

XtremeDSP Development Kit Pro User Guide

NT107-0246 - Issue 1

Document Name: XtremeDSP Development Kit Pro User Guide

Document Number: NT107-0246

Issue Number: Issue I

Date of Issue: 22/12/04

Revision History:

Date	Issue Number	Revision
10/12/2004	I	Initial release

Trademark Information

The Nallatech Logo, the DIME logo, the DIME-II logo, FUSE, FIELD Upgradeable Systems Environment, DIME, DIME-II, XtremeDSP Development Kit Pro, and the “Bally”, Ben and “Strath” product name prefixes are all trademarks of Nallatech Limited. “The Algorithms to Hardware Company”, “Making Hardware Soft”, “FPGA-Centric Systems”, “the only logical solution” and “software defined systems” are Service Marks of Nallatech Limited.

All products or brand names mentioned herein are used for identification purposes only and are trademarks, registered trademarks, or service marks of their respective owners.

Copyright Information

This document, which is supplied in confidence, is the copyright property of Nallatech Limited. Neither the whole, nor any extract may be disclosed, loaned, copied or used for any purpose other than those purposes for which written permission was given at the time of release. Application for any uplifting or relaxation of these restrictions must be made in writing to Nallatech Limited, who may at their discretion refuse any such application or give it qualified or absolute approval.

Copyright ©1993 - 2005 Nallatech Limited

All Rights Reserved

Technical Support

For technical support issues please contact Xilinx (see below for contact details):

Contacting Xilinx:



Support:

WWW:

Go to <http://support.xilinx.com>



Email:

support@xilinx.com

Contacting Nallatech



WWW:

www.nallatech.com



Product Registration

Nallatech XtremeDSP Development Kit Pro support lounge. The XtremeDSP Development Kit Pro is provided with an initial 90-day access to the support lounge upon registration. This lounge provides access to Nallatech software updates and relevant application notes as they become available. Continued access, beyond the 90 days, to this lounge is available on establishment of a maintenance agreement with Nallatech. The support lounge is available on the internet at:

www.nallatech.com/solutions/products/kits

In the actions section of this web page you will see an option to register your product. Registration requires the serial number of the XtremeDSP Development Kit Pro. The serial number is located on the bottom of the small blue board case.



Contents

About this User Guide	xiii
Part I: Introduction	I
XtremeDSP Development Kit Pro Overview	3
XtremeDSP Development Kit Pro Key Features	4
XtremeDSP Development Kit Pro Functional Diagram	6
Introduction to Nallatech Systems and Solutions	9
Introduction	9
Motherboards and Modules	9
DIME-II Standard	10
FUSE Operating System	10
MATLAB	11
DIMEscript	11
DIMEtalk	11
Part II: XtremeDSP Development Kit Pro Features	13
Physical Layout	15
XtremeDSP Development Kit Pro Overview	15
XtremeDSP Development Kit Pro Board Case Layout	16
Front	16
Left Side	16
Right Side	17
XtremeDSP Development Kit Pro Hardware Physical Layout	17
Front View	17
Back View	18
ADCs	19
Introduction	19
ADC Architecture	20
Hardware	21
Analog ADC Inputs	21
ADC Clocking	22
Analog Performance	22
ADC Front End Configuration	23
Firmware	27
Timing Constraints	28

Software	29
DACs	31
Introduction	31
DAC Architecture	32
Hardware	33
PLL Clock Multiplier	33
DAC Modes of Operation	34
Analog DAC Outputs.....	35
Output Configurations.....	36
DAC Clocking.....	38
Firmware	39
Two's Complement vs. Offset Binary Format.....	40
Timing Constraints.....	41
Software	41
ZBT SRAM Memory	43
Introduction	43
Hardware	44
ZBT SRAM Clocking.....	49
Firmware	49
Software	49
Digital I/O	51
Introduction	51
Hardware	52
User I/O Header - Interfacing	52
P-Link Bus Header (J10).....	52
Adjacent Bus Header (J8)	52
Firmware	54
Software	54
LEDs	55
Introduction	55
Hardware	56
Physical Location of LEDs.....	56
Interface LEDs.....	57
User LEDs	58
Firmware	58
Software	59
FUSE Software LED Function	59
Listing of Calls in the FUSE API for LED Control	60
Resets	61
Introduction	61
Hardware	61

Reset Structure	61
Firmware	62
Software	63
FUSE Software Reset Function.....	63
Listing of Calls in the FUSE API for Reset Control.....	64
Clocks	65
Introduction	65
Hardware	66
Physical Clock Resources	66
Clocking Configuration	67
Source Descriptions	67
Inter-FPGA Clock Management.....	70
ADC and DAC Clocks.....	71
ZBT Clocks.....	71
Firmware	72
Using the Virtex-II Pro DCMs.....	72
Clocking the ADCs and DACs	75
Software	77
Setting the Clocks in FUSE.....	77
Listing of Calls in the FUSE API to Set the Clocks.....	78
Bus Structure	81
Introduction	81
Bus Naming.....	81
Hardware	82
Firmware	83
Software	83
Board and System Level Monitoring Capabilities	85
Introduction	85
Temperature Monitoring	86
JTAG Chain Access	94
General JTAG Header.....	95
Parallel-IV JTAG Header	95
Configuration Status Monitoring	97
Power Specifications	99
Introduction	99
External Standalone Power Supply Specification.....	99
PCI Power Specification.....	100
Hardware	101
Power Supply LEDs.....	103
Cooling.....	104
External Connectors	106
Enabling Power Supplies at Power-On (using External Power Supply and JTAG	

connectors)	106
Environmental Specifications	107
Specifications	107
Operating Temperatures.....	107
Storage Temperatures.....	107
Relative Humidity	107
Part III: System Level Design	109
Building a Host Interface	111
Introduction	111
Hardware	111
Design Partitioning	111
Interface Communications Bus	113
Firmware	114
Interface FPGA to User FPGA Interface Core	114
Implementing the Communications Mechanism	114
Communications Bus Protocol	115
Further Information on Interface Core.....	118
Software	118
FUSE API Interlinks	118
Interface Performance.....	119
Part IV: FPGA Configuration	121
FPGA Configuration	123
Configuration Options	123
Module and Device Numbering within FUSE	123
Key Steps for FPGA Configuration	124
Using the FUSE APIs or DIMEScript	124
Access through JTAG Headers.....	125
FPGA Configuration using the FUSE Probe Tool	125
FPGA Configuration using the FUSE APIs	125
FPGA Configuration using DIMEScript	126
FPGA Configuration using General JTAG Chain	126
Part V: Hardware Examples	129
Feature Examples	131
Introduction	131
Simple ADC Hookup Example	131
Example Overview	131
Source	131
Implementation	131
Running the Example	132
ADC to DAC Example	135

Example Overview	135
Source	135
Implementation	135
Running the Example	135
Host Interface Example	138
Example Overview	138
Source	138
Implementation	138
Running the Example	139
Relevant Application Notes	146
Part VI:Xilinx ISE Support.....	147
Common ISE Settings	149
Synthesis and Implementation Settings	149
Synthesis Options.....	149
Implementation Options.....	149
Part VII:Xilinx Impact Support.....	153
Impact Support	155
Introduction	155
Set up a Connection	155
Connect a JTAG Download Cable.....	155
Open the Card to Enable Power Supplies.....	155
Alternative Method of Enabling Power Supplies	156
Launch Impact	157
Part VIII:Chipscope ILA Support.....	161
Chipscope ILA Support	163
Introduction	163
Connect a JTAG Download Cable	163
XtremeDSP Development Kit Pro Chipscope Example	164
Overview	164
Creating the ICON and ILA Cores.....	164
Manually Instantiating the ICON and ILA Cores in your Design.....	169
Analyzing the Design.....	171
Part IX:Xilinx Embedded Developer's Support.....	179
Xilinx Embedded Developer's Kit (EDK) Support	181
Introduction	181
Design Notes	181
Resets.....	181
Using the ZBT as a Peripheral.....	181
XMD Support.....	182
EDK Example	183

Load the FPGA Design with FUSE	184
Connect via XMD	185
Start the Software Debugger	186
Part X:Xilinx System Generator Support.....	189
System Generator Support	191
Introduction	191
Overview of System Generator	191
Using the HDL Netlist Flow (QAM Example)	192
Example Overview	192
Simulating the QAM Example.....	192
Using the Cosimulation Flow (MAC FIR Example)	200
Example Overview	200
Simulating the MAC FIR Cosimulation Example	200
Cosimulation Flow	202
Chipscope Use in System Generator	208
Example Overview	208
Chipscope Pro Overview	208
Chipscope in System Generator.....	209
Real-time debug	215
Importing Data Into MATLAB Workspace From Chipscope	220
System Generator and UltraController Reference	Design 220
Reference Design Contents	220
Running the Examples	222
Part XI:Board Firmware	223
Changing the Firmware	225
Introduction	225
The Firmware Utilities	225
Performing a Firmware Update	226
Converting from 5VIO PCI and USB firmware to 5VIO and 3.3VIO PCI firmware	226
Converting from PCI Firmware only to 5VIO PCI and USB firmware.....	227
Part XII:Reference Information	229
Pinout Information	231
Local Bus Pinouts - Interface Communications	231
Adjacent Out Bus - Interface Communications	232
Adjacent Header (J8)	233
PLINKS 0: PLINK Header (J10)	234
PLINKS 7: Nallatech RS232 Connections in Part	234
Nallatech RS232 Test Header (J9 and J12).....	235
DIME-II Control and Monitoring Signals	235
DIME-II Specific Pins	235

User LEDs	235
On Board Temperature Sensor	236
ZBT SRAM	236
Clock and Control Signals for ZBT.....	236
ZBT Address Signals for ZBT Bank A	237
ZBT Data Signals for ZBT Bank A.....	237
ZBT Address Signals for Bank B	238
ZBT Data Signals for Bank B.....	239
Inter-FPGA Clock Infrastructure Signals	240
Clock sources arriving at CLOCK FPGA.....	240
Clock Feedback Signals	240
Clocking Pinouts for DACs and ADCs.....	240
DAC Signal Pinouts	241
ADC Signal Pinouts	242
User I/O Header (J16 on module)	242
Part XIII: Troubleshooting.....	243
FAQ	245
XtremeDSP Development Kit Pro FAQs	245

About this User Guide

This User Guide provides detailed information on the XtremeDSP Development Kit Pro. It allows you to become acquainted with the Kit, its features and the functionality it provides. Before reading this manual it is recommended you read the *XtremeDSP Development Kit Pro Getting Started Guide* supplied with the Kit for installation procedures. The Kit User Guide provides details on the individual features available and how they are used, as well as information on support for certain tools. At the end of the User Guide you will find a reference section containing pinout information and a troubleshooting guide.

Symbols Used

Throughout this guide there are symbols to draw attention to important information:



The red arrow symbol indicates a set of procedures to follow, such as installing software or setting up hardware.



