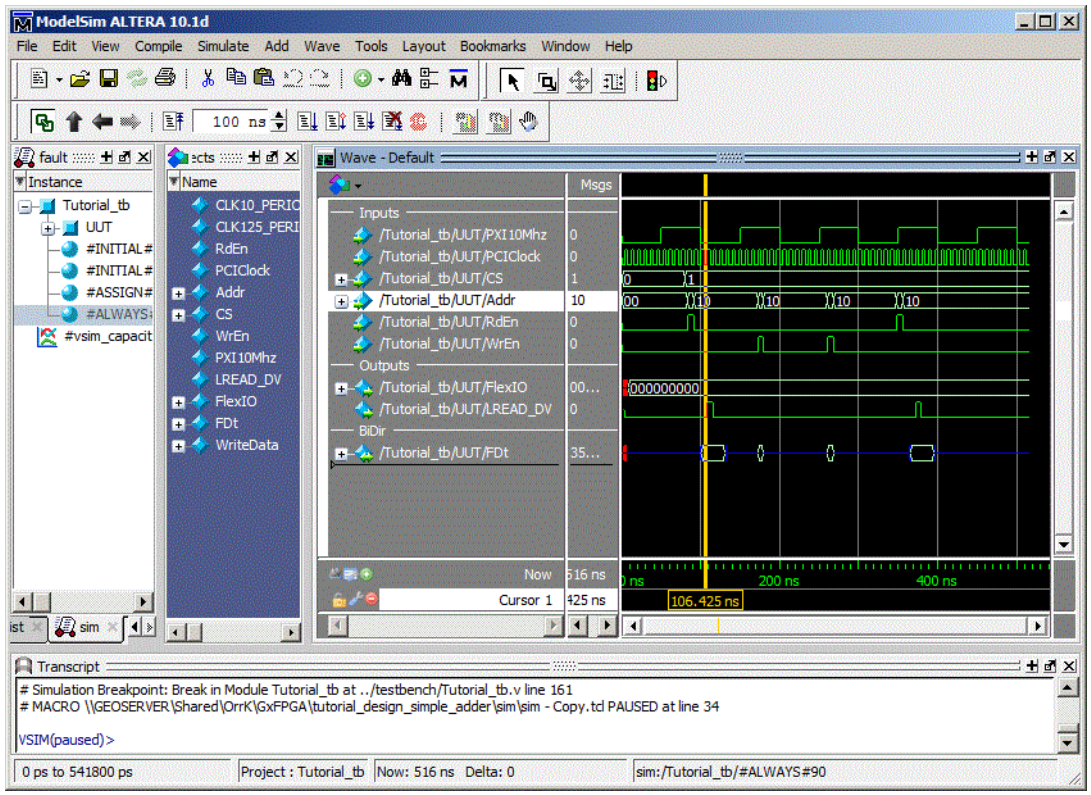


Figure 5-35: Simulation of Design

Load Gx3700 with SVF File

Start the **GX3700 Panel** (from the Windows **Start** menu, **Marvin Test Solutions**, **GxFpga**) and initialize the instrument. Next, click on the **Volatile** radio box and then click on the Browse Button (…) to select the newly generated SVF file (**tutorial_design_top.svf**). Finally click on the **Load** button to begin programming the card. You will see the progress bar indicate the status of the load. Once the load has completed, the status bar should be unfilled.

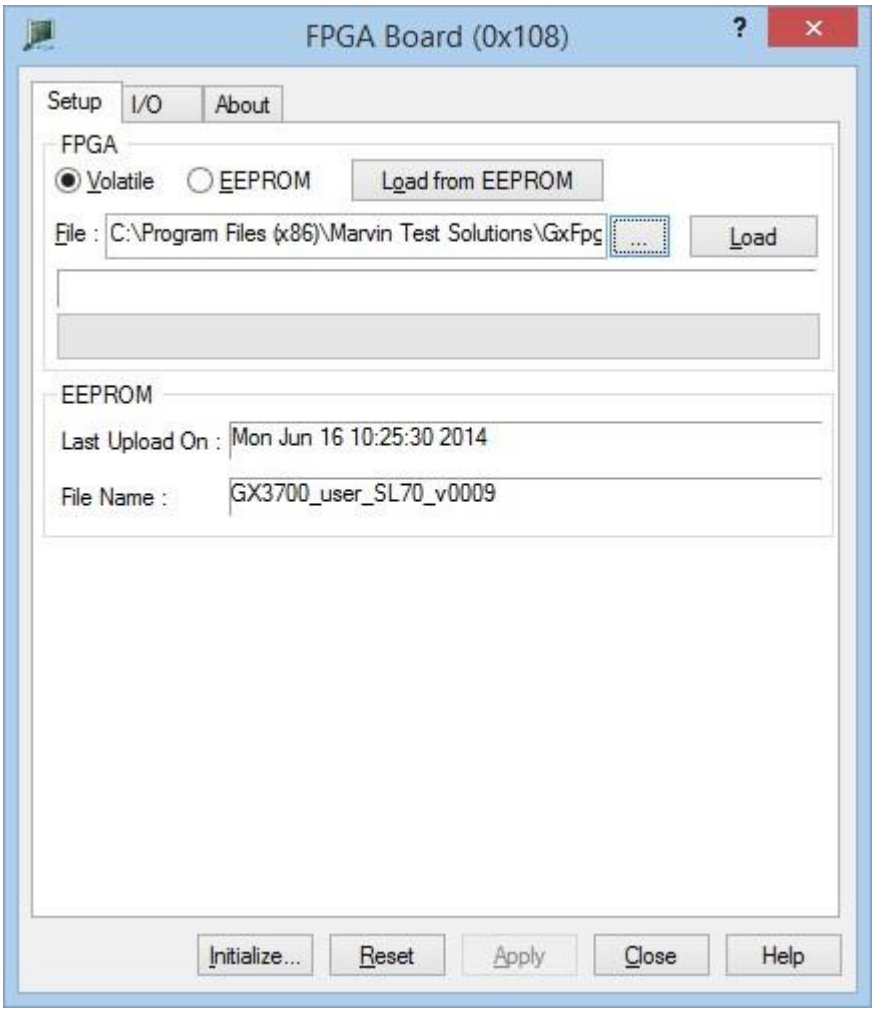


Figure 5-36: Software Front Panel

Testing the Design

Now that the design has been completed, compiled and loaded into the GX3700, we can move on to the testing.

There are two ways to access the FPGA, either through the software front panel or through the driver API DLL. We will demonstrate the programming method using ATEasy to access the driver API DLL.

Adder Testing

The software front panel will be used to test Phase 1 of the design which adds two 32 bit numbers together. Click on the **I/O Tab** to get started. The Adder phase is controlled through the FPGA Register space.

Offset 0x0 points to the first 32-bit number that will be summed and offset 0x4 points to the second 32-bit number that will be summed. Write values to both these locations.

The sum can be obtained by reading the 32-bit value at offset 0x8. Verify that the correct sum is read back as shown in Figure 5-31:

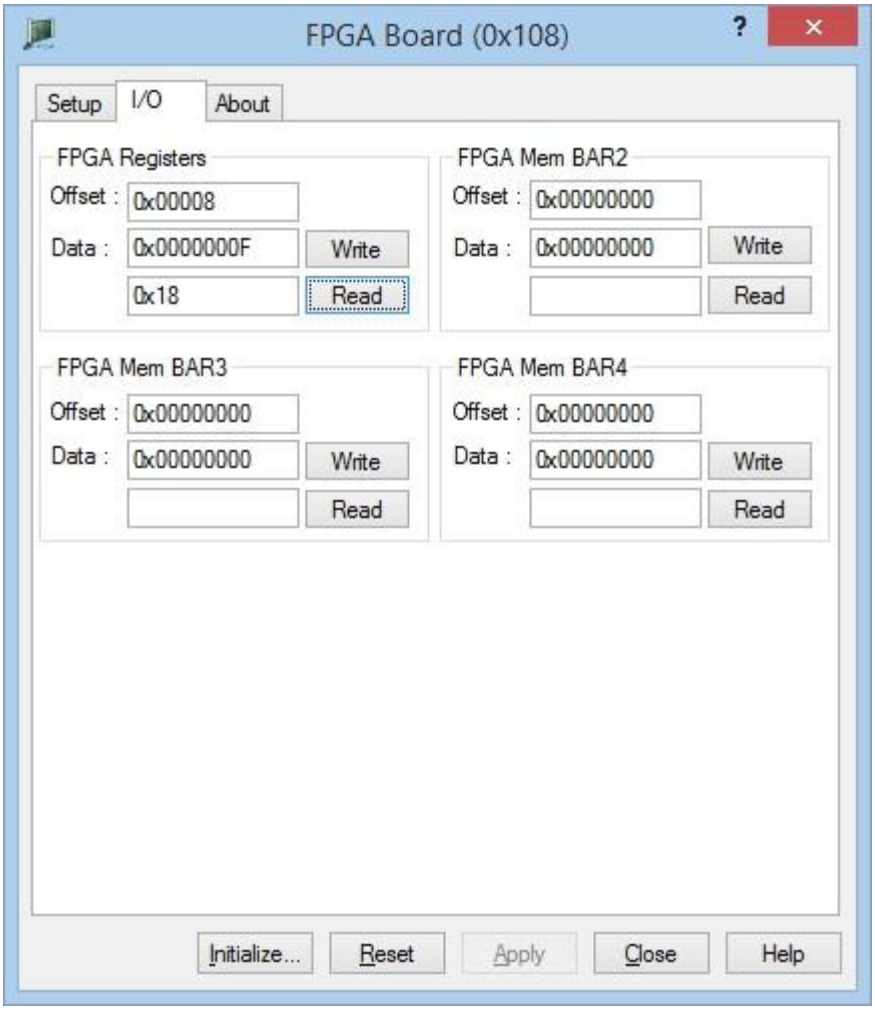


Figure 5-37: Using the Software Front Panel to read back the Sum

Clock Mux Testing

The software front panel will once again be used to test Phase 2 of the design. This part of the design uses a Mux to select between the PCI Clock and the 10 Mhz reference clock. The selected clock is output to I/O Channel 63 which

is located on pin 31 on the Flex I/O J2 connector of the GX3700. The Mux is controlled through the FPGA Register space.

Writing a 0x0 to offset 0xC will route the PCI/PCIe Clock signal to I/O Channel 63. Writing 0x1 to the same offset will route the 10 Mhz clock to this same channel. Try switching between both values while monitoring pin 31 of J2 with an oscilloscope. You should see the appropriate clock signals.

Digital Sequencer Testing

For this test, connect an oscilloscope to I/O Channel 65 (pin 1 of J3) to monitor the output signal of the sequencer. You can access the FPGA memory through the software front panel or through ATEasy. When using the software front panel, write values to the first 32 double words of the FPGA Memory space (offsets 0x0, 0x4, 0x8, 0xC etc). As you write to these locations, the data patterns being output on I/O Channel 1 should be updating dynamically. If you fill the 32-double word memory with a clock pattern (alternating 1's and 0's), you should be able to measure a frequency of 100Khz.

When using ATEasy, include the GxFPGA.drv driver and set it up with the correct slot number. Add a variable called **i** of type **long**. You can then run the following code to write to the FPGA memory:

```
REDIM adwData[32]
adwData[0] = 1
For i=0 to 31
    FPGA Write Memory(i*4, 4, adwData[i])
Next
```

This code will set the first double word to 1 and the rest to 0's resulting in a frequency of 6.25 Khz.

Chapter 6 - GXFPGA Verilog Tutorial

Introduction

This tutorial will go over the basic workflow to start designing and loading a FPGA configuration for the Gx3700. The example provides creation of a project using Verilog sources and coding. The "Tutorial design top reg.doc" contains the design register map.

The tutorial contents will entail:

- Downloading and installing the FPGA design tool
- Creating a new FPGA Design project with the Stratix III as the target device
- Setup the pin assignment to work with the GX3700 and Stratix III FPGA
- Use the Quartus IDE to import and design an example FPGA configuration
- Compile the project and generate the SVF and RPD programming files
- Loading the board with the generated programming files
- Testing the design using the Gx3700 Front Panel software and ATEasy
- The example configuration is broken down into three phases, each with a distinct function:
    - **Phase 1:** Take two values located in PCI Registers and generate a Sum (Adder) which can then be read through a third PCI Register.
    - **Phase 2:** 2 to 1 multiplexer to choose between the 10 MHz Clock and the PCI Clock which will be output on one of the FlexIO pins. The clock will be selected through a PCI Register.
- The source code for the examples in this chapter is provided in the Examples\Quartus\Gx3700 folder.

Downloading Altera Design FPGA Design Tools

The Marvin Test Solutions Gx3700 User programmable FPGA board can be designed using the free Altera Quartus II Web Edition or Subscription Edition design tool. This FPGA design tool allows end users to generate fully featured FPGA designs that can be downloaded to the Gx3700 board using the Marvin Test Solutions GXFPGA software API or software front panel. Other 3rd party tools can also be used to design the FPGA. Before proceeding with this tutorial, you must have Altera Quartus II **v11.0 SP1** installed on your PC. More information about this tool and how to download it can be found at http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html.

Create New Project

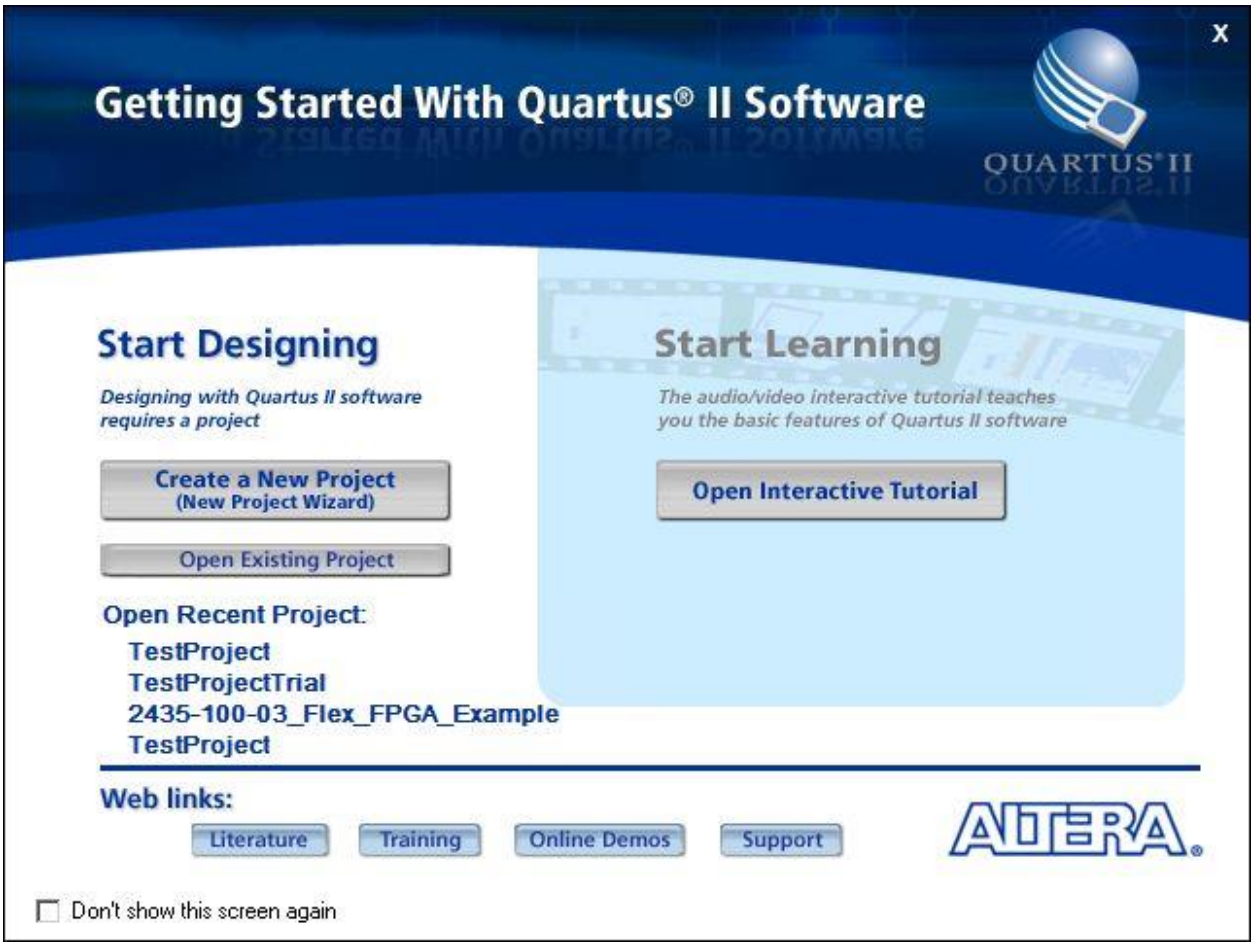


Figure 6-1: Quartus II Start Dialog

After installing Quartus II Web Edition, start the application and select C**reate a new Project** to start the New Project Wizard or select **File, New**, **New Quartus Project**.

Click on **Next** and then select the Project Folder and enter **tutorial_design_top** as the project name.

Click on **Next** twice (skip the adding files window).

Device Selection

The next window will allow you to select the FPGA target device. Select **Stratix III** as the Family and **EP3SL50F780C3** when using the GX3700 or **EP3SL70F780C3** for the GX3700e. For newer GXFPGA boards, the device ID will be displayed in the instrument front panel **About** page.

Click on **Next** twice (skip the Specify Tools window).

A window summarizing all the choices made for the creation of this project is shown. Click on **Finish**.

Pin Assignment Setup

You should now have an empty skeleton project loaded in Quartus II. Before you can get started on the FPGA design, you must assign the FPGA pins distinct names so that you can reference them in your design. This can be accomplished by running a **TCL script** which contains all the information necessary to configure the pin assignments as well as settings the project to either schematic entry or Verilog entry. These pin assignments are unique to this Stratix III FPGA and the GX3700 in particular. The following table lists all the pin assignments and their respective descriptions. The Pin Alias's listed in the table are the pin names you will be using in your design to reference the actual hardware pins on the FPGA.

Pin Assignments Table

Pin Alias (Node Name)	Description
Clocks	
10Mhz	Input. 10 MHz Reference Clock Signal from the PXI Backplane
PCIClock	Input. 33 MHz PCI Bus clock or 125MHz PCI Express application clock.
RefClk	Input. 80 Mhz Reference Clock onboard the GX3700
PCI Bus	
Addr[2..19]	Input. The PCI Address lines from the PCI bus
FDt[0..31]	Bidir. PCI Data lines from the PCI bus
CS[1..3]	Input. Chip Select lines from the PCI bus. CS[1] is for FPGA registers, CS[2] is for internal SRAM, CS[3] is currently not used.
LEXT	Input. External SRAM chip select. This is chip select for external SRAM on PCB.
RdEn	Input. PCI Read Enable line from the PCI bus
WrEn	Input. PCI Write Enable line from the PCI bus
LREAD_DV	Output. Read data valid. This is data valid for FDt(31:0) data bus.
LUW	Input. Currently not used. Upper Word.
LLW	Input. Currently not used. Lower Word.
LRESET	Input. Currently not used. Reset coming from PXI bridge FPGA.
PXI Bus	
PxiTrig[0..7]	Bidir. PXI Bus trigger signals
StarTrig	Output. PXI Star Trigger signal. This signal can be re-defined by the user as bi-directional.
PXI_LBL6	Bidir. PXI Local Bus Left 6. This is local bus according to PXIe spec.
PXI_LBR6	Bidir. PXI Local Bus Right 6. This is local bus according to PXIe spec.
PXIe_DSTARA	Input. PXIe DSTAR trigger A. This is DSTAR trigger according to PXIe spec.
PXIe_DSTARB	Input. PXIe DSTAR trigger B. This is DSTAR trigger according to PXIe spec.
PXIe_DSTARC	Output. PXIe DSTAR trigger C. This is DSTAR trigger according to PXIe spec.
PXIE_100M	Input. PXIe 100MHz clock. This is 100MHz clock according to PXIe spec.
PXIE_SYNC100	Input. PXIe Sync100. This is Sync100 signal according to PXIe spec.
I/O	
FlexIO[1..160]	Bidir. The physical IO Channels including 4 global clock inputs (2 differential pairs).

External Flash	
Fsm_a[1..23]	Output. Address bus shared by external SRAM and flash.
Fsd[0..31]	Bidir. Data bus shared by external SRAM and flash.
Flash_ce_n	Output. Flash chip enable.
Flash_oe_n	Output. Flash output enable.
Flash_we_n	Output. Flash write enable.
Flash_reset_n	Output. Flash chip reset
Flash_byte_n	Output. Flash byte/word select.
Flash_busy_n	Input. Flash busy
External SRAM	
Sram_be_n[0..3]	Output. External SRAM byte enable.
Sram_ce_n	Output. External SRAM chip select.
Sram_oe_n	Output. External SRAM output enable.
Sram_we_n	Output. External SRAM write enable.
RX DMA FIFO I/F	
RX_DMA_DAT[0..31]	Input. Receive DMA data coming from PC host.
RX_DMA_DV	Input. Receive DMA data valid.
RX_DMA_FIFOFULL	Output. Receive DMA FIFO full. This will throttle data from PC host.
RX_DMA_SP1	Output. Spare. Currently not used.
RX_DMA_SP2	Output. Spare. Currently not used.
TX DMA FIFO I/F	
TX_DMA_DAT[0..31]	Output. Transmit DMA data from memory going to PC host.
TX_DMA_DV	Output. Transmit DMA data valid.
TX_DMA_FIFOEMPTY	Output. Transmit DMA FIFO empty. When empty and is sending data to PC host, the DMA engine in PXI bridge FPGA will assert FIFO read enable TX_DMA_FIFO_RD.
TX_DMA_FIFO_RD	Input. Transmit DMA FIFO read enable.
Misc	
Spare[0..7]	Bidir. Do Not Use. Spares connected to PXI bridge FPGA.
IRQ	Output. Interrupt output pin going to PXI bridge FPGA IRQ = 1 means interrupt will be generated to PC host. IRQ = 0 means no interrupt.
FSpr[0..3]	Bidir. Spare Signals connected to Expansion Board
MClr	Input. FPGA Master Clear, Active High
TP[0..5]	Bidir. Connected to test header J7 on the GX3700 PCB
ACTIVE_LED_N	Output. Active LED. Connect to LD1 LED on board. '0' = LED on, '1' = LED off.

Table 6-1: Pin Assignments Table

Verilog project

In order to configure the project as Verilog and configure the pin assignment the TCL configuration script should be added to the project. To add the script to the project, click on **Project | Add/Remove Files in Project…** In the dialog box, click on the **…** button and browse for GX3700Verilog.tcl file in the "C:\Program Files\Marvin Test Solutions\GxFpga\" folder. On some systems, it is recommended to move the desired TCL file to your project's source location prior to adding it to the project. Click Open and then the **Add** button.

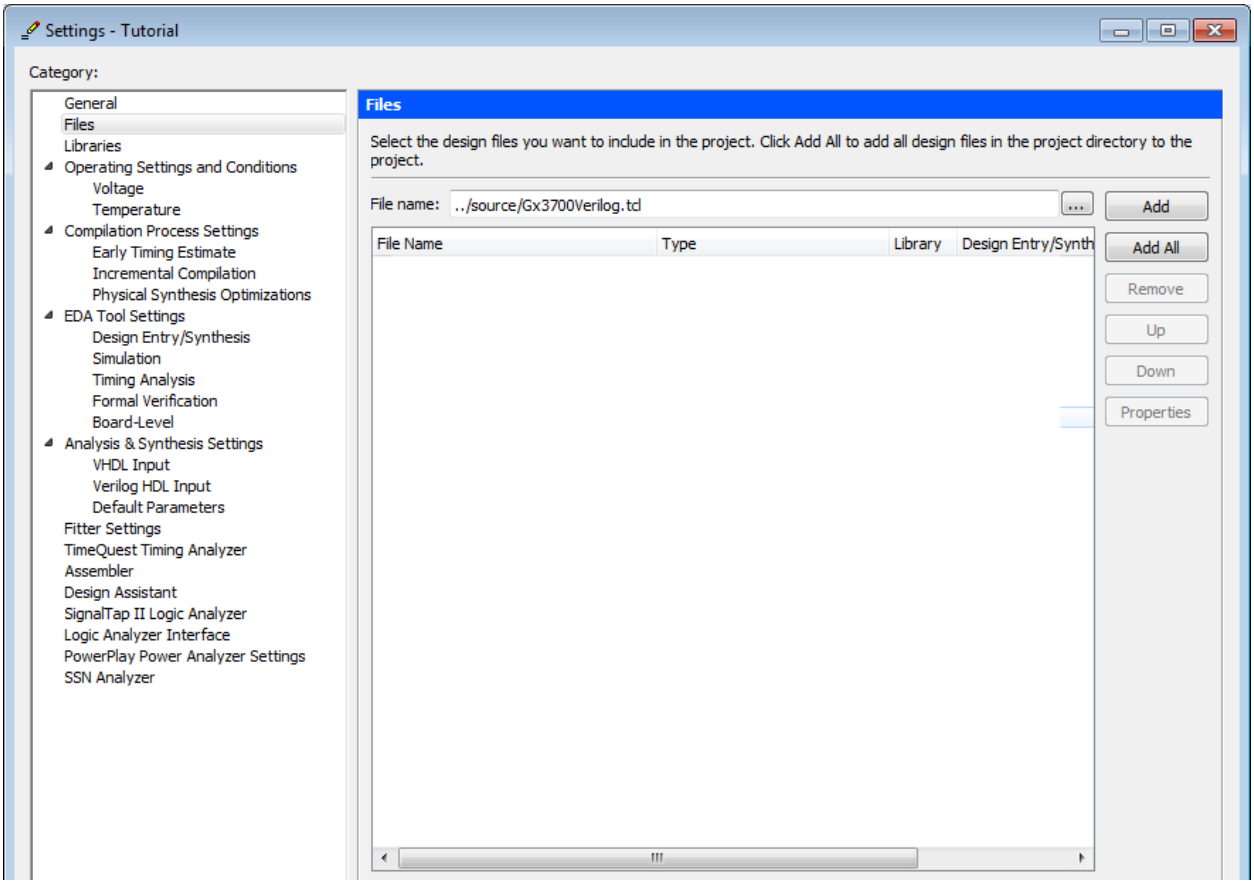


Figure 6-2: Add Tcl Script to Project

Then click on Tools | TCL Scripts … Select the configuration script file, **GX3700Verilog.tcl** and click on **Run**. This will configure your FPGA pin assignments.

Note: The TCL file will automatically add all the source files needed for the tutorial design to the Quartus II project.

You can view the pin assignments by running the Pin Planner application which is found in the Tasks list as highlighted below:

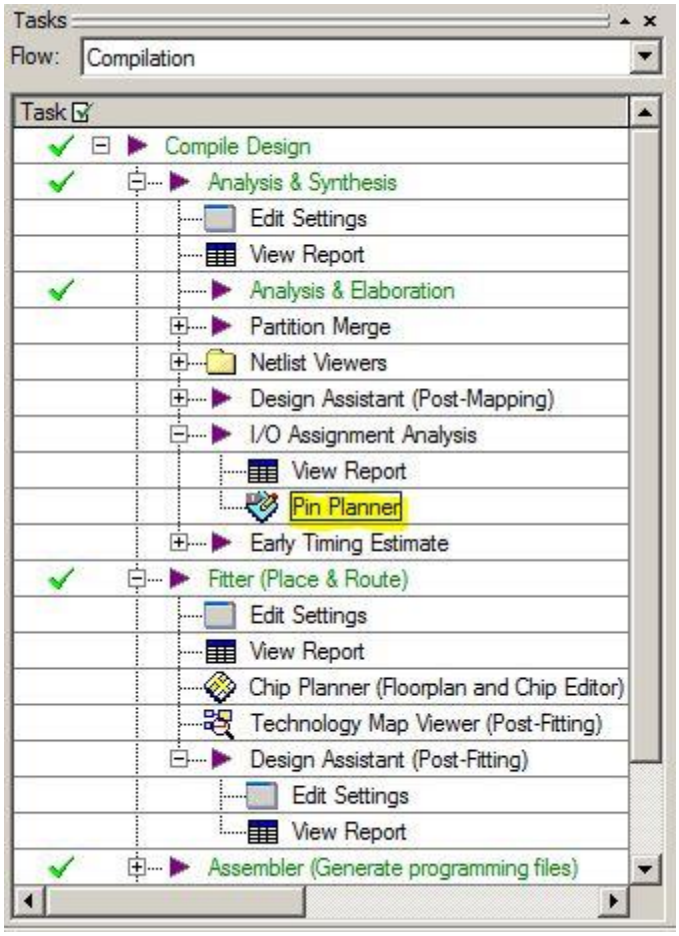


Figure 6-3: Task Flow

The Pin Planner will display a matrix of the physical FPGA pins and their mapped names as well as the I/O standard supported by the pin. These mapped names are used in the FPGA design, as wire names and I/O pins, to connect to the physical connections of the FPGA.

Creating Design File with Verilog

At this point you will have successfully created an FPGA design based on the source codes provided. This section will walk you through the steps of creating modeled components in several modules.

Note: There is more than one way to accomplish the following designs.

Phase 1: Creating the FPGA design - 32 bit Full Adder

This design will take two double word (32 bit) values, located in the first two double words in the Register space (byte offset 0x0 and 0x4), and add them together. The sum of the two values will be immediately output to the third double word in the Register space (byte offset 0x8). The sources for all referenced components are installed with the GXFPGA software package to C:\Program Files\Marvin Test Solutions\GxFpga\Examples\Quartus\Gx3700\Tutorial_Verilog\source

Components Used

- **adder.v** – An n-bit full adder
- **and_gate.v** - A two input and gate, the first input in n-bit width
- **d_flipflop.v** – A n-bit D flip-flop
- **decoder.v** – An n-bit decoder
- **or_gate2.v** - A two input or gate, the first input in n-bit width
- **or_gate4.v** - A four input or gate

Top-level Verilog file

In order to open the Verilog text editor, click on **File** menu, and then **New** the following dialog appears.

Select Verilog HDL File:

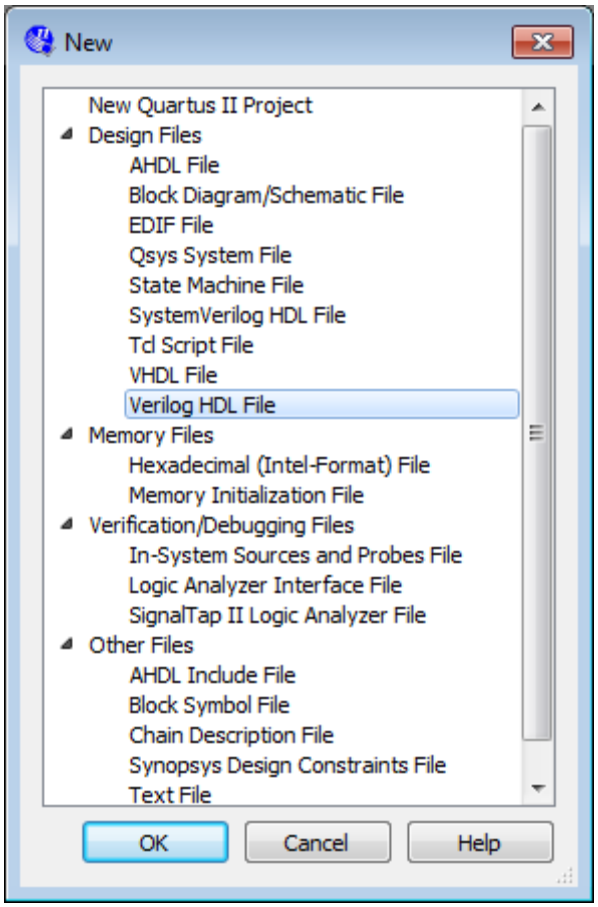


Figure 6-4: New File Dialog Box

Top-level inputs and outputs

The top-level object for this project will be named **tutorial_design_top.v**. Start by creating module prototype with the proper inputs and outputs. The inputs and outputs all correspond to pin on the FPGA.

```
//-------------------------------------------------------------------
// Design Name : GXFPGA Verilog Tutorial
// Function    : Demonstrates functionality described in the
//             Verilog Tutorial chapter of the GXFPGA User's Guide.
//-------------------------------------------------------------------

module tutorial_design_top(Addr, WrEn, CS, PCIClock, PXI10Mhz, RdEn, FlexIO, LREAD_DV, FDt);

input [6:2] Addr;
input WrEn;
input [2:1] CS;
input PCIClock;
input PXI10Mhz;
input RdEn;

output [65:33] FlexIO;
output LREAD_DV; // Read Data Valid flag

inout [31:0] FDt;

endmodule
```

Figure 6-5: GXFPGA Verilog Tutorial Prototype

The first step is creating the circuitry required to decode the PCI Address when data is to be written from the PC to the FPGA. This circuit will be used in all three functions of this example project. The signals required for PCI Write access will be the **PCI Clock**, **Write Enable**, **Chip Select 1**, and some **PCI Address lines**. The PCI Address lines 5 to 2 will be fed to a decoder which will generate a 32-bit value, and the result will be ANDed with the Chip Select 1 bit. Each Chip Select bit represents a certain PCI BAR access (GX3700 has two bars, memory and register memories). Bit 1 represents BAR1 of the PCI memory space (bit 2 for BAR2). BAR1 is the general-purpose Control Register BAR for the GX3700. The results of the AND operation will be once again ANDed to the Write Enable PCI signal.

To create the address decoder, we'll need to model the D Flip-flop (to latch the inputs), the And gate, and the decoder. For each module that we add, you should use the New File Dialog to add a Verilog HDL file to create the blank file. When saving, give the file the same name as the module. The source for the referenced modules follows:

```
//------------------------------------------------------------------
// Design Name : and_gate
// Function    : A two input and gate, the first input in n-bit width
//------------------------------------------------------------------

module and_gate (out, in1, in2);
parameter width=1;

output [width-1:0] out;
input [width-1:0] in1;
input in2;

assign out = in2 ? in1 : 0;

endmodule
```

Figure 6-6: and_gate.v source

When saving, give the file the same name as the entity. The source for the referenced entity follows:

```
//-----------------------------------------------------
// Design Name : d_flipflop
// Function    : A n-bit D flip-flop
//-----------------------------------------------------

module d_flipflop (d, clk, ena, clrn, q);
parameter width = 1;

output [width-1:0] q;
input clk, ena, clrn;
input [width-1:0] d;

reg [width-1:0] q;

always @ ( posedge clk or negedge clrn )
begin
    if (~clrn)
        q <= 'b0;
    else if (ena)
        q <= d;
end

endmodule
```

Figure 6-7: d_flipflop.v source

When saving, give the file the same name as the module. The source for the referenced modules follows:

```
//-----------------------------------------------------
// Design Name : decoder
// Function    : An n-bit decoder
//-----------------------------------------------------

module decoder (decoder_in, enable, decoder_out);
parameter input_bit = 2;

output [2 ** input_bit-1:0] decoder_out ;
input [input_bit-1:0] decoder_in;
input enable;

assign decoder_out = enable ? (1 << decoder_in) : 'b0 ;

endmodule
```

Figure 6-8: decoder.v source

In tutorial_design_top.v, we will now write the code to describe our PCI Address Decoder Circuit. Latch both the Address and Write Enable lines using the PCI Clock. Decode the 5 bit Address lines into a 32-bit bus named DecodedAddr. This decoded bus is ANDed with the FPGA's CS[1] to define our PCI Address Decoded Select lines.