The blue 'i' symbol indicates useful or important information.



The red '!' symbol indicates a warning, which requires special attention.

User Guide Format

The User Guide is divided into **Sections**, which are grouped into **Parts**. The parts divide the document as follows:

- **Introduction:** Provides an introduction to the User Guide and the key aspects of the underlying hardware and software architecture.
- **XtremeDSP Development Kit Pro Features:** This part provides details on the key features provided in the Kit such as the use of the ADC and DAC components.
- **System Level Design:** Provides information on creating a complete system that uses the Kit hardware and the FUSE API in a host application.
- **Configuration:** Discusses the different ways in which the Kit's FPGAs may be configured.
- **Hardware Examples:** Provides a number of implemented examples of using particular hardware features.

- Xilinx ISE Support: Details support and use of the Kit within the Xilinx ISE tool flow.
- Xilinx Impact Support: Details support for the Xilinx Impact programming tool.
- Xilinx ChipScope ILA Support: Details support for the Xilinx ChipScope ILA tool.
- Xilinx Embedded Developers Kit: Provides details on support for the Xilinx EDK tools.
- Xilinx System Generator Support: Provides details on support for the Xilinx System Generator tool.
- Board Firmware: Gives details on how to update or change the firmware for the host interface on the hardware.
- Reference Information: Provides full user pinout details for the main FPGAs available for programming.
- Troubleshooting: Contains help and a FAQ for common queries.

Related Documentation

There are a number of additional sources of information on specific products used in the Kit. Generally these are included in the documents folder in the CDs indicated in brackets.

- | | |
|------------------|---|
| • Analog Devices | AD6645 ADC Datasheet (XtremeDSP Kit CD) |
| • Analog Devices | AD9772A DAC Datasheet (XtremeDSP Kit CD) |
| • Maxim | 1617 Datasheet (XtremeDSP Kit CD) |
| • Micron | ZBT SRAM Datasheet (XtremeDSP Kit CD) |
| • Nallatech | BenADDA Datasheet (XtremeDSP Kit CD) |
| • Nallatech | DIMEscript User Guide (FUSE CD) |
| • Nallatech | FUSE C-C++ API Developers Guide (FUSE CD) |
| • Nallatech | FUSE System Software User Guide (FUSE CD) |
| • Nallatech | PCI to User FPGA Interface Core Application Note (FUSE CD) |
| • Xilinx | Virtex-II and Virtex-II Pro Datasheet (available on Xilinx Website) |

Abbreviations

- | | |
|----------------|---|
| • ADC: | Analog to Digital Converter |
| • API: | Application Program Interface |
| • ATX: | Advanced Technology eXtended |
| • BIST: | Built-In Self Test |
| • DAC: | Digital to Analog Converter |
| • DCM: | Digital Clock Manager |
| • DIME: | DSP and Image Processing Modules for Enhanced FPGAs |
| • DLL: | Delay Locked Loop |
| • EDK: | Embedded Developer's Kit |
| • ENOB: | Effective Number of Bits |
| • FPGA: | Field Programmable Gate Array |
| • FPS: | Fixed Power Supply |

- **FUSE:** Field Upgradeable System Environment
- **GUI:** Graphical User Interface
- **ILA:** (Xilinx) Integrated Logic Analyzer
- **I/O:** Input/Output
- **JTAG:** Joint Test Action Group
- **LVDS:** Low Voltage Differential Signalling
- **LVTTL:** Low Voltage Transistor-Transistor Logic
- **PCB:** Printed Circuit Board
- **PCI:** Peripheral Component Interconnect
- **PLL:** Phase Locked Loop
- **PPS:** Programmable Power Supply
- **SNR:** Signal to Noise Ratio
- **SRAM:** Static Random Access Memory
- **VCO:** Voltage Controlled Oscillator
- **VHDL:** VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
- **ZBT:** Zero Bus Turnaround memory

Typographical Conventions

The following typographical convention are used in this manual:

- **Red text** indicates a cross-reference to information within the document set you are currently reading. Click the red text to go to the referenced item. To return to the original page, right click anywhere on the current page and select **Go To Previous View**.
- **Blue underlined text** indicates a link to a Web page. Click blue-underlined text to browse the specified Web site.
- *Italics* denotes the following items:
 - References to other documents:

See the *FUSE System Software User Guide* for more information.
 - Emphasis in text:

Enable Loopback should *not* be enabled until all other registers have been set up.

FUSE Naming Conventions

Please note that the XtremeDSP Development Kit Pro clocks are named differently in the FUSE System Software compared to this User Guide. The clock naming conventions are shown in [Table I on page xvi](#).

Clock Names in FUSE	Clock Names in Documentation
System Clock (SYSCLK)	Clock A (CLK A)
DSP Clock (DSPCLK)	Clock B (CLK B)
Pixel Clock (PIXCLK)	Clock C (CLK C)

Table I: FUSE Naming Conventions

For more information on how the clocks are named please see [“DIME-II System Clocks” on page 65](#).

Scope

Please note that this User Guide provides information exclusively on the XtremeDSP Development Kit Pro. It does not provide details on previous versions of the XtremeDSP Development Kit.

Comments and Suggestions

At the back of this book, you will find a remarks form. We welcome any comments you may have on our product or its documentation. Your remarks will be examined thoroughly and taken into account for future versions of Nallatech products.



NALLATECH
The High Performance FPGA Solutions Company

Part I: Introduction

This part of the User Guide provides an introduction to the XtremeDSP Development Kit Pro and outlines its key features and functionality.

Section I

XtremeDSP Development Kit Pro Overview

In this section:

- XtremeDSP Development Kit Pro Key Features
 - XtremeDSP Development Kit Pro Functional Diagram
-



For installation instructions please refer to the *Getting Started Guide* supplied in the XtremeDSP Development Kit Pro CD wallet and also on the XtremeDSP Development Kit Pro CD. This guide covers all installation procedures for the Kit including hardware installation and FUSE software installation.

I.1 XtremeDSP Development Kit Pro Key Features

The XtremeDSP Development Kit Pro serves as an ideal development platform for the Virtex-II Pro FPGA technology and provides an entry into the scalable DIME-II systems available from Nallatech. Its dual channel high performance ADCs and DACs, as well as the user programmable Virtex-II device are ideal to implement high performance signal processing applications such as Software Defined Radio, 3G Wireless, Networking, HDTV or Video Imaging.



Figure 1: XtremeDSP Development Kit Pro

The key features of the Kit include:

Hardware

- XtremeDSP development board consisting of a motherboard populated with a module (daughter card) in a blue stand-alone board case. The motherboard is referred to as the "BenONE-Kit Motherboard" and the module is referred to as the "BenADDA DIME-II module".
- BenONE-Kit Motherboard
 - Supports the supplied BenADDA DIME-II module only
 - Spartan-II FPGA for 3.3V/5V PCI or USB interface
 - Host interfacing via 3.3V/5V PCI 32 bit/33-MHz or USB v1.1 interfaces
 - Status LEDs
 - JTAG configuration headers
 - User 0.1" pitch pin headers connected directly to user programmable FPGA I/O
- BenADDA DIME-II module
 - Virtex-II Pro User FPGA: XC2VP30-4FFI152
 - 2 independent ADC channels: AD6645 ADC (14-bits up to 105 MSPS)
 - 2 independent DAC channels: AD9772 DAC (14-bits up to 160MSPS)

- Support for external clock, on board oscillator and programmable clocks
- Two banks of ZBT-SRAM (133MHz, 512Kx32 bits per bank)
- Multiple Clocking Options: Internal & External
- Status LEDs
- External power supply (US Mains cable with separate UK, European or Australian mains adaptors)
- Wide ranging input (90 - 264Vac), multiple output, power supply, generating;
 - +5 Volts @ 5A, +12 Volts @ 2A, -12 Volts @ 800mA
- USB v1.1 compatible cable, 2 metres long
- 5 MCX to BNC cables for connecting to the ADC / DAC and external clock connectors
- PCI Backplate and 2 screws
- 2x BNC jack to jack adaptors for use in loop back configurations
- Large blue Kit carrying case

'XtremeDSP Installation Pack' containing

- Nallatech FUSE (Field Upgradeable Systems Environment) Software CD. Provides the ability to control and configure FPGAs, and provides facilities to transfer data between the Kit and a host PC via a GUI or a C based API
- Nallatech XtremeDSP Development Kit Pro CD that provides documentation on the Nallatech hardware as well as hardware support files for the use within the Nallatech FUSE environment
- Optional Nallatech evaluation software

'Xilinx XtremeDSP Software Evaluation CD Kit' (supplied separately)

- Evaluation for the Xilinx Foundation ISE
- Evaluation for the Xilinx System Generator for DSP
- Evaluation for MATLAB® Simulink

1.2 XtremeDSP Development Kit Pro Functional Diagram

The XtremeDSP Development Kit Pro features three Xilinx FPGAs - a Virtex-II Pro User FPGA, a Virtex-II FPGA for clock management and a Spartan-II Interface FPGA. The Virtex-II device is available exclusively for user designs whilst the Spartan-II is supplied pre-configured with firmware for PCI/USB interfacing. The PCI/USB interfacing firmware and low level drivers abstract the PCI/USB interfacing from the user resulting in a simplified design process for user designs/applications. The Interface FPGA communicates directly with the larger User FPGA (XC2VP30-4FF1152) via a dedicated communications bus that is made up of the LBUS and ADJOUT busses shown in Figure 2 on page 6.

The Virtex-II Pro XC2VP30-4FF1152 device is intended to be used for the main part of a user's design. The Virtex-II XC2V80-4CS144 is intended to be used as a clock configuration device in a design.

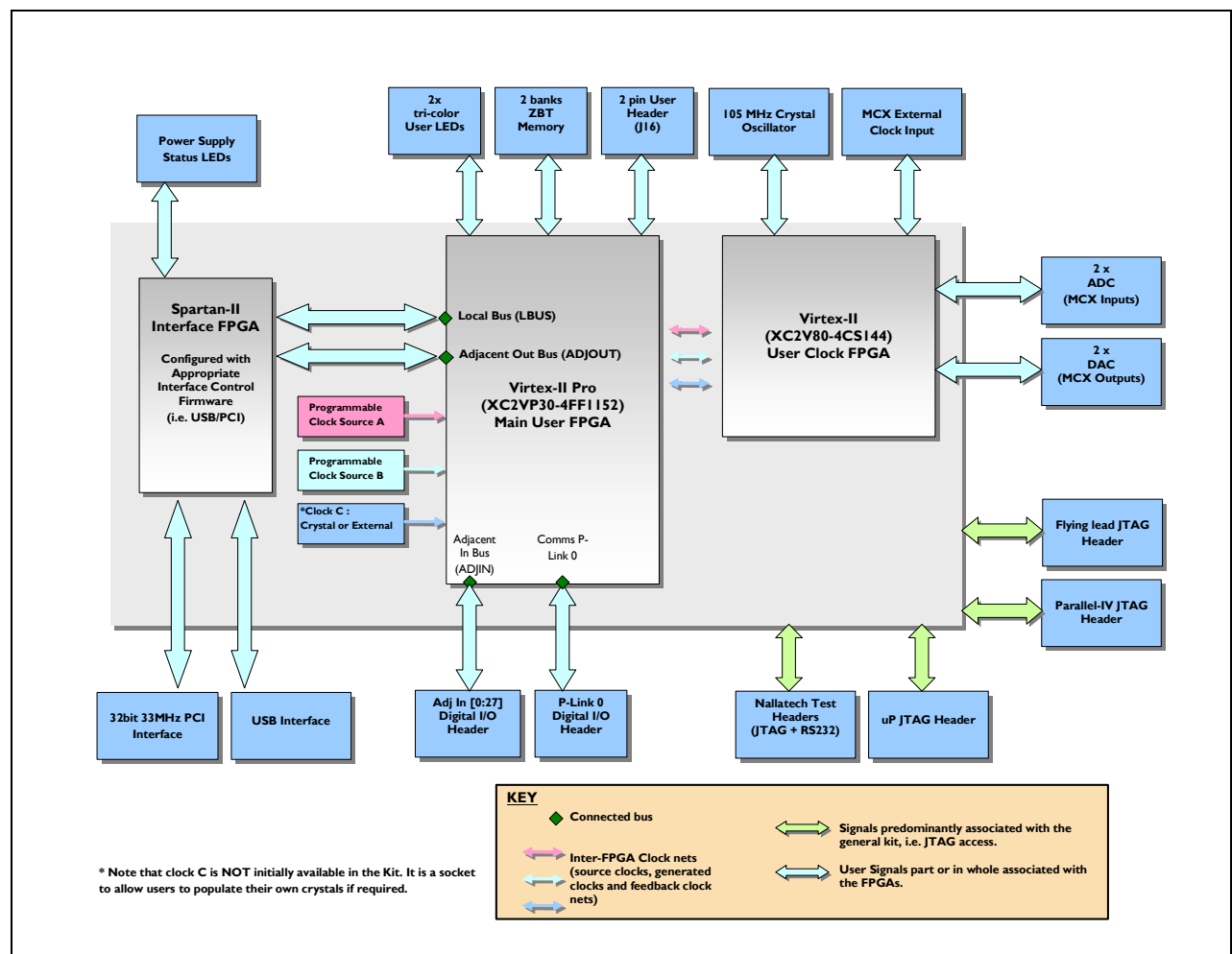


Figure 2: XtremeDSP Development Kit Pro Functional Diagram

For more information on:

- communications between the User FPGA and the Spartan-II Interface FPGA, refer to “**Interface FPGA to User FPGA Interface Core**” on page 114.
- ADCs, refer to the features section “**ADCs**” on page 19.
- DACs, refer to the features section “**DACs**” on page 31.
- ZBT SRAM memory, refer to the features section “**ZBT SRAM Memory**” on page 43.
- status LEDs, refer to the features section “**LEDs**” on page 55.

- user 0.1" pitch pin headers, refer to the features section “Digital I/O” on page 51.
- JTAG headers, refer to the features section “Board and System Level Monitoring Capabilities” on page 85.
- clocking options, refer to the features section “Clocks” on page 65.
- temperature sense capabilities, refer to the features section “Board and System Level Monitoring Capabilities” on page 85.

Section 2

Introduction to Nallatech Systems and Solutions

In this section:

- Introduction
 - Motherboards and Modules
 - DIME-II Standard
 - FUSE
 - MATLAB
 - DIMEScript
 - DIMEtalk
-

2.1 Introduction

Nallatech provides professional, low risk, high-performance FPGA computing solutions for customers' challenging applications. Our customers benefit from the advantages Nallatech solutions provide, with the reassurance that they are using a complete solution from a trusted supplier with over ten years experience in FPGA systems technology. This is backed by a committed team worldwide, who are dedicated to delivering world-class product solutions, customer service and support, to a global customer base.

2.2 Motherboards and Modules

Nallatech provide a comprehensive range of motherboards and modules which can be used in a variety of applications. The XtremeDSP Development Kit Pro provides a PCI/USB motherboard (BenONE) that can support a single specific module. The module used is the BenADDA that provides FPGA resources in addition to ADC inputs and DAC outputs. Nallatech motherboards and modules are designed to a standard called DIME-II.

2.3 DIME-II Standard

DIME-II is a modular standard developed specifically for Field Programmable Gate Arrays (FPGAs) which allows design engineers to develop re-programmable systems with the opportunity to alter or experiment with the partitioning of their designs at any stage of the design cycle. DIME-II has been introduced in response to ever increasing bandwidth and performance requirements. This standard addresses the high performance needs for future generations of reconfigurable computers. Software support for DIME-II hardware is provided by FUSE.

2.4 FUSE Operating System

Nallatech's FUSE System Software provides configuration, control and communications functionality between host systems and Nallatech FPGA computing hardware. This enables developers to design complex processing systems, with seamless integration between software, hardware and FPGA applications.

FUSE provides several interfaces, including the scripting language DIMEscript, FUSE Probe Tool and the FUSE development APIs for C/C++, with optional APIs also available for Java and MATLAB®. FUSE is available as FUSE for Windows® and FUSE for Linux®. In the XtremeDSP Development Kit Pro, FUSE for Windows® is only provided as part of the Kit. Key features include:

- fast and simple device configuration
- multiple card support
- multiple interface support
- interfacing and control of Nallatech hardware features

Figure 3 on page 10 provides an overview of the FUSE operating system.

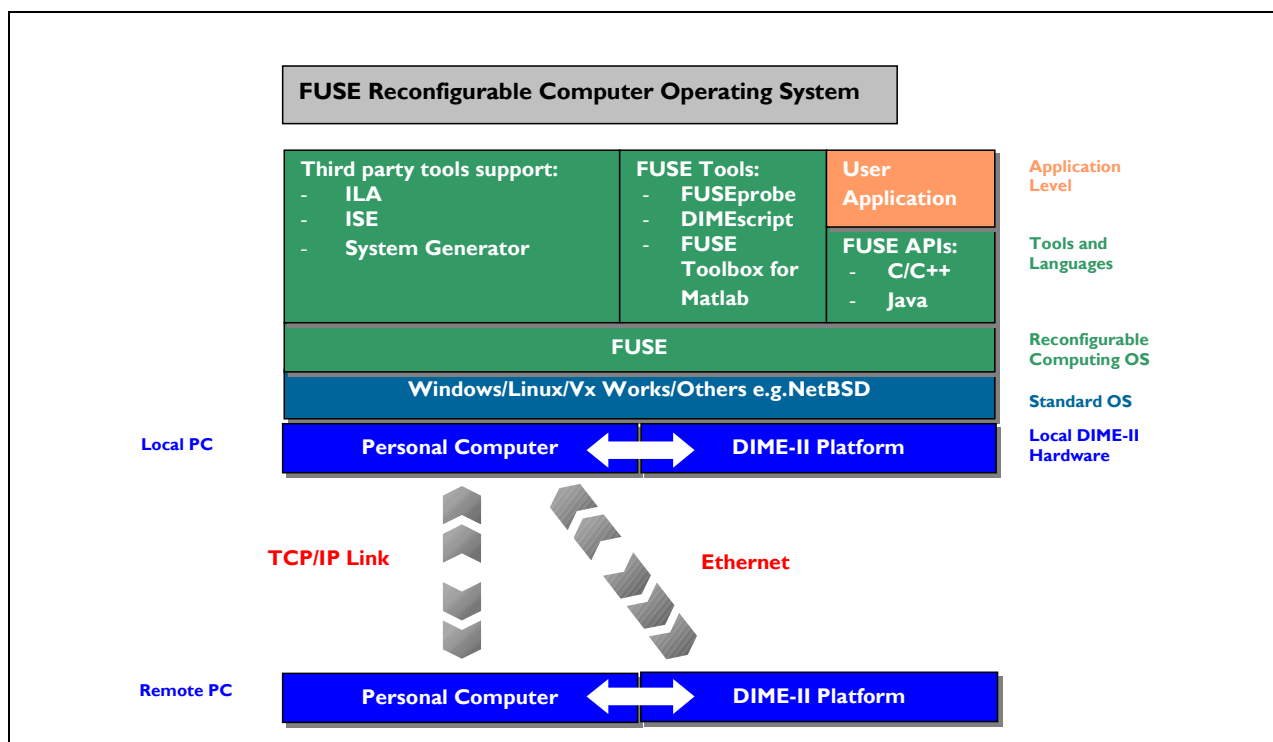


Figure 3: FUSE Overview

Further details on FUSE are provided in the documentation supplied on the FUSE CD or installed as part of the FUSE installation process.

2.5 MATLAB

The FUSE Toolbox for MATLAB® is another level of integration that allows the user to develop applications for a Nallatech DIME based system straight from the MATLAB® environment. Each function of the toolbox is a wrapping of the corresponding function from the FUSE C/C++ library where appropriate. The software interface is implemented as a set of mex-files (see *FUSE Toolbox for MATLAB Developers Guide*). These mex-files provide access to the C/C++ FUSE API through calls from the MATLAB environment. An evaluation copy of FUSE Toolbox for MATLAB is provided with the Kit.

2.6 DIMEscript

DIMEscript has been developed by Nallatech as a simple method of accessing motherboards and modules without the need to resort to programming using high level languages like C/C++. DIMEscript is an interpreted language which means that the language is read in line-by-line and appropriate actions taken. This, in turn, means that any errors in the script are only found when the relevant line is executed.

This is in contrast to a compiled language where the required action is checked in advance and made into a more machine friendly form. In the case of the compiled language, syntax and other features can be fully checked before running the code. DIMEscript enables users to:

- open a Nallatech card
- read data from the card
- write data to the card
- access various specific card functions

The intention of DIMEscript is to provide a useful scripting language to control DIME based systems without having to create say a C based application which may be overkill for a simple setup or specific test.

2.7 DIMEtalk

DIMEtalk is a design tool for implementing communications networks within FPGAs. DIMEtalk provides an integrated design environment which allows users to implement communications networks within an FPGA system using simple block components such as routers, nodes and edge components. DIMEtalk also provides a library of software functions, which are accessible via the FUSE software. These functions provide easy access to the FPGA network from a software environment.

A DIMEtalk network is composed of routers helping nodes communicate using packets. Each DIMEtalk connected process is represented by a node which acts as a doorway into the network. The simplest form of node can be thought of as an area of memory accessible by processes on the DIMEtalk network and the associated process behind the node. An evaluation copy of DIMEtalk is provided with the Kit.

Figure 4 on page 12 represents an example DIMETalk network.