Additionally, we will define our Write Enable (WE) lines in this code block. We will use this later, along with Read Enable, to read and write to registers.

```
// PCI Address Decoder Circuit
wire [4:0] LatchedAddr;
wire LatchedWrEn, LatchedRdEn;
wire [31:0] DecodedAddr, WE, Sel;
d_flipflop            inst23(WrEn, PCIClock, nc_ena, nc_rst, LatchedWrEn);
d_flipflop #(5)       inst24(Addr, PCIClock, nc_ena, nc_rst, LatchedAddr);
decoder #(5)   inst(LatchedAddr, nc_ena, DecodedAddr);
and_gate #(32) inst2(Sel, DecodedAddr, CS[1]);
and_gate #(32) inst3(WE, Sel, LatchedWrEn);
```

Figure 6-9: PCI Address Decoder Circuit

You will notice that we used a few undefined symbols in this last section: **nc_ena** and **nc_rst**. These are placeholders for enable and reset lines that our various components can take advantage of. For this tutorial, I have chosen not to use enable or reset lines at all so we should add the following code to tutorial_design_top.v to explicit set these wires to always enabled, never reset.

```
wire nc_rst, nc_ena;
assign nc_rst = 1'b1;  // No reset
assign nc_ena = 1'b1;  // Always enabled
```

Now that the PCI address decoder circuit is complete, we can feed the appropriate bits from the WE bus to D Flip Flops that will store data clocked in from the PCI data lines. For example, the first double word in PCI memory (representing the first number to be summed) will be written to a D Flip Flop with its enable line tied to WE[0] (the first bit in the WE bus). The second double word to be added will be written to another D Flip Flop with its enable line tied to WE[1]. Finally, the PCI Clock signal (33Mhz) will be used as the clock source of the D Flip Flops. Note that each bit of the Sel and WE buses represent a consecutive double word address (bit 0 corresponds with byte 0, bit 1 corresponds with byte 4, bit 2 corresponds with byte 8 etc.).

First we start by creating an extend circuit to deal with any timing issues with the WE signal. Then we will create some Flip Flops to latch inputs to the adders. We will use a placeholder named **LatchedFDt** as the input to the D Flip Flops. Eventually the PCI data lines will drive these inputs. Wire the outputs of the D Flip Flops to the Adder component. The output of the adder, **Sum**, will be used as an output later.

```
//-----------------------------------------------------
// Design Name : or_gate4
// Function    : A four input or gate
//-----------------------------------------------------

module or_gate4 (out, in1, in2, in3, in4);
parameter width=1;
output [width-1:0] out;
input [width-1:0] in1, in2, in3, in4;

assign out=in1|in2|in3|in4;

endmodule
```

Figure 6-10: or_gate4.v source

```
//-----------------------------------------------------
// Design Name : or_gate2
// Function    : A two input or gate, the first input in n-bit width
//-----------------------------------------------------

module or_gate2 (out, in1, in2);
parameter width=1;
output [width-1:0] out;
input [width-1:0] in1, in2;

assign out=in1|in2;

endmodule
```

Figure 6-11: or_gate2.v source

```
//-----------------------------------------------------
// Design Name : adder
// Function    : An n-bit full adder
//-----------------------------------------------------

module adder (dataa, datab, result);
parameter width = 1;

output [width-1:0] result;
input [width-1:0] dataa, datab;

assign result = dataa+datab;

endmodule
```

Figure 6-12: adder.v source

```
// WE[31..0] extend circuit - Extend write enable to mitigate timing issues
wire [31:0] LatchedWE, LatchedWE2, LatchedWE3, WE_EXT;
d_flipflop #(32)       inst26(WE, PCIClock, nc_ena, nc_rst, LatchedWE);
d_flipflop #(32)       inst27(LatchedWE, PCIClock, nc_ena, nc_rst, LatchedWE2);
d_flipflop #(32)       inst28(LatchedWE2, PCIClock, nc_ena, nc_rst, LatchedWE3);
or_gate4 #(32) inst30(WE_EXT, WE, LatchedWE, LatchedWE2, LatchedWE3);

// Adder circuit - Latch the addends and include adder
wire [31:0] Sum, Addend1, Addend2;
d_flipflop #(32)       inst4(LatchedFDt, PCIClock, WE_EXT[0], nc_rst, Addend1);
d_flipflop #(32)       inst5(LatchedFDt, PCIClock, WE_EXT[1], nc_rst, Addend2);
adder #(32)    inst7(Addend1, Addend2, Sum);
```

Figure 6-13: WE Extend Circuit and Adder Circuit

Before moving on we must first extend the RdEn signal. Add the following to the tutorial_design_top.v:

```
// RdEn to 2 PCI Circuit
wire RdEn_Extend;
wire [31:0] RE;
or_gate2               inst1(RdEn_Extend, RdEn, LatchedRdEn);
d_flipflop             inst8(RdEn, PCIClock, nc_ena, nc_rst, LatchedRdEn);
and_gate #(32) inst12(RE, Sel, RdEn_Extend);
d_flipflop             inst21(LatchedRdEn, PCIClock, nc_ena, nc_rst, LREAD_DV);

// RE[31..0] extend circuit - Extend read enable to mitigate timing issues
wire [31:0] LatchedRE, LatchedRE2, LatchedRE3, RE_EXT;
d_flipflop #(32)       inst18(RE, PCIClock, nc_ena, nc_rst, LatchedRE);
d_flipflop #(32)       inst19(LatchedRE, PCIClock, nc_ena, nc_rst, LatchedRE2);
d_flipflop #(32)       inst20(LatchedRE2, PCIClock, nc_ena, nc_rst, LatchedRE3);
or_gate4 #(32) inst22(RE_EXT, RE, LatchedRE, LatchedRE2, LatchedRE3);
```

Figure 6-14: RdEn to 2 PCI Circuit and RE Extend Circuit

We also create a Read Data Valid output pin, **LREAD_DV**. This comes from a D-Flipflop with the PCIClock as an input clock and the RdEn as the input data. The D-Flip Flop also creates our extender for our ReadEnable.

The inputs to the D Flips Flops can now be wired to the PCI data lines (FDt). We need to clean up the FDt signal as is comes back into our circuit by adding the D-FlipFlop.

```
// Tri-stated FDt pins
wire [31:0] FDt, LatchedFDt;
reg [31:0] FDt_out_value;
reg [31:0] FDt_in_value;
assign FDt = RE_EXT ? FDt_out_value : 32'bz;
d_flipflop #(32)       inst25(FDt_in_value, PCIClock, nc_ena, nc_rst, LatchedFDt);
always @(posedge PCIClock) begin
    if (RE_EXT[2]==1'b1)
        FDt_out_value <= Sum;
    else if (RE_EXT[0]==1'b1)
        FDt_out_value <= Addend1;
    else if (RE_EXT[1]==1'b1)
        FDt_out_value <= Addend2;
    else if (RE_EXT[31]==1'b1)
        FDt_out_value <= result;

    FDt_in_value <= FDt; //store the input value
end
```

Figure 6-15: FDt in/out signal assignment

Now that the design has been completed, a revision number should be added so that the end user can read it back from the PCI bus at the 32nd register double word location (byte address 0x7C).

Including a revision number constant to the design is a Marvin Test Solutions standard practice that we recommend end users to follow. The revision constant is 32 bits long and is read as a hexadecimal number such as 0x3564A000. The first two digits of the hexadecimal number represent the company, in this case 35 is for Marvin Test Solutions designs. The next two digits are the design specific code, 64 in this case. And the last 4 digits, A000, is the revision of the design.

Add the following to tutorial_design_top.v:

```
// Add revision constant
reg [31:0] result = 32'h3564A000;
```

Figure 6-16: Symbol Properties

Phase 2: Creating the FPGA Design - 2 to 1 Clock Mux

This design will output either the PCI Clock (33Mhz) or the 10Mhz clock to FlexIO Channel 65 (Check connectors tables for the correct pin location) depending on what was written to the 4th double word in the PCI register space (byte offset 0xC). A 1 will select the 10Mhz clock signal, and a 0 will select the PCI clock signal.

Design

You will now build upon the tutorial project to add the functionality of a 2 to 1 Clock Mux. The 10Mhz clock will be brought into the design by an input pin. The PCI Clock signal input pin is already present in the Phase 1 circuit, so this will be reused. FlexIO[65] (IO Channel 65) will be used to output the selected clock to the outside world.

```
// Clock Mux Circuit
wire LatchedFDt0, PCIClock;
d_flipflop            inst6(FDt[0], PCIClock, WE_EXT[3], nc_rst, LatchedFDt0);
assign FlexIO[65] = LatchedFDt0 ? PXI10Mhz : PCIClock;
```

Figure 6-17: Clock Mux Circuit

FDt[0] is the first bit of the PCI data bus. This bit can either be 0 or 1, to indicate which clock source to choose. WE_EXT[3] is the 4th bit from the decoded PCI Address. When this bit is high, it indicates that the PCI Bus is addressing the 4th double word (byte offset 0xC) of the Register space for the GX3700. In our case, the value of this double word is used to select which clock is selected by our Mux.

At this point the design is complete, continue with the next sections to generate SVF or RPD files and load your design to the GX3700.

Configure Project to Output SVF and RPD Files

To ensure that a SVF file is generated upon project compilation, go to the **Assignments**, **Device ...** and click on the **Device and Pin Options** button. Then click on the **Programming Files** and verify that the **Serial Vector Format File** checkbox has been selected.

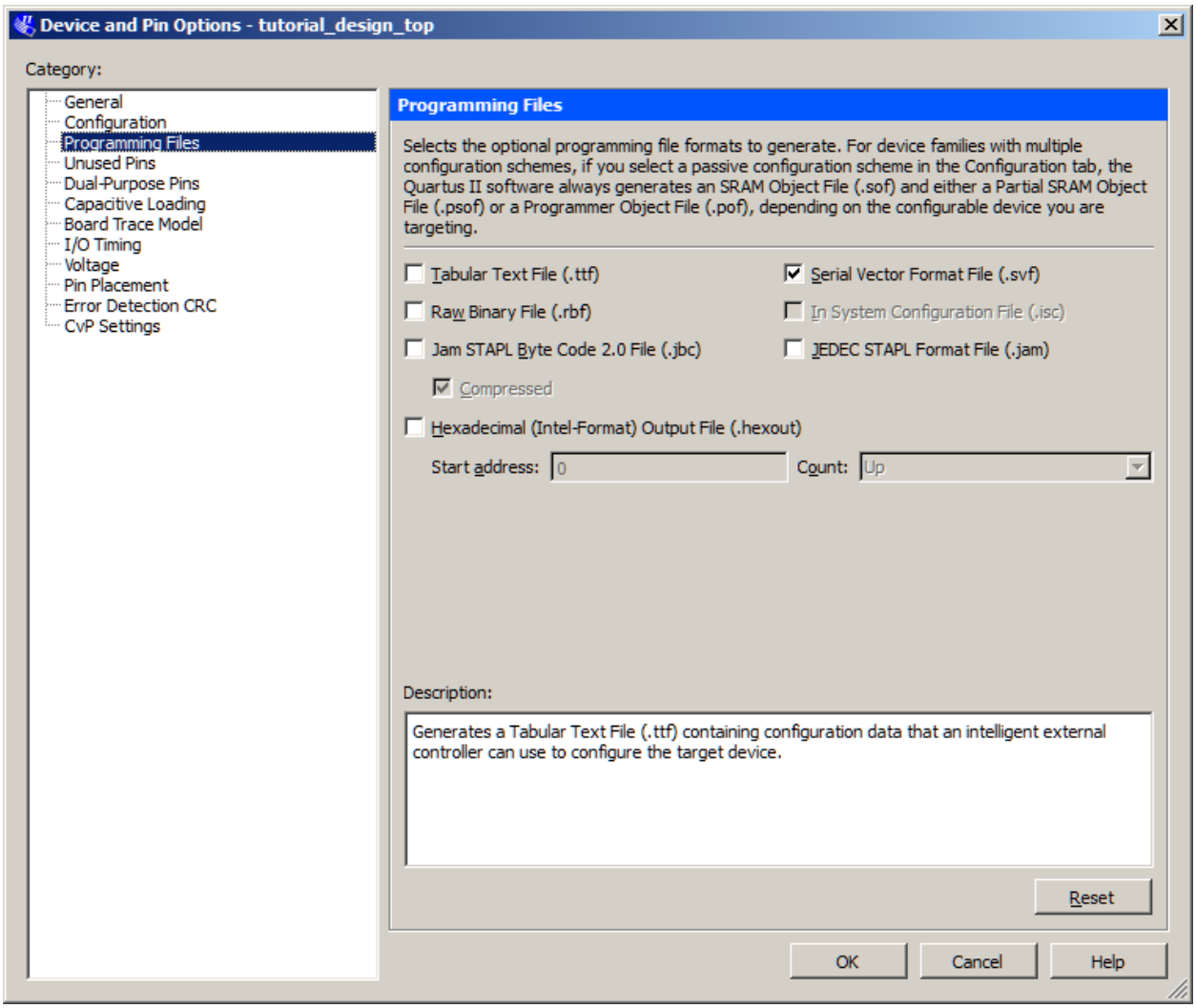


Figure 6-18: Select SVF as output file

Now click on the **Configuration** choose **Active Serial Configuration Scheme**, check **Use Configuration Device** checkbox and select **EPCS64** as the configuration device from the drop down selection. Finally click on **OK** twice to exit the settings dialog boxes.

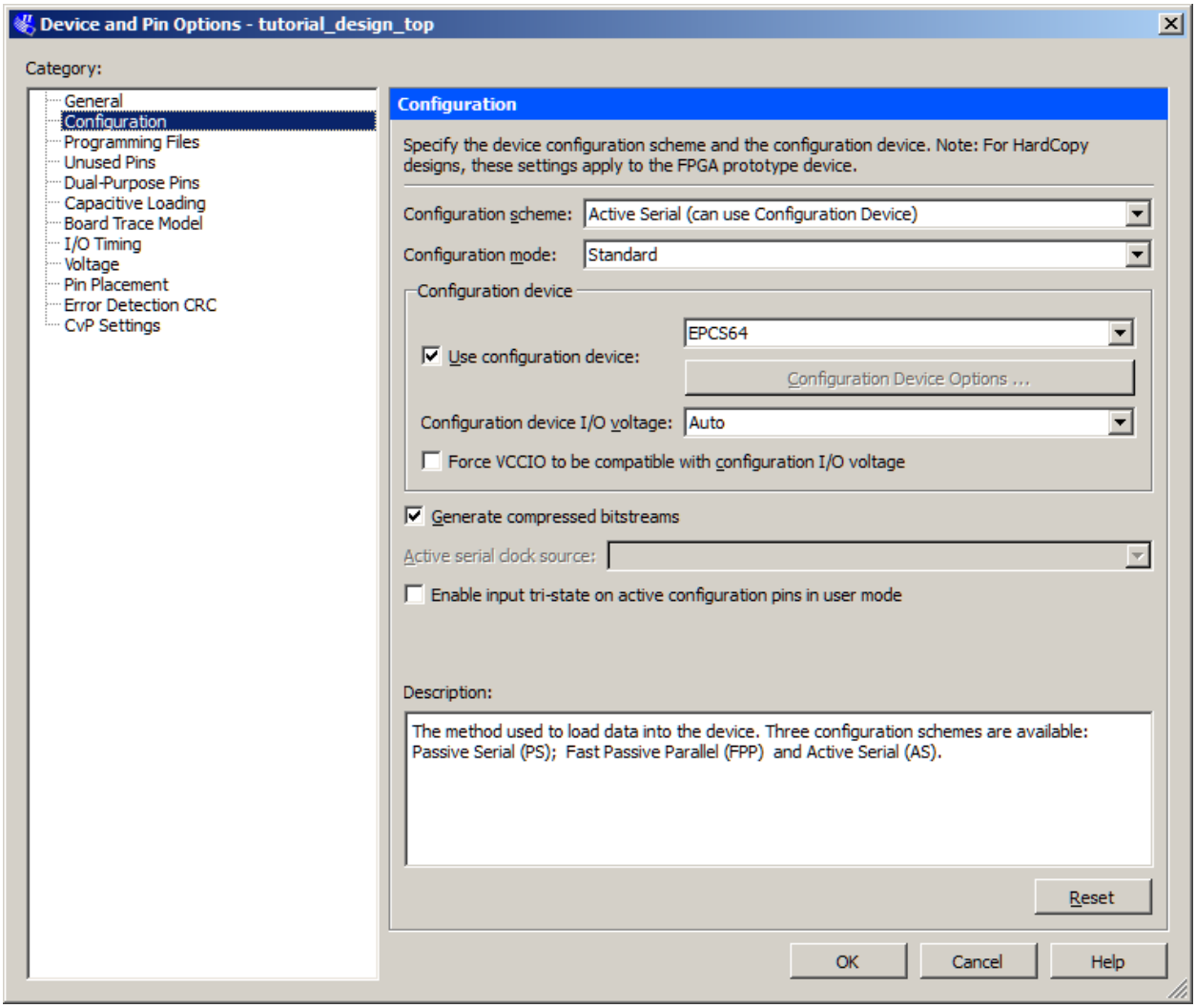


Figure 6-19: Select Configuration Device

Compile an Example Project and Build RPD and SVF Files

Click on **Processing** menu tab and choose **Start Compilation** to start the compilation process for the example project. After the process, has ran successfully, you should now see in Quartus II something similar to the figure below. The green check marks indicate success and the red X indicates failure. The process will succeed only when there is no error. There may or may not be any warning. If there is any warning, make sure that it is OK for the design before moving forward. For this tutorial design, ignore all warnings.

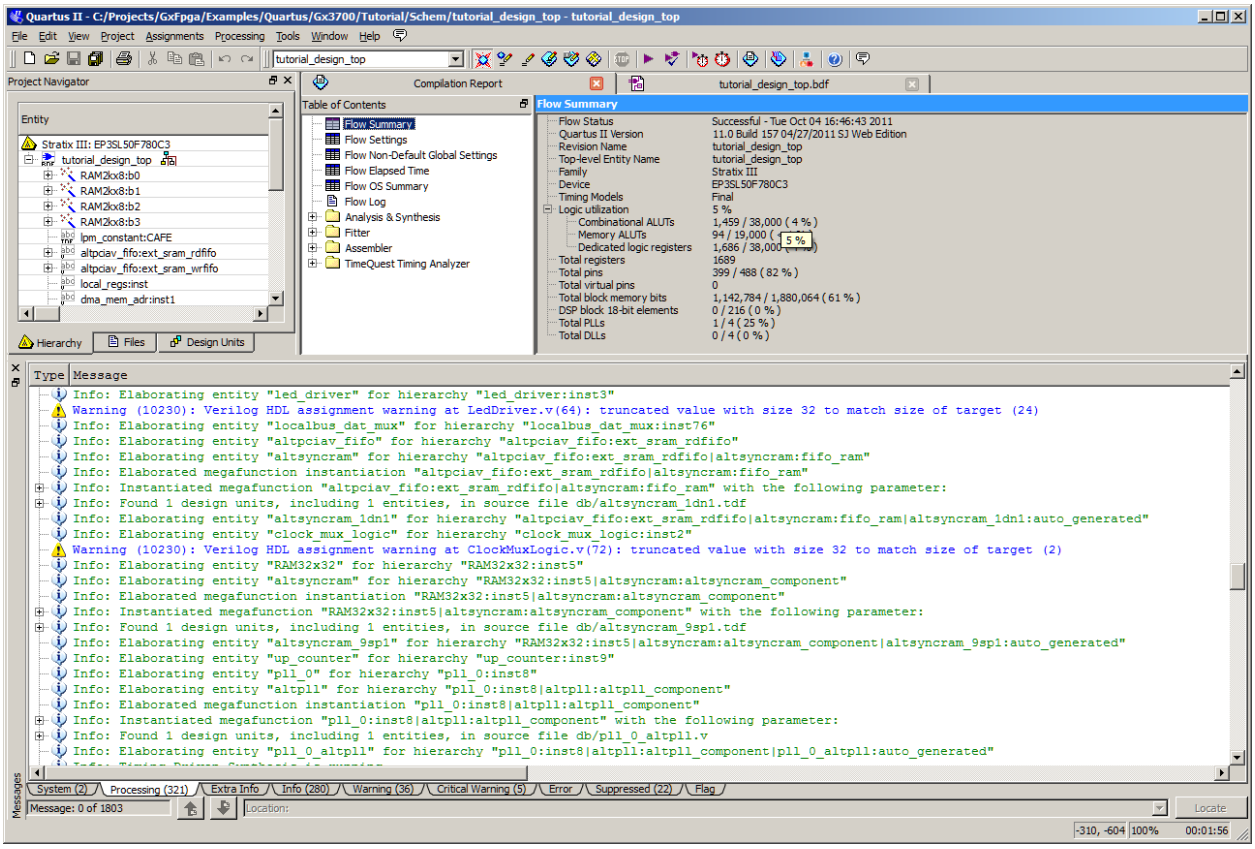


Figure 6-20: Compilation Tools and Status

The SVF file will be generated after the project compilation has finished. The **Compilation Task** window will show green check marks next to each major task to indicate completion.

In order to generate RPD file go to **File**, **Convert Programming Files …**

Select **Raw Programming Data File (.rpd)** as the **Programming file type** and **tutorial_design_top.rpd** as the **File Name**. Click on the **Add File** button and select **tutorial_design_top.pof**. The .pof file should now appear below the **POF Data** node as shown below. Finally, click the **Generate** button to create the RPD file.

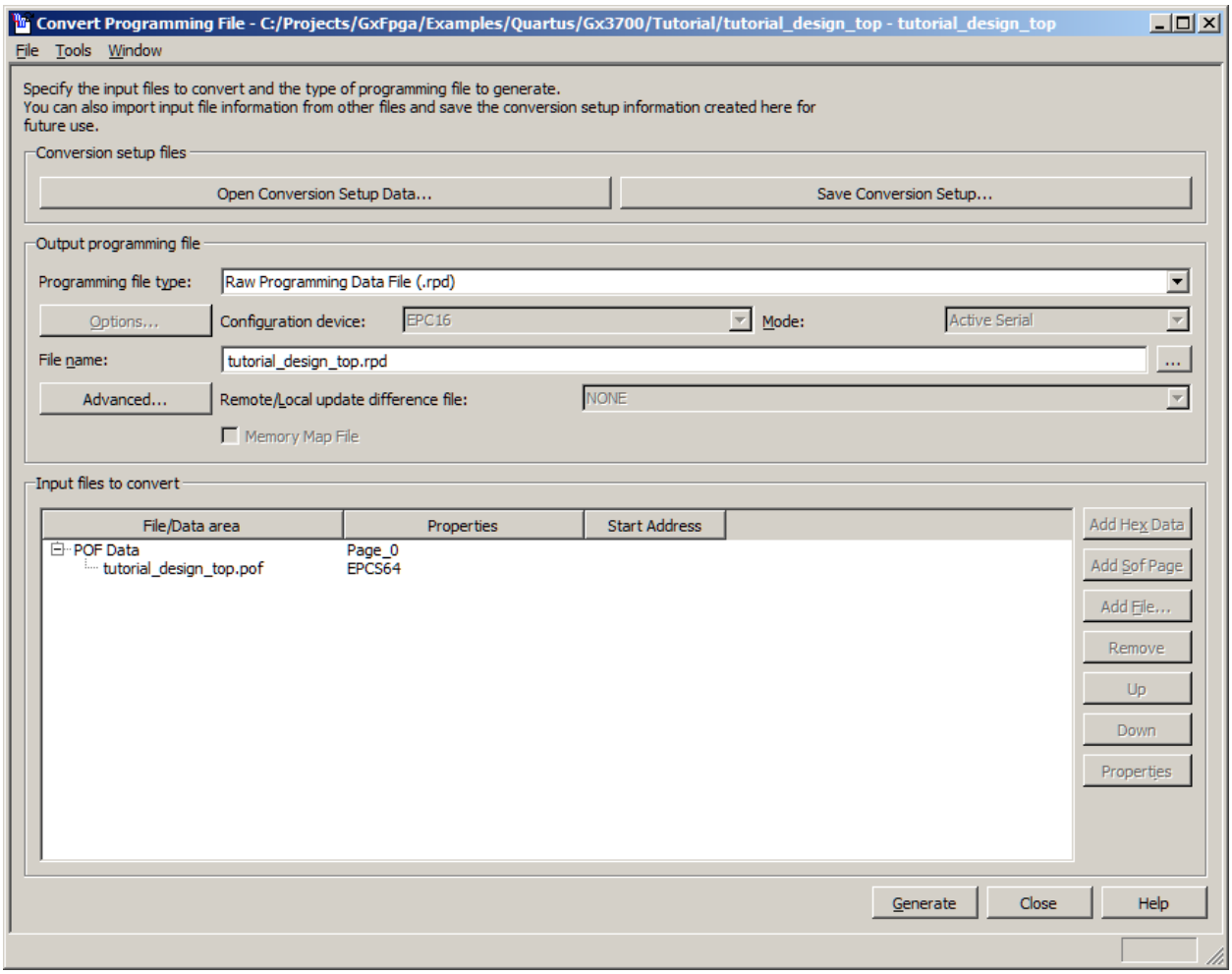


Figure 6-21: Convert Programming Files Dialog Box

Load Gx3700 with SVF File

Start the **GX3700 Panel** (from the Windows **Start** menu, **Marvin Test Solutions**, **GxFpga**) and initialize the instrument. Next, click on the **Volatile** radio box and then click on the Browse Button (…) to select the newly generated SVF file (**tutorial_design_top.svf**). Finally click on the **Load** button to begin programming the card. You will see the progress bar indicate the status of the load. Once the load has completed, the status bar should be unfilled.

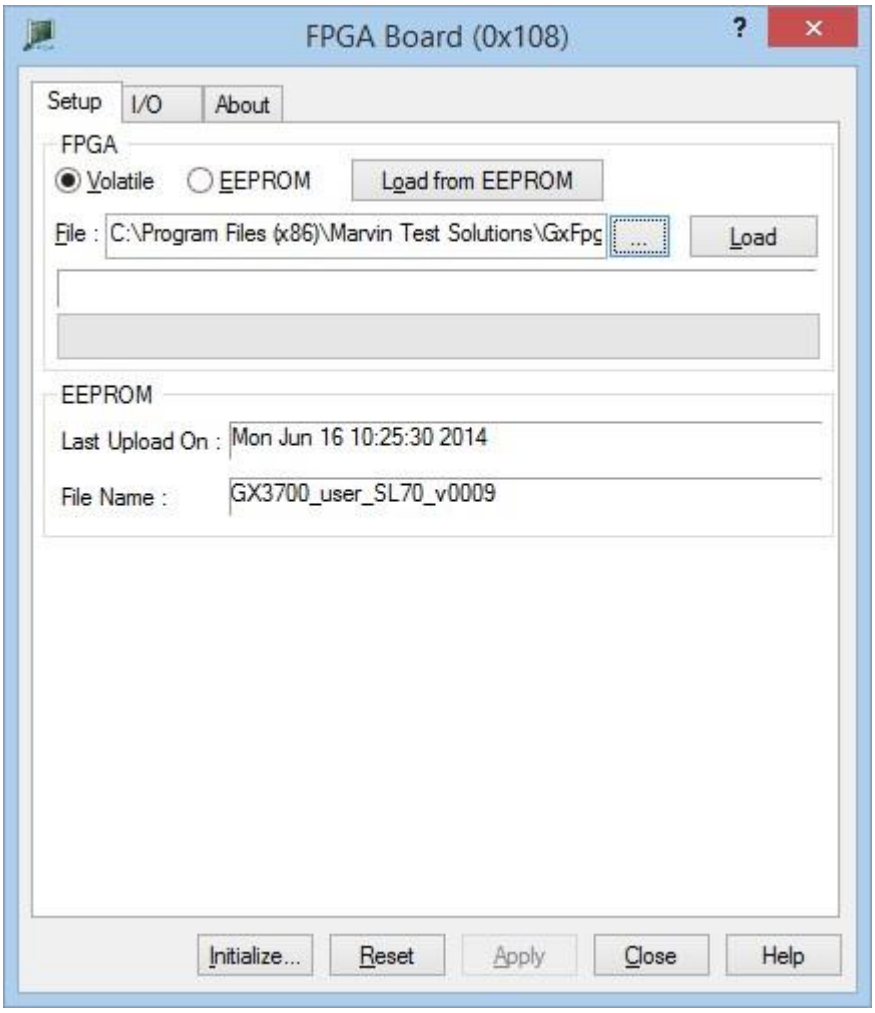


Figure 6-22: Software Front Panel

Testing the Design

Now that the design has been completed, compiled and loaded into the GX3700, we can move on to the testing. There are two ways to access the FPGA, either through the software front panel or through the driver API DLL. We will demonstrate the programming method using ATEasy to access the driver API DLL.

Adder Testing

The software front panel will be used to test Phase 1 of the design which adds two 32 bit numbers together. Click on the **I/O Tab** to get started. The Adder phase is controlled through the FPGA Register space. Offset 0x0 points to the first 32 bit number that will be summed and offset 0x4 points to the second 32 bit number that will be summed. Write values to both these locations. The sum can be obtained by reading the 32-bit value at offset 0x8. Verify that the correct sum is read back.

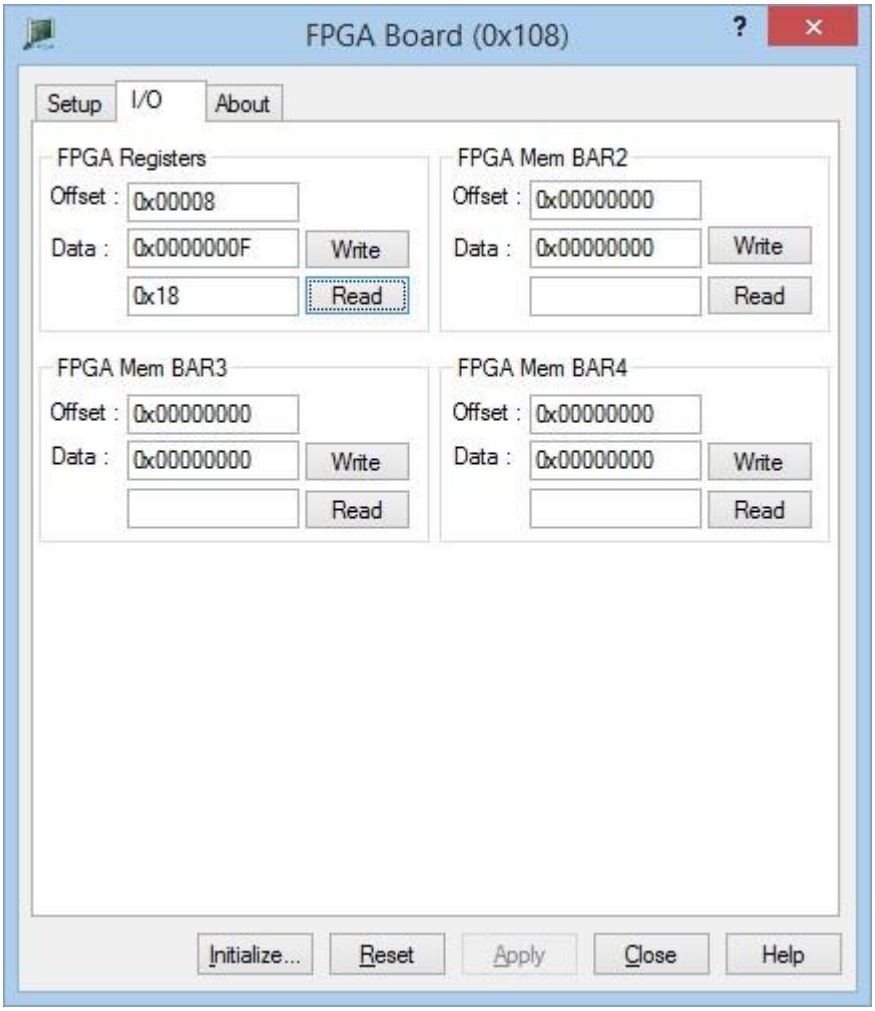


Figure 6-23: Using the Software Front Panel to read back the Sum

Clock Mux Testing

The software front panel will once again be used to test Phase 2 of the design. This part of the design uses a Mux to select between the PCI Clock and the 10 Mhz reference clock. The selected clock is output to I/O Channel 65. The Mux is controlled through the FPGA Register space.

Writing a 0x0 to offset 0xC will route the PCI/PCIe Clock signal to I/O Channel 65. Writing 0x1 to the same offset will route the 10 Mhz clock to this same channel. Try switching between both values while monitoring pin with an oscilloscope. You should see the appropriate clock signals.

Chapter 7 - GXFPGA VHDL Tutorial

Introduction

This tutorial will go over the basic workflow to start designing and loading a FPGA configuration for the Gx3700. The example provides creation of a project using VHDL sources and coding. The "Tutorial design top reg.doc" contains the design register map.

The tutorial contents will entail:

- Downloading and installing the FPGA design tool
- Creating a new FPGA Design project with the Stratix III as the target device
- Setup the pin assignment to work with the GX3700 and Stratix III FPGA
- Use the Quartus IDE to create an example FPGA configuration
- Compile the project and generate the SVF and RPD programming files
- Loading the board with the generated programming files
- Testing the design using the Gx3700 Front Panel software and ATEasy
- The example configuration is broken down into three phases, each with a distinct function:
    - **Phase 1:** Take two values located in PCI Registers and generate a Sum (Adder) which can then be read through a third PCI Register.
    - **Phase 2:** 2 to 1 multiplexer to choose between the 10 MHz Clock and the PCI Clock which will be output on one of the FlexIO pins. The clock will be selected through a PCI Register.
- The source code for the examples in this chapter is provided in the Examples\Quartus\Gx3700 folder.

Downloading Altera Design FPGA Design Tools

The Marvin Test Solutions Gx3700 User programmable FPGA board can be designed using the free Altera Quartus II Web Edition or Subscription Edition design tool. This FPGA design tool allows end users to generate fully featured FPGA designs that can be downloaded to the Gx3700 board using the Marvin Test Solutions GXFPGA software API or software front panel. Other 3rd party tools can also be used to design the FPGA. Before proceeding with this tutorial, you must have Altera Quartus II **v11.0 SP1** installed on your PC. More information about this tool and how to download it can be found at http://www.altera.com/products/software/quartus-ii/web-edition/qts-we-index.html.