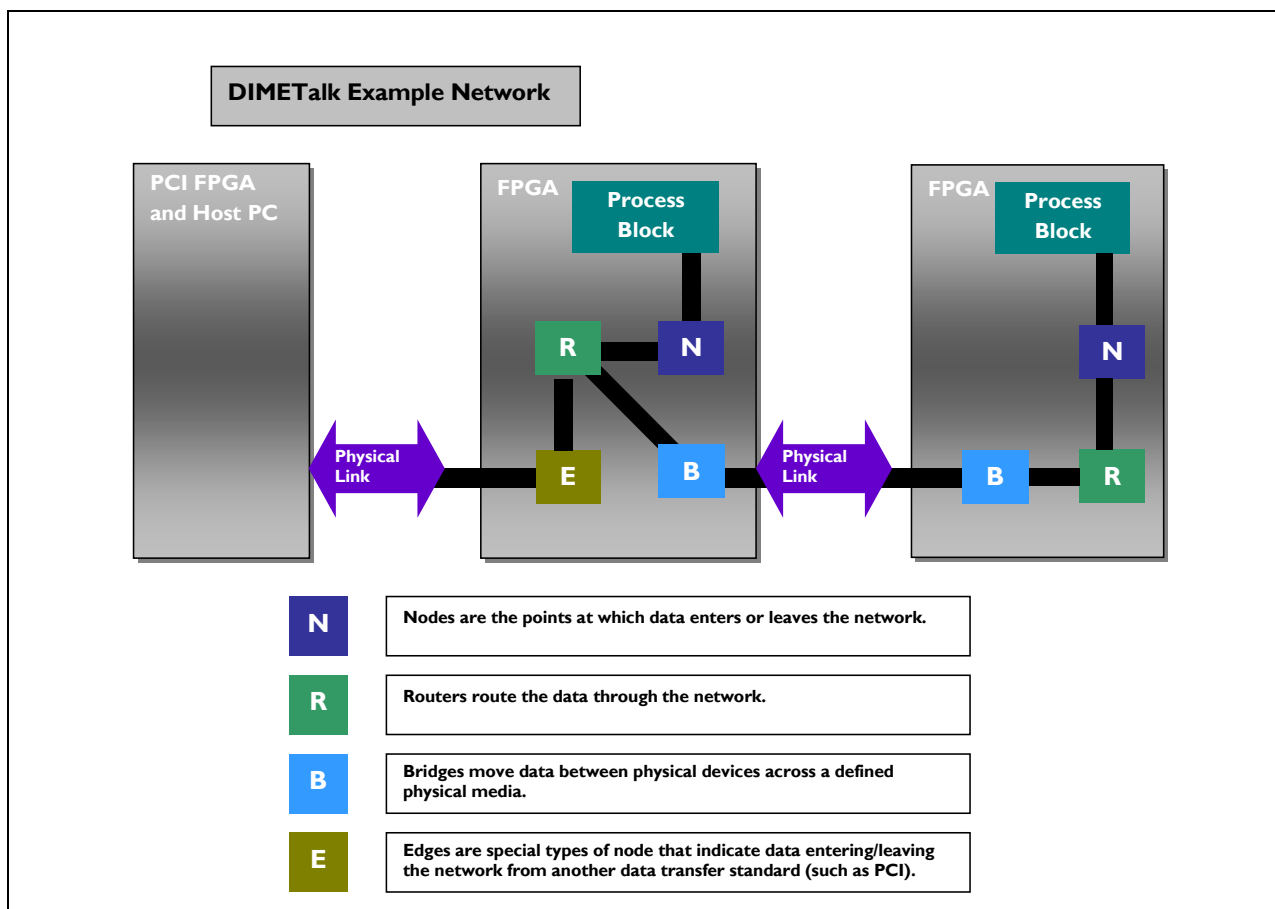


Figure 4: DIMETalk Example Network



NALLATECH
The High Performance FPGA Solutions Company

Part II: XtremeDSP Development Kit Pro Features

This part of the User Guide provides information on the key features of the XtremeDSP Development Kit Pro including: physical layout, ADCs, DACs, ZBT memory, digital I/O, LEDs, resets, clocks, bus structure, system monitoring, power and environmental considerations.

Section 3

Physical Layout

In this section:

- XtremeDSP Development Kit Pro Overview
- XtremeDSP Development Kit Pro Board Case Layout
- XtremeDSP Development Kit Pro Physical Layout

3.1 XtremeDSP Development Kit Pro Overview

Table 2 on page 15 lists the individual items that are supplied in the Kit.

Item #	Description	Quantity
1	BenONE-PCI DIME-II Motherboard PECA	1
2	XtremeDSP Development Kit Pro Board Case	1
3	Australian Travel Adaptor	1
4	UK Travel Adaptor	1
5	EURO Travel Adaptor	1
6	US Power Cord	1
7	A-B USB Cable 2m Long	1
8	MCX to BNC Cable 1m Long	5
9	BenADDA DIME-II Module PECA	1
10	PCI Backplate assembly	1
11	BNC Jack to Jack Adaptor	2
12	Power Supply Unit	1
13	XtremeDSP Development Kit Pro Carry Case	1
14	XtremeDSP Development Kit Pro CD Wallet	1

Table 2: Kit Contents

3.2 XtremeDSP Development Kit Pro Board Case Layout

The main hardware for the Kit is fitted inside a small blue case which provides protection for the board and also a degree of EMI shielding. [Figure 5 on page 16](#) highlights the key features of the board case.



Cooling to the main FPGA is provided by a small fan that draws cold air in through a main fan vent. The FPGA can be configured with a design that will create a large amount of heat. Therefore it may be necessary to remove the lid of the case when running certain high power designs. Please refer to [“Cooling” on page 104](#) for more details.

3.2.1 Front

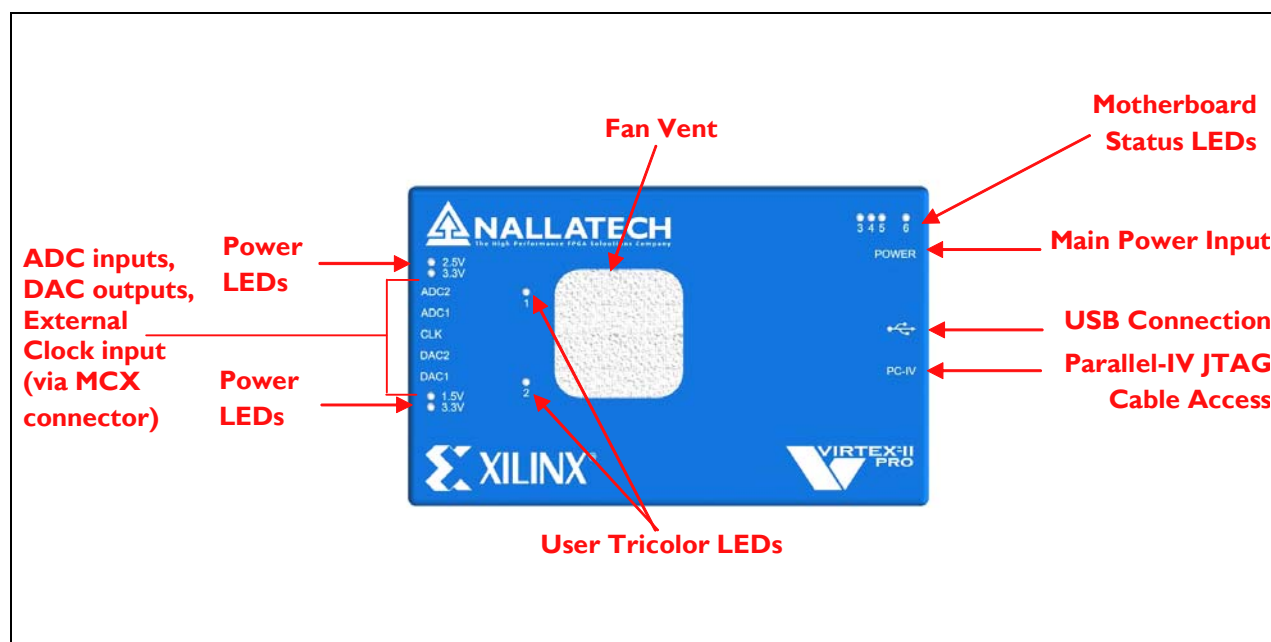


Figure 5: Board Case - Front

3.2.2 Left Side

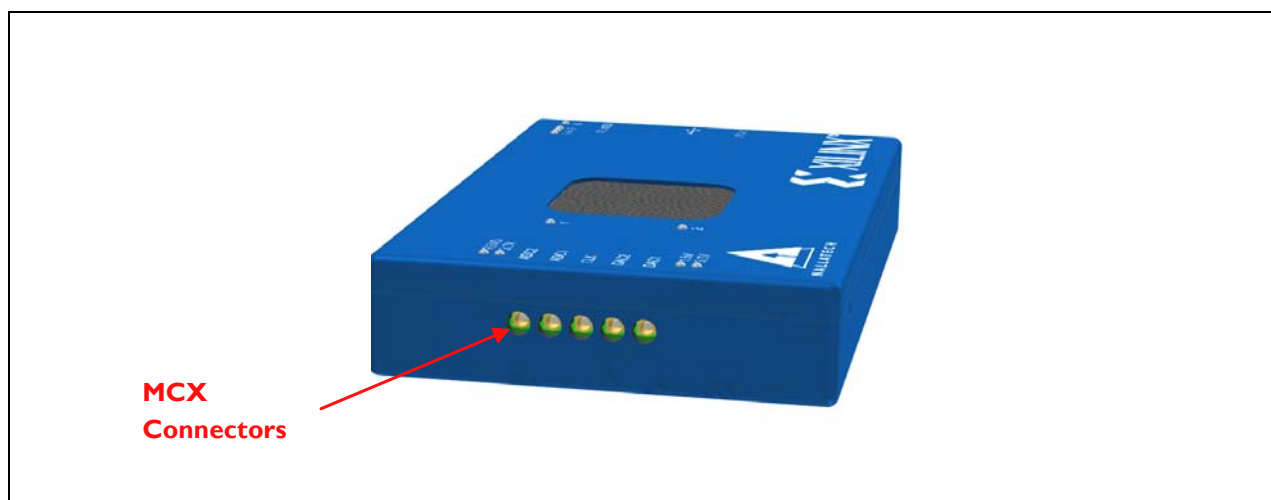


Figure 6: Board Case - left side showing MCX Connectors

3.2.3 Right Side

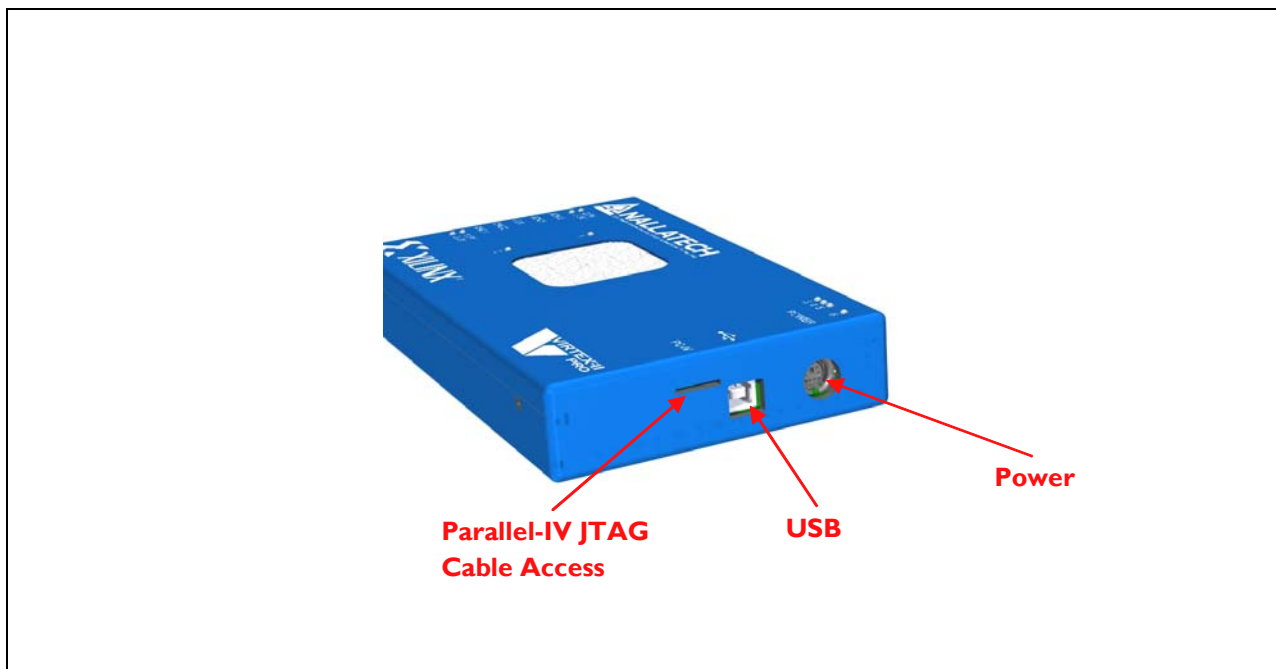


Figure 7: Board Case - right side showing USB, power and Parallel-IV Access.

3.3 XtremeDSP Development Kit Pro Hardware Physical Layout

3.3.1 Front View

Figure 8 on page 17 highlights the key features on the front of the board.

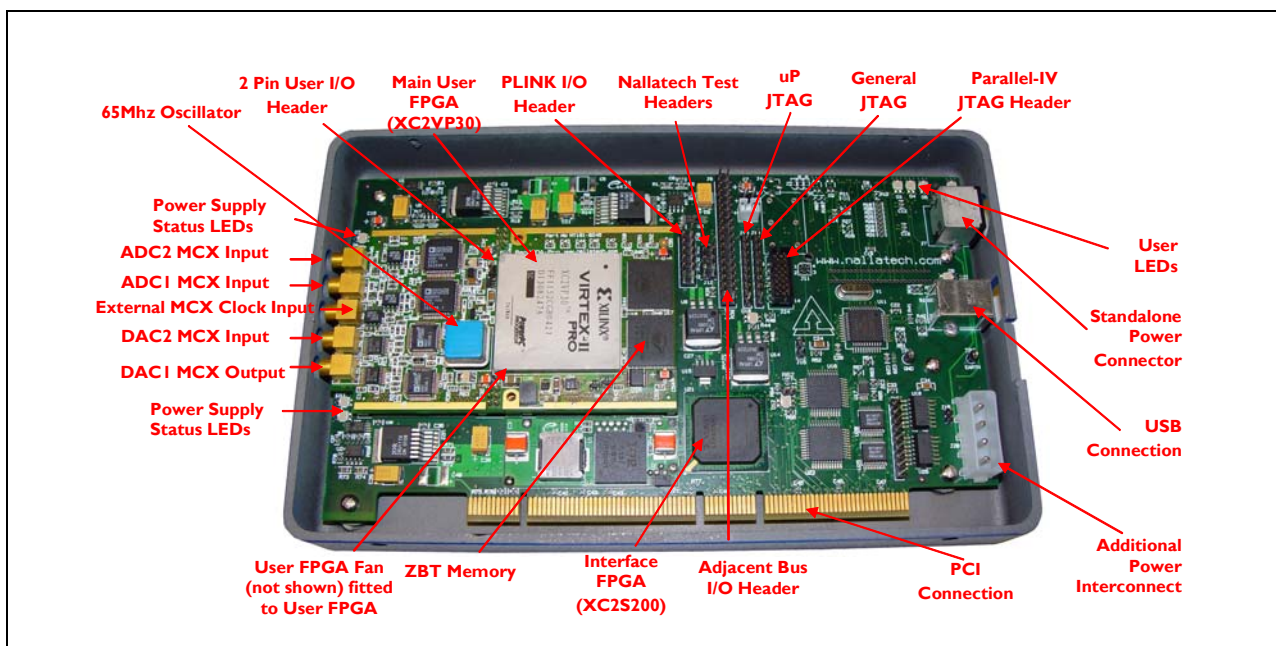


Figure 8: Front View of Board Physical Layout



The Clock FPGA (XC2V80-4CS144) is located on the underside of the module and is not visible in Figure 8 on page 17.

3.3.2 Back View

There are no specific user features on the back of the board. Figure 9 on page 18 is displayed for reference only.

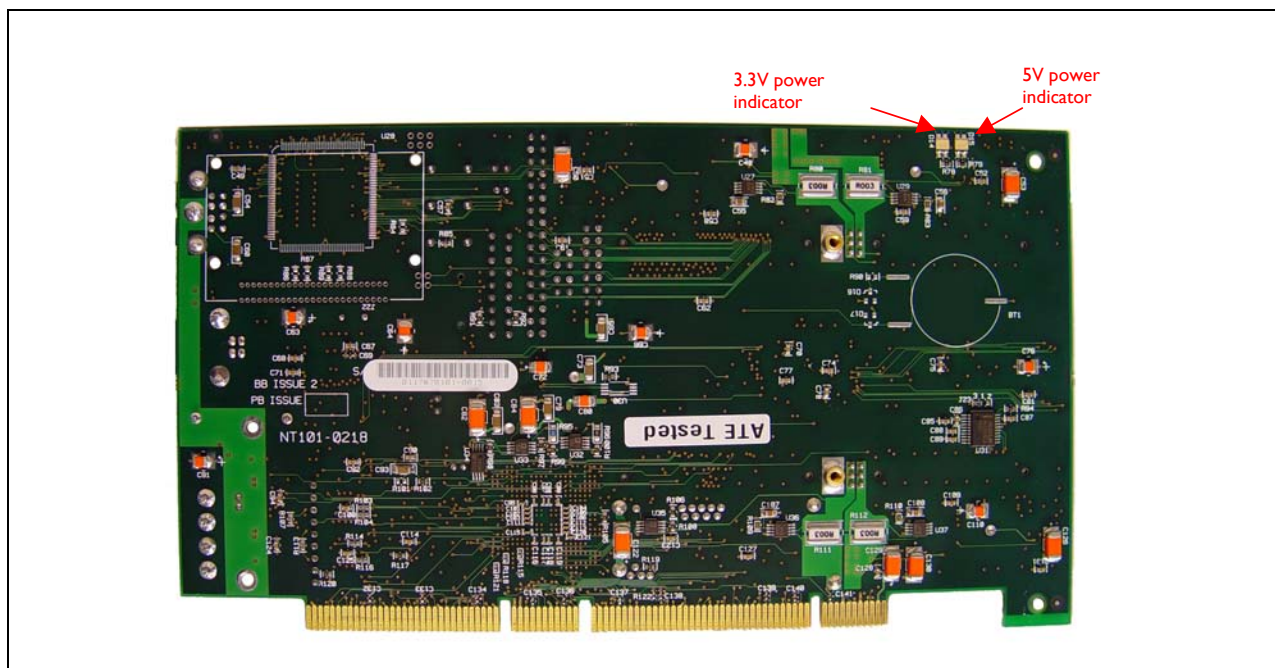


Figure 9: Back View of Board Physical Layout

Section 4

ADCs

In this section:

- Introduction to the ADCs
- Hardware Aspects
- Firmware Aspects
- Software Aspects

4.1 Introduction

The BenADDA DIME-II module used in the XtremeDSP Development Kit Pro has two analog input channels, with each channel providing independent data and control signals to the FPGA. Two sets of 14-bit wide data are fed from two ADCs (AD6645) devices, each of which has an isolated supply and ground plane. [Figure 10 on page 19](#) illustrates the interfacing between one of the ADCs and the FPGA.

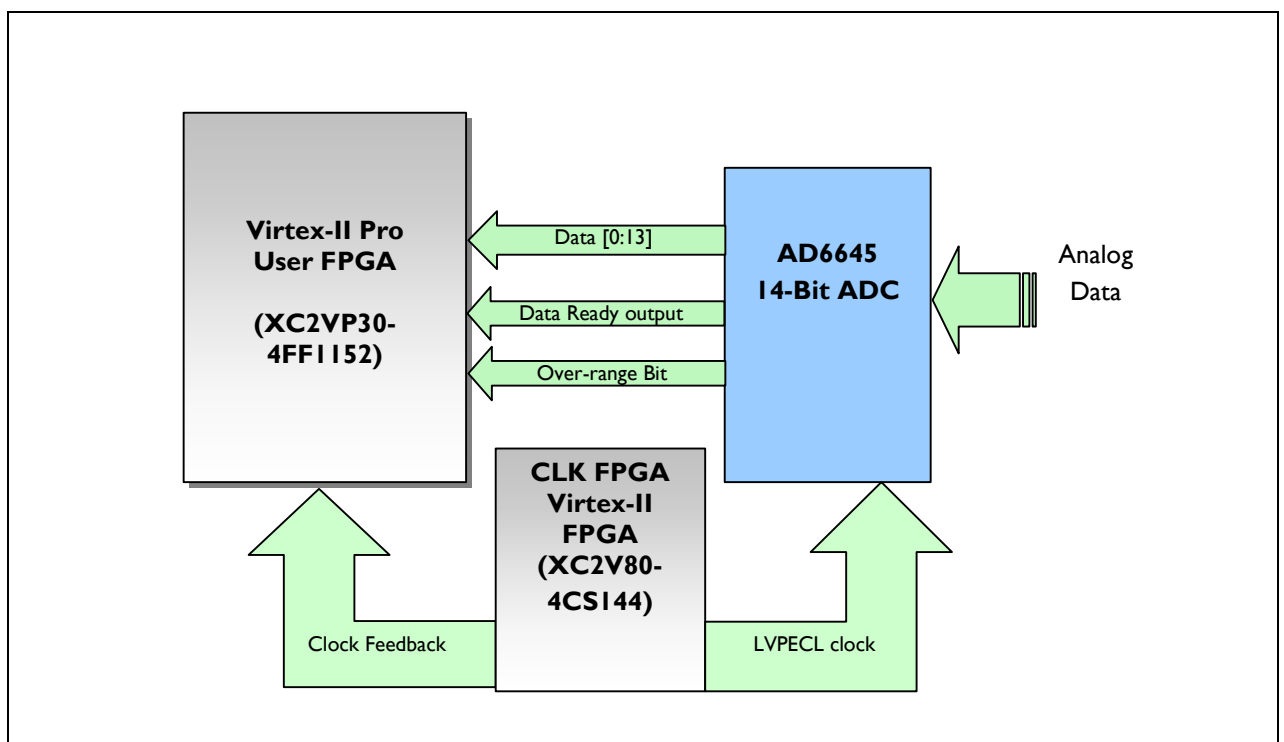


Figure 10: ADC to FPGA Interface

The main features of the on board ADC channels are:

- 14-bit ADC resolution, 2's complement format.
- 105MSPS sampling data rate.
- Single-ended 50Ω impedance analog inputs or with population changes differential inputs.
- 3rd order filter on analog inputs (-3dB point of 58MHz).
- ADCs clocked differentially.