Create New Project

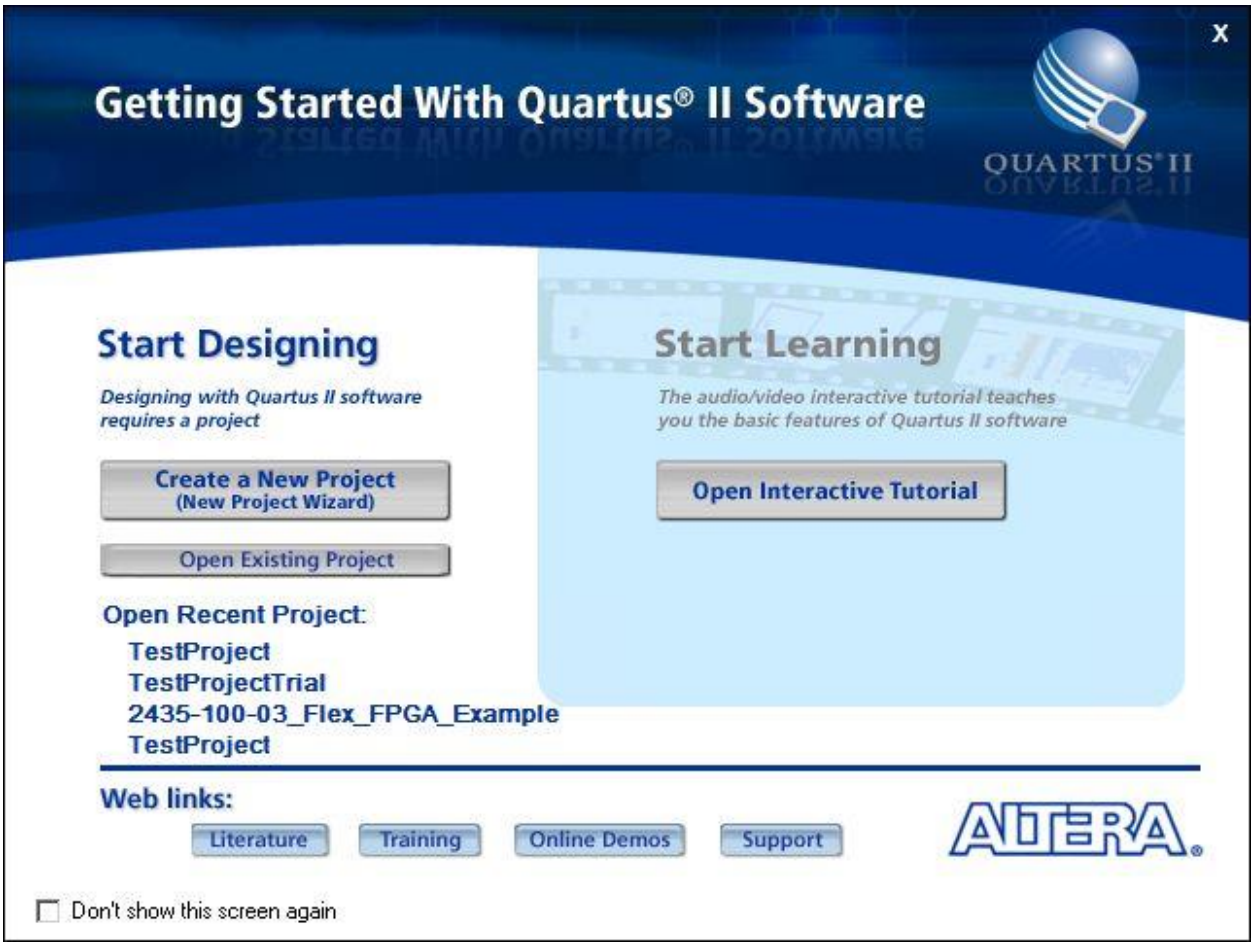


Figure 7-1: Quartus II Start Dialog

After installing Quartus II Web Edition, start the application and select C**reate a new Project** to start the New Project Wizard or select **File, New**, **New Quartus Project**.

Click on **Next** and then select the Project Folder and enter **tutorial_design_top** as the project name.

Click on **Next** twice (skip the adding files window).

Device Selection

The next window will allow you to select the FPGA target device. Select **Stratix III** as the Family and **EP3SL50F780C3** when using the GX3700 or **EP3SL70F780C3** for the GX3700e. For newer GXFPGA boards, the device ID will be displayed in the instrument front panel **About** page.

Click on **Next** twice (skip the Specify Tools window).

A window summarizing all the choices made for the creation of this project is shown. Click on **Finish**.

Pin Assignment Setup

You should now have an empty skeleton project loaded in Quartus II. Before you can get started on the FPGA design, you must assign the FPGA pins distinct names so that you can reference them in your design. This can be accomplished by running a **TCL script** which contains all the information necessary to configure the pin assignments as well as settings the project to either schematic entry or Verilog entry. These pin assignments are unique to this Stratix III FPGA and the GX3700 in particular. The following table lists all the pin assignments and their respective descriptions. The Pin Alias's listed in the table are the pin names you will be using in your design to reference the actual hardware pins on the FPGA.

Pin Assignments Table

Pin Alias (Node Name)	Description
Clocks	
10Mhz	Input. 10 MHz Reference Clock Signal from the PXI Backplane
PCIClock	Input. 33 MHz PCI Bus clock or 125MHz PCI Express application clock.
RefClk	Input. 80 Mhz Reference Clock onboard the GX3700
PCI Bus	
Addr[2..19]	Input. The PCI Address lines from the PCI bus
FDt[0..31]	Bidir. PCI Data lines from the PCI bus
CS[1..3]	Input. Chip Select lines from the PCI bus. CS[1] is for FPGA registers, CS[2] is for internal SRAM, CS[3] is currently not used.
LEXT	Input. External SRAM chip select. This is chip select for external SRAM on PCB.
RdEn	Input. PCI Read Enable line from the PCI bus
WrEn	Input. PCI Write Enable line from the PCI bus
LREAD_DV	Output. Read data valid. This is data valid for FDt(31:0) data bus.
LUW	Input. Currently not used. Upper Word.
LLW	Input. Currently not used. Lower Word.
LRESET	Input. Currently not used. Reset coming from PXI bridge FPGA.
PXI Bus	
PxiTrig[0..7]	Bidir. PXI Bus trigger signals
StarTrig	Output. PXI Star Trigger signal. This signal can be re-defined by the user as bi-directional.
PXI_LBL6	Bidir. PXI Local Bus Left 6. This is local bus according to PXIe spec.
PXI_LBR6	Bidir. PXI Local Bus Right 6. This is local bus according to PXIe spec.
PXIe_DSTARA	Input. PXIe DSTAR trigger A. This is DSTAR trigger according to PXIe spec.
PXIe_DSTARB	Input. PXIe DSTAR trigger B. This is DSTAR trigger according to PXIe spec.
PXIe_DSTARC	Output. PXIe DSTAR trigger C. This is DSTAR trigger according to PXIe spec.
PXIE_100M	Input. PXIe 100MHz clock. This is 100MHz clock according to PXIe spec.
PXIE_SYNC100	Input. PXIe Sync100. This is Sync100 signal according to PXIe spec.
I/O	
FlexIO[1..160]	Bidir. The physical IO Channels including 4 global clock inputs (2 differential pairs).

External Flash	
Fsm_a[1..23]	Output. Address bus shared by external SRAM and flash.
Fsd[0..31]	Bidir. Data bus shared by external SRAM and flash.
Flash_ce_n	Output. Flash chip enable.
Flash_oe_n	Output. Flash output enable.
Flash_we_n	Output. Flash write enable.
Flash_reset_n	Output. Flash chip reset
Flash_byte_n	Output. Flash byte/word select.
Flash_busy_n	Input. Flash busy
External SRAM	
Sram_be_n[0..3]	Output. External SRAM byte enable.
Sram_ce_n	Output. External SRAM chip select.
Sram_oe_n	Output. External SRAM output enable.
Sram_we_n	Output. External SRAM write enable.
RX DMA FIFO I/F	
RX_DMA_DAT[0..31]	Input. Receive DMA data coming from PC host.
RX_DMA_DV	Input. Receive DMA data valid.
RX_DMA_FIFOFULL	Output. Receive DMA FIFO full. This will throttle data from PC host.
RX_DMA_SP1	Output. Spare. Currently not used.
RX_DMA_SP2	Output. Spare. Currently not used.
TX DMA FIFO I/F	
TX_DMA_DAT[0..31]	Output. Transmit DMA data from memory going to PC host.
TX_DMA_DV	Output. Transmit DMA data valid.
TX_DMA_FIFOEMPTY	Output. Transmit DMA FIFO empty. When empty and is sending data to PC host, the DMA engine in PXI bridge FPGA will assert FIFO read enable TX_DMA_FIFO_RD.
TX_DMA_FIFO_RD	Input. Transmit DMA FIFO read enable.
Misc	
Spare[0..7]	Bidir. Do Not Use. Spares connected to PXI bridge FPGA.
IRQ	Output. Interrupt output pin going to PXI bridge FPGA IRQ = 1 means interrupt will be generated to PC host. IRQ = 0 means no interrupt.
FSpr[0..3]	Bidir. Spare Signals connected to Expansion Board
MClr	Input. FPGA Master Clear, Active High
TP[0..5]	Bidir. Connected to test header J7 on the GX3700 PCB
ACTIVE_LED_N	Output. Active LED. Connect to LD1 LED on board. '0' = LED on, '1' = LED off.

Table 7-1: Pin Assignments Table

Schematic entry project

In order to configure the project as schematic entry and configure the pin assignment the TCL configuration script should be added to the project. To add the script to the project, click on **Project | Add/Remove Files in Project…** In the dialog box, click on the **…** button and browse for GX3700VHDL.tcl file in the "C:\Program Files\Marvin Test Solutions\GxFpga\" folder. On some systems, you may need to Click Open and then the **Add** button.

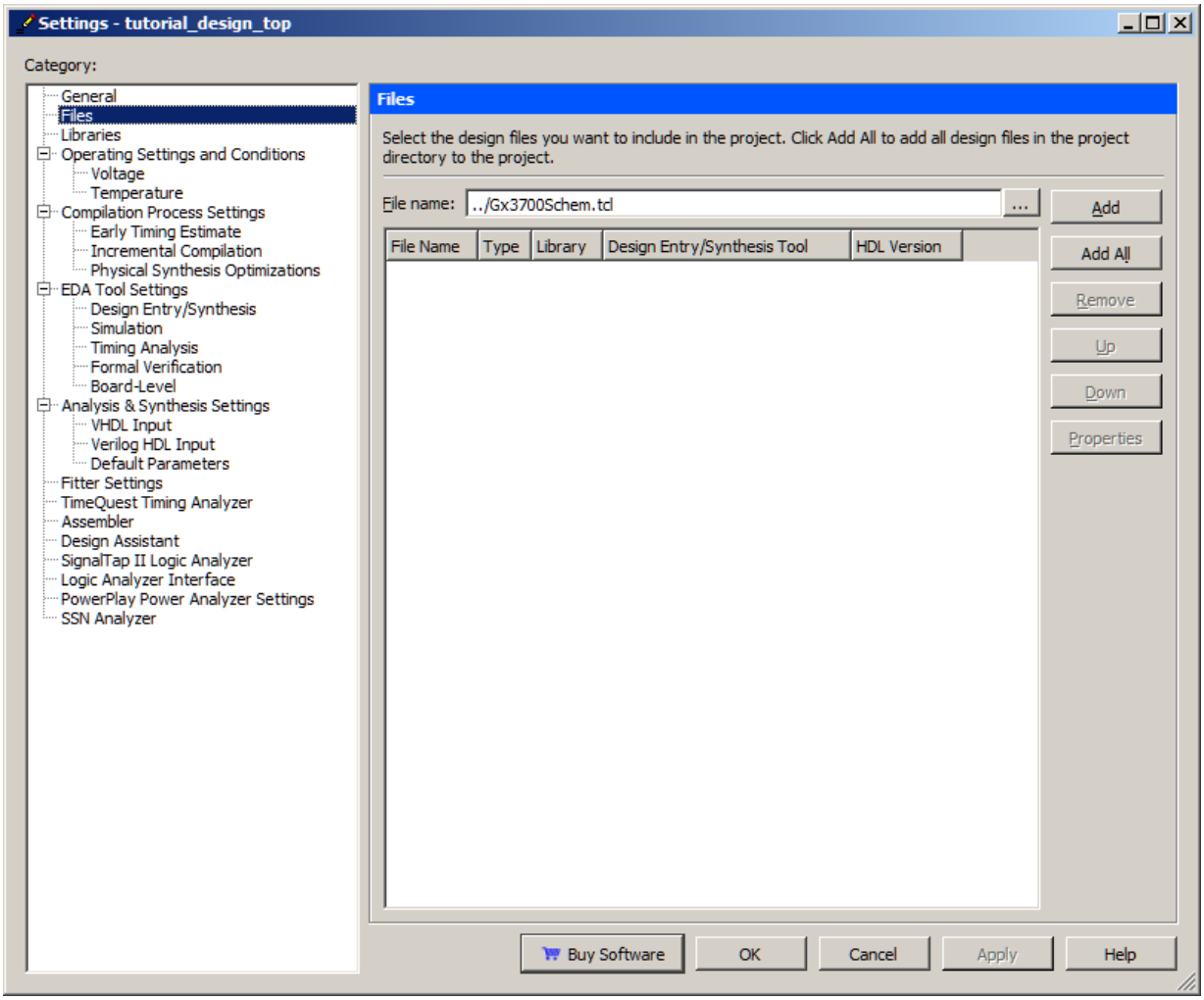


Figure 7-2: Add Tcl Script to Project

Then click on Tools | TCL Scripts … Select the configuration script file, **GX3700VHDL.tcl** and click on **Run**. This will configure your FPGA pin assignments.

Note: The TCL file will automatically add all the source files needed for the tutorial design to the Quartus II project.

You can view the pin assignments by running the Pin Planner application which is found in the Tasks list as highlighted below:

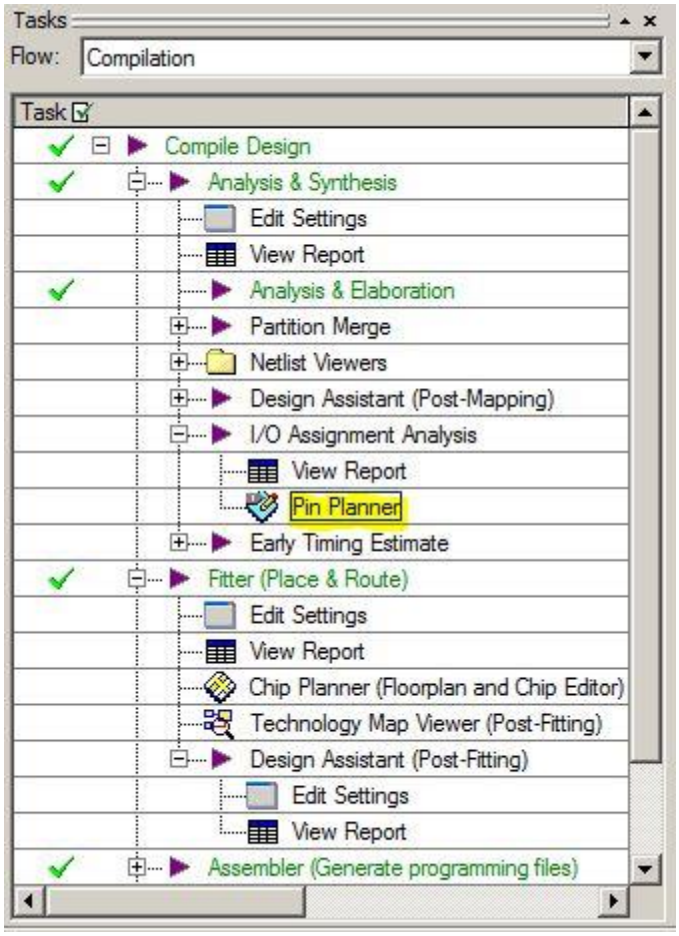


Figure 7-3: Task Flow

The Pin Planner will display a matrix of the physical FPGA pins and their mapped names as well as the I/O standard supported by the pin. These mapped names are used in the FPGA design, as wire names and I/O pins, to connect to the physical connections of the FPGA.

Creating Design File with VHDL

This section will walk you through the steps of creating modeled components in several modules.

Note: There is more than one way to accomplish the following designs.

Phase 1: Creating the FPGA design - 32 bit Full Adder

This design will take two double word (32 bit) values, located in the first two double words in the Register space (byte offset 0x0 and 0x4), and add them together. The sum of the two values will be immediately output to the third double word in the Register space (byte offset 0x8). The sources for all referenced components are installed with the GXFPGA software package to C:\Program Files\Marvin Test Solutions\GxFpga\Examples\Quartus\Gx3700\Tutorial_VHDL\source

Components Used

- **d_flipflop_1.vhd** – A 1-bit D flip-flop
- **d_flipflop_n.vhd** - A n-bit D flip-flop
- **decoder.vhd** – A 5 to 32 decoder (structural)
- **or_gate2.vhd** – A two input or gate
- **or_gate4.vhd** - A four input or gate
- **adder.vhd** – An n-bit full adder
- **and_gate_1.vhd** – A two input 1-bit and gate
- **and_gate_n.vhd** – A two input variable-width and gate

Top-level VHDL file

In order to open the VHDL text editor, click on **File** menu, and then **New** the following dialog appears.

Select VHDL File:

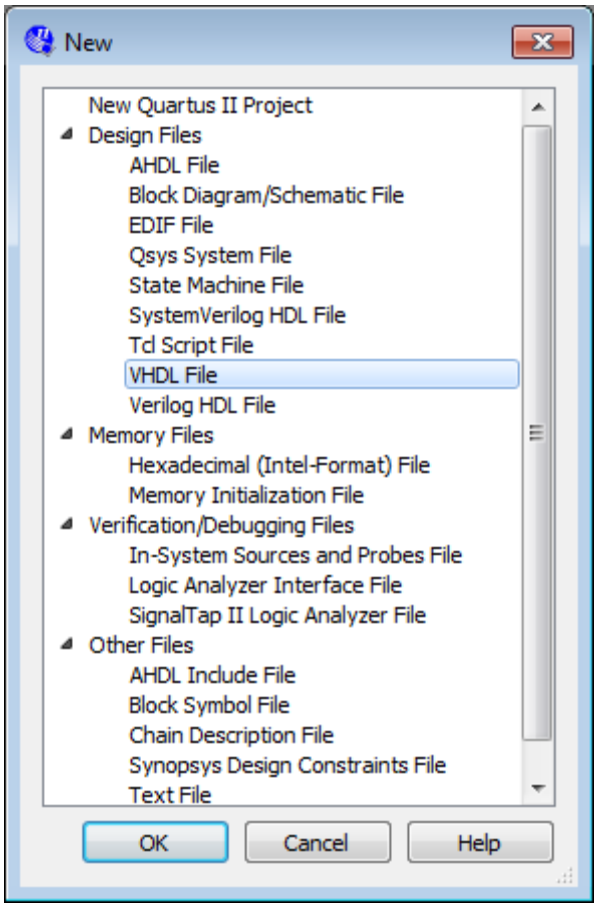


Figure 7-4: New File Dialog Box

Top-level inputs and outputs

The top-level object for this project will be named **tutorial_design_top.vhd**. Start by creating module prototype with the proper inputs and outputs. The inputs and outputs all correspond to pin on the FPGA.

```
--------------------------------------------------------------------
-- Design Name : GXFPGA VHDL Tutorial
-- Function    : Demonstrates functionality described in the
--           VHDL Tutorial chapter of the GXFPGA User's Guide.
--------------------------------------------------------------------

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY tutorial_design_top IS
    PORT (Addr          : IN    STD_LOGIC_VECTOR(6 downto 2);
        CS              : IN    STD_LOGIC_VECTOR(2 downto 1);
        WrEn, RdEn      : IN    STD_LOGIC;
        PCIClock, PXI10Mhz     : IN       STD_LOGIC;
        FlexIO          : OUT   STD_LOGIC_VECTOR(65 downto 33);
        LREAD_DV        : OUT   STD_LOGIC;
        FDt             : INOUT STD_LOGIC_VECTOR(31 downto 0));
END tutorial_design_top;
```

Figure 7-5: GXFPGA VHDL Tutorial Prototype

The first step is creating the circuitry required to decode the PCI Address when data is to be written from the PC to the FPGA. This circuit will be used in all three functions of this example project. The signals required for PCI Write access will be the **PCI Clock**, **Write Enable**, **Chip Select 1**, and some **PCI Address lines**. The PCI Address lines 5 to 2 will be fed to a decoder which will generate a 32-bit value, and the result will be ANDed with the Chip Select 1 bit. Each Chip Select bit represents a certain PCI BAR access (GX3700 has two bars, memory and register memories). Bit 1 represents BAR1 of the PCI memory space (bit 2 for BAR2). BAR1 is the general-purpose Control Register BAR for the GX3700. The results of the AND operation will be once again ANDed to the Write Enable PCI signal.

To create the address decoder, we'll need to model the D Flip-flop (to latch the inputs), the AND gate, and the decoder. For each module that we add, you should use the New File Dialog to add a Verilog HDL file to create the blank file. When saving, give the file the same name as the module. The source for the referenced modules follows:

```
----------------------------------------------------
-- Design Name : and_gate_n
-- Function    : A two input and gate, the first input in n-bit width
----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY and_gate_n IS
GENERIC (n_width : NATURAL := 32);
    PORT (Input1       : IN  STD_LOGIC_VECTOR (n_width-1 downto 0);
          Input2       : IN  STD_LOGIC;
        Output         : OUT STD_LOGIC_VECTOR (n_width-1 downto 0));
END and_gate_n;

ARCHITECTURE Behavior OF and_gate_n IS
BEGIN
    PROCESS (Input1, Input2) BEGIN
        IF Input2 = '1' THEN
                Output <= Input1;
        ELSE
                FOR i IN 0 TO n_width-1 LOOP
                        Output(i) <= '0';
                END LOOP;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 7-6: and_gate_n.vhd source

```
-----------------------------------------------------
-- Design Name : d_flipflop_1
-- Function    : A 1-bit D flip-flop
-----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_flipflop_1 IS
  PORT (D       : IN  STD_LOGIC;
    Clock       : IN  STD_LOGIC;
    Enable      : IN  STD_LOGIC;
    Clearn      : IN  STD_LOGIC;
    Q           : OUT STD_LOGIC);
END d_flipflop_1;

ARCHITECTURE Behavior OF d_flipflop_1 IS
BEGIN
    PROCESS(Clock)
    BEGIN
IF (rising_edge(Clock)) THEN
               IF (Clearn = '1') THEN
                       Q <= '0';
               ELSE
                       IF (Enable = '1') THEN
                               Q <= D;
                       END IF;
               END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 7-7: d_flipflop_1.vhd source

```
-------------------------------------------------------
-- Design Name : d_flipflop_n
-- Function    : A n-bit D flip-flop
-------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY d_flipflop_n IS
    GENERIC (n_width   : INTEGER RANGE 2 TO 32 := 32 );
    PORT (D            : IN  STD_LOGIC_VECTOR (n_width-1 downto 0);
             Clock     : IN  STD_LOGIC;
        Enable         : IN  STD_LOGIC;
        Clearn         : IN  STD_LOGIC;
        Q              : OUT STD_LOGIC_VECTOR (n_width-1 downto 0));
END d_flipflop_n;

ARCHITECTURE Behavior OF d_flipflop_n IS
BEGIN
    PROCESS(Clock)
    BEGIN
        if (rising_edge(Clock)) then
                if (Clearn = '1') then
                               A: FOR i IN 0 TO n_width-1 loop
                               Q(i) <= '0';
                               END LOOP;
                ELSE
                       IF (Enable = '1') THEN
                               Q <= D;
                       END IF;
                END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 7-8: d_flipflop_n.vhd source

```
----------------------------------------------------
-- Design Name : decoder
-- Function    : An 5 to 32 decoder (non-behavioral)
----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

entity decoder is
   port(Decoder_In     : IN    STD_LOGIC_VECTOR(4 downto 0);
        Decoder_Out    : OUT STD_LOGIC_VECTOR(31 downto 0));
end decoder;

ARCHITECTURE Behavior OF decoder IS
BEGIN
Decoder_Out <="00000000000000000000000000000001" when Decoder_In="00000" else
    "00000000000000000000000000000010" when Decoder_In="00001" else
    "00000000000000000000000000000100" when Decoder_In="00010" else
    "00000000000000000000000000001000" when Decoder_In="00011" else
    "00000000000000000000000000010000" when Decoder_In="00100" else

This entity was abbreviated due to its repetitive nature.

    "00001000000000000000000000000000" when Decoder_In="11011" else
    "00010000000000000000000000000000" when Decoder_In="11100" else
    "00100000000000000000000000000000" when Decoder_In="11101" else
    "01000000000000000000000000000000" when Decoder_In="11110" else
    "10000000000000000000000000000000" when Decoder_In="11111";
END Behavior;
```

Figure 7-9: decoder.vhd source

```
----------------------------------------------------
-- Design Name : and_gate_1
-- Function    : A two input and gate
----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY and_gate_1 IS
    PORT (Input1   : IN  STD_LOGIC;
          Input2   : IN  STD_LOGIC;
          Output   : OUT STD_LOGIC);
END and_gate_1;

ARCHITECTURE Behavior OF and_gate_1 IS
BEGIN
    PROCESS (Input1, Input2)
    BEGIN
        IF Input2 = '1' THEN
                Output <= Input1;
        ELSE
                Output <= '0';
        END IF;
    END PROCESS;
END Behavior;
```

Figure 7-10: and_gate_1.vhd source

```
----------------------------------------------------
-- Design Name : adder
-- Function    : An n-bit full adder
----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY adder IS
    GENERIC(n_width           : NATURAL := 32);
      PORT (DataA, DataB      : IN  STD_LOGIC_VECTOR(n_width-1 downto 0);
        Cin                   : IN  STD_LOGIC;
        Result                : OUT STD_LOGIC_VECTOR(n_width-1 downto 0);
        Cout                  : OUT STD_LOGIC);
END adder;

ARCHITECTURE Behavior OF adder IS
BEGIN
    adder: PROCESS (DataA, DataB, Cin)
        variable carry : STD_LOGIC;
        variable isum  : STD_LOGIC_VECTOR(n_width-1 downto 0);
    BEGIN
        carry := Cin;
        for i in 0 to n_width-1 loop
                isum(i) := DataA(i) xor DataB(i) xor carry;
carry   := (DataA(i) and DataB(i)) or (DataA(i) and carry) or (DataB(i) and carry);
        end loop;
        Result <= isum;
      Cout     <= carry;
END PROCESS adder;
END Behavior;
```

Figure 7-11: adder.vhd source

```
----------------------------------------------------
-- Design Name : or_gate2
-- Function    : A two input or gate
----------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY or_gate2 IS
   PORT (Input1   : IN  STD_LOGIC;
       Input2     : IN  STD_LOGIC;
       Output     : OUT STD_LOGIC);
END or_gate2;

ARCHITECTURE Behavior OF or_gate2 IS
BEGIN
    Output <= Input1 or Input2;
END Behavior;
```

Figure 7-12: or_gate2.vhd source

```
-------------------------------------------------------
-- Design Name : or_gate4
-- Function    : A four input or gate
-------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY or_gate4 IS
    GENERIC (n_width : NATURAL := 32);
       PORT (Input1 : IN  STD_LOGIC_VECTOR (n_width-1 downto 0);
        Input2 : IN  STD_LOGIC_VECTOR (n_width-1 downto 0);
        Input3 : IN  STD_LOGIC_VECTOR (n_width-1 downto 0);
        Input4 : IN  STD_LOGIC_VECTOR (n_width-1 downto 0);
        Output : OUT STD_LOGIC_VECTOR (n_width-1 downto 0));
END or_gate4;

ARCHITECTURE Behavior OF or_gate4 IS
BEGIN
    Output <= Input1 or Input2 or Input3 or Input4;
END Behavior;
```

Figure 7-13: or_gate4.vhd source

In tutorial_design_top.v, we will now write the code to describe our PCI Address Decoder Circuit. Latch both the Address and Write Enable lines using the PCI Clock. Decode the 5 bit Address lines into a 32-bit bus named DecodedAddr. This decoded bus is ANDed with the FPGA's CS[1] to define our PCI Address Decoded Select lines.

Additionally, we will define our Write Enable (WE) lines in this code block. We will use this later, along with Read Enable, to read and write to registers.

```
-- PCI Address Decoder Circuit
inst: decoder          PORT MAP (LatchedAddr, DecodedAddr);
inst2: and_gate_n      GENERIC MAP (32) PORT MAP (DecodedAddr, CS(1), Sel);
inst3: and_gate_n      GENERIC MAP (32) PORT MAP (Sel, LatchedWrEn, WE);
inst23: d_flipflop_1   PORT MAP (WrEn, PCIClock, NC_Ena, NC_Rst, LatchedWrEn);
inst24: d_flipflop_n    GENERIC MAP (5) PORT MAP (Addr, PCIClock, NC_Ena, NC_Rst, LatchedAddr);
```

Figure 7-14: PCI Address Decoder Circuit

You will notice that we used a few undefined symbols in this last section: **nc_ena** and **nc_rst**. These are placeholders for enable and reset lines that our various components can take advantage of. For this tutorial, I have chosen not to use enable or reset lines at all so we should add the following code to tutorial_design_top.v to explicit set these wires to always enabled, never reset.

```
SIGNAL NC_Cin, NC_Rst : STD_LOGIC := '0';
SIGNAL Res1, NC_Ena : STD_LOGIC := '1';
```

Now that the PCI address decoder circuit is complete, we can feed the appropriate bits from the WE bus to D Flip Flops that will store data clocked in from the PCI data lines. For example, the first double word in PCI memory (representing the first number to be summed) will be written to a D Flip Flop with its enable line tied to WE[0] (the first bit in the WE bus). The second double word to be added will be written to another D Flip Flop with its enable line tied to WE[1]. Finally, the PCI Clock signal (33Mhz) will be used as the clock source of the D Flip Flops. Note that each bit of the Sel and WE buses represent a consecutive double word address (bit 0 corresponds with byte 0, bit 1 corresponds with byte 4, bit 2 corresponds with byte 8 etc.)

First we start by creating an extend circuit to deal with any timing issues with the WE signal. Then we will create some Flip Flops to latch inputs to the adders. We will use a placeholder named **LatchedFDt** as the input to the D Flip Flops. Eventually the PCI data lines will drive these inputs. Wire the outputs of the D Flip Flops to the Adder component. The output of the adder, **Sum**, will be used as an output later.

```
-- WE extend circuit - Extend write enable to mitigate timing issues
inst26: d_flipflop_n    GENERIC MAP (32) PORT MAP (WE, PCIClock, NC_Ena, NC_Rst, LatchedWE);
inst27: d_flipflop_n    GENERIC MAP (32) PORT MAP (LatchedWE, PCIClock, NC_Ena, NC_Rst,
    LatchedWE2);
inst28: d_flipflop_n    GENERIC MAP (32) PORT MAP (LatchedWE2, PCIClock, NC_Ena, NC_Rst,
    LatchedWE3);
inst30: or_gate4        GENERIC MAP (32) PORT MAP (LatchedWE, LatchedWE2, LatchedWE3, WE,
    WE_EXT);

-- Adder circuit - Latch the addends and include adder
inst4: d_flipflop_n     GENERIC MAP (32) PORT MAP (FDt_LoopBack, PCIClock, WE_EXT(0), NC_Rst,
    AdderA);
inst5: d_flipflop_n     GENERIC MAP (32) PORT MAP (FDt_LoopBack, PCIClock, WE_EXT(1), NC_Rst,
    AdderB);
inst7: adder            GENERIC MAP (32) PORT MAP (AdderA, AdderB, NC_Cin, AdderBuff, NC_Cout);
```

Figure 7-15: WE Extend Circuit and Adder Circuit

Before moving on we must first extend the RdEn signal. Add the following to the tutorial_design_top.v:

```
-- RdEn to 2 PCI Circuit
inst1: or_gate2         PORT MAP (RdEn, LatchedRdEn, RdEn_Extend);
inst8: d_flipflop_1     PORT MAP (RdEn, PCIClock, NC_Ena, NC_Rst, LatchedRdEn);
inst12: and_gate_n      GENERIC MAP (32) PORT MAP (Sel, RdEn_Extend, RE);
inst21: d_flipflop_1    PORT MAP (LatchedRdEn, PCIClock, NC_Ena, NC_Rst, LREAD_DV);
```

Figure 7-16: RdEn to 2 PCI Circuit

```
-- RE extend circuit - Extend read enable to mitigate timing issues
inst18: d_flipflop_n    GENERIC MAP (32) PORT MAP (RE, PCIClock, NC_Ena, NC_Rst, LatchedRE);
inst19: d_flipflop_n    GENERIC MAP (32) PORT MAP (LatchedRE, PCIClock, NC_Ena, NC_Rst,
    LatchedRE2);
inst20: d_flipflop_n    GENERIC MAP (32) PORT MAP (LatchedRE2, PCIClock, NC_Ena, NC_Rst,
    LatchedRE3);
inst22: or_gate4        GENERIC MAP (32) PORT MAP (LatchedRE, LatchedRE2, LatchedRE3, RE,
    RE_EXT);
```

Figure 7-17: RE Extend Circuit

We also create a Read Data Valid output pin, **LREAD_DV**. This comes from a D-Flipflop with the PCIClock as an input clock and the RdEn as the input data. The D-Flip Flop also creates our extender for our ReadEnable.

The inputs to the D Flips Flops can now be wired to the PCI data lines (FDt). We need to clean up the FDt signal as is comes back into our circuit by adding the D-FlipFlop.

```
-- Tri-state FDt when not reading registers
FDt <= FDt_out_value when RE_EXT /= X"00000000" else (others => 'Z');

process (PCIClock, RE_EXT, AdderA, AdderB, AdderBuff, LPM_CONSTANT, FDt)
begin
    if (RE_EXT(2)='1')
        then FDt_out_value <= AdderBuff;
    elsif (RE_EXT(0)='1')
        then FDt_out_value <= AdderA;
    elsif (RE_EXT(1)='1')
        then FDt_out_value <= AdderB;
    elsif (RE_EXT(31)='1')
        then FDt_out_value <= LPM_CONSTANT;
    end if;
    FDt_in_value <= FDt; --store the input value
end process;
```

Figure 7-18: FDt in/out signal assignment

Now that the design has been completed, a revision number should be added so that the end user can read it back from the PCI bus at the 32nd register double word location (byte address 0x7C).

Including a revision number constant to the design is a Marvin Test Solutions standard practice that we recommend end users to follow. The revision constant is 32 bits long and is read as a hexadecimal number such as 0x3564A000. The first two digits of the hexadecimal number represent the company, in this case 35 is for Marvin Test Solutions designs. The next two digits are the design specific code, 64 in this case. And the last 4 digits, A000, is the revision of the design.

Add the following to tutorial_design_top.vhd in the section where signals are defined:

```
// Add revision constant
SIGNAL LPM_CONSTANT : STD_LOGIC_VECTOR(31 downto 0) := X"3564A000";
```

Figure 7-19: Symbol Properties

Phase 2: Creating the FPGA Design - 2 to 1 Clock Mux

This design will output either the PCI Clock (33Mhz) or the 10Mhz clock to FlexIO Channel 65 (Check connectors tables for the correct pin location) depending on what was written to the 4th double word in the PCI register space (byte offset 0xC). A 1 will select the 10Mhz clock signal, and a 0 will select the PCI clock signal.

Design

You will now build upon the tutorial project to add the functionality of a 2 to 1 Clock Mux. The 10Mhz clock will be brought into the design by an input pin. The PCI Clock signal input pin is already present in the Phase 1 circuit, so this will be reused. FlexIO[65] (IO Channel 65) will be used to output the selected clock to the outside world.