4.1.1 ADC Architecture

The ADC (AD6645) is straightforward to operate - the user is only required to apply data and a clock input. There are no set-up or control signals. **Figure 11 on page 20** shows the internal architecture of the ADC.

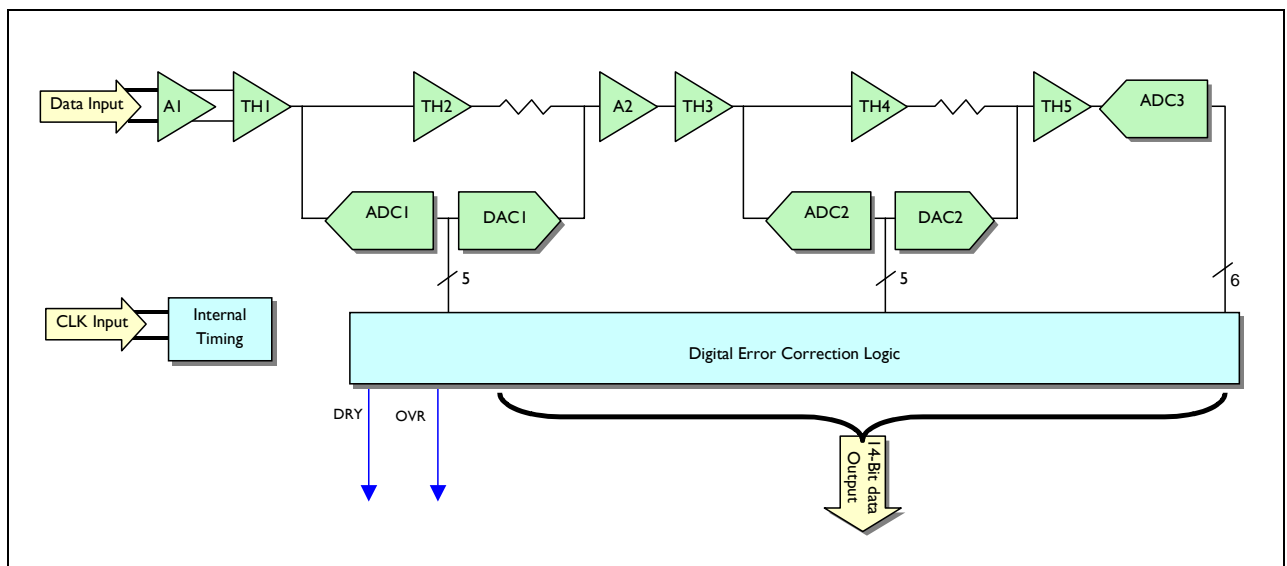


Figure 11: ADC (AD6645) Internal Architecture

Theory of ADC (AD6645) Operation

The AD6645 has complementary analog inputs; each input is centred at 2.4V and should swing $\pm 0.55V$ around this 2.4V reference. This means that the differential analog input signal will be 2.2V_{pp} as both input signals (AIN and AIN#) are 180 degrees out of phase with each other.

When data arrives at the AD6645, both analog inputs are buffered prior to the first track-and-hold (TH1). The analog signals are held in TH1 while the ENCODE (CLK) pulse is high and then data is applied to the input of a 5-bit coarse ADC (ADC1). The digital output of ADC1 is fed into the 5-bit DAC1. The output from the DAC1 is subtracted from the delayed analog signal at the input of TH3 to generate a first residue signal. The purpose of TH2 is to provide a pipeline delay to compensate for the digital delay of ADC1.

This first residue signal is then applied to the second conversion stage. Again a similar process is achieved through this stage, which finally leads onto obtaining a second residue signal that is applied to a third 6-bit ADC. Finally the digital outputs of ADC1, ADC2 and ADC3 are added together and corrected in the digital error correction logic to generate the final output.

4.2 Hardware

4.2.1 Analog ADC Inputs

The inputs to the ADC devices are connected via MCX connectors on the front of the module. The standard shipped configuration exhibits 50Ω single-ended inputs, each featuring a 3rd order anti-aliasing filter with a -3dB point at 58MHz. The AD6645 ADC inputs are connected to the AD8138 op-amp which is connected directly to the MCX input. The recommended maximum signal magnitude at the MCX input to attain best performance characteristics is:

$$2\text{ V p-p or } \pm 1\text{ V}$$

The Kit has five through-hole MCX connectors that allow interfacing to and from the module. All Analog Input and Output signals to the BenADDA DIME-II module are conducted via four MCX connectors on top of the module. The fifth MCX connector provides an input source for an external clock. [Figure 12 on page 21](#) outlines the positioning of these connectors.

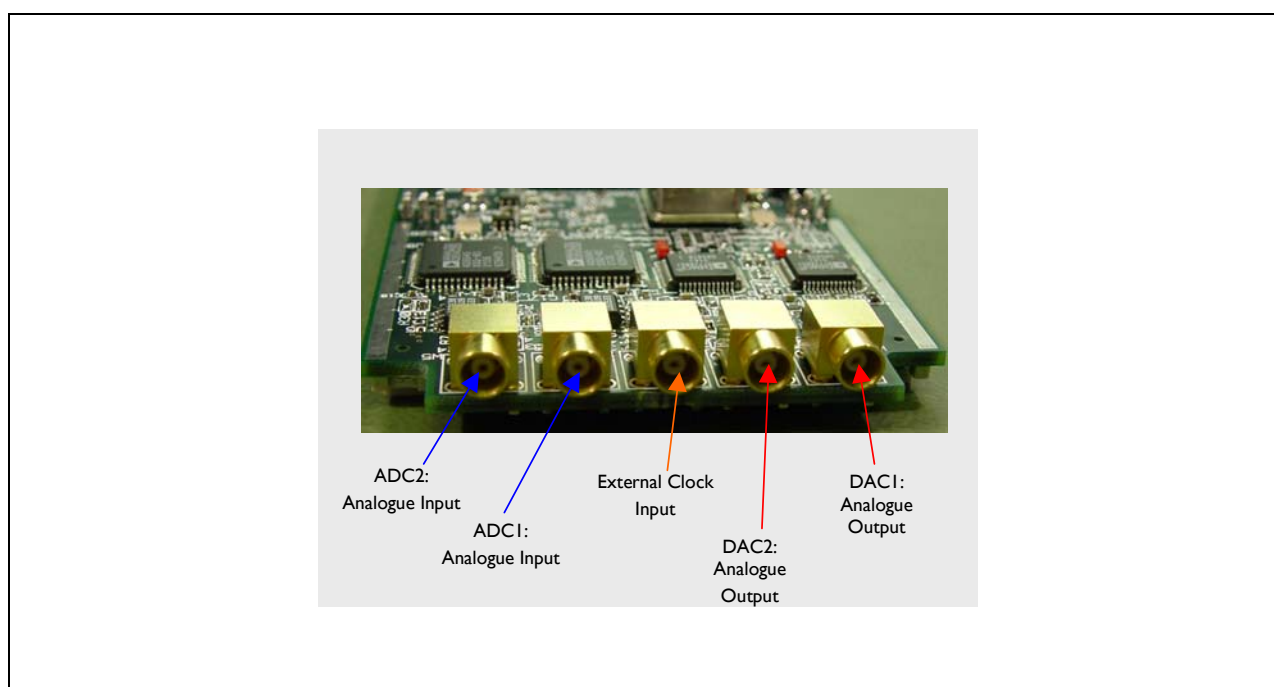


Figure 12: MCX Connectors

[Figure 12 on page 21](#) shows there are two Analog Input channels, two Analog Output channels and one external clock source input. Details on the DAC inputs and the external clock input are provided in [“DACs” on page 31](#).



Please note when the Kit is still in the blue board case the function of each MCX connector is labelled on the lid.

Interfacing to MCX connectors via supplied cable

The Kit is supplied with five cables that are suitable for connecting between the on board MCX connectors and the user input/output BNC connections. The cable assembly is shown in [Figure 13 on page 22](#).

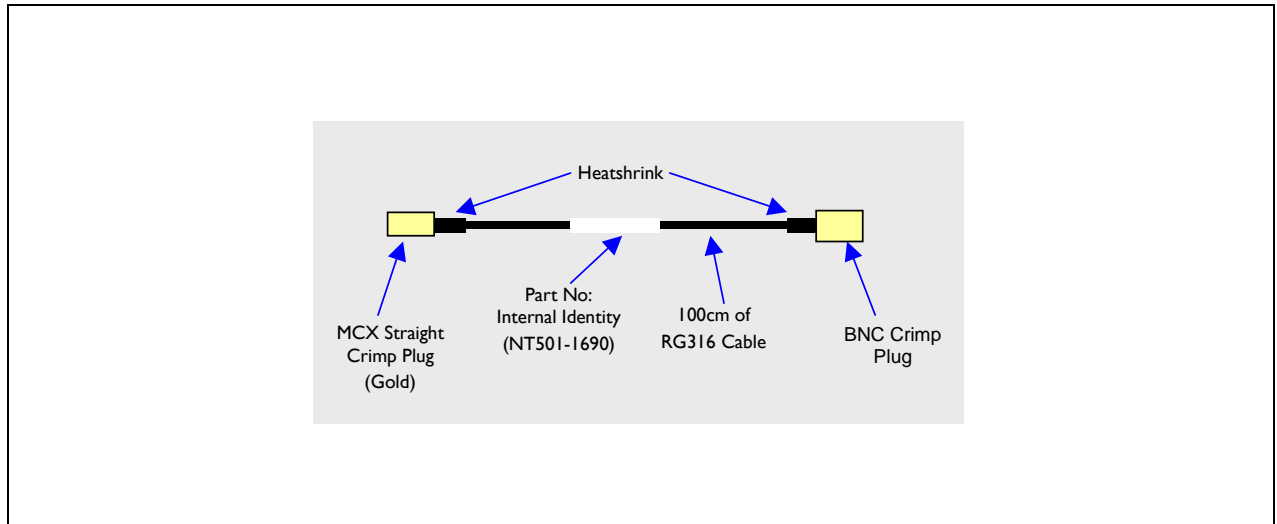


Figure 13: Supplied Cable Assembly

The MCX connectors are a "push" fit design; therefore the MCX crimp plug on the supplied cable pushes into the MCX connector on the Kit hardware.

4.2.2 ADC Clocking

Each ADC device is clocked directly by an independent differential, LVPECL signal. This LVPECL signal is driven from the Virtex-II XC2V80-4CS144 FPGA (Clock FPGA) which is dedicated to managing the various methods for clocking each ADC and DAC device in the Kit. The way the ADCs are clocked depends on the bitfile that is assigned to the dedicated Clock FPGA. A number of clock sources can be used through the Clock FPGA including:

- On board 105MHz crystal.
- External clock input via the middle MCX connector.
- Clocks from the programmable oscillators available in the Kit.



Please note that the ADC devices (AD6645) can only support a clock input of up to 105MHz. This is important if you wish to use one of the DIME Clocks (i.e. CLKA, CLKB, CLKC) that can be set higher than 105MHz.

Please refer to ["Clocks" on page 65](#) for details on how the Clock FPGA can be used.

4.2.3 Analog Performance

The ADC channels on the BenADDA DIME-II module can typically resolve between 11 and 12 bits using the on board 105MHz oscillator. The ENOB (Effective Number of Bits) characteristics were obtained using signals at -1dBFS. An improvement on these figures would be expected if the input signal levels were reduced slightly. The SNR (Signal to Noise Ratio) of the AD6645 is, at best, 74.5dB suggesting that the maximum number of bits attainable is 12.1.



Please note that the measured ENOB in the BIST(Built in Self Test) will be slightly reduced due to slight loss/interference pickup as a result of the BNC jack-to-jack adaptors.

The ENOB characteristics worsen slightly when using an external clock. The ADC channels can resolve between 11 and 12.5 bits. The frequency responses of both ADC channels display a -3dB point of 58MHz. Each channel features a 3rd order passive low pass filter which, with a theoretical cut-off frequency of 58MHz, would expect to suppress signals at 350MHz by 60dB. The actual attenuation at this frequency is -59dB. The pass band ripple measures 0.15dB whilst the channel to channel frequency response variation is limited to 0.085dB over $f_s/2$.

Both ADC channels display a measure of crosstalk: -80.5dB for Channel 1; -73dB for Channel 2. However, the Effective Resolution in both cases measures 12.4 bits and 12.3 bits respectively. These figures are higher than the ENOB characteristics and suggest there is very little contribution to the pass band noise.

For more detailed information on the analog performance of the hardware please refer to the *BenADDA Datasheet* supplied on the XtremeDSP Development Kit Pro CD at the location 'CD ROM Drive\Documentation\Datasheets\'.

4.2.4 ADC Front End Configuration

The BenADDA DIME-II module used in the Kit has been designed to take either single-ended or differential analog inputs. The choice between single-ended or differential inputs is made when the board is built. Therefore in the case of the XtremeDSP Development Kit Pro, the module configuration has been set for single-ended inputs.

The analog input signal is dc-coupled through a differential op-amp (AD8138), which is fed into the actual ADC (AD6645). The op-amp has been configured to support either single-ended or differential inputs and always outputs a differential signal. This means that all data will be input to the AD6645 differentially which helps to reduce noise induced on the input signal. The hardware has also been designed with a 3rd order filter on the front end of the ADC which helps to reduce the overall noise induced on the input signal, thereby improving the resolution at the output of the ADC circuit.



Please note that the following information on specific builds for different front-end configurations is for reference only. Attempted repair or alteration of the goods as supplied by Nallatech immediately invalidates the warranty. Nallatech will not provide support upon alteration and cannot guarantee performance characteristics subsequently obtained.

Channel I

The schematic diagram shows a differential amplifier circuit. The input signal is connected to a 51 Ω resistor, which is then connected to the non-inverting input (+) of the AD8138 op-amp. The inverting input (-) is connected to a 510 Ω resistor, which is also connected to the output of the op-amp. The op-amp is configured with a feedback network consisting of a 510 Ω resistor and a capacitor C_5 in parallel. The output of the op-amp is connected to a network of resistors R_{19} and R_{20} , inductors L_2 and L_3 , and a capacitor C_{15} . The final output is differential, labeled $\overline{\text{AIN}}$ and AIN . The circuit also includes a 51 Ω resistor at the input, a 510 Ω resistor at the input, and a capacitor C_4 at the output.

Filter	-3dB Point	R ₁₉	R ₂₀	C ₄	C ₅	C ₁₅	L ₂	L ₃	Roll off / decade
Present ^a	57.9MHz	33Ω	33Ω	3.3pF	3.3pF	33pF	82nH	82nH	-60dB
Absent	250MHz ^b	24Ω	24Ω	1pF +/- 0.25pF	1pF +/- 0.25pF	DNP ^c	0Ω ^d	0Ω	-

- a. Default Build
- b. Analog BW of AD6645
- c. DNP = Do Not Populate
- d. Resistor Value



Channel 2

The BenADDA DIME-II module front end for Channel 2 is illustrated in [Figure 15 on page 25](#).

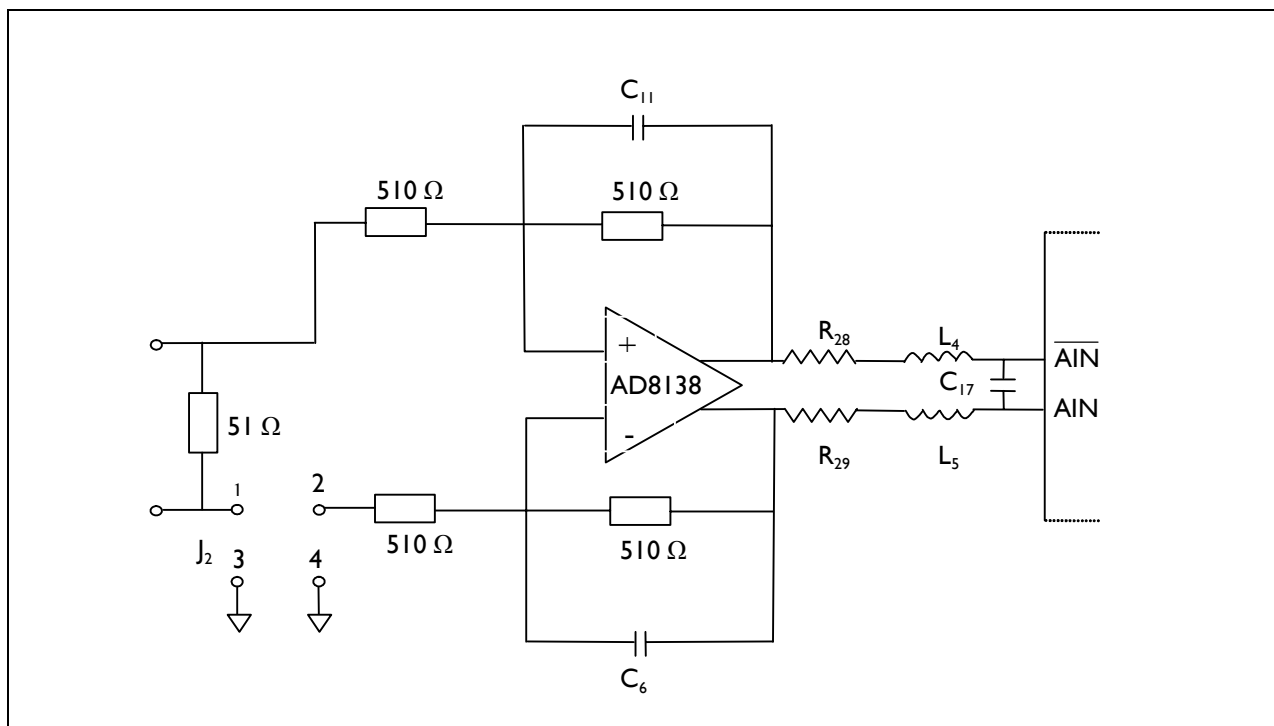


Figure 15: BenADDA Front End Schematic for Channel 2

The adjustable filter components in [Figure 15 on page 25](#) have the reference designators identified. [Table 4 on page 25](#) shows the possible filtering options and the related component values. Please refer to the assembly drawing on [page 27](#) for the location of these reference designators.

Filter	-3dB Point	R ₂₈	R ₂₉	C ₆	C ₁₁	C ₁₇	L ₄	L ₅	Roll off / decade
Present ^a	57.9MHz	33Ω	33Ω	3.3pF	3.3pF	33pF	82nH	82nH	-60dB
Absent	250MHz ^b	24Ω	24Ω	1pF +/- 0.25pF	1pF +/- 0.25pF	DNP ^c	0Ω ^d	0Ω	-

Table 4: Filter Component Values for Channel 2

- a. Default Build
- b. Analog BW of AD6644
- c. DNP = Do Not Populate
- d. Resistor Value



Note that component values have 1% tolerance.

Coupling Specifications

The BenADDA DIME-II module front end can be configured for Single-ended or Differential coupling. The configurable jumper J1 in [Figure 14 on page 24](#) and J2 in [Figure 15 on page 25](#), determine the coupling arrangement. [Table 5 on page 26](#) displays the jumper settings for Single-Ended coupling whilst [Table 6 on page 26](#) displays the jumper settings for Differential coupling.

Jumper J1		Jumper J2	
Position 1-3	0Ω	Position 1-3	0Ω
Position 2-4	24Ω	Position 2-4	24Ω

Table 5: Single-Ended Configuration

Jumper J1		Jumper J2	
Position 1-2	0Ω	Position 1-2	0Ω
Position 3-4	DNP ^a	Position 3-4	DNP

Table 6: Differential Configuration

a. DNP = Do Not Populate

Default Configuration Settings

Unless otherwise stated, the front end of the BenADDA DIME-II module in the Kit is built to the configuration described in [Table 7 on page 26](#) and [Table 8 on page 26](#). The default settings include a filter and single-ended coupling.

FilterChannel 1		FilterChannel 2	
R ₁₉	33Ω	R ₂₈	33Ω
R ₂₀	33Ω	R ₂₉	33Ω
C ₄	3.3pF	C ₆	3.3pF
C ₅	3.3pF	C ₁₁	3.3pF
C ₁₅	33pF	C ₁₇	33pF
L ₂	82nH	L ₄	82nH
L ₃	82nH	L ₅	82nH

Table 7: Default Filter Configuration



Note that component values have 1% tolerance.

Jumper J1		Jumper J2	
Position 1-3	0Ω	Position 1-3	0Ω
Position 2-4	24Ω	Position 2-4	24Ω

Table 8: Default Coupling Configuration

Assembly Drawing

The reference designators for the filters and the jumpers are highlighted in **Figure 16 on page 27**.