```
-- Clock Mux Circuit
process (WE_EXT(3))
begin
    if (rising_edge(WE_EXT(3)))
            then LatchedFDt0 <= FDt(0);
    end if;
end process;

FlexIO(65) <= PCIClock when LatchedFDt0='0' else PXI10Mhz;
```

Figure 7-20: Clock Mux Circuit

FDt[0] is the first bit of the PCI data bus. This bit can either be 0 or 1, to indicate which clock source to choose. WE_EXT[3] is the 4th bit from the decoded PCI Address. When this bit is high, it indicates that the PCI Bus is addressing the 4th double word (byte offset 0xC) of the Register space for the GX3700. In our case, the value of this double word is used to select which clock is selected by our Mux.

At this point the design is complete, continue with the next sections to generate SVF or RPD files and load your design to the GX3700.

Configure Project to Output SVF and RPD Files

To ensure that a SVF file is generated upon project compilation, go to the **Assignments**, **Device ...** and click on the **Device and Pin Options** button. Then click on the **Programming Files** and verify that the **Serial Vector Format File** checkbox has been selected.

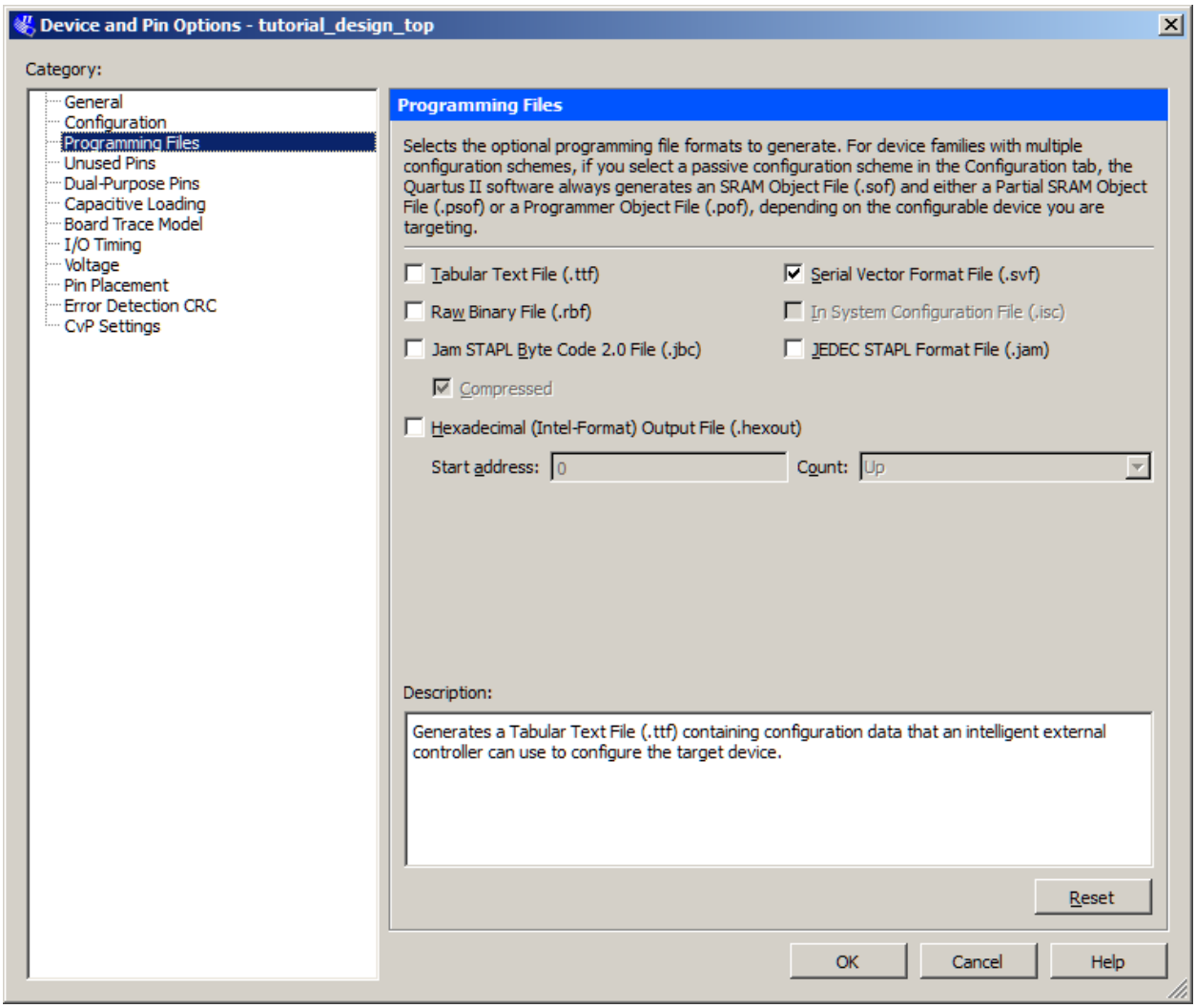


Figure 7-21: Select SVF as output file

Now click on the **Configuration** choose **Active Serial Configuration Scheme**, check **Use Configuration Device** checkbox and select **EPCS64** as the configuration device from the drop down selection. Finally click on **OK** twice to exit the settings dialog boxes.

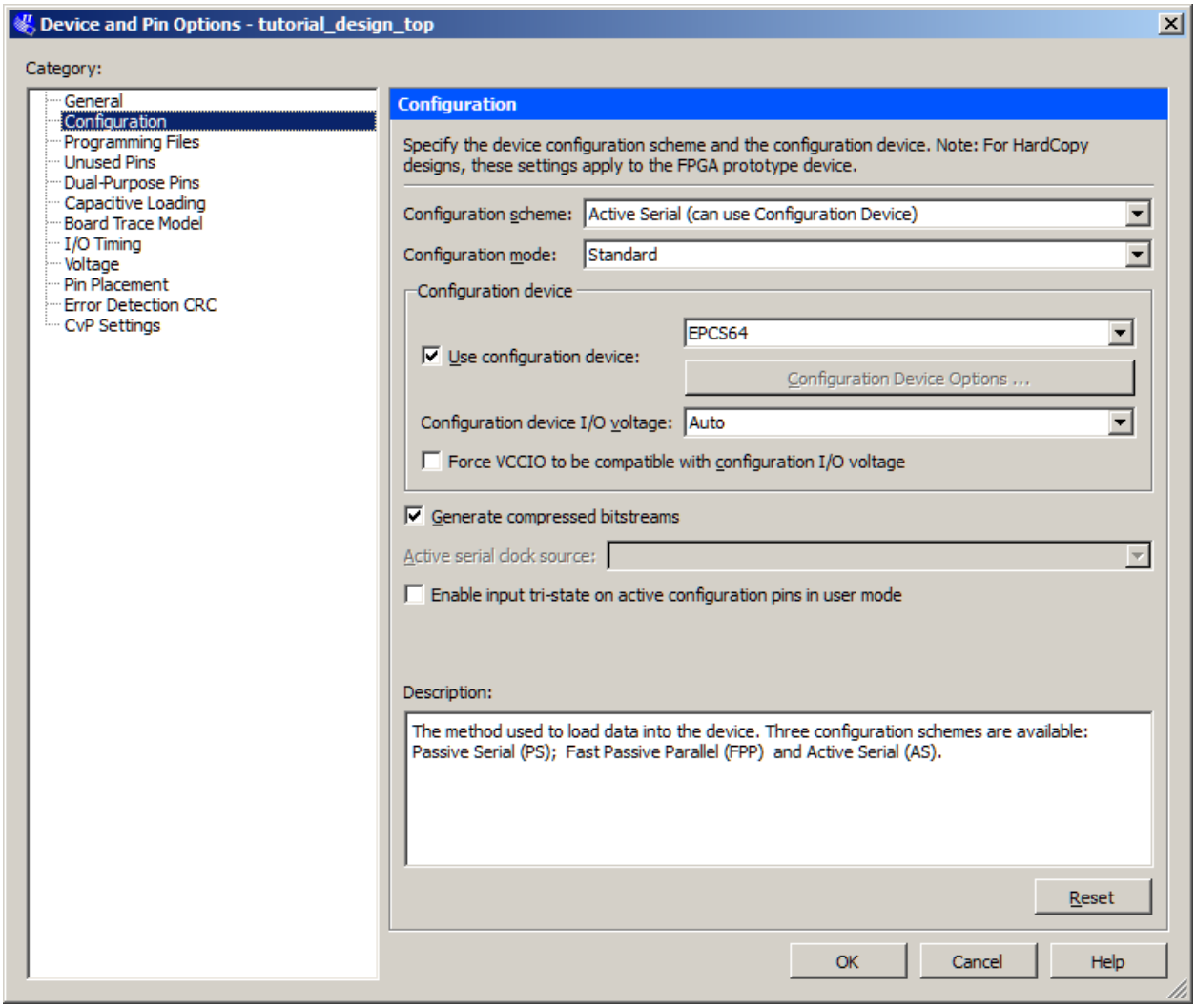


Figure 7-22: Select Configuration Device

Compile an Example Project and Build RPD and SVF Files

Click on **Processing** menu tab and choose **Start Compilation** to start the compilation process for the example project. After the process, has ran successfully, you should now see in Quartus II something similar to the figure below. The green check marks indicate success and the red X indicates failure. The process will succeed only when there is no error. There may or may not be any warning. If there is any warning, make sure that it is OK for the design before moving forward. For this tutorial design, ignore all warnings.

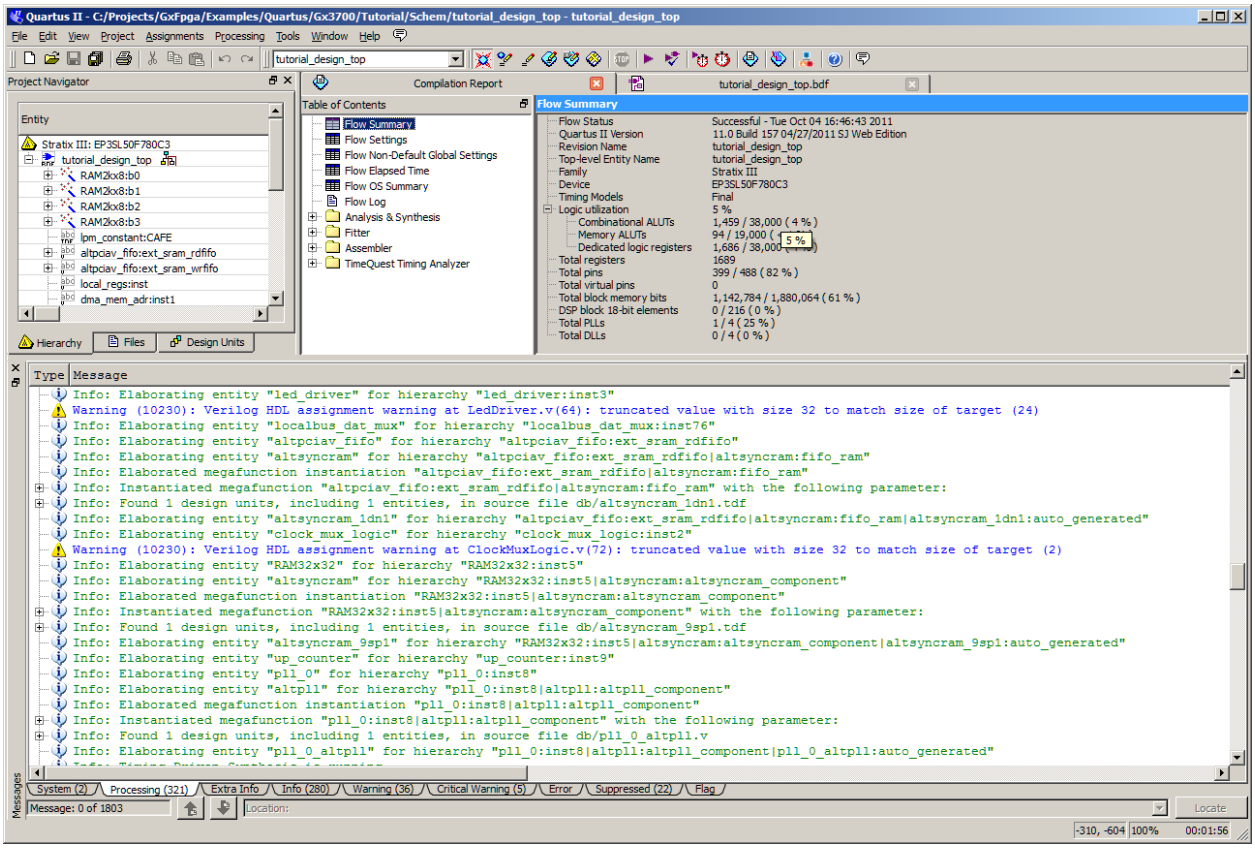


Figure 7-23: Compilation Tools and Status

The SVF file will be generated after the project compilation has finished. The **Compilation Task** window will show green check marks next to each major task to indicate completion.

In order to generate RPD file go to **File**, **Convert Programming Files …**

Select **Raw Programming Data File (.rpd)** as the **Programming file type** and **tutorial_design_top.rpd** as the **File Name**. Click on the **Add File** button and select **tutorial_design_top.pof**. The .pof file should now appear below the **POF Data** node as shown below. Finally, click the **Generate** button to create the RPD file.

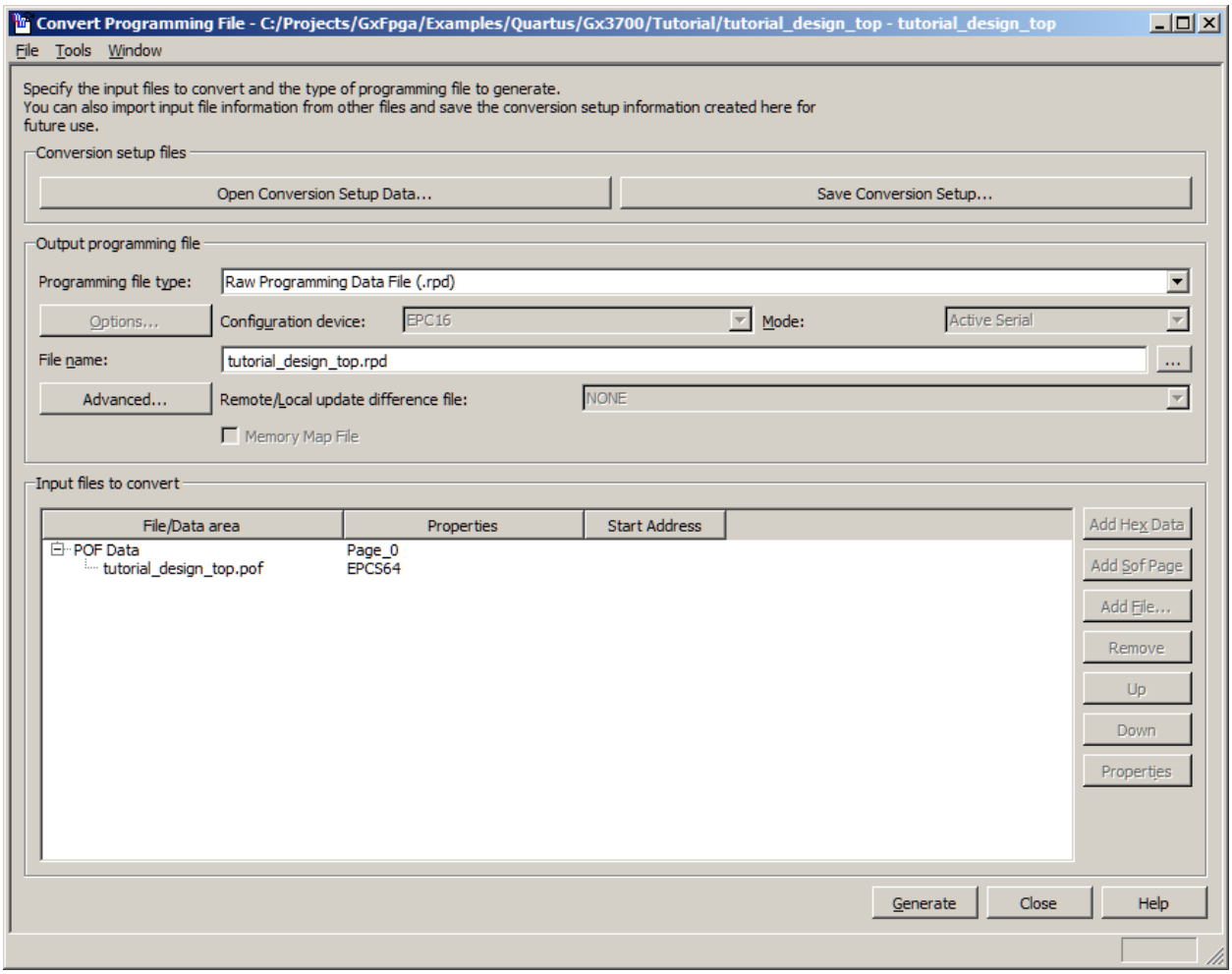


Figure 7-24: Convert Programming Files Dialog Box

Simulating the Design

To simulate the design, we will use **ModelSim** application from Altera. You can download the software for free form the Altera website. There is a test bench for this tutorial that is already created for you inside the **GXFPGA\Examples\Quartus\GX3700\Tutorial_VHDL.**

Follow these steps to simulate the design:

1. Open the ModelSim application:

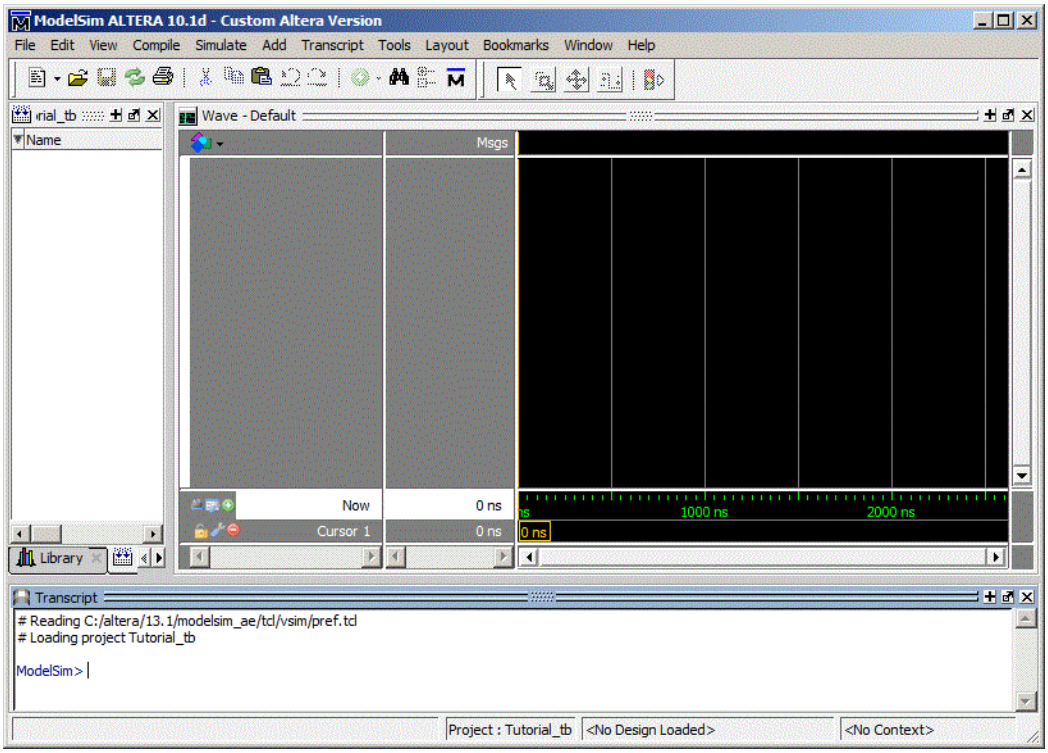


Figure 7-25: ModelSim Main Window

2. Click **File→ Change Directory** and choose the **sim** folder under the Tutorial folder. At this point the ModelSim should display the simulation pins:

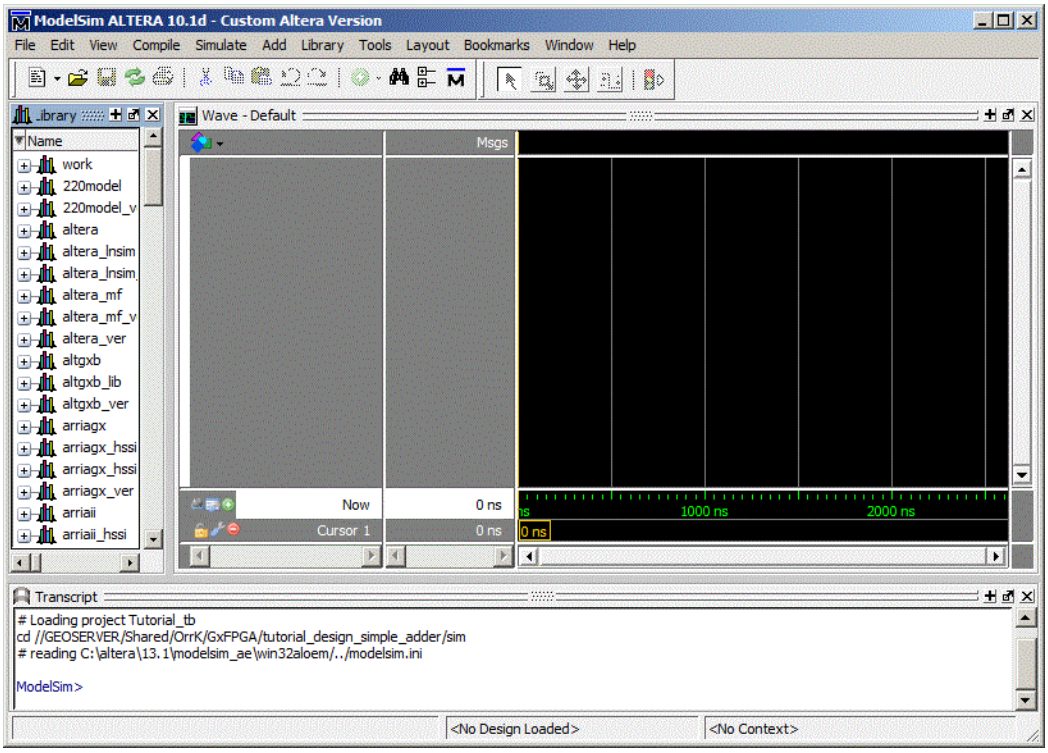


Figure 7-26: ModelSim Tutorial Simulation

3. Click **Tools→ Tcl→Execute Macro…** and choose **sim.tcl,** and click **Open.** When ModelSim asks to close the current project, click **Yes**. The screen should appear like the screen below:

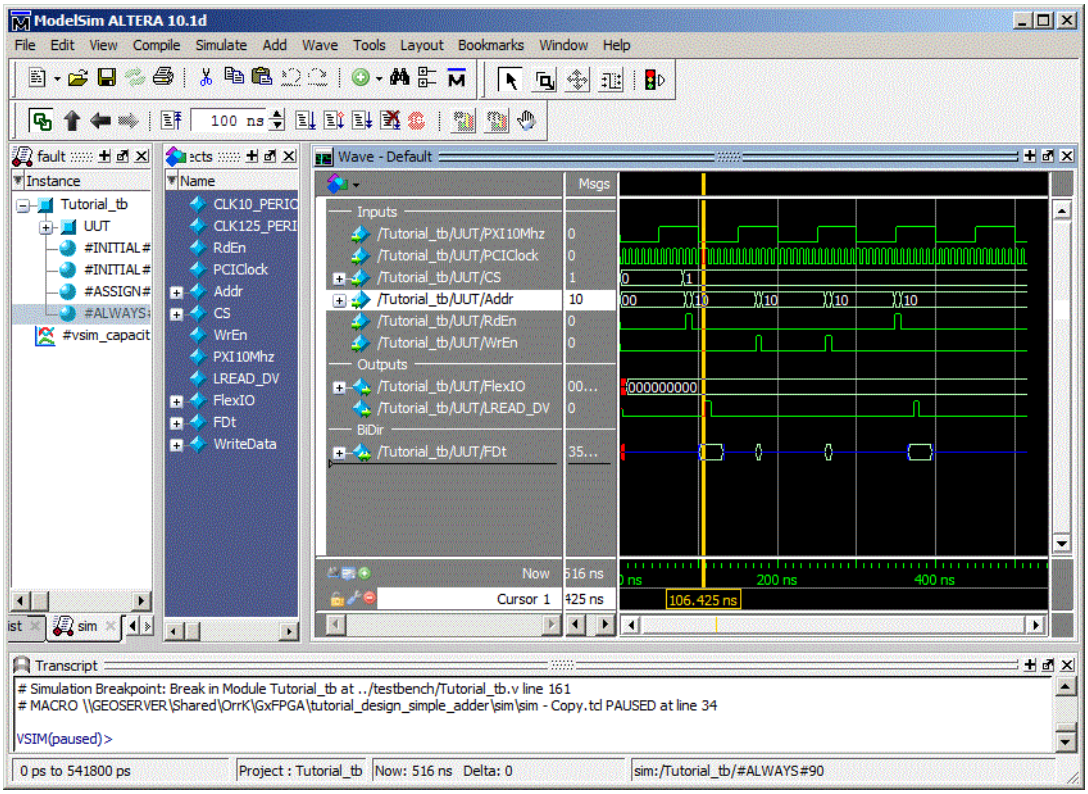


Figure 7-27: Simulation of Design

Load Gx3700 with SVF File

Start the **GX3700 Panel** (from the Windows **Start** menu, **Marvin Test Solutions**, **GxFpga**) and initialize the instrument. Next, click on the **Volatile** radio box and then click on the Browse Button (…) to select the newly generated SVF file (**tutorial_design_top.svf**). Finally click on the **Load** button to begin programming the card. You will see the progress bar indicate the status of the load. Once the load has completed, the status bar should be unfilled.

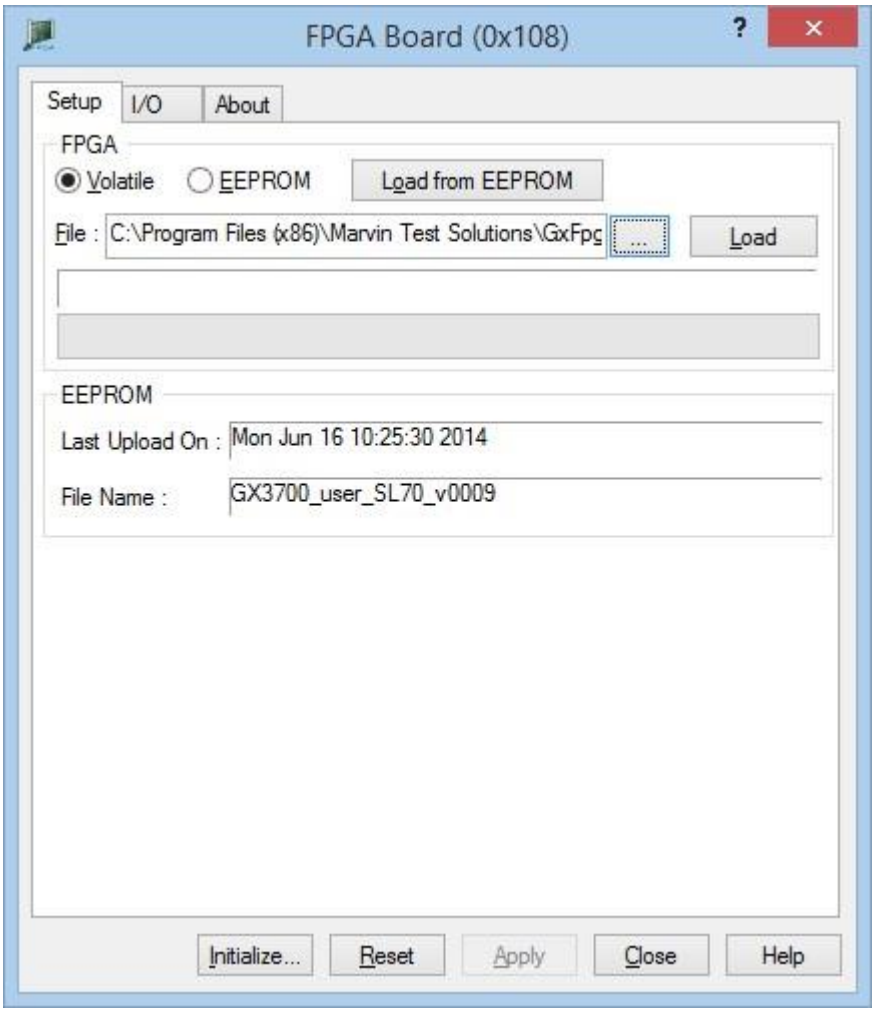


Figure 7-28: Software Front Panel

Testing the Design

Now that the design has been completed, compiled and loaded into the GX3700, we can move on to the testing.

There are two ways to access the FPGA, either through the software front panel or through the driver API DLL. We will demonstrate the programming method using ATEasy to access the driver API DLL.

Adder Testing

The software front panel will be used to test Phase 1 of the design which adds two 32 bit numbers together. Click on the **I/O Tab** to get started. The Adder phase is controlled through the FPGA Register space.

Offset 0x0 points to the first 32 bit number that will be summed and offset 0x4 points to the second 32 bit number that will be summed. Write values to both these locations.

The sum can be obtained by reading the 32 bit value at offset 0x8. Verify that the correct sum is read back as shown in Figure 5-31.

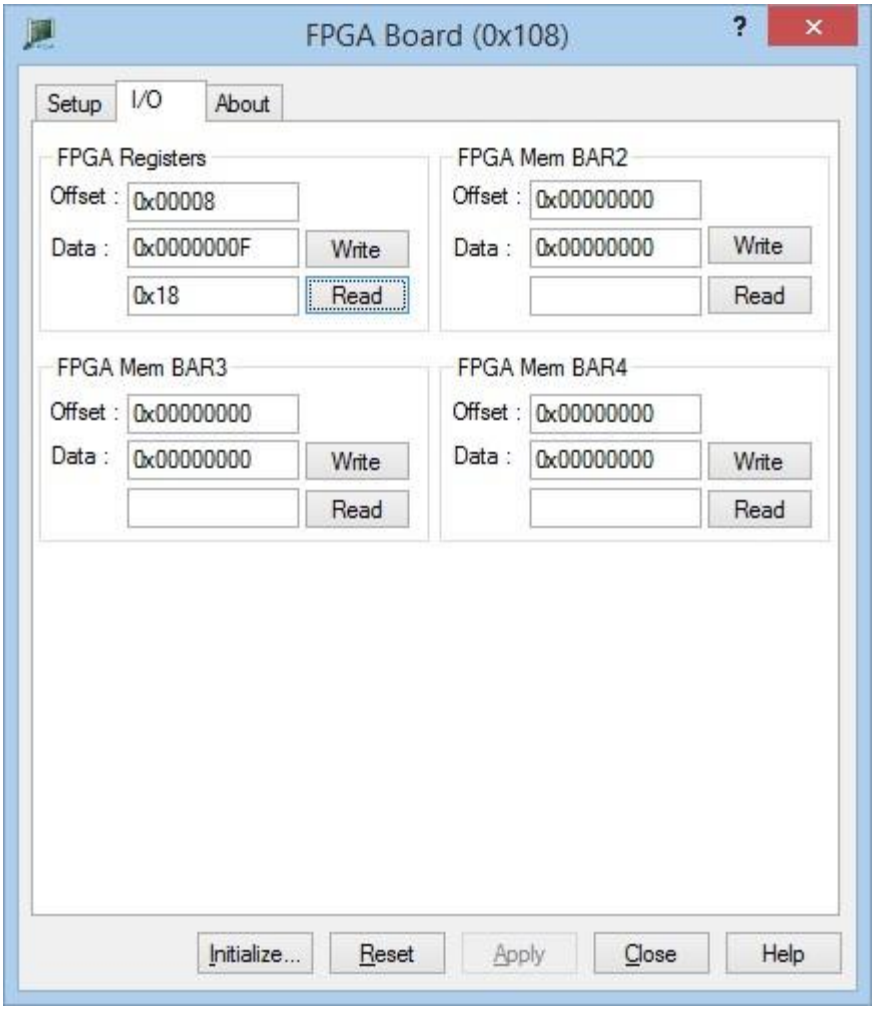


Figure 7-29: Using the Software Front Panel to read back the Sum

Clock Mux Testing

The software front panel will once again be used to test Phase 2 of the design. This part of the design uses a Mux to select between the PCI Clock and the 10 Mhz reference clock. The selected clock is output to I/O Channel 63 which is located on pin 31 on the Flex I/O J2 connector of the GX3700. The Mux is controlled through the FPGA Register space.

Writing a 0x0 to offset 0xC will route the PCI/PCIe Clock signal to I/O Channel 63. Writing 0x1 to the same offset will route the 10 Mhz clock to this same channel. Try switching between both values while monitoring pin 31 of J2 with an oscilloscope. You should see the appropriate clock signals.

Chapter 8 - GX3700 Expansion Boards

The GX3700 requires a piggy-back expansion board to connect to the outside world, a simple; feed through expansion board is provided. Custom expansion boards can be developed by customers. The following information is provided to assist the user with developing expansion boards. This information is for both, GX3700 and 3700e.

Expansion Board Design Guide

The expansion board mates with the GX3700 using one connector (P8) and two mounting holes. Two other connectors – J1 and J2 – exist on the expansion board and are attached to the front panel when the expansion board is mounted. Figure 6-1 depicts a bottom view of the expansion board and Figure 6-2 and Figure 6-3 detail the complete GX3700 with the feed through expansion board assembly.

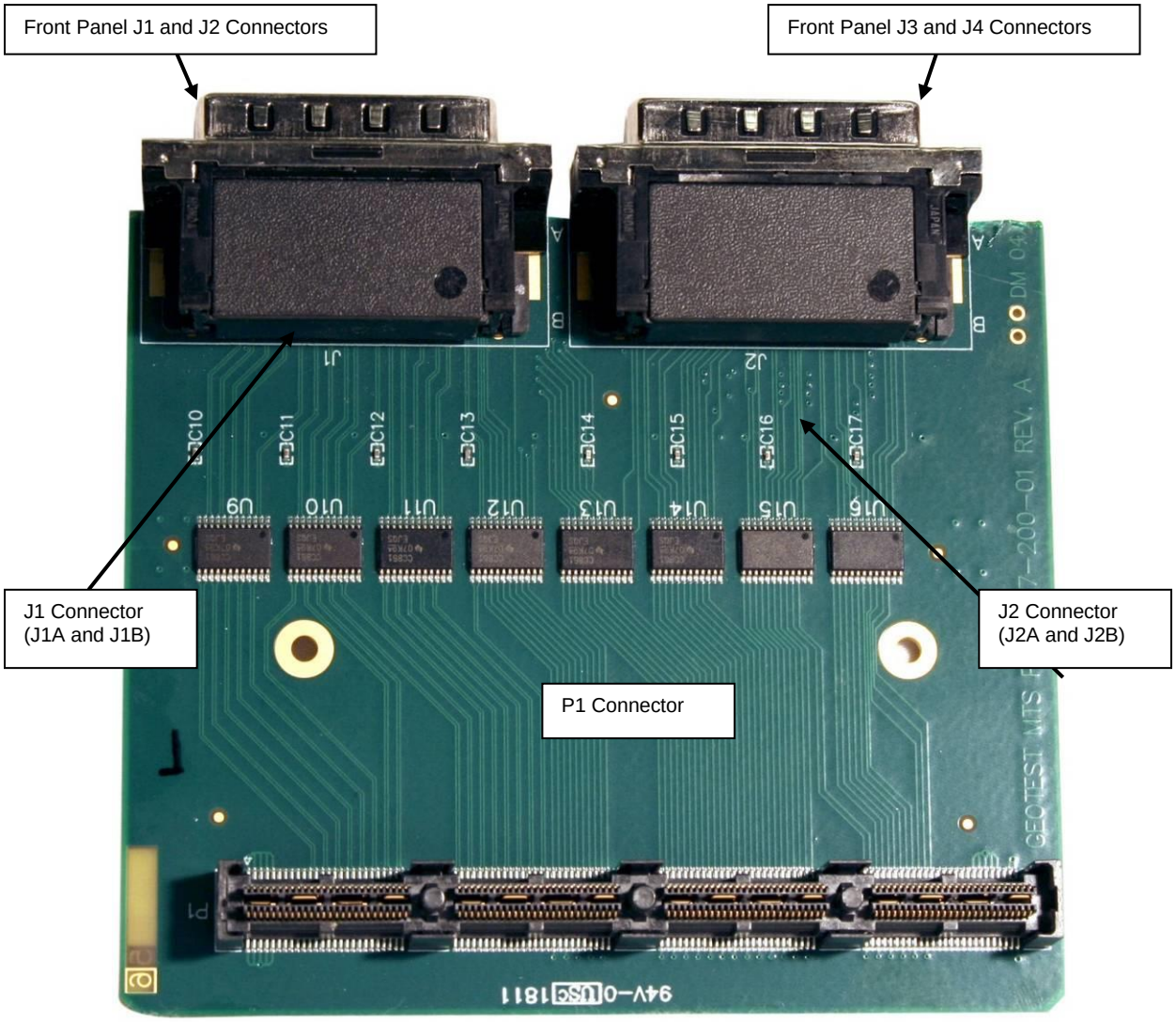


Figure 8-1: GX3701 Expansion Board – Bottom View

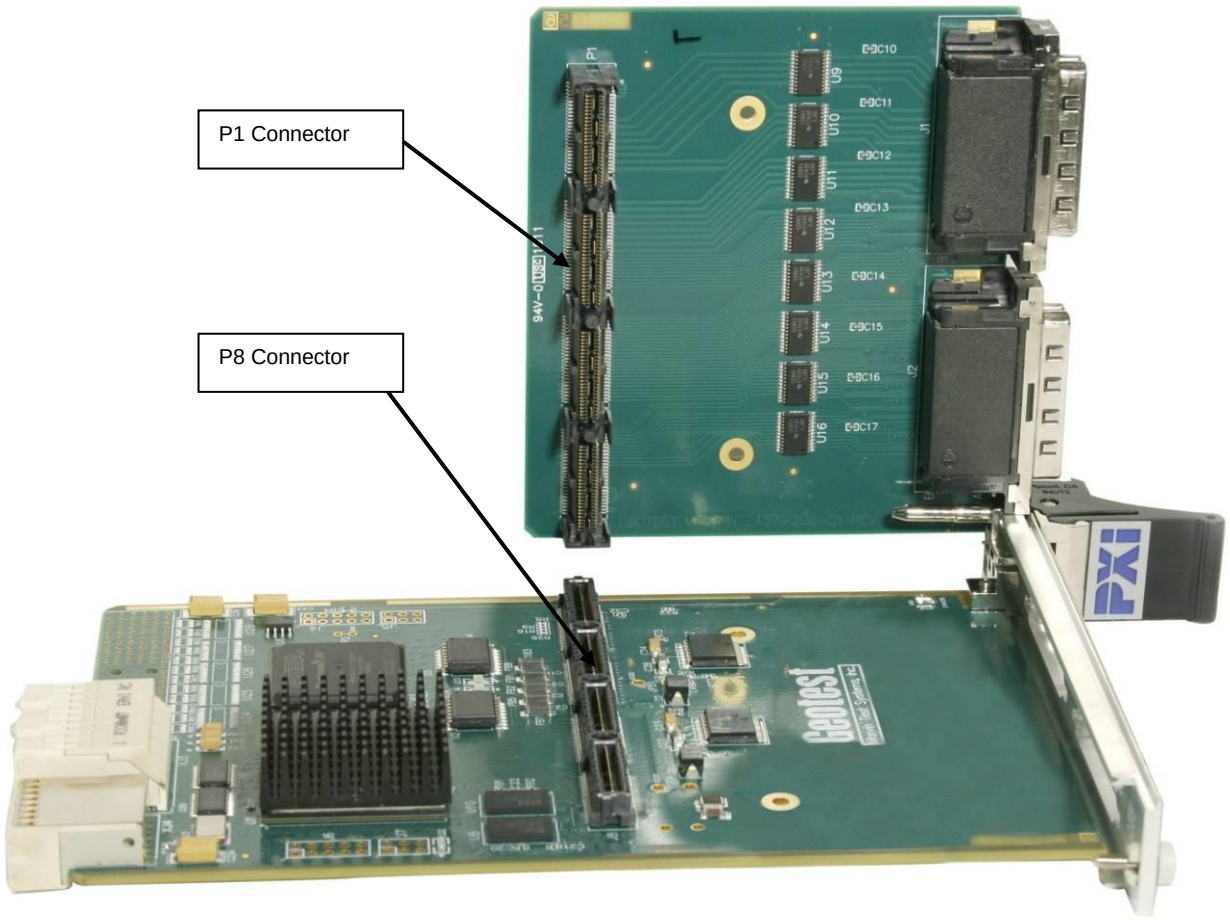


Figure 8-2: GX3700e Assembly with Expansion Board

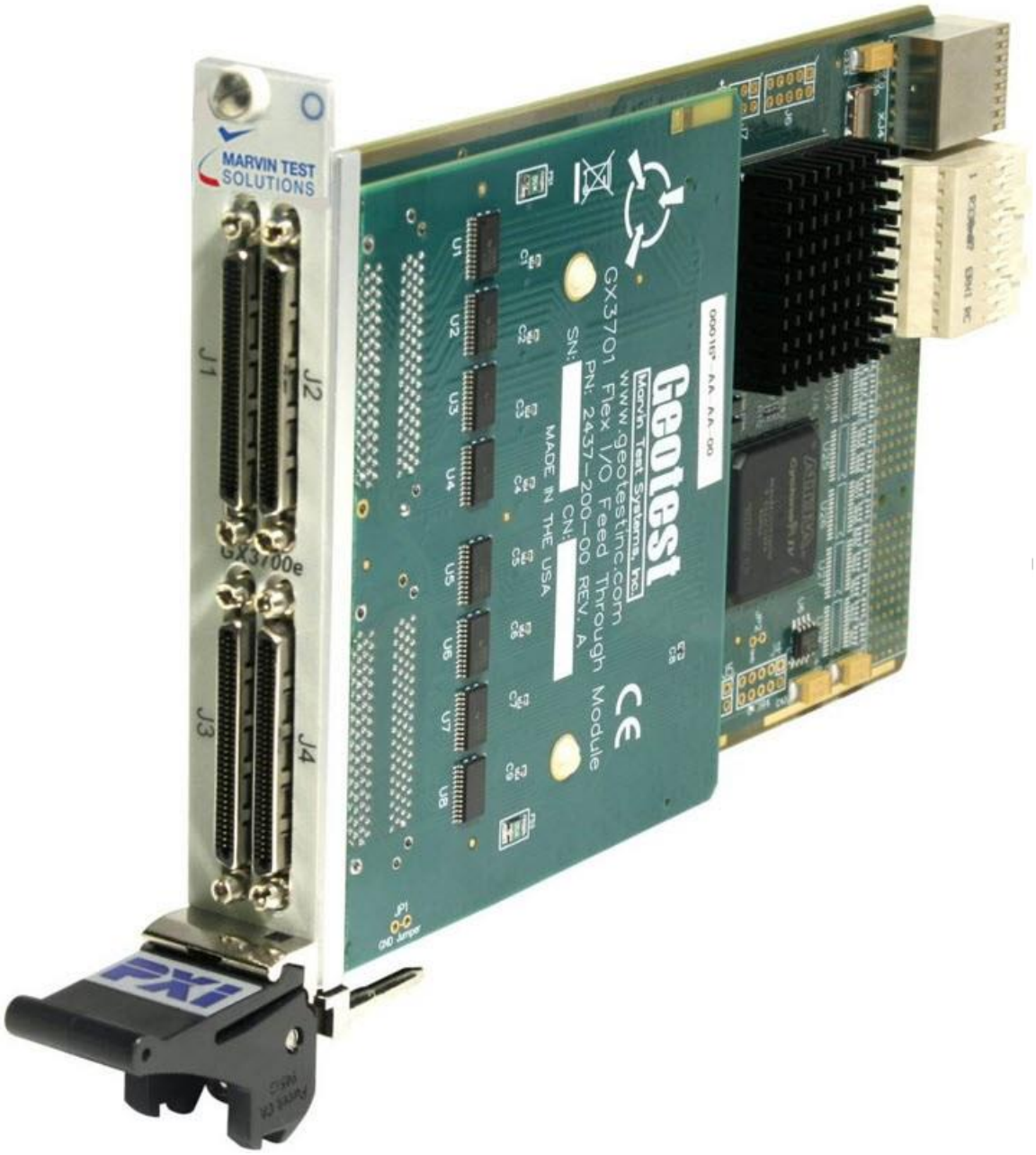


Figure 8-3: GX3700e with Expansion Board Mounted

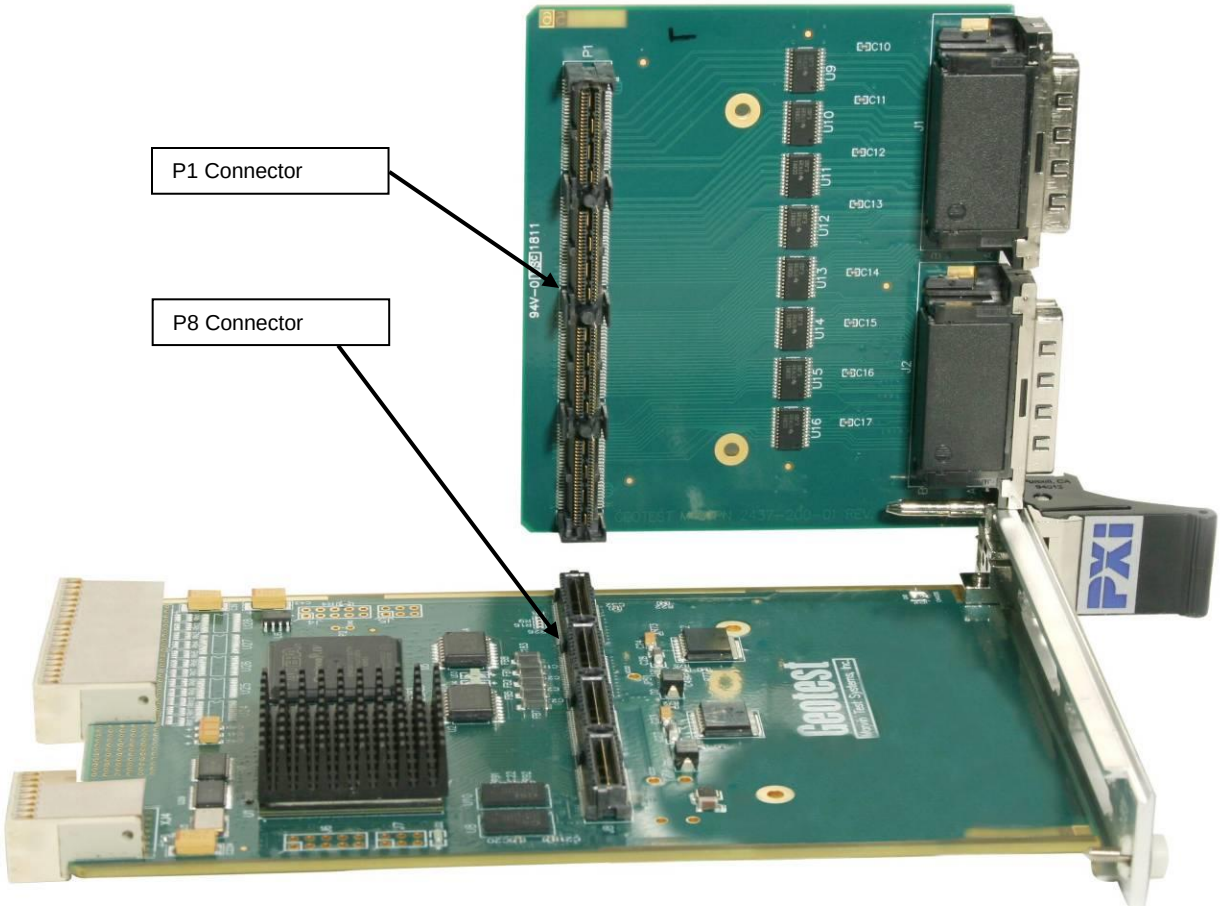


Figure 8-4: GX3700 Assembly with Expansion Board

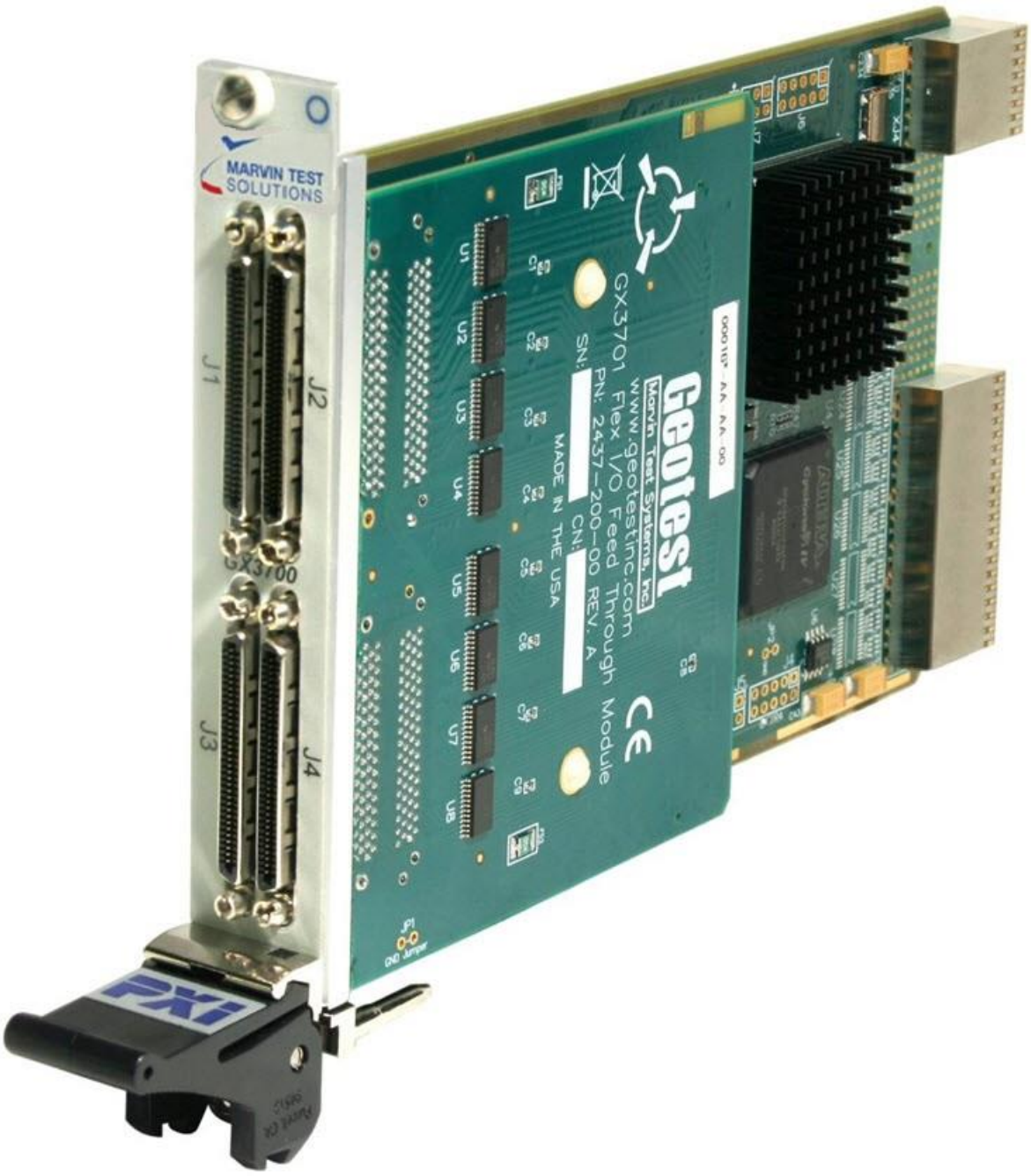


Figure 8-5: GX3700 with Expansion Board Mounted

Mechanical Layout Guide

The locations of the mounting holes and connectors are critical to ensure a proper fit between the GX3700 and the expansion board. Figure 6-6 describes the mechanical details of a typical board and the locations of connectors and mounting holes. The figure presents a transparent view of the board from the top, with dimensions for critical component locations. The coordinates for the connectors are pointing to the component reference point. For P1 it is the middle between pads 1 and 2 of the footprint, as shown in Figure 6-7. For J1 and J2 it is the center of pad A1 of the footprint, as shown in Figure 6-8.

Note: Dimensions are in mils unless noted otherwise.

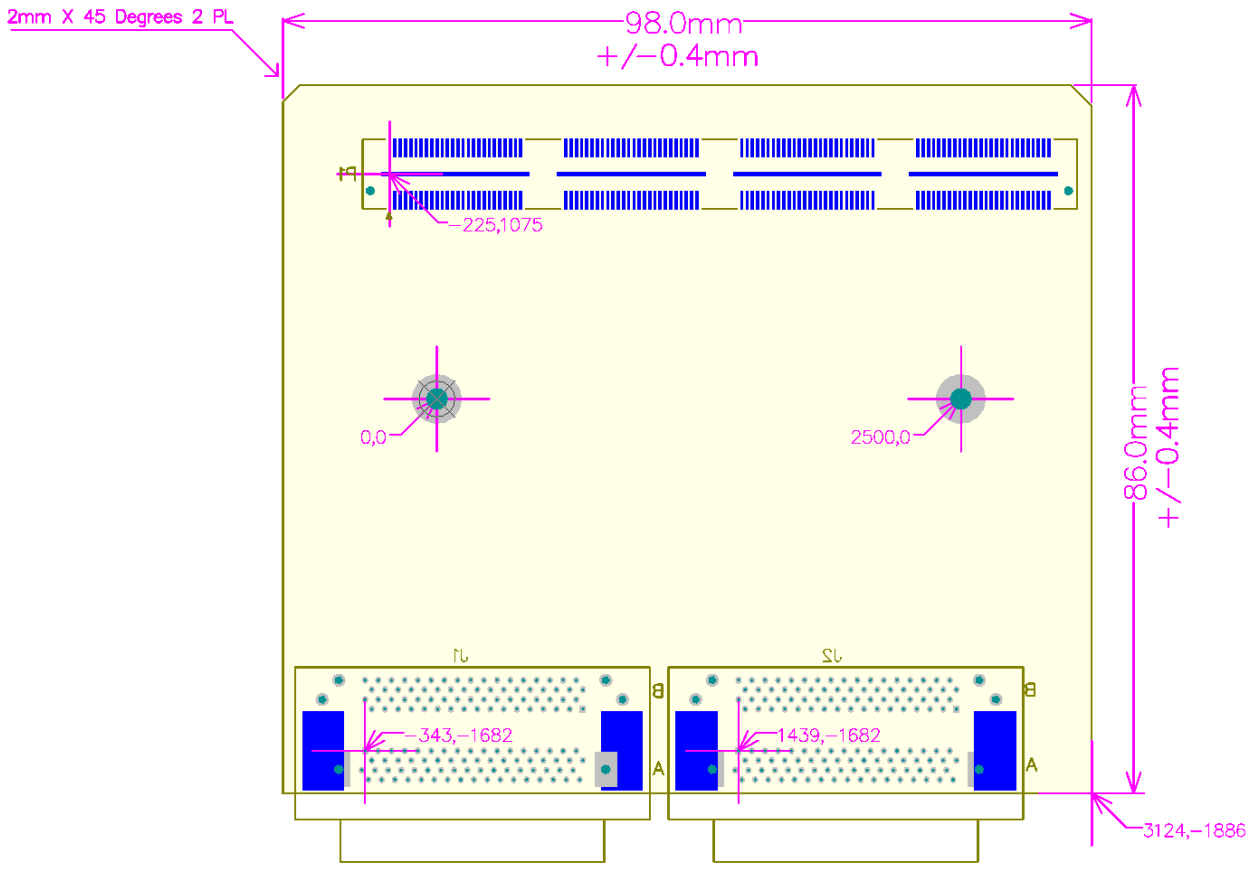


Figure 8-6: Mechanical Details – Top View of Typical Board. Dimensions are in mils unless noted otherwise.

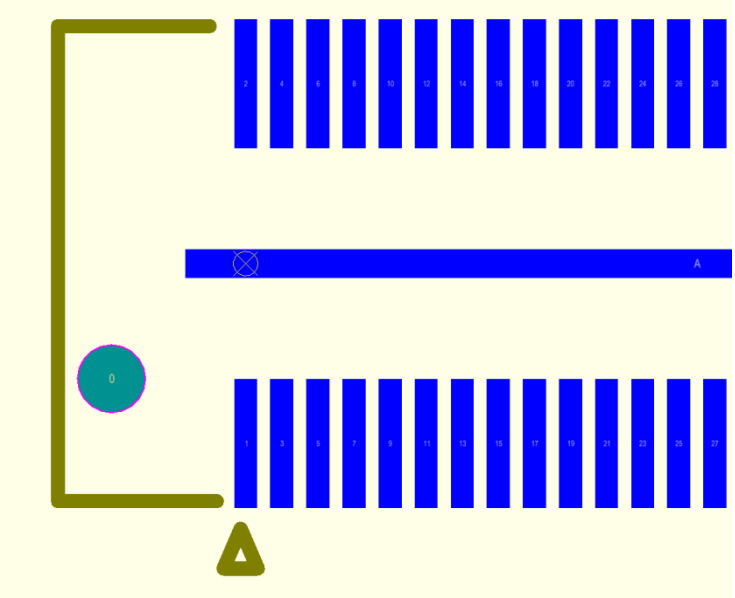


Figure 8-7: Component P1 Reference Point

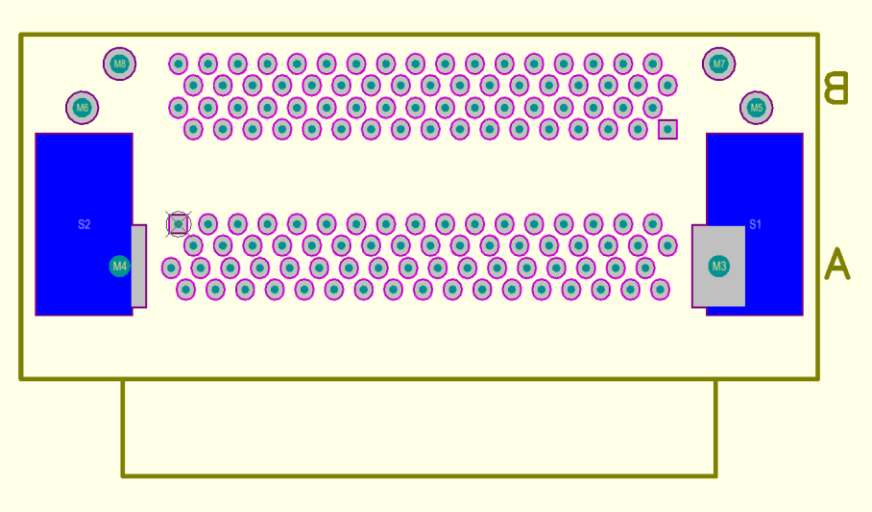


Figure 8-8: Components J1, J2 Reference Point

Figure 6-9 describes the recommended maximum dimensions for the expansion board and the recommended maximum component height. The maximum board area is about 110 Sq centimeters or about 17 sq inches.

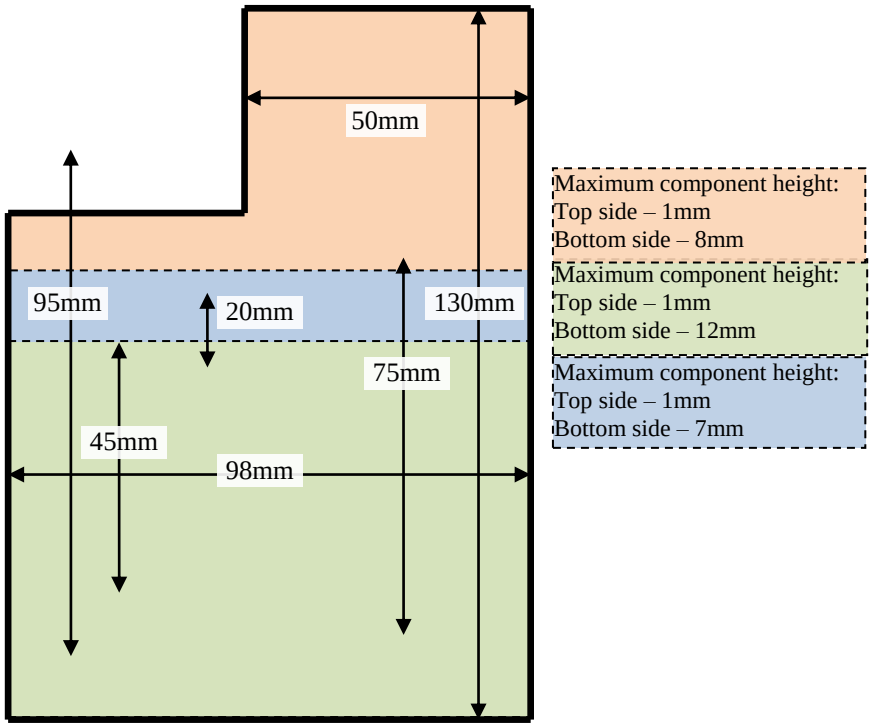


Figure 8-9: Mechanical Details – Top view, Maximum Board Dimensions

Expansion Board Connectors and Electrical Requirements

P1 is a High Speed Terminal Strip with Rugged Ground Plan manufactured by Samtec (http://www.samtec.com/). It has a middle bar that is used for ground and power connections. The part number for P1 is QFS-104-06.25-SL-D-A. Figure 6-10 shows a schematic diagram of P1.

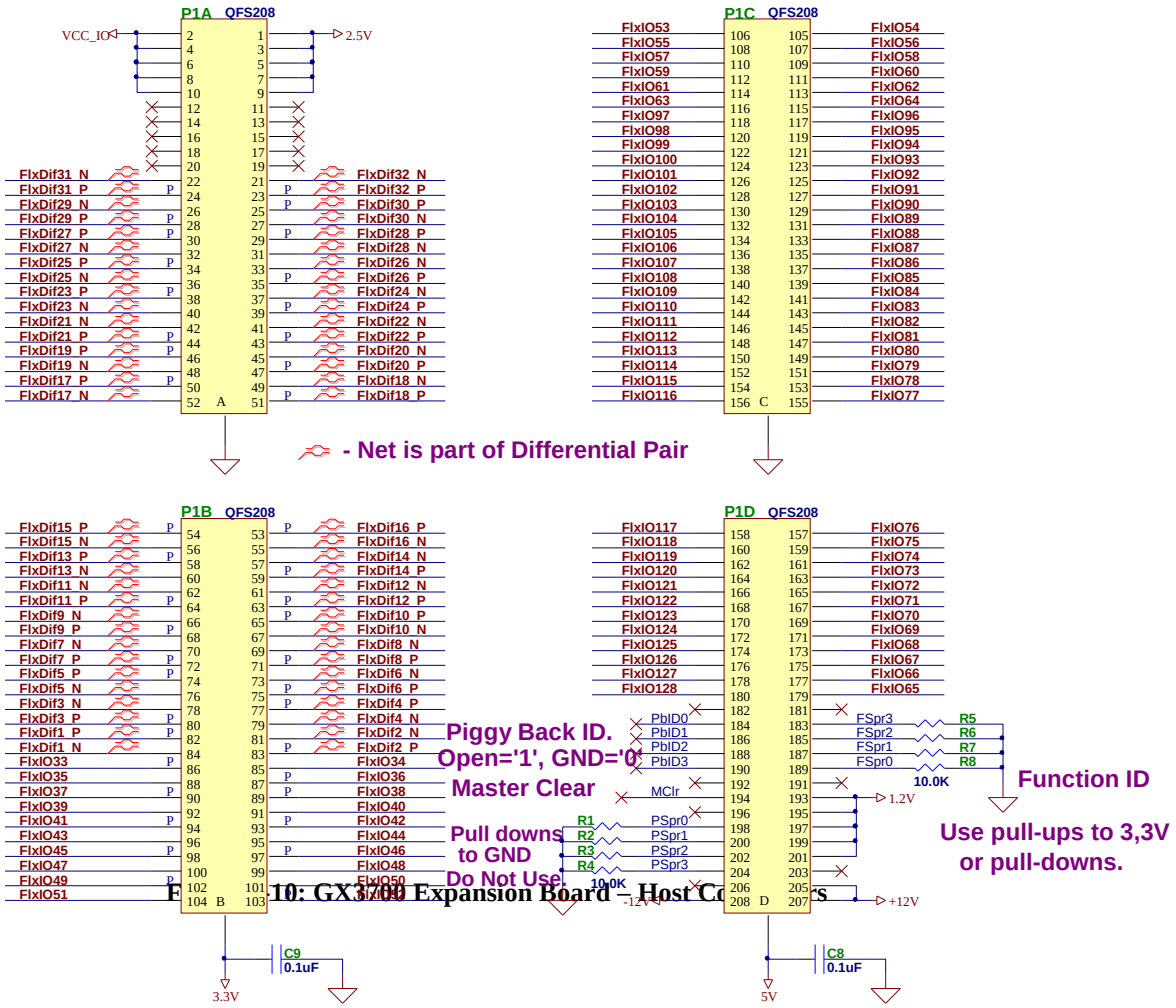


Figure 6-10: GX3700 Expansion Board – Host Connectors

J1 and J2 are used to connect the expansion board signals to the user application. Each one is a dual VHDCI 68 pins. There are few vendors for these connectors; one option is Honda PN HDRA-E68W1LFDTC-SL+. Each connector has two parts A and B. J1A corresponds to J2 on the front panel, J1B to J1, J2A to J4 and J2B to J3 on the front panel. Customers can us other connectors for their application but that will require changing the design of the front panel.

The following table lists the assignments for the expansion board signals.

P1 Expansion Board Connector Pin Assignment

The following table describes the GX3700 expansion board P1pin mapping to the front panel user connectors J1-J4, and FPGA pins:

Expansion Board Connector	Front Panel User Connector	FPGA pin Description	FPGA Pin Name	Remark
P1-82	J1-1	Flex Diff 1P	AG22	
P1-84	J1-35	Flex Diff 1N	AH22	
P1-83	J1-2	Flex Diff 2P	AG19	
P1-81	J1-36	Flex Diff 2N	AF19	
P1-80	J1-3	Flex Diff 3P	AH21	
P1-78	J1-37	Flex Diff 3N	AH20	
P1-77	J1-4	Flex Diff 4P	AD18	
P1-79	J1-38	Flex Diff 4N	AE19	
P1-74	J1-5	Flex Diff 5P	AG18	
P1-76	J1-39	Flex Diff 5N	AH19	
P1-75	J1-6	Flex Diff 6P	AE18	
P1-73	J1-40	Flex Diff 6N	AF17	
P1-72	J1-7	Flex Diff 7P	AH18	
P1-70	J1-41	Flex Diff 7N	AH17	
P1-71	J1-8	Flex Diff 8P	AE17	
P1-69	J1-42	Flex Diff 8N	AF16	
P1-68	J1-9	Flex Diff 9P	AG16	
P1-66	J1-43	Flex Diff 9N	AH16	
P1-65	J1-10	Flex Diff 10P	AD16	
P1-67	J1-44	Flex Diff 10N	AE16	
P1-64	J1-11	Flex Diff 11P	AG15	Dedicated Clock Input
P1-62	J1-45	Flex Diff 11N	AH15	Dedicated Clock Input
P1-63	J1-12	Flex Diff 12P	AD13	
P1-61	J1-46	Flex Diff 12N	AE13	
P1-58	J1-13	Flex Diff 13P	AG13	Dedicated Clock Input
P1-60	J1-47	Flex Diff 13N	AH14	Dedicated Clock Input
P1-59	J1-14	Flex Diff 14P	AD12	
P1-57	J1-48	Flex Diff 14N	AE12	
P1-54	J1-15	Flex Diff 15P	AG12	
P1-56	J1-49	Flex Diff 15N	AH13	
P1-53	J1-16	Flex Diff 16P	AF10	
P1-55	J1-50	Flex Diff 16N	AF11	
P1-50	J1-17	Flex Diff 17P	AH11	

Expansion Board Connector	Front Panel User Connector	FPGA pin Description	FPGA Pin Name	Remark
P1-52	J1-51	Flex Diff 17N	AH12	
P1-51	J1-18	Flex Diff 18P	AE9	
P1-49	J1-52	Flex Diff 18N	AF9	
P1-46	J1-19	Flex Diff 19P	AG10	
P1-48	J1-53	Flex Diff 19N	AH10	
P1-47	J1-20	Flex Diff 20P	AF8	
P1-45	J1-54	Flex Diff 20N	AE8	
P1-44	J1-21	Flex Diff 21P	AG9	
P1-42	J1-55	Flex Diff 21N	AH8	
P1-43	J1-22	Flex Diff 22P	AE6	
P1-41	J1-56	Flex Diff 22N	AF6	
P1-38	J1-23	Flex Diff 23P	AG7	
P1-40	J1-57	Flex Diff 23N	AH7	
P1-39	J1-24	Flex Diff 24P	AE5	
P1-37	J1-58	Flex Diff 24N	AF5	
P1-34	J1-25	Flex Diff 25P	AG6	
P1-36	J1-59	Flex Diff 25N	AH6	
P1-35	J1-26	Flex Diff 26P	AF2	
P1-33	J1-60	Flex Diff 26N	AG1	
P1-30	J1-27	Flex Diff 27P	AH4	
P1-32	J1-61	Flex Diff 27N	AH5	
P1-29	J1-28	Flex Diff 28P	AE2	
P1-31	J1-62	Flex Diff 28N	AF1	
P1-28	J1-29	Flex Diff 29P	AG4	
P1-26	J1-63	Flex Diff 29N	AH3	
P1-25	J1-30	Flex Diff 30P	AD1	
P1-27	J1-64	Flex Diff 30N	AE1	
P1-24	J1-31	Flex Diff 31P	AG3	
P1-22	J1-65	Flex Diff 31N	AH2	
P1-23	J1-32	Flex Diff 32P	AC2	
P1-21	J1-66	Flex Diff 32N	AC1	
P1-A,C	J1-34,68	GND		Power
P1-B	J1-33,67	User 3.3V		Power
P1-86	J2-1	FlexIO33	AH23	Routed to Expansion as Flex Diff 33P
P1-85	J2-2	FlexIO34	AF20	Routed to Expansion as Flex Diff 34N
P1-88	J2-3	FlexIO35	AH24	Routed to Expansion as Flex Diff 33N
P1-87	J2-4	FlexIO36	AE20	Routed to Expansion as Flex Diff 34P

Expansion Board Connector	Front Panel User Connector	FPGA pin Description	FPGA Pin Name	Remark
P1-90	J2-5	FlexIO37	AG25	Routed to Expansion as Flex Diff 35P
P1-89	J2-6	FlexIO38	AE21	Routed to Expansion as Flex Diff 36P
P1-92	J2-7	FlexIO39	AH25	Routed to Expansion as Flex Diff 35N
P1-91	J2-8	FlexIO40	AF21	Routed to Expansion as Flex Diff 36N
P1-94	J2-9	FlexIO41	AH26	Routed to Expansion as Flex Diff 37P
P1-93	J2-10	FlexIO42	AD22	Routed to Expansion as Flex Diff 38P
P1-96	J2-11	FlexIO43	AG27	Routed to Expansion as Flex Diff 37N
P1-95	J2-12	FlexIO44	AE22	Routed to Expansion as Flex Diff 38N
P1-98	J2-13	FlexIO45	AH27	Routed to Expansion as Flex Diff 39P
P1-97	J2-14	FlexIO46	AG24	Routed to Expansion as Flex Diff 40P
P1-100	J2-15	FlexIO47	AF26	Routed to Expansion as Flex Diff 39N
P1-099	J2-16	FlexIO48	AF23	Routed to Expansion as Flex Diff 40N
P1-102	J2-17	FlexIO49	AE23	Routed to Expansion as Flex Diff 41P
P1-101	J2-18	FlexIO50	AF24	Routed to Expansion as Flex Diff 42N
P1-104	J2-19	FlexIO51	AD24	Routed to Expansion as Flex Diff 41N
P1-103	J2-20	FlexIO52	AE24	Routed to Expansion as Flex Diff 42P
P1-106	J2-21	FlexIO53	AB1	
P1-105	J2-22	FlexIO54	B1	
P1-108	J2-23	FlexIO55	AB2	
P1-107	J2-24	FlexIO56	C1	
P1-110	J2-25	FlexIO57	AE4	
P1-109	J2-26	FlexIO58	D1	
P1-112	J2-27	FlexIO59	AD6	
P1-111	J2-28	FlexIO60	D2	
P1-114	J2-29	FlexIO61	AE7	
P1-113	J2-30	FlexIO62	E1	
P1-116	J2-31	FlexIO63	AD7	
P1-115	J2-32	FlexIO64	E2	
P1-A,C	J2-34-66,68	GND		Power
P1-B	J2-33,67	User 3.3V		Power
P1-179	J3-1	FlexIO65	AA18	
P1-177	J3-2	FlexIO66	Y17	
P1-175	J3-3	FlexIO67	AB17	
P1-173	J3-4	FlexIO68	AC17	
P1-171	J3-5	FlexIO69	AB16	
P1-169	J3-6	FlexIO70	AC16	
P1-167	J3-7	FlexIO71	Y15	

Expansion Board Connector	Front Panel User Connector	FPGA pin Description	FPGA Pin Name	Remark
P1-165	J3-8	FlexIO72	AA15	
P1-163	J3-9	FlexIO73	Y14	
P1-161	J3-10	FlexIO74	Y13	
P1-159	J3-11	FlexIO75	AA13	
P1-157	J3-12	FlexIO76	AB13	
P1-155	J3-13	FlexIO77	AA1	
P1-153	J3-14	FlexIO78	Y2	
P1-151	J3-15	FlexIO79	Y1	
P1-149	J3-16	FlexIO80	W2	
P1-147	J3-17	FlexIO81	W1	
P1-145	J3-18	FlexIO82	V3	
P1-143	J3-19	FlexIO83	V1	
P1-141	J3-20	FlexIO84	U3	
P1-139	J3-21	FlexIO85	T2	
P1-137	J3-22	FlexIO86	N2	
P1-135	J3-23	FlexIO87	L2	
P1-133	J3-24	FlexIO88	L1	
P1-131	J3-25	FlexIO89	K2	
P1-129	J3-26	FlexIO90	K1	
P1-127	J3-27	FlexIO91	J1	
P1-125	J3-28	FlexIO92	H2	
P1-123	J3-29	FlexIO93	H1	
P1-121	J3-30	FlexIO94	G2	
P1-119	J3-31	FlexIO95	G1	
P1-117	J3-32	FlexIO96	F1	
P1-A,C	J3-34-66,68	GND		Power
P1-D	J3-33,67	User 5V		Power
P1-118	J4-1	FlexIO97	AC7	
P1-120	J4-2	FlexIO98	AB7	
P1-122	J4-3	FlexIO99	AC8	
P1-124	J4-4	FlexIO100	AB8	
P1-126	J4-5	FlexIO101	AH9	
P1-128	J4-6	FlexIO102	AD9	
P1-130	J4-7	FlexIO103	AC9	
P1-132	J4-8	FlexIO104	AB9	
P1-134	J4-9	FlexIO105	AA9	
P1-136	J4-10	FlexIO106	Y9	

Expansion Board Connector	Front Panel User Connector	FPGA pin Description	FPGA Pin Name	Remark
P1-138	J4-11	FlexIO107	AE10	
P1-140	J4-12	FlexIO108	AC10	
P1-142	J4-13	FlexIO109	AA10	
P1-144	J4-14	FlexIO110	Y10	
P1-146	J4-15	FlexIO111	AE11	
P1-148	J4-16	FlexIO112	AC11	
P1-150	J4-17	FlexIO113	AB11	
P1-152	J4-18	FlexIO114	Y11	
P1-154	J4-19	FlexIO115	AF12	
P1-156	J4-20	FlexIO116	AC12	
P1-158	J4-21	FlexIO117	Y18	
P1-160	J4-22	FlexIO118	AD19	
P1-162	J4-23	FlexIO119	AC19	
P1-164	J4-24	FlexIO120	AB19	
P1-166	J4-25	FlexIO121	AA19	
P1-168	J4-26	FlexIO122	Y19	
P1-170	J4-27	FlexIO123	AC20	
P1-172	J4-28	FlexIO124	AB20	
P1-174	J4-29	FlexIO125	AG21	
P1-176	J4-30	FlexIO126	AD21	
P1-178	J4-31	FlexIO127	AC21	
P1-180	J4-32	FlexIO128	AB21	
P1-A,C	J4-34-66,68	GND		Power
P1-D	J4-33,67	User 5V		Power
P1-1,3,5,7,9	N/A	2.5V		Power
P1-2,4,6,8,10	N/A	VCC_IO		VCC I/O of the I/O banks of FPGA used on expansion board. Selectable on the GX3700 carrier by jumper as 1.2V, 2.5V or 3.3V.
P1-193,195, 197,199,201	N/A	1.2V		Power
P1-205,207	N/A	+12V		Power
P1-208	N/A	-12V		Power
P1-194	N/A	MClr		Input, Master Clear
P1-198		PSpr0		Do Not Use
P1-200		PSpr1		Do Not Use
P1-202		PSpr2		Do Not Use
P1-204		PSpr3		Do Not Use

Expansion Board Connector	Front Panel User Connector	FPGA pin Description	FPGA Pin Name	Remark
P1-184		PbID0		Output, Piggy Back ID. Pull up on carrier
P1-186		PbID1		Output, Piggy Back ID. Pull up on carrier
P1-188		PbID2		Output, Piggy Back ID. Pull up on carrier
P1-200		PbID3		Output, Piggy Back ID. Pull up on carrier
P1-189		FSpr0	L22	Output, Spare. Can be used as Function ID
P1-187		FSpr1	J16	Output, Spare. Can be used as Function ID
P1-185		FSpr2	J15	Output, Spare. Can be used as Function ID
P1-183		FSpr3	J14	Output, Spare. Can be used as Function ID

Table 8-1: Expansion Board P1 Pin Assignments

Notes for Expansion Board P1 connector:

1. Maximum 1A per pin.
2. PSpr[3..0] are reserved. Should be connected to ground using 1K-50K resistors.
3. PbID[3..0] are used to identify the expansion board. Leave pins unconnected for logic ‘1’ or connect to ground for logic ‘0’. The GX3700 software driver can read these pins to identify the specific expansion board installed.
4. FSpr[3..0] are spare pins connected to the user FPGA. Should be connected to ground or 3.3V using 1K-50K resistors if not used in the design. Can also be used as an additional identification field.
5. MClr is a Master Clear input to the Expansion board. It is active high and is asserted by the controller at power-up or by a software command at any time.
6. The Flex I/O signals must never be driven more than VCC_IO. If higher voltage logic is used in the Expansion board design, these signals must be protected.
7. During the user FPGA configuration phase, the Flex I/O pins have a weak pull-up that may cause an un-intentional condition in the Expansion board. Pull-down resistors should be used where necessary.

GX3701 Expansion Board

The GX3700/GX3700e is provided with the GX3701– Flex I/O Feed Through Expansion Module. The GX3701 provides connection and isolation between the FPGA I/O and the 4 connectors located on the module's front panel using bus switches. The GX3700 FPGA supports LVTTL, LVDS logic levels and can also be configured for 1.2 / 2.5 / 3.3 V logic level. Each channel can be configured as an input or output or isolated. Via the user FPGA, each I/O can be programmed for a specific logic level and current level. There are no active buffers on this board.

GX3701 Programming

Use the GXFPGA **GxFpgaXXX** driver functions to program the board. The functions are described in details in Chapter 9. Some of the functions are also available from the software front panel.

GX3701 Expansion Board Specification

Number of Channels	160 I/O; up to 84 I/O can be configured as 42 differential I/O channels 4 I/O are single-ended or 2 differential clock inputs
Logic Family	LVTTL, LVDS, configurable for 1.2 / 2.5 / 3.3 V logic; 5 V compatible; user programmable via the FPGA
Output Current	+/- 12 mA, sink or source, max. Programmable via the FPGA
Input Leakage Current	+/- 10 uA
Power On State	Default is disconnected at power on (unprogrammed FPGA) or defined by FPGA program
Input Protection	Overvoltage: -0.5 V to 7.0 V (input) Short circuit: up to 8 outputs may be shorted at a time

GX3702 Expansion Board

The GX3702 provides the same functionality as the GX3701 except the I/O configuration (connectors J1 – J4) is compatible with National Instruments' PXI 7811R and PXI 7813R modules' I/O.

Connections to the GX3700 may be made with 68-pin VHDCI male plug connector. Shielded cables with matching connectors are available from MTS. The I/O connections for the GX3702 are detailed in the tables below.

J1 – Flex I/O Bank A Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	GND	18	GND	35	Flex I/O 0	52	Flex I/O 17
2	GND	19	GND	36	Flex I/O 1	53	Flex I/O 18
3	GND	20	GND	37	Flex I/O 2	54	Flex I/O 19
4	GND	21	GND	38	Flex I/O 3	55	Flex I/O 20
5	GND	22	GND	39	Flex I/O 4	56	Flex I/O 21
6	GND	23	GND	40	Flex I/O 5	57	Flex I/O 22
7	GND	24	GND	41	Flex I/O 6	58	Flex I/O 23
8	GND	25	GND	42	Flex I/O 7	59	Flex I/O 24
9	GND	26	GND	43	Flex I/O 8	60	Flex I/O 25
10	GND	27		44	Flex I/O 9	61	Flex I/O 26
11	GND	28		45	Flex I/O 10	62	Flex I/O 27
12	GND	29	Flex I/O 28	46	Flex I/O 11	63	Flex I/O 29
13	GND	30	Flex I/O 30	47	Flex I/O12	64	Flex I/O 31
14	GND	31	Flex I/O 32	48	Flex I/O 13	65	Flex I/O 33
15	GND	32	Flex I/O 34	49	Flex I/O 14	66	Flex I/O 35
16	GND	33	Flex I/O 36	50	Flex I/O 15	67	Flex I/O 37
17	GND	34	Flex I/O 38	51	Flex I/O 16	68	Flex I/O 39

Table 8-2: J1 Flex IO Bank A Pin Out

J2 – Flex I/O Bank B Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	GND	18	GND	35	Flex I/O 40	52	Flex I/O 57
2	GND	19	GND	36	Flex I/O 41	53	Flex I/O 58
3	GND	20	GND	37	Flex I/O 42	54	Flex I/O 59
4	GND	21	GND	38	Flex I/O 43	55	Flex I/O 60
5	GND	22	GND	39	Flex I/O 44	56	Flex I/O 61
6	GND	23	GND	40	Flex I/O 45	57	Flex I/O 62
7	GND	24	GND	41	Flex I/O 46	58	Flex I/O 63
8	GND	25	GND	42	Flex I/O 47	59	Flex I/O 64
9	GND	26	GND	43	Flex I/O 48	60	Flex I/O 65
10	GND	27		44	Flex I/O 49	61	Flex I/O 66
11	GND	28		45	Flex I/O 50	62	Flex I/O 67
12	GND	29	Flex I/O 68	46	Flex I/O 51	63	Flex I/O 69
13	GND	30	Flex I/O 70	47	Flex I/O 52	64	Flex I/O 71
14	GND	31	Flex I/O 72	48	Flex I/O 53	65	Flex I/O 73
15	GND	32	Flex I/O 74	49	Flex I/O 54	66	Flex I/O 75
16	GND	33	Flex I/O 76	50	Flex I/O 55	67	Flex I/O 77
17	GND	34	Flex I/O 78	51	Flex I/O 56	68	Flex I/O 79

Table 8-3: J2 Flex IO Bank B Pin Out

J3 – Flex I/O Bank C Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	GND	18	GND	35	Flex I/O 80	52	Flex I/O 97
2	GND	19	GND	36	Flex I/O 81	53	Flex I/O 98
3	GND	20	GND	37	Flex I/O 82	54	Flex I/O 99
4	GND	21	GND	38	Flex I/O 83	55	Flex I/O 100
5	GND	22	GND	39	Flex I/O 84	56	Flex I/O 101
6	GND	23	GND	40	Flex I/O 85	57	Flex I/O 102
7	GND	24	GND	41	Flex I/O 86	58	Flex I/O 103
8	GND	25	GND	42	Flex I/O 87	59	Flex I/O 104
9	GND	26	GND	43	Flex I/O 88	60	Flex I/O 105
10	GND	27		44	Flex I/O 89	61	Flex I/O 106
11	GND	28		45	Flex I/O 90	62	Flex I/O 107
12	GND	29	Flex I/O 108	46	Flex I/O 91	63	Flex I/O 109
13	GND	30	Flex I/O 110	47	Flex I/O 92	64	Flex I/O 111
14	GND	31	Flex I/O 112	48	Flex I/O 93	65	Flex I/O 113
15	GND	32	Flex I/O 114	49	Flex I/O 94	66	Flex I/O 115
16	GND	33	Flex I/O 116	50	Flex I/O 95	67	Flex I/O 117
17	GND	34	Flex I/O 118	51	Flex I/O 96	68	Flex I/O 119

Table 8-4: J3 Flex IO Bank C Pin Out

J4 – Flex I/O Bank D Connector

Pin#	Function	Pin#	Function	Pin#	Function	Pin#	Function
1	GND	18	GND	35	Flex I/O 120	52	Flex I/O 137
2	GND	19	GND	36	Flex I/O 121	53	Flex I/O 138
3	GND	20	GND	37	Flex I/O 122	54	Flex I/O 139
4	GND	21	GND	38	Flex I/O 123	55	Flex I/O 140
5	GND	22	GND	39	Flex I/O 124	56	Flex I/O 141
6	GND	23	GND	40	Flex I/O 125	57	Flex I/O 142
7	GND	24	GND	41	Flex I/O 126	58	Flex I/O 143
8	GND	25	GND	42	Flex I/O 127	59	Flex I/O 144
9	GND	26	GND	43	Flex I/O 128	60	Flex I/O 145
10	GND	27		44	Flex I/O 129	61	Flex I/O 146
11	GND	28		45	Flex I/O 130	62	Flex I/O 147
12	GND	29	Flex I/O 148	46	Flex I/O 131	63	Flex I/O 149
13	GND	30	Flex I/O 150	47	Flex I/O 132	64	Flex I/O 151
14	GND	31	Flex I/O 152	48	Flex I/O 133	65	Flex I/O 153
15	GND	32	Flex I/O 154	49	Flex I/O 134	66	Flex I/O 155
16	GND	33	Flex I/O 156	50	Flex I/O 135	67	Flex I/O 157
17	GND	34	Flex I/O 158	51	Flex I/O 136	68	Flex I/O 159

Table 8-5: J4 Flex IO Bank D Pin Out

I/O: Input/ Output, **R**: Reserved, **DNU**: Do Not Use, **P**: Power/GND

GX3702 Expansion Board Specification

Number of Channels	160 I/O; up to 84 I/O can be configured as 42 differential I/O channels 4 I/O are single-ended or 2 differential clock inputs
Logic Family	LVTTL, LVDS, configurable for 1.2 / 2.5 / 3.3 V logic; 5 V compatible, programmable per pin via the FPGA
Output Current	+/- 12 mA, sink or source, max; programmable via the FPGA
Input Leakage Current	+/- 10 uA
Power On State	Default is disconnected at power on (unprogrammed FPGA) or defined by FPGA program
Input Protection	Overvoltage: -0.5 V to 7.0 V (input) Short circuit: up to 8 outputs may be shorted at a time

GX3788 Expansion Board

The GX3700/GX3700e can be equipped with the GX3788– Digital and Analog I/O Expansion Module. The GX3788 is a user configurable, FPGA-based, 3U PXI multi-function card which supports digital and analog test capabilities. The GX3788 is based on the GX3700 FPGA card and includes an integral daughter board which provides (8) differential input, 16-bit, 250 MS/s A to D converters and (8), 16-bit, 1 MS/s, D to A converters. The module's FPGA is pre-programmed, providing access to all digital and analog functions. Alternatively, users can program or modify the FPGA, allowing user to adapt the module to their own specific test needs. The design of the FPGA employs Altera's free Quartus II Web Edition tool set. Once the user has compiled the FPGA design, the configuration file can be loaded into the FPGA directly or via an on-board EEPROM. The digital and analog I/O lines are routed to the 4 front connections (J1 to J4)

The GX3788's digital I/O signals are TTL compatible and can be programmed as inputs or outputs. The A to D channels can be configured as 8 differential, or 16 single ended inputs and support a sampling rate of up to 250 KS/s. Alternately, two channel operation can support a sampling rate of 1 MS/s. The D to A channels support a simultaneous sampling rate of 1 MS/s. The FPGA device supports up to four phase lock loops for clock synthesis, clock generation and for support of the I/O interface. An on-board 80 MHz oscillator is available for use with the FGPA device or alternatively, the PXI 10 MHz clock can be used as a clock reference by the FPGA.

GX3788 Programming

Use the GXFPGA **Gx3788xxx** driver functions to program the board. The functions are described in detail in Chapter 9. Some of the functions are also available from the software front panel (**DAQ** page).

GX3788 Digital and Analog Multi-Function Expansion Board Specification

Digital I/O Channels	
Logic Families	LVTTL, LVDS, configurable for 1.2 / 2.5 / 3.3 V logic; 5 V compatible, programmable per pin via the FPGA
Current	±12.0 mA, max. Programmable per pin via the FPGA
Input Leakage Current	±10 µA
Power on State	Default is disconnected at power on (unprogrammed FPGA) or defined by FPGA program
Number of Channels	32 Differential digital I/O lines 64 Single-ended digital I/O line
FIFO Depth	2047 Samples
Maximum FIFO Clock Rate	10 MHz
Clock Sources	PXI triggers, Ext Trigger, Star X, PXI Clk10, PXI Clk100 (Express version), DSTAR (Express version), Local bus
Protection	Overvoltage: -0.5 V to 7.0 V (input) Short circuit: up to 8 outputs may be shorted at a time
Analog Input Channels	
Number of Channels	8 differential or 16 single-ended
Sample Rate	250 KS/s (simultaneous) or 1 MS/s (two channels)
Bus Transfer Modes	DMA, Interrupt, Register I/O
Resolution	16-bits
Accuracy	+/- 13.60V Range: +/- 7.50mV +/- 10.24V Range: +/- 6.50mV +/- 5.12V Range: +/- 4.50mV +/- 2.56V Range: +/- 4.0mV
Input Voltage Ranges (FS VDC)	± 13.6* ± 10.24 ± 5.12 ± 2.56 ± 1.28 ± 0.64 * Uses the gain value for the 20.48 VDC range
Input Impedance	500 M ohms
Analog BW (3 dB)	8 MHz
Over Voltage Protection	± 24V
CMRR, DC to 60 Hz	90 dB
Channel to Channel Crosstalk	-120 dB (adj. ch.), Fin = 10 KHz
Triggering	Trigger in / Trigger out (FPGA controlled)

Analog Output Channels	
Number of Channels	8
Conversion Rate	1 MS/s (simultaneous)
Resolution	16-bits
Output Accuracy	+/- 6.0 mV
Output Range	± 10 V
Output Drive Current	3 mA
Short Circuit Current	8 mA
Output Slew Rate	6 V/us
Timing Sources	
PXI Bus	10 MHz
Internal	80 MHz oscillator, ±20 ppm
FPGA and Memory	
FPGA Type	Altera Stratix III, EP3SL50F780
Number of PLLs	Four
Logic Elements	47.5 K
Internal Memory	1.836 Mb
On-Board Memory	256 K x 32 SSRAM
On-Board Flash	16 MB
Power	
3.3 VDC	3.6 A (typ); 4.9 A (max)
5 VDC	0.045 A (max)
User 3.3 V (@ J1, J2 connector)	1 A, max
User 5 V (@ J3, J4 connector)	1 A, max
Environmental	
Operating Temperature	0 °C to +50 °C
Storage Temperature	-20 °C to +70 °C
Size	3U PXI
Weight	200 g

Chapter 9 - Function Reference

Introduction

The GXFPGA driver functions reference chapter is organized in alphabetical order. Each function is presented starting with the syntax of the function, a short description of the function parameters description and type followed by a Comments, an Example (written in C), and a See Also sections.

All function parameters follow the same rules:

- Strings are ASCIIZ (null or zero character terminated).
- Most function's first parameter is *nHandle* (16-bit integer). This parameter is required for operating the board and is returned by the **GxFpgaInitialize** or the **GxFpgaInitializeVisa** functions. The *nHandle* is used to identify the board when calling a function for programming and controlling the operation of that board.
- All functions return a status with the last parameter named *pnStatus*. The *pnStatus* is zero if the function was successful, or less than a zero on error. The description of the error is available using the **GxFpgaGetErrorString** function or by using a predefined constant, defined in the driver interface files: GXFPGA.H, GXFPGA.BAS, GXFPGA.VB, GXFPGA.PAS or GXFPGA.DRV.
- Parameter name are prefixed as follows:

Prefix	Type	Example
a	Array, prefix this before the simple type.	*anArray* (Array of Short)
n	Short (signed 16-bit)	nMode
d	Double - 8 bytes floating point	dReading
dw	Double word (unsigned 32-bit)	dwTimeout
l	Long (signed 32-bit)	lBits
p	Pointer. Usually used to return a value. Prefix this before the simple type.	pnStatus
sz	Null (zero value character) terminated string	szMsg
w	Unsigned short (unsigned 16-bit)	wParam
hwnd	Window handle (32-bit integer).	hwndPanel

Table 9-1: Parameter Prefixes

GXFPGA Functions

The following list is a summary of functions available for the GX3700:

Driver Functions	Description
General Functions	
GxFpgaInitialize	Initializes the driver for the board at the specified slot number using HW. The function returns a handle that can be used with other GXFPGA functions to program the board
GxFpgaInitializeVisa	Initializes the driver for the specified slot using VISA. The function returns a handle that can be used with other GXFPGA functions to program the board.
GxFpgaReset	Resets the GX3700 interface FPGA and User FPGA to their default state.
GxFpgaGetBoardSummary	Returns the board summary.
GxFpgaGetBoardType	Returns the board type.
GxFpgaGetDriverSummary	Returns the driver name and version.
GxFpgaGetErrorString	Returns the error string associated with the specified error number.
GxFpgaPanel	Opens the instrument panel dialog to used to interactively control the board.
FPGA Settings Functions	
GxFpgaGetEepromSummary	Returns the timestamp and filename of the last FPGA configuration written to EEPROM.
GxFpgaGetExpansionBoardID	Returns the current Expansion Board ID.
GxFpgaLoad	Loads the volatile FPGA or the non volatile EEPROM with FPGA configuration data in the form of SVF or RPD files respectively.
GxFpgaLoadFromEeprom	Loads the FPGA with the contents of the EEPROM.
GxFpgaLoadStatus	Returns the progress of the last asynchronous load in percentage.
GxFpgaLoadStatusMessage	Returns a string describes the current load progress of the last asynchronous load.
GxFpgaRead	Reads the specified number of data elements from the User's FPGA specified BAR memory.
GxFpgaReadRegister	Reads a 32 bit User's FPGA register.
GxFpgaWrite	Writes the specified number of data elements to the User's FPGA specified BAR memory.
GxFpgaWriteRegister	Writes a buffer of 32 bit double words to the User's FPGA's register space.
Event (Interrupt) Functions	
GxFpgaSetEvent	Enables or disables an event handler
GxFpgaDiscardEvents	Clears the events queue
GxFpgaWaitOnEvent	Waits until event received or timeout occurred

Driver Functions	Description
DMA Functions	
GxFpgaDmaFreeMemory	Free the DMA block of continues physical memory that was previously allocated when the user called **GxFpgaDmaTransfer** API
GxFpgaDmaGetTransferStatus	Returns the DMA transfer status register.
GxFpgaDmaTransfer	Transfers a block of data using DMA.
Upgrade firmware functions	
GxFpgaUpgradeFirmware	Upgrades the board's firmware.
GxFpgaUpgradeFirmwareStatus	Monitor the firmware upgrade process.
Gx3788 functions	
Gx3788Initialize	Initializes the driver for the board at the specified slot number using HW. The function returns a handle that can be used with other GX3788 functions to program the board
Gx3788InitializeVisa	Initializes the driver for the specified slot using VISA. The function returns a handle that can be used with other GX3788 functions to program the board.
Gx3788Reset	Resets the GX3788 to its default state.
Gx3788Panel	Opens the instrument panel dialog to used to interactively control the board.
Gx3788AnalogInGetGroundSource	Returns the analog input ground source
Gx3788AnalogInMeasureChannel	Measure voltage from analog input channel
Gx3788AnalogInScanGetChannelListIndex	Returns the analog input channel from index
Gx3788AnalogInScanGetCount	Returns the analog input scan count
Gx3788AnalogInScanGetSampleRate	Returns the analog input scan sample rate
Gx3788AnalogInScanGetLastRunCount	Returns the analog input scan count of the last run
Gx3788AnalogInScanIsRunning	Returns the status of the analog input scanning
Gx3788AnalogInScanReadMemoryRawData	Reads the analog input memory in the form of raw data
Gx3788AnalogInScanReadMemoryVoltages	Reads the analog input memory in the form of voltages
Gx3788AnalogInScanSetChannelListIndex	Sets the analog input channel at an index
Gx3788AnalogInScanSetCount	Sets the analog input scan count
Gx3788AnalogInScanSetSampleRate	Sets the analog input scan sample rate
Gx3788AnalogInScanStart	Starts the analog input scan process
Gx3788AnalogInSetGroundSource	Sets the analog input ground source
Gx3788AnalogOutGetOutputState	Returns the analog output state
Gx3788AnalogOutGetVoltage	Returns the analog output voltage value
Gx3788AnalogOutReset	Resets the analog output settings.
Gx3788AnalogOutSetOutputState	Sets the analog output state
Gx3788AnalogOutSetVoltage	Sets the analog output voltage value
Gx3788GetBoardSummary	Returns the board summary.
Gx3788GetCalibrationInfo	Returns the board calibration info

Driver Functions	Description
Gx3788PioGetPort	Returns the digital port output data value
Gx3788PioGetPortChannel	Returns the digital channel output data value
Gx3788PioGetPortDirection	Returns the digital port direction state
Gx3788PioGetPortChannelDirection	Returns the digital channel direction state
Gx3788PioReadPort	Reads the input state of 32 channels in the specified digital port
Gx3788PioReadPortChannel	Reads the input state of the specified digital port channel
Gx3788PioResetPort	Resets the digital port to default settings
Gx3788PioResetPortChannel	Resets the digital port channel to default settings
Gx3788PioSetPort	Sets the digital port output data value
Gx3788PioSetPortChannel	Sets the digital channel output data value
Gx3788PioSetPortDirection	Sets the digital port direction state
Gx3788PioSetPortChannelDirection	Sets the digital channel direction state
Gx3788TriggerGetOutputLevel	Returns trigger output level
Gx3788TriggerReadInputLevel	Reads back the trigger input level
Gx3788TriggerSetOutputLevel	Sets the trigger output level

GxFpgaDiscardEvents

Purpose

Clears the event queue.

Syntax

GxFpgaDiscardEvents (*nHandle*, *nEventType*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
nEventType	SHORT	Event type. Use the constant GT_EVENT_INTERRUPT (1). No other value is supported.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function clears the event queue and remove all pending events. Setting an event handler using the **GxFpgaSetEvent** automatically clears the event queue.

Example

The following example uses discard events to reset the queue after lengthy operation:

```
GxFpgaInitialize (1, &nHandle, &nStatus);
GxFpgaSetEvent(nHandle, GT_EVENT_INTERRUPT, TRUE, NULL, (PVOID)1, &nStatus);
while (TRUE)
{
    ! wait up to 1000 ms for the event
    GxFpgaWaitOnEvent(nHandle, GT_EVENT_INTERRUPT, 1000, &nStatus);
    if (nStatus!=0)    ! success event occurred
    {    printf("no event occurred - exiting");
         break;
    }
    else
    {   ! do something lengthy …
        ! now ready to receive more events
        GxFpgaDiscardEvents(nHandle, GT_EVENT_INTERRUPT, &nStatus);
}
```

See Also

GxFpgaInitialize, **GxFpgaGetErrorString, GxFpgaWaitOnEvent, GxFpgaSetEvent**

GxFpgaDmaFreeMemory

Purpose

Free the DMA block of continues physical memory that was previously allocated when the user called **GxFpgaDmaTransfer** API.

Syntax

GxFpgaDmaFreeMemory (*nHandle*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The first time the use calls the **GxFpgaDmaTransfer** API, a 64KB block of continues physical memory is allocated for the DMA usage. The user can free this block of physical memory back to the OS by calling this function.

Example

The following example free any previously allocated block of 64KB of continues physical memory.

```
SHORT   nStatus;
GxFpgaDmaFreeMemory (nHandle, &nStatus);
```

See Also

GxFpgaDmaTransfer, GxFpgaGetErrorString

GxFpgaDmaGetTransferStatus

Purpose

Returns the DMA transfer status register.

Syntax

GxFpgaDmaGetTransferStatus (*nHandle, pnDmaStatus, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pnDmaStatus	SHORT	0. No DMA Transfer 1. DMA Transfer is in progress.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example returns the DMA transfer status:

```
SHORT nDmaStatus;
GxFpgaDmaGetTransferStatus (nHandle, &nDmaStatus, &nStatus);
```

See Also

GxFpgaDmaTransfer, GxFpgaGetErrorString

GxFpgaDmaTransfer

Purpose

Transfers a block of data using DMA.

Syntax

GxFpgaDmaTransfer (*nHandle, bDmaRd, pvData, nElementSize, dwSize, dwMode, pvOp, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
bDmaRd	BOOL	Transfer operation: 0. GXFPGA_DMA_READ = DMA write opearation. The function will write the buffer data (*pvData*) content to the User's FLEX FPGA memory location. 1. GXFPGA_DMA_WRITE = DMA read opearation. The function will copy the speciread from the User's FLEX FPGA memory location to the buffer (*pvData*).
pvData	PVOID	Pointer to an array of data. The array must be greater or equal to *dwSize* parameter.
nElementSize	SHORT	The *pvData* buffer element size.
dwSize	DWORD	Number of elements in the *pvData* buffer. Maximum number of bytes that can be transferred at once is 65528.
dwMode	DWORD	Not used.
pvOp	PVOID	Not used.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function utilizes the built-in DMA capabilities in order to transfer data to or from the User's FLEX FPGA at the fastest speed.

Note: The user need to setup the path to the target memory as it design depended.

Example

The following example read a block of 256 bytes of data from User's FLEX FPGA memory location to the buffer:

```
DWORD adwData[256]
GxFpgaDmaTransfer (nHandle, GXFPGA_DMA_READ, 0, &adwData, 4, 256, 0, 0, &nStatus);
```

See Also

GxFpgaDmaGetTransferStatus, GxFpgaGetErrorString

GxFpgaGetBoardSummary

Purpose

Returns the board information.

Syntax

GxFpgaGetBoardSummary (*nHandle, pszSummary, nMaxLen, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pszSummary	PSTR	Buffer to contain the returned board info (null terminated) string.
nMaxLen	SHORT	*pszSummary* buffer size.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function returns the board information including the board firmware version, serial number and user FPGA part number.

The Gx3700 board comes installed with one of the following the following Stratix III user FPGA parts:

- EP3SL50F780
- EP3SL70F780
- EP3SL110F780
- EP3SL150F780
- EP3SL200F780
- EP3SL340F780
- EP3SE50F780
- EP3SE80F780
- EP3SE110F780
- EP3SLE260F780.

Example

The following example returns the board information:

```
CHAR szSummary[1024];

GxFpgaGetBoardSummary (nHandle, szSummary, 1024, &nStatus);
```

See Also

GxFpgaInitialize, GxFpgaGetEepromSummary, GxFpgaGetErrorString

GxFpgaGetBoardType

Purpose

Returns the board type.

Syntax

GxFpgaGetBoardType (*nHandle, pnType, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pnType	PSHORT	Returned board type: 0. GXFPGA_UNKNOWN_BOARD_TYPE: unknown board type 1. GXFPGA_BOARD_TYPE_GX3500: board type is GX3500 2. GXFPGA_BOARD_TYPE_GX3500E: board type is GX3500E 3. GXFPGA_BOARD_TYPE_GX3700: board type is GX3700 4. GXFPGA_BOARD_TYPE_GX3700E: board type is GX3700E
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example returns the board type:

```
SHORT nType;
GxFpgaGetBoardType(nHandle, &nType, &nStatus);
```

See Also

GxFpgaInitialize, GxFpgaGetEepromSummary, GxFpgaGetErrorString

GxFpgaGetEepromSummary

Purpose

Returns the timestamp and filename of the last FPGA configuration written to EEPROM.

Syntax

GxFpgaGetEepromSummary (*nHandle, pszSummary, nMaxLen, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pszSummary	PSTR	Buffer to contain a summary indicating last FPGA EEPROM write timestamp and file name.
nMaxLen	SHORT	*pszSummary* buffer size.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function returns the time stamp and file name indicating the last recorded EEPROM loading.

Example

The following example returns the EEPROM summary:

```
CHAR szSummary[1024];

GxFpgaGetEepromSummary (nHandle, szSummary, 1024, &nStatus);
```

See Also

GxFpgaLoad, GxFpgaGetBoardSummary, GxFpgaGetErrorString

GxFpgaGetDriverSummary

Purpose

Returns the driver name and version.

Syntax

GxFpgaGetDriverSummary (*pszSummary*, *nSummaryMaxLen*, *pdwVersion*, *pnStatus*)

Parameters

Name	Type	Comments
pszSummary	PSTR	Buffer to the returned driver summary string.
nSummaryMaxLen	SHORT	The size of the summary string buffer.
pdwVersion	PDWORD	Returned version number. The high order word specifies the major version number where the low order word specifies the minor version number.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The returned string is: "GXFPGA Driver for GX3700. Version 1.00, Copyright © 2009 Marvin Test Solutions – MTS inc.".

Example

The following example prints the driver version:

```
CHAR sz[128];
DWORD dwVersion;
SHORT nStatus;

GxFpgaGetDriverSummary (sz, sizeof sz, &dwVersion, &nStatus);
printf("Driver Version %d.%d", (INT)(dwVersion>>16), (INT)
       dwVersion &0xFFFF);
```

See Also

GxFpgaGetBoardSummary, GxFpgaGetErrorString

GxFpgaGetErrorString

Purpose

Returns the error string associated with the specified error number.

Syntax

GxFpgaGetErrorString (*nError , pszMsg*, *nErrorMaxLen*, *pnStatus*)

Parameters

Name	Type	Comments
nError	SHORT	Error number.
pszMsg	PSTR	Buffer to the returned error string.
nErrorMaxLen	SHORT	The size of the error string buffer.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function returns the error string associated with the *nError* as returned from other driver functions.

The following table displays the possible error values; not all errors apply to this board type:

Resource Errors

0	No error has occurred
-1	Unable to open the HW driver. Check if HW is properly installed
-2	Board does not exist in this slot/base address
-3	Different board exist in the specified PCI slot/base address
-4	PCI slot not configured properly. You may configure using the PciExplorer from the Windows Control Panel
-5	Unable to register the PCI device
-6	Unable to allocate system resource for the device
-7	Unable to allocate memory
-8	Unable to create panel
-9	Unable to create Windows timer
-10	Bad or Wrong board EEPROM
-11	Not in calibration mode
-12	Board is not calibrated
-13	Function is not supported by the specified board

General Parameter Errors

-20	Invalid or unknown error number
-21	Invalid parameter
-22	Illegal slot number
-23	Illegal board handle
-24	Illegal string length
-25	Illegal operation mode

-26 Parameter is out of the allowed range

VISA Errors

-30 Unable to Load VISA32/64.DLL, make sure VISA library is installed

-31 Unable to open default VISA resource manager, make sure VISA is properly installed

-32 Unable to open the specified VISA resource

-33 VISA viGetAttribute error

-34 VISA viInXX error

-35 VISA ViMapAddress error

Miscellaneous Errors

-41 Unable to enable interrupt or event

-42 Unable to disable interrupt or event

-43 Event or interrupt timeout

-44 Event or interrupt wait error

Board Specific Errors

-50 Offset is out of range

-51 File Name is not valid

-52 Programming file could not be opened

-53 User FPGA Volatile Programming error

-54 User FPGA EEPROM Programming error

-55 Cannot program through software, External Programmer Detected

-56 FPGA or EEPROM is currently being loaded and is busy

-57 FPGA could not be reloaded with the EEPROM data

-58 Size and Offset must be multiple of 4

-59 Expansion board required for function not found

-60 FPGA device program failure

-61 Mismatch the data width and number of bytes

-62 Offset must be multiple of 4

-63 Invalid data width, can be 1 byte, 2 bytes or 4 bytes

-64 Invalid DMA data size

-65 Invalid DMA board's offset

-66 Error: timeout when reading using DMA.

-67 Error: timeout when writing using DMA

-70 Invalid time stamp in on-board EEPROM

-71 Error: timeout when reading from the on-board EEPROM

-72 Error: timeout when writing to the on-board EEPROM

Example

The following example initializes the board. If the initialization failed, the following error string is printed:

```
CHAR    sz[256];
SHORT   nStatus, nHandle;
GxFpgaInitialize (3, &Handle, &Status);
if (nStatus<0)
{
    GxFpgaGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz);// prints the error string returns
}
```

GxFpgaGetExpansionBoardID

Purpose

Returns the current Expansion Board ID.

Syntax

GxFpgaGetExpansionBoardID (*nHandle, pucExpansionBoardID, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pucExpansionBoardID	PBYTE	Returned value that identifies the currently installed expansion board.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The returned expansion board ID identifies the type of expansion board being used:

ucExpansionBoardID	Type of board	Examples
0x1	PIO expansion board	GX3701, GX3709, GX3710
0xF	No expansion board installed	N/A

Comments

The expansion board ID is read from P8 pins 19, 21, 23 and 25 to from a 4 bit integer (0-15).

Example

The following example returns the expansion board ID to the *ucExpansionBoardID*:

```
BYTE ucExpansionBoardID;
GxFpgaGetExpansionBoardID (nHandle, &ucExpansionBoardID, &nStatus);
```

See Also

GxFpgaGetErrorString

GxFpgaInitialize

Purpose

Initializes the driver for the board at the specified slot number. The function returns a handle that can be used with other GXFPGA functions to program the board.

Syntax

GxFpgaInitialize (*nSlot, pnHandle, pnStatus*)

Parameters

Name	Type	Comments
nSlot	SHORT	GX3700 board slot number on the PXI bus.
pnHandle	PSHORT	Returned handle for the board. The handle is set to zero on error and <> 0 on success.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The **GxFpgaInitialize** function verifies whether or not the GX3700 board exists in the specified PXI slot. The function does not change any of the board settings. The function uses the HW driver to access and program the board.

The Marvin Test Solutions HW device driver is installed with the driver and is the default device driver. The function returns a handle that for use with other Counter functions to program the board. The function does not change any of the board settings.

The specified PXI slot number is displayed by the **PXI/PCI Explorer** applet that can be opened from the Windows **Control Panel**. You may also use the label on the chassis below the PXI slot where the board is installed. The function accepts two types of slot numbers:

- A combination of chassis number (chassis # x 256) with the chassis slot number. For example 0x105 (chassis 1 slot 5).
- Legacy *nSlot* as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet (1-255).

The returned handle *pnHandle* is used to identify the specified board with other GX3700 functions.

Example

The following example initializes two GX3700 boards at slot 1 and 2.

```
SHORT nHandle1, nHandle2, nStatus;
GxFpgaInitilize (1, &nHandle1, &nStatus);
GxFpgaInitilize (2, &nHandle2, &nStatus);
if (nHandle1==0 || nHandle2==0)
{
    printf("Unable to Initialize the board")
return;
}
```

See Also

GxFpgaInitializeVisa, GxFpgaReset, **GxFpgaGetErrorString**

GxFpgaInitializeVisa

Purpose

Initializes the driver for the specified PXI slot using the default VISA provider.

Syntax

GxFpgaInitializeVisa (*szVisaResource, pnHandle, pnStatus*)

Parameters

Name	Type	Comments
szVisaResource	LPCTSTR	String identifying the location of the specified board in order to establish a session.
pnHandle	PSHORT	Returned Handle (session identifier) that can be used to call any other operations of that resource
pnStatus	PSHORT	Returned status: 0 on success, 1 on failure.

Comments

The **GxFpgaInitializeVisa** opens a VISA session to the specified resource. The function uses the default VISA provider configured in your system to access the board. You must ensure that the default VISA provider support PXI/PCI devices and that the board is visible in the VISA resource manager before calling this function.

The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI Measurement and Automation (NI_MAX). It is also displayed by Marvin Test Solutions PXI/PCI Explorer as shown in the prior figure. The VISA resource string can be specified in several ways as follows:

- Using chassis, slot, for example: "PXI0::CHASSIS1::SLOT5"
- Using the PCI Bus/Device combination, for example: "PXI9::13::INSTR" (bus 9, device 9).
- Using alias, for example: "FPGA1". Use the PXI/PCI Explorer to set the device alias.

The function returns a board handle (session identifier) that can be used to call any other operations of that resource. The session is opened with VI_TMO_IMMEDIATE and VI_NO_LOCK VISA attributes. On terminating the application the driver automatically invokes **viClose**() terminating the session.

Example

The following example initializes a GX3700 boards at PXI bus 5 and device 11.

```
SHORT nHandle, nStatus;
GxFpgaInitializeVisa ("PXI5::11::INSTR", &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

See Also

GxFpgaInitialize, GxFpgaReset, **GxFpgaGetErrorString**

GxFpgaLoad

Purpose

Loads the volatile FPGA or the non-volatile EEPROM with FPGA configuration data in the form of SVF or RPD files respectively.

Syntax

GxFpgaLoad (*nHandle, nTarget, szFileName nMode, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
nTarget	SHORT	Target can be as follows: 0. GXFPGA_LOAD_TARGET_VOLATILE 1. GXFPGA_LOAD_TARGET_EEPROM
szFileName	LPCSTR	Path and file name of the file containing the FPGA configuration data. If the programming mode is Volatile, then the file will have a .SVF extension. If the programming mode is EEPROM, then the file will have an .RPD extension.
nMode	SHORT	The loading mode can be as follows: 0. GXFPGA_LOAD_MODE_SYNC 1. GXFPGA_LOAD_MODE_ASYNC
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function can operate in synchronous mode or asynchronous mode. Synchronous mode means that the function is blocking and does not return until after the load operation has completed. The Asynchronous mode means that the function is non-blocking and returns immediately and allows the calling program to check the load status by calling **GxFpgaLoadStatus.**

Use the **GxFpgaLoadFromEeprom** function to load the volatile memory from the EEPROM. By default, when the card is powered up the volatile memory will be automatically load the configuration from the EEPROM.

Example

The following example loads the volatile FPGA with a Serial Vector File (SVF) in synchronous mode

```
GxFpgaLoad(nHandle, GXFPGA_LOAD_TARGET_VOLATILE, "C:\\MyDesign.SVF", GXFPGA_LOAD_MODE_SYNC
    &nStatus);
```

See Also

GxFpgaLoadStatus, GxFpgaLoadStatusMessage, GxFpgaGetEepromSummary, GxFpgaLoadFromEeprom, GxFpgaGetErrorString

GxFpgaLoadFromEeprom

Purpose

Loads the FPGA with the contents of the EEPROM.

Syntax

GxFpgaLoadFromEeprom (*nHandle, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

By default, when JP2 jumper is present, when the card is powered up the volatile memory will be automatically loaded with the configuration from the EEPROM.

Example

The following example loads the FPGA with the contents of the EEPROM:

```
GxFpgaLoadFromEeprom (nHandle, &nStatus);
```

See Also

GxFpgaLoad, GxFpgaGetErrorString

GxFpgaLoadStatus

Purpose

Returns the progress of the last asynchronous load in percentage.

Syntax

GxFpgaLoadStatus (*nHandle, pnPercentCompleted, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pnPercentCompleted	PSHORT	The percent complete of the current load, 0-100.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

100 percent indicates that the load has completed. This function is used to check the load status after calling **GxFpgaLoad** in Asynchronous mode.

Example

The following load an FPGA file in asynchronous mode and prints the progress:

```
SHORT nPercentage=0, nPriorPrecentage, nStatus, n;
CHAR  szMsg[1024];

GxFpgaLoad(nHandle, GXFPGA_LOAD_TARGET_VOLATILE, "C:\\MyDesign.SVF", GXFPGA_LOAD_MODE_ASYNC
    &nStatus);
while (nStatus==0 && nPrecentage<100)
{   GxFpgaLoadStauts (nHandle, &nPercentage, &nStatus);
    GxFpgaLoadStautsMessage (nHandle, szMsg, sizeof szMsg, &n);
    if (nPrecentage!=nPriorPrecentage)
       printf("Load Complete=%i, Status=%s", nPrecentage, szMsg);
    nPriorPrecentage=nPrecentage;
    sleep(300);
}
printf("Load Complete=%i, Status=%s", nPrecentage, szMsg);
```

See Also

GxFpgaLoad, **GxFpgaLoadStatusMessage, GxFpgaGetErrorString**

GxFpgaLoadStatusMessage

Purpose

Returns a string describes the current load progress of the last asynchronous load.

Syntax

GxFpgaLoadStatusMessage (*nHandle, pszMsg, nMsgMaxLen, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pszMsg	PSTR	A buffer to the returned message describing the current load status.
nMsgMaxLen	SHORT	Size of the pszMsg.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function returns the current load status into the user-supplied buffer. You can use the function to display the status progress and result after calling **GxFpgaLoad** in Asynchronous mode.

Example

The following load an FPGA file in asynchronous mode and prints the progress:

```
SHORT nPercentage=0, nPriorPrecentage, nStatus, n;
CHAR  szMsg[1024];

GxFpgaLoad(nHandle, GXFPGA_LOAD_TARGET_VOLATILE, "C:\\MyDesign.SVF", GXFPGA_LOAD_MODE_ASYNC
    &nStatus);
while (nStatus==0 && nPrecentage<100)
{   GxFpgaLoadStauts (nHandle, &nPercentage, &nStatus);
    GxFpgaLoadStautsMessage (nHandle, szMsg, sizeof szMsg, &n);
    if (nPrecentage!=nPriorPrecentage)
       printf("Load Complete=%i, Status=%s", nPrecentage, szMsg);
    nPriorPrecentage=nPrecentage;
    sleep(300);
}
printf("Load Complete=%i, Status=%s", nPrecentage, szMsg);
```

See Also

GxFpgaLoad, **GxFpgaLoadStatus, GxFpgaGetErrorString**

GxFpgaPanel

Purpose

Opens a virtual panel used to interactively control the GX3700.

Syntax

GxFpgaPanel (pnHandle, hwndParent, nMode, phwndPanel, pnStatus)

Parameters

Name	Type	Comments
pnHandle	PSHORT	Handle to a GX3700 board.
hwndParent	HWND	Panel parent window handle. A value of 0 sets the desktop as the parent window.
nMode	SHORT	The mode in which the panel main window is created. 0 for modeless window and 1 for modal window.
phwndPanel	HWND	Returned window handle for the panel.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function is used to create the panel window. The panel window may be open as a modal or a modeless window depending on the *nMode* parameters.

If the mode is set to modal dialog (*nMode*=1), the panel will disable the parent window (*hwndParent*) and the function will return only after the window was closed by the user. In that case, the *pnHandle* may return the handle created by the user using the panel Initialize dialog. This handle may be used when calling other GXFPGA functions.

If a modeless dialog was created (*nMode*=0), the function returns immediately after creating the panel window returning the window handle to the panel - *phwndPanel.* It is the responsibility of calling program to dispatch windows messages to this window so that the window can respond to messages.

Example

The following example opens the panel in modal mode:

```
DWORD dwPanel;
SHORT nHandle=0, nStatus;

GxFpgaPanel(&nHandle, 0, 1, &dwPanel, &nStatus);
```

See Also

GxFpgaInitialize, GxFpgaGetErrorString

GxFpgaRead

Purpose

Reads the specified number of data elements from the User's FPGA specified BAR memory.

Syntax

GxFpgaRead (*nHandle, nMemoryBar, dwOffset, pvData, nElementSize, dwSize, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
nMemoryBar	SHORT	The board's specified memory mapped address space BAR number, values are as follows: 1. GXFPGA_MEMORY_BAR1: Memory mapped address space BAR 1. 2. GXFPGA_MEMORY_BAR2: Memory mapped address space BAR 2. 3. GXFPGA_MEMORY_BAR3: Memory mapped address space BAR 3. 4. GXFPGA_MEMORY_BAR4: Memory mapped address space BAR 4.
dwOffset	DWORD	The offset in the FPGA's shared memory space in terms of bytes, must be aligned to 4 bytes address.
pvData	PVOID	A buffer that will be written to the FPGA's shared memory. Buffer size must be as indicated by the *dwSize*.
nElementSize	SHORT	The data Size in bytes.
dwSize	DWORD	The number of data elements to be read from the memory location.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example read 100 DWORD data points from BAR2 memory space at offset 8:

```
DWORD adwData[100];
GxFpgaRead (nHandle, GXFPGA_MEMORY_BAR2, 0x8, &adwData, 4, 100, &nStatus);
```

See Also

GxFpgaWrite, GxFpgaWriteRegister, GxFpgaGetErrorString

GxFpgaReadRegister

Purpose

Reads a 32-bit FPGA register.

Syntax

GxFpgaReadRegister (*nHandle, dwOffset, pvData, dwSize, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
dwOffset	DWORD	The offset in the FPGA's register space in terms of bytes, must be aligned to 4 bytes address.
pvData	PVOID	A buffer that will contain the data read. Buffer size must be as indicated by the *dwSize*.
dwSize	DWORD	The number of bytes to be read from the memory location must be multiple of 4.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function will read one or more double words from the FPGA's registers. The offset to be read from must be 4 byte aligned.

The Maximum value of dwOffset is 0x400.

Example

```
DWORD adwData[100];
GxFpgaReadRegister (nHandle, 0x8, &adwData, 400, &nStatus);
```

See Also

GxFpgaWriteRegister, GxFpgaGetErrorString

GxFpgaReset

Purpose

Resets the GX3700 interface FPGA and User FPGA to their default state.

Syntax

GxFpgaReset (*nHandle*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example initializes and resets the GX3700 board:

```
GxFpgaInitialize (1, &nHandle, &nStatus);
GxFpgaReset (nHandle, &nStatus);
```

See Also

GxFpgaInitialize, **GxFpgaGetErrorString**

GxFpgaSetEvent

Purpose

Enables or disables an event handler.

Syntax

GxFpgaSetEvent (*nHandle, nEventType, bEnable, procCallback, pvUserData, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
nEventType	SHORT	Event type. Use the constant GT_EVENT_INTERRUPT (1). No other value is supported.
bEnable	BOOL	Enable (<>0) or disable (0) the event.
procCallback	PROCEDURE	Optional. User supplied procedure, called by the driver when an event occurred.
pvUserData	PVOID	User data (pointer or value) that is passed to the callback procedure when an event occurred.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

If NULL is passed in to the *procCallback* parameter, the only way to get notified that an event has occurred is to call the **GxFpgaWaitOnEvent** function.

The *procCallback* should be defined as follows:

GxFpgaCallback (*nHandle, nEventType,,pvUserData, pnStatus*) : Long

Example

The following example output whether an event received during 1 second:

```
GxFpgaInitialize (1, &nHandle, &nStatus);
GxFpgaSetEvent(nHandle, GT_EVENT_INTERRUPT, TRUE, NULL, (PVOID)1, &nStatus);
! wait up to 1000 ms for the event
GxFpgaWaitOnEvent(nHandle, GT_EVENT_INTERRUPT, 1000, &nStatus);
if (nStatus==0)        ! success event occurred
    printf("event occurred");
else
    printf("No event occurred");
GxFpgaSetEvent(nHandle, GT_EVENT_INTERRUPT, FALSE, NULL, (PVOID)1, &nStatus);
```

See Also

GxFpgaInitialize, **GxFpgaGetErrorString, GxFpgaWaitOnEvent, GxFpgaDiscardEvents**

GxFpgaUpgradeFirmware

Purpose

Upgrades the board's firmware.

Syntax

GxFpgaUpgradeFirmware (*nHandle, szFile, nMode, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
szFile	PCSTR	Path and file name of the firmware file. The firmware file extension is RPD.
nMode	SHORT	The upgrading firmware mode can be as follows: 0. GT_FIRMWARE_UPGRADE_MODE_SYNC: the function returns when upgrading firmware is done or in case of an error. 1. GT_ FIRMWARE_UPGRADE_MODE_ASYNC: the function returns immediately. The user can monitor the progress of upgrading firmware using the **GxFpgaUpgradeFirmwareStatus** API.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function used in order to upgrade the board's firmware. The firmware file can only be obtained by request from Marvin Test Solutions.

Note: Loading an incorrect firmware file to the board can permanently damage the board.

Example

The following example loads Upgrades the board's firmware using synchronous mode:

```
GxFpgaUpgradeFirmware (nHandle, "C:\\Gx3700Fw.rpd", GT_LOAD_MODE_SYNC, &nStatus);
```

See Also

GxFpgaUpgradeFirmwareStatus, GxFpgaGetErrorString

GxFpgaUpgradeFirmwareStatus

Purpose

Monitor the firmware upgrade process.

Syntax

GxFpgaUpgradeFirmwareStatus (*nHandle, pszMsg, nMsgMaxLen, pnProgress, pbIsDone, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
pszMsg	PSTR	Buffer to contain the message from the firmware upgrade process.
nMsgMaxLen	SHORT	*pszMsg* buffer size.
pnProgress	PSHORT	Returns the firmware upgrades progress.
pbIsDone	PBOOL	Returned TRUE if the firmware upgrades are done.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function is used in order to monitor the firmware upgrade process whenever the user called **GxFpgaUpgradeFirmware** API with GT_ FIRMWARE_UPGRADE_MODE_ASYNC mode.

Note: In order to prevent CPU over load if the function is called form within a loop, a delay of about 500mSec will be activated if the time differences between consecutive calls are less than 500mSec.

Example

The following example loads Upgrades the board's firmware using asynchronous mode, and ten monitors the firmware upgrade process:

```
CHAR    sz[256];
CHAR    szMsg[256];
BOOL    bIsDone=FALSE;
GxFpgaUpgradeFirmware (nHandle, "C:\\Gx3700Fw.rpd", GT_UPGRADE_FIRMWARE_MODE_ASYNC, &nStatus);
if (nStatus<0)
{   GxFpgaGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz);// prints the error string returns
}
While (bIsDone==FALSE || nStatus<0)
{   GxFpgaUpgradeFirmwareStatus (nHandle, szMsg, sizeof szMsg, &nProgress, &bIsDone, &nStatus);
    printf("Upgrade Progress %i", nProgress);
    sleep(1000);
}
if (nStatus<0)
{   GxFpgaGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz);// prints the error string returns
}
```

See Also

GxFpgaUpgradeFirmware, GxFpgaGetErrorString

GxFpgaWaitOnEvent

Purpose

Waits until event received or timeout occurred.

Syntax

GxFpgaWaitOnEvent (*nHandle, nEventType, lTimeout, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
nEventType	SHORT	Event type. Use the constant GT_EVENT_INTERRUPT (1). No other value is supported.
lTimeout	LONG	Timeout to wait in mill seconds.
pnStatus	PSHORT	Returned status: 0 on success (event occurred), negative number on failure.

Comments

The function suspends the current thread until an event occurred or until the specified timeout expired.

Example

The following example output whether an event received during 1 second:

```
GxFpgaInitialize (1, &nHandle, &nStatus);
GxFpgaSetEvent(nHandle, GT_EVENT_INTERRUPT, TRUE, NULL, (PVOID)1, &nStatus);
! wait up to 1000 ms for the event
GxFpgaWaitOnEvent(nHandle, GT_EVENT_INTERRUPT, 1000, &nStatus);
if (nStatus==0)        ! success event occurred
    printf("event occurred");
else if (nStatus==GT_EVENT_WAIT_TIMEOUT)
    printf("No event occurred (timeout)");
else
    printf("Event error");
GxFpgaSetEvent(nHandle, GT_EVENT_INTERRUPT, FALSE, NULL, (PVOID)1, &nStatus);
```

See Also

GxFpgaInitialize, **GxFpgaGetErrorString, GxFpgaSetEvent, GxFpgaDiscardEvents**

GxFpgaWrite

Purpose

Writes the specified number of data elements to the User's FPGA specified BAR memory.

Syntax

GxFpgaWrite (*nHandle, nMemoryBar, dwOffset, pvData, nElementSize, dwSize, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
nMemoryBar	SHORT	The board's specified memory mapped address space BAR number, values are as follows: 1. GXFPGA_MEMORY_BAR1: Memory mapped address space BAR 1. 2. GXFPGA_MEMORY_BAR2: Memory mapped address space BAR 2. 3. GXFPGA_MEMORY_BAR3: Memory mapped address space BAR 3. 4. GXFPGA_MEMORY_BAR4: Memory mapped address space BAR 4.
dwOffset	DWORD	The offset of User's FPGA memory space in bytes.
pvData	PVOID	A buffer that will be written to the FPGA's shared memory. Buffer size must be as indicated by the *dwSize*.
nElementSize	SHORT	The data Size in bytes.
dwSize	DWORD	The number of data elements to write to the memory location.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example writes 100 DWORD to the User's FPGA BAR1 memory to begin at offset 8:

```
DWORD adwData[100];
GxFpgaWrite (nHandle, GXFPGA_MEMORY_BAR1, 0x8, &adwData, 4, 100, &nStatus);
```

See Also

GxFpgaRead, GxFpgaWriteRegister, GxFpgaGetErrorString

GxFpgaWriteRegister

Purpose

Writes a buffer of 32 bit double words to the FPGA's register space.

Syntax

GxFpgaWriteRegister (*nHandle, dwOffset, pvData, dwSize, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3700 board.
dwOffset	DWORD	The offset in the FPGA's register space in terms of bytes, must be aligned to 4 bytes address.
pvData	PDWORD	A buffer that will be written to the FPGA's registers. Buffer size must be as indicated by the *dwSize*.
dwSize	DWORD	The number of bytes to be written to the registers must be multiple of 4.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function will write one or more double words to the FPGA's registers. The offset to be written to must be 4-byte aligned

The Maximum value of dwOffset is 0x400.

Example

The following example writes 400 bytes to the card register space at offset 8:

```
DWORD adwData[100];
GxFpgaWriteRegister (nHandle, 0x8, &adwData, 400, &nStatus);
```

See Also

GxFpgaReadRegister, GxFpgaGetErrorString

Gx3788Initialize

Purpose

Initializes the driver for the board at the specified slot number. The function returns a handle that can be used with other GX3788 function.

Syntax

Gx3788Initialize (*nSlot, pnHandle, pnStatus*)

Parameters

Name	Type	Comments
nSlot	SHORT	GX3788 board slot number on the PXI bus.
pnHandle	PSHORT	Returned handle for the board. The handle is set to zero on error and <> 0 on success.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The **Gx3788Initialize** function verifies whether or not the GX3788 board exists in the specified PXI slot. The function does not change any of the board settings. The function uses the HW driver to access and program the board.

The Marvin Test Solutions HW device driver is installed with the driver and is the default device driver. The function returns a handle that for use with other Counter functions to program the board. The function does not change any of the board settings.

The specified PXI slot number is displayed by the **PXI/PCI Explorer** applet that can be opened from the Windows **Control Panel**. You may also use the label on the chassis below the PXI slot where the board is installed. The function accepts two types of slot numbers:

- A combination of chassis number (chassis # x 256) with the chassis slot number. For example, 0x105 (chassis 1 slot 5).
- Legacy *nSlot* as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet (1-255).

The returned handle *pnHandle* is used to identify the specified board with other GX3788 functions.

Example

The following example initializes two GX3788 boards at slot 1 and 2.

```
SHORT nHandle1, nHandle2, nStatus;
Gx3788Initilize (1, &nHandle1, &nStatus);
Gx3788Initilize (2, &nHandle2, &nStatus);
if (nHandle1==0 || nHandle2==0)
{
    printf("Unable to Initialize the board")
return;
}
```

See Also

Gx3788InitializeVisa, Gx3788Reset, **GxFpgaGetErrorString**

Gx3788InitializeVisa

Purpose

Initializes the driver for the specified PXI slot using the default VISA provider.

Syntax

Gx3788InitializeVisa (*szVisaResource, pnHandle, pnStatus*)

Parameters

Name	Type	Comments
szVisaResource	LPCTSTR	String identifying the location of the specified board in order to establish a session.
pnHandle	PSHORT	Returned Handle (session identifier) that can be used to call any other operations of that resource
pnStatus	PSHORT	Returned status: 0 on success, 1 on failure.

Comments

The **Gx3788InitializeVisa** opens a VISA session to the specified resource. The function uses the default VISA provider configured in your system to access the board. You must ensure that the default VISA provider support PXI/PCI devices and that the board is visible in the VISA resource manager before calling this function.

The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI Measurement and Automation (NI_MAX). It is also displayed by Marvin Test Solutions PXI/PCI Explorer as shown in the prior figure. The VISA resource string can be specified in several ways as follows:

- Using chassis, slot, for example: "PXI0::CHASSIS1::SLOT5"
- Using the PCI Bus/Device combination, for example: "PXI9::13::INSTR" (bus 9, device 9).
- Using alias, for example: "FPGA1". Use the PXI/PCI Explorer to set the device alias.

The function returns a board handle (session identifier) that can be used to call any other operations of that resource. The session is opened with VI_TMO_IMMEDIATE and VI_NO_LOCK VISA attributes. On terminating the application the driver automatically invokes **viClose**() terminating the session.

Example

The following example initializes a GX3788 boards at PXI bus 5 and device 11.

```
SHORT nHandle, nStatus;
Gx3788InitializeVisa ("PXI5::11::INSTR", &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

See Also

Gx3788Initialize, Gx3788Reset, **GxFpgaGetErrorString**

Gx3788Reset

Purpose

Resets the GX3788 to their default state.

Syntax

Gx3788Reset (*nHandle*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function will reset the following settings:

- Digital Port Data to 0x0
- Digital Port Direction to 0x0 (input)
- Analoug Output Channel Voltages to 0.0 V

Example

The following example initializes and resets the GX3788 board:

```
Gx3788Initialize (1, &nHandle, &nStatus);
Gx3788Reset (nHandle, &nStatus);
```

See Also

Gx3788Initialize, **GxFpgaGetErrorString**

Gx3788GetBoardSummary

Purpose

Returns the board information.

Syntax

Gx3788GetBoardSummary (*nHandle, pszSummary, nMaxLen, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pszSummary	PSTR	Buffer to contain the returned board info (null terminated) string.
nMaxLen	SHORT	*pszSummary* buffer size.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The function returns the board information of the Gx3788 which includes the FPGA version, and serial number.

Example

The following example returns the board information:

```
CHAR szSummary[1024];
Gx3788GetBoardSummary (nHandle, szSummary, 1024, &nStatus);
```

See Also

Gx3788Initialize, GxFpgaGetErrorString, Gx3788GetCalibrationInfo

Gx3788GetCalibrationInfo

Purpose

Returns the calibration information.

Syntax

Gx3788GetCalibrationInfo (*nHandle, pszCalibrationInfo, nInfoMaxLen, pnDaysUntilExpire, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	**Handle to a GX3788 board**
pszCalibrationInfo	PSTR	Buffer to contain the returned board's calibration information (null terminated) string.
nInfoMaxLen	SHORT	Size of the buffer to contain the calibration information string.
pnDaysUntilExpire	PSHORT	Returns the number of days until or from expiration, if number is > 0 then calibration is current otherwise past due.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The returned board's calibration information has the following fields:

Model: model number, e.g. "GX3788"

Serial Number: serial number, e.g. 37880016

Control Number: Marvin Test Solutions control number, e.g. "BB-BA-00"

Production Calibration Date: Wed Oct 24 12:30:25 2010

Calibration Date: Wed Oct 24 12:31:58 2010

Recommended Interval: 1 year

Next Calibration Date: Fri Oct 24 12:31:58 2011

Status: calibration status can be either "Expired" followed by the number of days past expiration or "Current" followed by number of days until expire.

Calibration License: can be either "Installed" with the calibration license number or "Not Installed".

Example

The following example returns the calibration information:

```
CHAR szInfo[2048];
SHORT nDays, nStatus;
Gx3788GetCalibrationInfo (nHandle, sz, 2048, &nDays, &nStatus);
```

See Also

Gx3788Initialize, GxFpgaGetErrorString, Gx3788GetBoardSummary

Gx3788AnalogInGetGroundSource

Purpose

Returns the analog input ground source

Syntax

Gx3788AnalogInGetGroundSource (*nHandle, nChannels, pnGroundSource, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nChannels	SHORT	Select analog in channel group to query: 0. GX3788_ANALOG_IN_CHANNELS_0_7: Channel group 0 to 7 1. GX3788_ANALOG_IN_CHANNELS_8_15: Channel group 8 ot 15
pnGroundSource	PSHORT	Returns the analog input Ground Source: 0. GX3788_ANALOG_IN_DIGITAL_GND: Digital Ground 1. GX3788_ANALOG_IN_ANALOG_GND: Analog Ground
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the ground source of analog channels 0 through 7:

```
Gx3788AnalogInGetGroundSource (nHandle, GX3788_ANALOG_IN_CHANNELS_0_7, &nGroundSource, &nStatus);
```

See Also

Gx3788AnalogInSetGroundSource, **GxFpgaGetErrorString**

Gx3788AnalogInMeasureChannel

Purpose

Measures the voltage on a particular analog channel

Syntax

Gx3788AnalogInMeasureChannel (*nHandle, nMode, nChannel, nVoltageRange, pdVoltage, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nMode	SHORT	Select analog in channel group to query: 0. GX3788_ANALOG_IN_DIFFERENTIAL 1. GX3788_ANALOG_IN_SINGLE_ENDED
nChannel	SHORT	Selects the analog input channel to measure Use the following constants when using single ended mode: 0. GX3788_ANALOG_IN_0 1. GX3788_ANALOG_IN_1 2. GX3788_ANALOG_IN_2 3. GX3788_ANALOG_IN_3 4. GX3788_ANALOG_IN_4 5. GX3788_ANALOG_IN_5 6. GX3788_ANALOG_IN_6 7. GX3788_ANALOG_IN_7 8. GX3788_ANALOG_IN_8 9. GX3788_ANALOG_IN_9 10. GX3788_ANALOG_IN_10 11. GX3788_ANALOG_IN_11 12. GX3788_ANALOG_IN_12 13. GX3788_ANALOG_IN_13 14. GX3788_ANALOG_IN_14 15. GX3788_ANALOG_IN_15 Use the following constants when using differential mode: 0. GX3788_ANALOG_IN_DIFF_0_AND_1 1. GX3788_ANALOG_IN_DIFF_2_AND_3 2. GX3788_ANALOG_IN_DIFF_4_AND_5 3. GX3788_ANALOG_IN_DIFF_6_AND_7 4. GX3788_ANALOG_IN_DIFF_8_AND_9 5. GX3788_ANALOG_IN_DIFF_10_AND_11 6. GX3788_ANALOG_IN_DIFF_12_AND_13 7. GX3788_ANALOG_IN_DIFF_14_AND_15
nVoltageRange	SHORT	Returns the analog input Ground Source: 0. GX3788_ANALOG_IN_RANGE_NEG_13p60V_TO_POS_13p60V 1. GX3788_ANALOG_IN_RANGE_NEG_10p24V_TO_POS_10p24V 2. GX3788_ANALOG_IN_RANGE_NEG_5p12V_TO_POS_5p12V 3. GX3788_ANALOG_IN_RANGE_NEG_2p56V_TO_POS_2p56V 4. GX3788_ANALOG_IN_RANGE_NEG_1p28V_TO_POS_1p28V

		5. GX3788_ANALOG_IN_RANGE_NEG_0p64V_TO_POS_0p64V
pdVoltage	PDOUBLE	Returns the measured voltage
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

Each analog in channel has the ability to be measured for voltage. The detection circuit can be set to use one of 7 voltage ranges (**nVoltageRange** parameter). The measurement can also be taken as a single ended input or a differential pair of inputs (**nMode** parameter). When the differential mode is used, a pair of analog input channels are used together to provide the differential input. The differential pairs are defined as channels 0 and 1, channels 2 and 3, etc. For example, if the **GX3788_ANALOG_IN_DIFFERENTIAL** constant is passed to the **nMode** and the **GX3788_ANALOG_IN_DIFF_4_AND_5** contant is passed in to the **nChannel** parameter, the differential pair will be channels 4 and 5.

Example

The following example gets the ground source of analog channels 0 through 7:

```
Gx3788AnalogInMeasureChannel (nHandle, GX3788_ANALOG_IN_SINGLE_ENDED, 0,
    GX3788_ANALOG_IN_RANGE_NEG_10p24V_TO_POS_10p24V, &dVoltage, &nStatus);
printf("Analog In Channel 0 measurement = %f Voltage", dVoltage);
```

See Also

Gx3788Initialize, Gx3788Reset, **GxFpgaGetErrorString**

Gx3788AnalogInScanGetChannelListIndex

Purpose

Return scan channel list entry

Syntax

Gx3788AnalogInScanGetChannelListIndex (*nHandle, dwScanChannelIndex, pdwChannel, pnRange, pnMode, pbIsLastChannel, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
dwScanChannelIndex	DWORD	Index of entry in channel list to return (0-63)
pdwChannel	PDWORD	Returns the channel number stored at the selected index in the scan channel list (0-15)
pnRange	PSHORT	Returns the range of the input channel stored at the selected index in the scan channel list 0. GX3788_ANALOG_IN_RANGE_NEG_13p60V_TO_POS_13p60V 1. GX3788_ANALOG_IN_RANGE_NEG_10p24V_TO_POS_10p24V 2. GX3788_ANALOG_IN_RANGE_NEG_5p12V_TO_POS_5p12V 3. GX3788_ANALOG_IN_RANGE_NEG_2p56V_TO_POS_2p56V 4. GX3788_ANALOG_IN_RANGE_NEG_1p28V_TO_POS_1p28V 5. GX3788_ANALOG_IN_RANGE_NEG_0p64V_TO_POS_0p64V
pnMode	PSHORT	Returns the mode of the input channel stored at the selected index in the scan channel list 0. GX3788_ANALOG_IN_DIFFERENTIAL 1. GX3788_ANALOG_IN_SINGLE_ENDED
pbIsLastChannel	PBOOL	Returns the last channel flag stored at the selected index in the scan channel list
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The analog in scan operation requires that a channel list be set up prior to initiating the operation. This function allows the user to create a channel list, including the analog in channel number, voltage range, and mode. The channel list is then used by the sequencer to acquire samples (one per channel that is defined in the channel list) for each sample clock period. Note that the same channel number can be repeated in a channel list, resulting in mulitple samples being taken on the same channel within a sample clock period. Use the **bIsLastChannel** parameter to indicate which channel list index should be considered by the sequencer to be the last.

Example

The following example gets the analog in scan channel list at index 4:

```
Gx3788AnalogInScanSetChannelListIndex (nHandle, 4, &nRange, &nMode, &bIsLastChannel, &nStatus);
```

See Also

Gx3788AnalogInScanSetChannelListIndex, Gx3788AnalogInScanSetCount, **Gx3788AnalogInScanGetCount , GxFpgaGetErrorString**

Gx3788AnalogInScanGetCount

Purpose

Returns the analog input measurement count

Syntax

Gx3788AnalogInScanGetCount (*nHandle, pdwMeasureCount, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pdwMeasureCount	PDWORD	Returns the number of voltage measurements to take during a scan operation
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The scan count is the number of sample clock periods that will be executed. During each clock period, the sequencer will capture samples for each channel defined in the channel list. The channel list can be modified by calling **Gx3788AnalogInScanSetChannelListIndex**

Example

The following example gets the analog in scan count:

```
Gx3788AnalogInScanGetCount (nHandle, &dwMeasurementCount, &nStatus);
```

See Also

Gx3788AnalogInScanSetCount, **GxFpgaGetErrorString**

Gx3788AnalogInScanGetLastRunCount

Purpose

Returns the analog input measurement count from the last scan operation

Syntax

Gx3788AnalogInScanGetLastRunCount (*nHandle, pdwMeasureCount, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pdwMeasureCount	PDWORD	Returns the number of voltage measurements taken during the last scan operation
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the analog in scan count of the last scan operation:

```
Gx3788AnalogInScanGetLastRunCount (nHandle, &dwMeasurementCount, &nStatus);
```

See Also

Gx3788AnalogInScanSetCount, **Gx3788AnalogInScanGetCount, GxFpgaGetErrorString**

Gx3788AnalogInScanGetSampleRate

Purpose

Returns the analog input measurement sample rate

Syntax

Gx3788AnalogInScanGetSampleRate (*nHandle*, *pdSampleRate*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pdSampleRate	PDOUBLE	Returns the sample rate of the analog in scan operation in Hz
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the analog in scan rate:

```
Gx3788AnalogInScanGetSampleRate (nHandle, &dSampleRate, &nStatus);
```

See Also

Gx3788AnalogInScanSetSampleRate, **GxFpgaGetErrorString**

Gx3788AnalogInScanIsRunning

Purpose

Returns the analog input measurement sample rate

Syntax

Gx3788AnalogInScanIsRunning (*nHandle*, *pbIsRunning*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pbIsRunning	PBOOL	Return TRUE if an analog in scan operation is in progress and FALSE if it is not
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example checks if a scan operation is in progress:

```
Gx3788AnalogInScanIsRunning (nHandle, &bRunning, &nStatus);
if (bRunning)
    printf("Analog In Scan in progress...");
else
    printf("Analog In Scan not running");
```

See Also

Gx3788AnalogInScanStart, **GxFpgaGetErrorString**

Gx3788AnalogInScanReadMemoryRawData

Purpose

Read recorded voltage samples from the last scan operation

Syntax

Gx3788AnalogInScanReadMemoryRawData (*nHandle*, *dwMemoryStart*, *dwCount*, *pawData*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
dwMemoryStart	DWORD	The starting offset in memory for the scan sample memory read operation
dwCount	DWORD	The number of samples to read from memory
pawData	PWORD	Return the samples in the form of raw samples within an array
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The sample memory contains the captured voltage samples after a scan operation has completed. This function returns the samples in their raw 16-bit form. Each set of sample(s) (acquired from one or more channels as defined in the channel list), are stored sequentially in memory. For example, if the channel list defines channels 6, 8, 3, and 4, and the sample count was set to 3, then the sample memory will contain voltage samples in the following order:

6,8,3,4...6,8,3,4...6,8,3,4.

Example

The following example reads 10 samples from memory start at offset 0:

```
WORD awData[10];
Gx3788AnalogInScanReadMemoryRawData (nHandle, 0, 10, awData, &nStatus);
```

See Also

Gx3788AnalogInScanSetChannelListIndex, Gx3788AnalogInScanSetCount, Gx3788AnalogInScanStart GxFpgaGetErrorString

Gx3788AnalogInScanReadMemoryVoltages

Purpose

Read recorded voltage samples from the last scan operation

Syntax

Gx3788AnalogInScanReadMemoryVoltages (*nHandle, dwMemoryStart, dwCount, padData, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
dwMemoryStart	DWORD	The starting offset in memory for the scan sample memory read operation
dwCount	DWORD	The number of samples to read from memory
padData	PDOUBLE	Return the samples in the form of voltages within an array
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The sample memory contains the captured voltage samples after a scan operation has completed. This function returns the samples in the form of voltages. Each set of sample(s) (acquired from one or more channels as defined in the channel list), are stored sequentially in memory. For example, if the channel list defines channels 6, 8, 3, and 4, and the sample count was set to 3, then the sample memory will contain voltage samples in the following order:

6,8,3,4...6,8,3,4...6,8,3,4.

Example

The following example reads 10 samples from memory start at offset 0:

```
DOUBLE adData[10];
Gx3788AnalogInScanReadMemoryVoltages (nHandle, 0, 10, adData, &nStatus);
```

See Also

Gx3788AnalogInScanSetChannelListIndex, Gx3788AnalogInScanSetCount, Gx3788AnalogInScanStart
GxFpgaGetErrorString

Gx3788AnalogInScanSetChannelListIndex

Purpose

Modify scan channel list

Syntax

Gx3788AnalogInScanSetChannelListIndex (*nHandle, dwScanChannelIndex, dwChannel, nRange, nMode, bIsLastChannel, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
dwScanChannelIndex	DWORD	Index in channel list to modify (0-63)
dwChannel	DWORD	Channel number to add to the scan channel list (0-15)
nRange	SHORT	Sets the range of the input channel at the selected index in the scan channel list 0. GX3788_ANALOG_IN_RANGE_NEG_13p60V_TO_POS_13p60V 1. GX3788_ANALOG_IN_RANGE_NEG_10p24V_TO_POS_10p24V 2. GX3788_ANALOG_IN_RANGE_NEG_5p12V_TO_POS_5p12V 3. GX3788_ANALOG_IN_RANGE_NEG_2p56V_TO_POS_2p56V 4. GX3788_ANALOG_IN_RANGE_NEG_1p28V_TO_POS_1p28V 5. GX3788_ANALOG_IN_RANGE_NEG_0p64V_TO_POS_0p64V
nMode	SHORT	Sets the mode of the input channel at the selected index in the scan channel list 0. GX3788_ANALOG_IN_DIFFERENTIAL 1. GX3788_ANALOG_IN_SINGLE_ENDED
bIsLastChannel	BOOL	Marks the channel as the last in the channel list at the selected channel index
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The analog in scan operation requires that a channel list be set up prior to initiating the operation. This function allows the user to create a channel list, including the analog in channel number, voltage range, and mode. The channel list is then used by the sequencer to acquire samples (one per channel that is defined in the channel list) for each sample clock period. Note that the same channel number can be repeated in a channel list, resulting in mulitple samples being taken on the same channel within a sample clock period. Use the **bIsLastChannel** parameter to indicate which channel list index should be considered by the sequencer to be the last.

Example

The following example sets the analog in scan channel list at index 4 to analog in channel 5 with a range of +/- 13.60V, and single ended mode:

```
Gx3788AnalogInScanSetChannelListIndex (nHandle, 4,
    GX3788_ANALOG_IN_RANGE_NEG_13p60V_TO_POS_13p60V, GX3788_ANALOG_IN_SINGLE_ENDED, FALSE,
    &nStatus);
```

See Also

Gx3788AnalogInScanGetChannelListIndex, Gx3788AnalogInScanSetCount, **GxFpgaGetErrorString**

Gx3788AnalogInScanSetCount

Purpose

Sets the analog input measurement count

Syntax

Gx3788AnalogInScanSetCount (*nHandle, dwMeasureCount, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
dwMeasureCount	DWORD	Sets the number of voltage measurements to take during a scan operation
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

The scan count is the number of sample clock periods that will be executed. During each clock period, the sequencer will capture samples for each channel defined in the channel list. The channel list can be modified by calling **Gx3788AnalogInScanSetChannelListIndex**

Example

The following example sets the analog in scan count to 5:

```
Gx3788AnalogInScanSetCount (nHandle, 5, &nStatus);
```

See Also

Gx3788AnalogInScanGetCount, **GxFpgaGetErrorString**

Gx3788AnalogInScanSetSampleRate

Purpose

Sets the analog input measurement sample rate

Syntax

Gx3788AnalogInScanSetSampleRate (*nHandle, dSampleRate, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
dSampleRate	DOUBLE	Sets the sample rate of the analog in scan operation in Hz
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the analog in scan rate to 100 Hz:

```
Gx3788AnalogInScanSetSamppleRate (nHandle, 100, &nStatus);
```

See Also

Gx3788AnalogInScanGetSampleRate, **GxFpgaGetErrorString**

Gx3788AnalogInScanStart

Purpose

Starts an analog in scan operation

Syntax

Gx3788AnalogInScanStart (*nHandle, nScanMode, dwMemoryStart, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nScanMode	SHORT	Sets the scan operation mode: 0. GX3788_ANALOG_IN_DIFFERENTIAL: Diffrential mode 1. GX3788_ANALOG_IN_SINGLE_ENDED: Single ended mode
dwMemoryStart	DWORD	Sets the address in sample memory to start from
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example starts an analog in scan operation using single ended mode, starting at memory address 0:

```
Gx3788AnalogInScanStart (nHandle, GX3788_ANALOG_IN_SINGLE_ENDED, 0, &nStatus);
```

See Also

Gx3788AnalogInScanSetSampleRate, Gx3788AnalogInScanGetSampleRate, **GxFpgaGetErrorString**

Gx3788AnalogInSetGroundSource

Purpose

Sets the analog input ground source

Syntax

Gx3788AnalogInSetGroundSource (*nHandle*, *nChannels*, *nGroundSource*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nChannels	SHORT	Select analog in channel group to query: 0. GX3788_ANALOG_IN_CHANNELS_0_7: Channel group 0 to 7 1. GX3788_ANALOG_IN_CHANNELS_8_15: Channel group 8 ot 15
nGroundSource	SHORT	Analog input Ground Source: 0. GX3788_ANALOG_IN_DIGITAL_GND: Digital Ground 1. GX3788_ANALOG_IN_ANALOG_GND: Analog Ground
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the ground source of analog channels 0 through 7 to digital ground:

```
Gx3788AnalogInSetGroundSource (nHandle, GX3788_ANALOG_IN_CHANNELS_0_7,
    GX3788_ANALOG_IN_DIGITAL_GND, &nStatus);
```

See Also

Gx3788AnalogInGetGroundSource, **GxFpgaGetErrorString**

Gx3788AnalogOutGetOutputState

Purpose

Returns the analog output channel state

Syntax

Gx3788AnalogOutGetOutputState (*nHandle, pbOutputEnable, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pbOutputEnable	PBOOL	Returns the analog output state. 0. GX3788_ANALOG_OUT_ENABLE: All the analog ouput channels are enabled (driving voltage). 1. GX3788_ANALOG_OUT_DISABLE: All the analog output channels are disabled (not driving voltage).
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the analog output channel state:

```
Gx3788AnalogOutGetOutputState (nHandle, &bAnalogOutputState, &nStatus);
```

See Also

Gx3788AnalogOutSetOutputState, Gx3788AnalogOutSetVoltage, Gx3788AnalogOutGetVoltage, **GxFpgaGetErrorString**

Gx3788AnalogOutGetVoltage

Purpose

Returns the analog output channel voltage

Syntax

Gx3788AnalogOutGetVoltage (*nHandle, nChannel, pdVoltage, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nChannel	SHORT	Selects analog output channel to set (0-7)
pdVoltage	PDOUBLE	Returned voltage setting of the selected channel
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the analog output channel 4 voltage:

```
Gx3788AnalogOutGetVoltage (nHandle, 4, &dVoltage, &nStatus);
```

See Also

Gx3788AnalogOutSetVoltage, **GxFpgaGetErrorString**

Gx3788AnalogOutReset

Purpose

Sets all the analog output channels to default settings

Syntax

Gx3788AnalogOutReset (*nHandle*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function sets all analog output channels to 0 volts and disables all analog output channels.

Example

The following example resets analog output channels:

```
Gx3788AnalogOutReset (nHandle, &nStatus);
```

See Also

Gx3788AnalogOutSetVoltage, Gx3788AnalogOutGetVoltage, **GxFpgaGetErrorString**

Gx3788AnalogOutSetOutputState

Purpose

Sets the analog output channel state

Syntax

Gx3788AnalogOutSetOutputState (*nHandle, bOutputEnable, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
bOutputEnable	BOOL	Sets the analog output state. 0. GX3788_ANALOG_OUT_ENABLE: All the analog ouput channels are enabled (driving voltage). 1. GX3788_ANALOG_OUT_DISABLE: All the analog output channels are disabled (not driving voltage).
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example enables all the analog output channels:

```
Gx3788AnalogOutSetOutputState (nHandle, GX3788_ANALOG_OUT_ENABLE, &nStatus);
```

See Also

Gx3788AnalogOutGetOutputState, Gx3788AnalogOutSetVoltage, Gx3788AnalogOutGetVoltage, **GxFpgaGetErrorString**

Gx3788AnalogOutSetVoltage

Purpose

Sets the analog output channel voltage

Syntax

Gx3788AnalogOutSetVoltage (*nHandle, nChannel, dVoltage, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nChannel	SHORT	Selects analog output channel to set (0-7)
dVoltage	DOUBLE	Voltage to set the analog output channel
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the analog output channel 4 to 6.5 volts:

```
Gx3788AnalogOutSetVoltage (nHandle, 4, 6.5, &nStatus);
```

See Also

Gx3788AnalogOutGetVoltage, **GxFpgaGetErrorString**

Gx3788PioGetPort

Purpose

Returns the output states of a selected digital port

Syntax

Gx3788PioGetPort (*nHandle, nPort, pdwValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to get: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
pdwValue	PDWORD	Returns the output states of the selected digital port, each bit corresponds to a channel, when the channel is in high state - 1 will be returned for that channel/bit, low state – 0 will returned
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the digital port 0 output states:

```
Gx3788PioGetPort (nHandle, GX3788_PIO_PORT0, &dwValue, &nStatus);
```

See Also

Gx3788PioSetPort, **GxFpgaGetErrorString**

Gx3788PioGetPortChannel

Purpose

Returns the output state of a selected digital port

Syntax

Gx3788PioGetPortChannel (*nHandle, nPort, nChannel, pbValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to get: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
nChannel	SHORT	Selects the channel within the selected port to query (0-31)
pbValue	PBOOL	Returns the output state of the selected digital channel 0. FALSE: Digital low level 1. TRUE: Digital high level
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the digital port 0, channel 15 output state:

```
Gx3788PioGetPortChannel (nHandle, GX3788_PIO_PORT0, 15, &pbValue, &nStatus);
```

See Also

Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, Gx3788PioSetPort, GxFpgaGetErrorString

Gx3788PioGetPortChannelDirection

Purpose

Returns the digital channel direction state

Syntax

Gx3788PioGetPortChannelDirection (*nHandle, nPort, nChannel, pbValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to get: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
nChannel	SHORT	Selects the channel within the selected port to set (0-31)
pbValue	PBOOL	Returns the direction setting of a selected digital channel 0. FALSE: Digital channel is an input 1. TRUE: Digital channel is an output
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the digital port 0, channel 15 direction setting:

```
Gx3788PioGetPortChannelDirection (nHandle, GX3788_PIO_PORT0, 15, &bDirection, &nStatus);
```

See Also

Gx3788PioSetPortChannelDirection, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, Gx3788PioSetPort, Gx3788PioGetPort, GxFpgaGetErrorString

Gx3788PioGetPortDirection

Purpose

Returns the direction (input or output) settings of a selected digital port

Syntax

Gx3788PioGetPortDirection (*nHandle, nPort, pdwDirection, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to get: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
pdwDirection	PDWORD	Returns the direction settings of the selected digital port 0. GX3788_PIO_PORT_DIRECTION_IN: Digital channel is an input 1. GX3788_PIO_PORT_DIRECTION_OUT: Digital channel is an output
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the digital port 0 direction settings:

```
Gx3788PioGetPortDirection (nHandle, GX3788_PIO_PORT0, &dwDirection, &nStatus);
```

See Also

Gx3788PioSetPortDirection, Gx3788PioSetPort, Gx3788PioGetPort, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, GxFpgaGetErrorString

Gx3788PioReadPort

Purpose

Reads the input state of 32 channels in the specified digital port

Syntax

Gx3788PioReadPort (*nHandle, nPort, pdwValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to read from: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
pdwValue	PDWORD	Returns the read input states of the selected digital port
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example reads the digital port 0 input states:

```
Gx3788PioReadPort (nHandle, GX3788_PIO_PORT0, &dwValue, &nStatus);
```

See Also

Gx3788PioGetPort, Gx3788PioSetPort, GxFpgaGetErrorString

Gx3788PioReadPortChannel

Purpose

Reads the input state of the specified digital port channel

Syntax

Gx3788PioReadPortChannel (*nHandle, nPort, nChannel, pbValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to read from: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
nChannel	SHORT	Selects the channel within the selected port to read from (0-31)
pbValue	PBOOL	Returns the read input states of the selected digital channel 0. FALSE: Digital low level 1. TRUE: Digital high level
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example reads the digital port 0, channel 15 input state:

```
Gx3788PioReadPortChannel (nHandle, GX3788_PIO_PORT0, 15, &bValue, &nStatus);
```

See Also

Gx3788PioReadPort, Gx3788PioGetPort, Gx3788PioSetPort, Gx3788PioGetPortChannel, Gx3788PioSetPortChannel, GxFpgaGetErrorString

Gx3788PioResetPort

Purpose

Sets the selected digital port to default

Syntax

Gx3788PioResetPort (*nHandle, nPort, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to set: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function sets all channel output values to 0 and all channel directions to input within the selected port.

Example

The following example resets digital port 0:

```
Gx3788PioResetPort (nHandle, GX3788_PIO_PORT0, &nStatus);
```

See Also

Gx3788PioResetPortChannel, Gx3788PioSetPort, Gx3788PioGetPort, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, GxFpgaGetErrorString

Gx3788PioResetPortChannel

Purpose

Sets the selected digital channel to default

Syntax

Gx3788PioResetPortChannel (*nHandle, nPort, nChannel, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to reset: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
nChannel	SHORT	Selects the channel within the selected port to reset (0-31)
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Comments

This function the selected channel output value to 0 and direction to input.

Example

The following example resets digital port 0, channel 15:

```
Gx3788PioResetPortChannel (nHandle, GX3788_PIO_PORT0, 15, &nStatus);
```

See Also

Gx3788Reset, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, Gx3788PioSetPort, Gx3788PioGetPort, GxFpgaGetErrorString

Gx3788PioSetPort

Purpose

Sets the output states of a selected digital port

Syntax

Gx3788PioSetPort (*nHandle, nPort, dwValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to set: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
dwValue	DWORD	Sets the output states of the selected digital port, each bit represents a channel within the port, bit 0, channel 1, when the bit is high the state will be high and 0 for low.
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the digital port 0 output states to 0xF:

```
Gx3788PioSetPort (nHandle, GX3788_PIO_PORT0, 0xF, &nStatus);
```

See Also

Gx3788PioGetPort, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, GxFpgaGetErrorString

Gx3788PioSetPortChannel

Purpose

Sets the output state of a selected digital port

Syntax

Gx3788PioSetPortChannel (*nHandle, nPort, nChannel, bValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to set: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
nChannel	SHORT	Selects the channel within the selected port to set (0-31)
bValue	PBOOL	Sets the output state of a selected digital channel 0. FALSE: Digital low level 1. TRUE: Digital high level
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the digital port 0, channel 15, output state to high:

```
Gx3788PioSetPortChannel (nHandle, GX3788_PIO_PORT0, 15, TRUE, &nStatus);
```

See Also

Gx3788PioGetPortChannel, Gx3788PioSetPort, Gx3788PioGetPort, GxFpgaGetErrorString

Gx3788PioSetPortChannelDirection

Purpose

Sets the output state of a selected digital port

Syntax

Gx3788PioSetPortChannelDirection (*nHandle, nPort, nChannel, bValue, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to set: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
nChannel	SHORT	Selects the channel within the selected port to set (0-31)
bValue	BOOL	Sets the direction setting of a selected digital channel 0. FALSE: Digital channel is an input 1. TRUE: Digital channel is an output
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the digital port 0, channel 15 direction to output:

```
Gx3788PioSetPortChannelDirection (nHandle, GX3788_PIO_PORT0, 15, TRUE, &nStatus);
```

See Also

Gx3788PioGetPortChannelDirection, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, Gx3788PioSetPort, Gx3788PioGetPort, GxFpgaGetErrorString

Gx3788PioSetPortDirection

Purpose

Sets the direction (input or output) settings of a selected digital port

Syntax

Gx3788PioSetPortDirection (*nHandle, nPort, dwDirection, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nPort	SHORT	Select the digital port to set: 0. GX3788_PIO_PORT0: Digital Port 0 1. GX3788_PIO_PORT1: Digital Port 1 2. GX3788_PIO_PORT2: Digital Port 2
dwDirection	DWORD	Sets the direction settings of the selected digital port 0. GX3788_PIO_PORT_DIRECTION_IN: Digital channel is an input 1. GX3788_PIO_PORT_DIRECTION_OUT: Digital channel is an output
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets channels 0 to 4 of digital port 0 to output and channels 5 to 31 to input:

```
Gx3788PioSetPortDirection (nHandle, GX3788_PIO_PORT0, 0x1F, &nStatus);
```

See Also

Gx3788PioGetPortDirection, Gx3788PioSetPort, Gx3788PioGetPort, Gx3788PioSetPortChannel, Gx3788PioGetPortChannel, GxFpgaGetErrorString

Gx3788TriggerGetOutputLevel

Purpose

Returns the output level of the trigger output line

Syntax

Gx3788TriggerGetOutputLevel (*nHandle, pnTriggerLevel, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pnTriggerLevel	PSHORT	Returns the output trigger level: 0. GX3788_TRIGGER_LEVEL_LOW: Output Trigger level is set to Low 1. GX3788_TRIGGER_LEVEL_HIGH: Output Trigger leve is set to High
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example gets the trigger output level:

```
Gx3788TriggerSetOuputLevel (nHandle, &nTriggerLevel, &nStatus);
```

See Also

Gx3788TriggerSetOutpuLevel, Gx3788TriggerReadInputLevel, GxFpgaGetErrorString

Gx3788TriggerReadInputLevel

Purpose

Reads the level of the trigger input line

Syntax

Gx3788TriggerReadInputLevel (*nHandle, pnTriggerLevel, pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
pnTriggerLevel	PSHORT	Returns the input trigger level: 0. GX3788_TRIGGER_LEVEL_LOW: Input Trigger level reads back Low 1. GX3788_TRIGGER_LEVEL_HIGH: Input Trigger level reads back High
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example reads the input trigger level:

```
Gx3788TriggerReadInputLevel (nHandle, &nTriggerLevel, &nStatus);
```

See Also

Gx3788TriggerGetOutpuLevel, Gx3788TriggerSetOutpuLevel, GxFpgaGetErrorString

Gx3788TriggerSetOutputLevel

Purpose

Sets the output level of the trigger output line

Syntax

Gx3788TriggerSetOutputLevel (*nHandle*, *nTriggerLevel*, *pnStatus*)

Parameters

Name	Type	Comments
nHandle	SHORT	Handle for a GX3788 board.
nTriggerLevel	SHORT	Sets the output trigger level: 0. GX3788_TRIGGER_LEVEL_LOW: Output Trigger level is set to Low 1. GX3788_TRIGGER_LEVEL_HIGH: Output Trigger leve is set to High
pnStatus	PSHORT	Returned status: 0 on success, negative number on failure.

Example

The following example sets the trigger output level to high:

```
Gx3788TriggerSetOuputLevel (nHandle, GX3788_TRIGGER_LEVEL_HIGH, &nStatus);
```

See Also

Gx3788TriggerGetOutpuLevel, Gx3788TriggerReadInputLevel, GxFpgaGetErrorString

Index

.

.NET ii

A

Adder Circuit 61

Adder Circuit with PCI Bus Connection ... 62, 94, 120

Adder Components 53

Adder Wizard 59, 60

Altera 3, 47, 81, 103

Applications 4

Architecture 1

Architecture

ATEasy ..... ii, 25, 43, 44

B

Board Description 1, 5

Board Handle 46

Board Installation 27

Borland 43

Borland-Delphi 43

Bus Wiring Tool 57

C

C/C++ 43

Calibration 191

Components Used 53, 87, 109

Connector 30, 35

Connectors 29, 30, 35, 141, 149

Corrupt files 24

Creating the FPGA Design 66, 67, 95, 121

Cyclone III 48, 82, 104

D

D Flip Flops 58, 93, 119

Decoder Properties 56

Delphi ..... ii, 43

Design 55, 89

Design File 53, 87, 109

Device Selection 48, 82, 104

Directories 25

Distributing the Driver 46

DMA FIFO Interface Timing 12

Driver

  Directory 25

  Files 25

Dynamic Digital Sequencer Circuit 70

E

Error-Handling 46

ESD 27

Examples 46

Expansion Board Connector 142

Expansion Board Design Guide 133

Expansion Boards 133

F

Features 3

Function Reference 155

G

Getting Started 23

GX3700Schem.tcl 51, 85, 107

GX3701 ..... 148, 152

GX3701 Specification 148, 151, 153

Gx3788AnalogInGetGroundSource 192

Gx3788AnalogInMeasureChannel 193

Gx3788AnalogInScanGetChannelListIndex 195

Gx3788AnalogInScanGetCount 196

Gx3788AnalogInScanGetLastRunCount 197

Gx3788AnalogInScanGetSampleRate 198

Gx3788AnalogInScanIsRunning 199

Gx3788AnalogInScanReadMemoryRawData 200

Gx3788AnalogInScanReadMemoryVoltages 201

Gx3788AnalogInScanSetChannelListIndex 202

Gx3788AnalogInScanSetCount 203

Gx3788AnalogInScanSetSampleRate 204
Gx3788AnalogInScanStart 205
Gx3788AnalogInSetGroundSource 206
Gx3788AnalogOutGetOutputState 207
Gx3788AnalogOutGetVoltage 208
Gx3788AnalogOutReset 209
Gx3788AnalogOutSetOutputState 210
Gx3788AnalogOutSetVoltage 211
Gx3788GetBoardSummary 190
Gx3788GetCalibrationInfo 191
Gx3788Initialize 187
Gx3788InitializeVisa 188
Gx3788PioGetPort 212
Gx3788PioGetPortChannel 213
Gx3788PioGetPortChannelDirection 214
Gx3788PioGetPortDirection 215
Gx3788PioReadPort 216
Gx3788PioReadPortChannel 217
Gx3788PioResetPort 218
Gx3788PioResetPortChannel 219
Gx3788PioSetPort 220
Gx3788PioSetPortChannel 221
Gx3788PioSetPortChannelDirection 222
Gx3788PioSetPortDirection 223
Gx3788Reset 189
Gx3788TriggerGetOutputLevel 224
Gx3788TriggerReadInputLevel 225
Gx3788TriggerSetOutputLevel 226
GXFPGA 1, 24
  Driver-Description 43
  Header-file 43
GXFPGA Driver 43
GXFPGA driver functions 45
GXFPGA Functions 156
GXFPGA Software 25
GXFPGA.bas 43
GXFPGA.dll 44
GXFPGA.EXE 24
GXFPGA.h 43
GXFPGA.lib 43
GXFPGA.llb 44
GXFPGA.pas 43
GXFPGA.vb 43
GXFPGA64.DLL 43
GXFPGA64.lib 43
GXFPGABC.lib 43
GxFpgaDiscardEvent 159
GxFpgaDiscardEvents 156
GxFpgaDmaFreeMemory 157, 160
GxFpgaDmaGetTransferStatus 157, 161
GxFpgaDmaTransfer 157, 162
GxFpgaGetBoardSummary 156, 157, 158, 163
GxFpgaGetBoardType 156, 164
GxFpgaGetDriverSummary 46, 156, 166
GxFpgaGetEepromSummary 156, 165
GxFpgaGetErrorString 46, 155, 156, 167, 169
GxFpgaGetExpansionBoardID 156, 170
GxFpgaInitialize 16, 26, 45, 46, 155, 156, 171
GxFpgaInitializeVisa 16, 26, 45, 46, 155, 156, 172
GxFpgaLoad 156, 173
GxFpgaLoadFromEeprom 156, 174
GxFpgaLoadStatus 156, 175
GxFpgaLoadStatusMessage 156, 176
GxFpgaPanel 156, 157, 177
GxFpgaRead 156
GxFpgaReadRegister 156, 179
GxFpgaReset 156, 157, 180
GxFpgaSetEvent 156, 181
GxFpgaUpgradeFirmware 157, 182
GxFpgaUpgradeFirmwareStatus 157, 183
GxFpgaWaitOnEvent 156, 184
GxFpgaWrite 156, 178, 185
GxFpgaWriteRegister 156, 186

H

Handle 27, 28, 45, 46

HW 24, 25, 29, 43, 46

I

If You Need Help i

Installation 23, 24

Precautions-Static-Electricity 27

Procedures-All-Boards 27, 29

Installation Directories 25

Interfaces 23

Inter-FPGA Bus Interface Timing 11

Introduction 3

J

J1 .....30, 31, 35, 149, 150, 151

TTL I/O Connector 149, 150, 151

J2 .....30, 32, 35, 36

J3 .....30, 33, 35, 37

J4 .....30, 34, 35, 38

JP2 39

JP3 39

JP4 39

JP5 39

Jumpers.....39

L

LabVIEW 44

LabVIEW/Real Time.....44

Linux 44

M

Mechanical Guide.....138

MegaWizard Plug-In 55

N

nHandle 44

O

OnError.....44

Open Schematic view Dialog Box.....54, 88, 110

Overview 3

P

Packing List 23

Panel 15, 17, 18, 19, 21, 24, 45, 177

Panel About Page.....21

Part / Model Number 23

Pascal.....43

PCI.....25

PCI Address Decoder Circuit 57, 91, 118

Pin Assignment.....49, 83, 105

PLL Wizard Dialog Box.....68

Plug & Play.....29

programming 43

Programming

Borland-Delphi 43

Error-Handling.....46

Visual.....43

Programming Examples 46

Programming the GX3700.....43

PXI.....5, 7, 24, 26, 27, 28, 29, 45

PXI System.....26

PXI/PCI Explorer ..16, 26, 45, 46, 171, 172, 187, 188

PXIeSYS.INI 16

PXISYS.INI.....16

Q

Quartus 47, 48, 49, 81, 82, 83, 103, 104, 105

R

RAM Wizard Dialog Box.....69

README.TXT 25

Removing a Board 29

Reset 46

RPD 71, 73, 96, 98, 122, 124

S

Safety and Handling i

Schematic entry project 51, 85, 107

Schematic view.....54, 88, 110

Setup.....24, 25

Setup Maintenance 24

Setup-and-Installation.....23
Slot.....16, 24, 27, 29, 45
Software.....24
Specifications 1, 13
Specifications
SVF..... 71, 73, 78, 96, 98, 100, 122, 124, 129
Symbol Insert Dialog Box 55, 89, 90, 91, 111, 112, 113, 114, 115, 116, 117, 118
Symbol Properties.....63, 94, 120
System
Directory.....25
System Requirements 24

T

Task Flow 52, 86, 108
TCL script.....49, 83, 105
Testing the Design 75, 79, 101, 126, 130
TTL I/O Connector 149, 150, 151
TTL I/O Connector - Default mode (J1 pins 27 and 28 Logic low)..... 149, 150, 151

V

Virtual Panel 15, 16, 17, 18, 19, 21, 24, 177
Initialize Dialog 15, 16
VISA 16, 25, 26, 45, 46, 156, 157, 171, 172, 187, 188
Visual.....43
Visual Basic..... ii, 43
Visual Basic .NET 43
Visual C++ ..... ii, 43

W

Warranty i