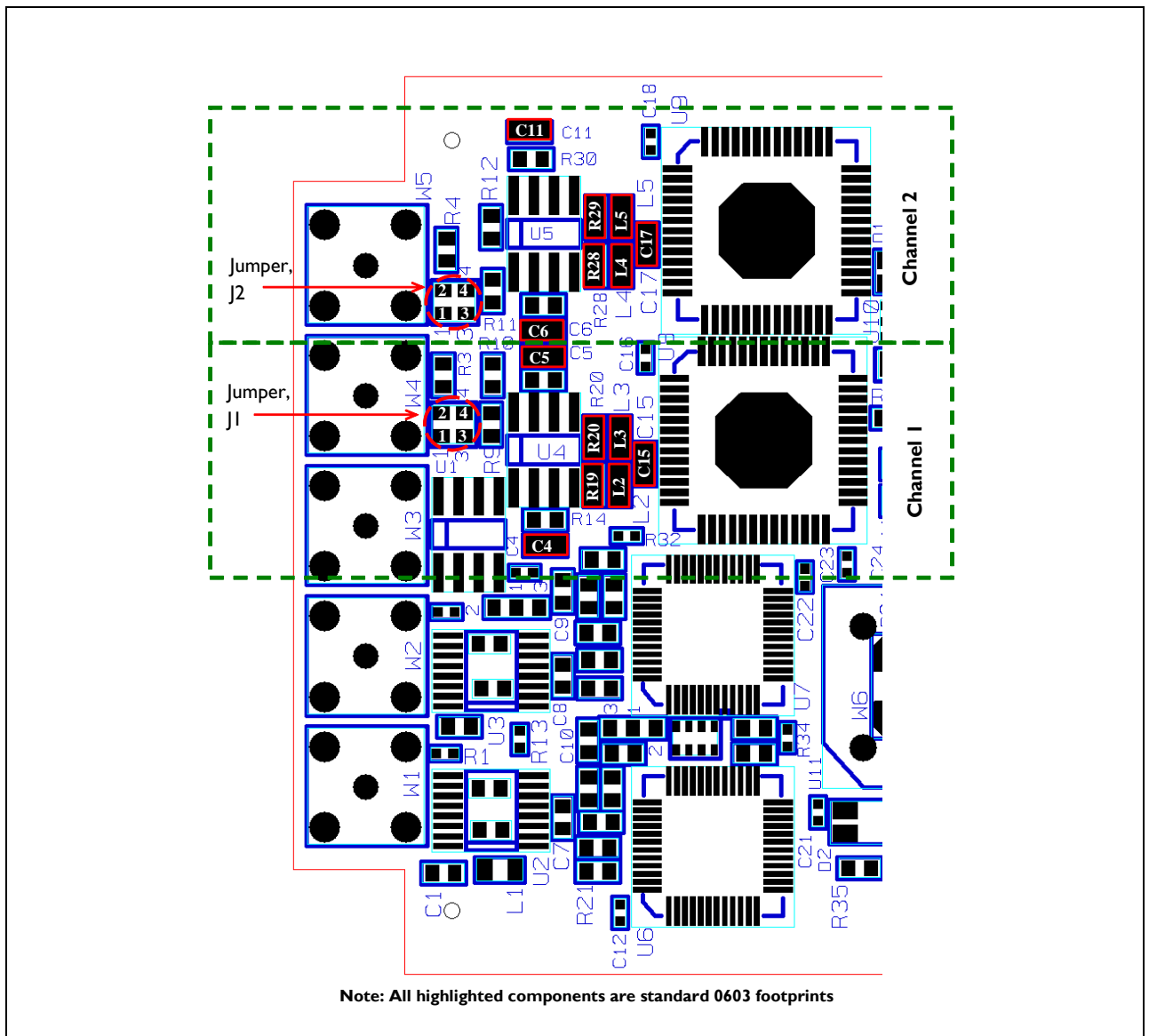


Figure 16: Assembly Drawing

4.3 Firmware

The ADC is connected to the main User FPGA via a number of signals. The following signals are highlighted here for ADC1 although there are a corresponding set of signals for ADC2.

ADC1_D(13 downto 0)

This is the two-complement output of the ADC. Bit 13 is the most significant bit.

ADCI_DRY

This is the Data Ready Output from the ADC. This is an inverted and delayed version of the encode clock. Any change in the duty cycle of the clock will correspondingly change the duty cycle of DRY.

ADCI_OVR

Overrange Bit; high indicates analog input exceeds \pm Full Scale (FS).



Please note that the output of the ADC is in twos-complement format.

The only signals you require are the actual data bits i.e. ADC1_D(13 downto 0) for ADC1 and ADC2_D(13 downto 0) for ADC2. It is recommended that when these signals come into your FPGA design they are registered/clocked in by your relevant design clock. **Figure 17 on page 28** shows an example.

```
-----
-- register the adc inputs
-----
adcDataRegisters : process (CLK1_FB, RESET1)
begin
  if RESET1 = '0' then
    ADC1_Di <= (others => '0');
    ADC2_Di <= (others => '0');
  elsif CLK1_FB = '1' and CLK1_FB'event then
    ADC1_Di <= ADC1_D;
    ADC2_Di <= ADC2_D;
  end if;
end process;
```

Figure 17: Registering of the ADC Inputs

The full version of this code snippet can be found in the XtremeDSP Development Kit Pro CD at the following location: 'CD ROM Drive\Examples\simple_adc_hookup'.

The main use of the ADC1_OVR and ADC2_OVR signals is to detect if the input is out of range. This can be used as a valid signal to part of your design. The use of this signal depends upon your application. The ADC1_DRY and ADC2_DRY signals indicate when there is valid data on the databuses from the ADCs. This is only of use when the ADC is in a different clock domain from the input registers that are capturing the databus values. The DRY signals can be used to indicate when it is valid to capture data from the ADC data outputs.

4.3.1 Timing Constraints

It is necessary to account for the setup and hold times required for the ADC devices. These timings are given in full in the *Analog Devices AD6645 Datasheet* on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Documentation\Datasheets'.

The timing requirements depend on your design although you should account for a 1ns delay for traversing the PCB in your calculations. For example in the 'simple_adc_hookup' example shown in **Figure 17 on page 28** the following timing constraint on the data signals is used.

NET "adcI_d<*>" OFFSET = IN 5 ns BEFORE "clkI_fb";

This is a suggested constraint and the actual constraint depends on the design.

4.4 Software

Please refer to the *XtremeDSP Analog Capture Datasheet* on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Application_Notes\NT302-0035_XtremeDSP_Kit_Analogue_Capture\Documents'. This provides an example design for Analog capture using the XtremeDSP Development Kit Pro.

Section 5

DACs

In this section:

- Introduction to the DACs
- Hardware Aspects
- Firmware Aspects
- Software Aspects

5.1 Introduction

The BenADDA DIME-II module used in the XtremeDSP Development Kit Pro has two analog output channels, with each channel having independent data and control signals from the FPGA. Two sets of 14-bit wide data busses are fed to the two DACs (AD9772A devices), each of which has an isolated supply and ground plane. [Figure 18 on page 31](#) illustrates the interfacing between one of the DACs and the FPGA.

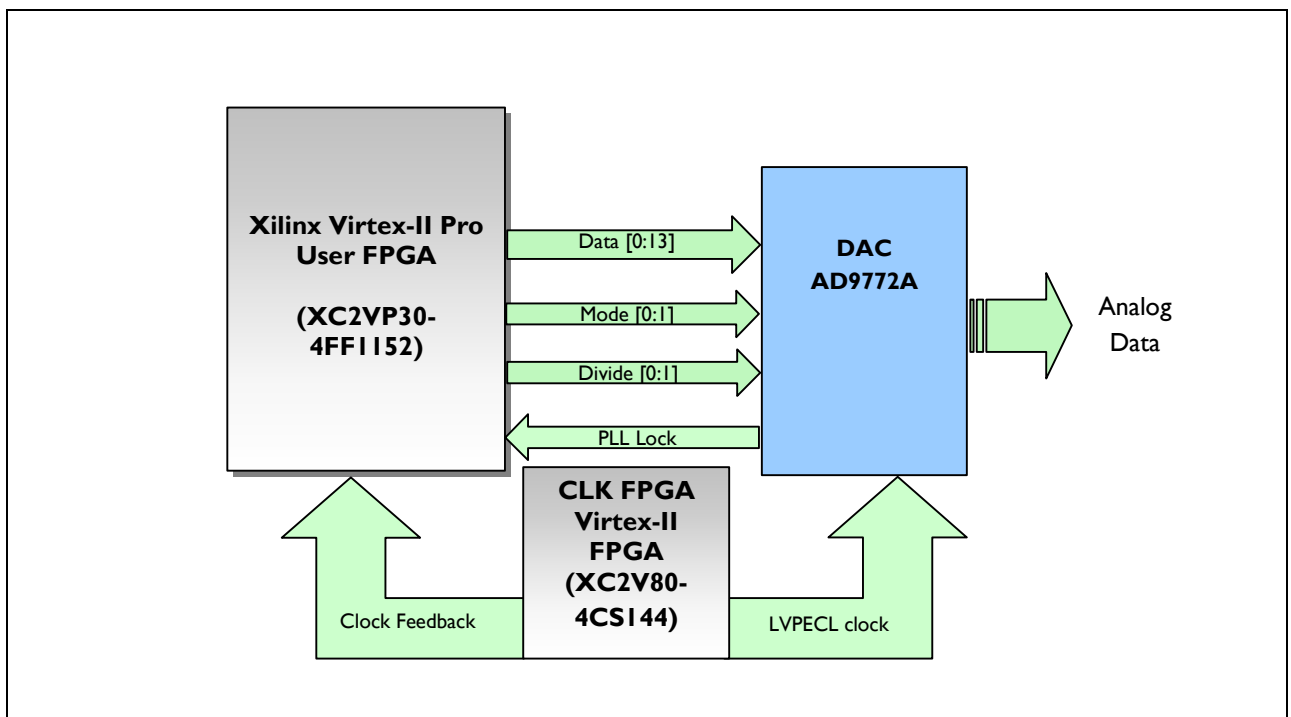


Figure 18: DAC Interface

The DAC device offers 14-bit resolution and a maximum conversion rate of 160MSPS. Additional control signals exist between the DAC and the FPGA to enable full control of the DACs' functionality.

The main features of the AD9772A are:

- 14-bit DAC resolution. Note that they are offset-binary input - for more details please see ["Firmware" on page 39](#).
- 160MSPS max input data rate.
- LVPECL clock inputs from the XC2V80-4CS144 Clock FPGA.
- internal Phase-Locked Loop (PLL) clock multiplier device feature.
- single ended (DC coupled) 50Ω outputs via MCX connectors as standard.

5.1.1 DAC Architecture

The AD9772A's architecture comprises four key areas as shown in [Figure 19 on page 32](#).

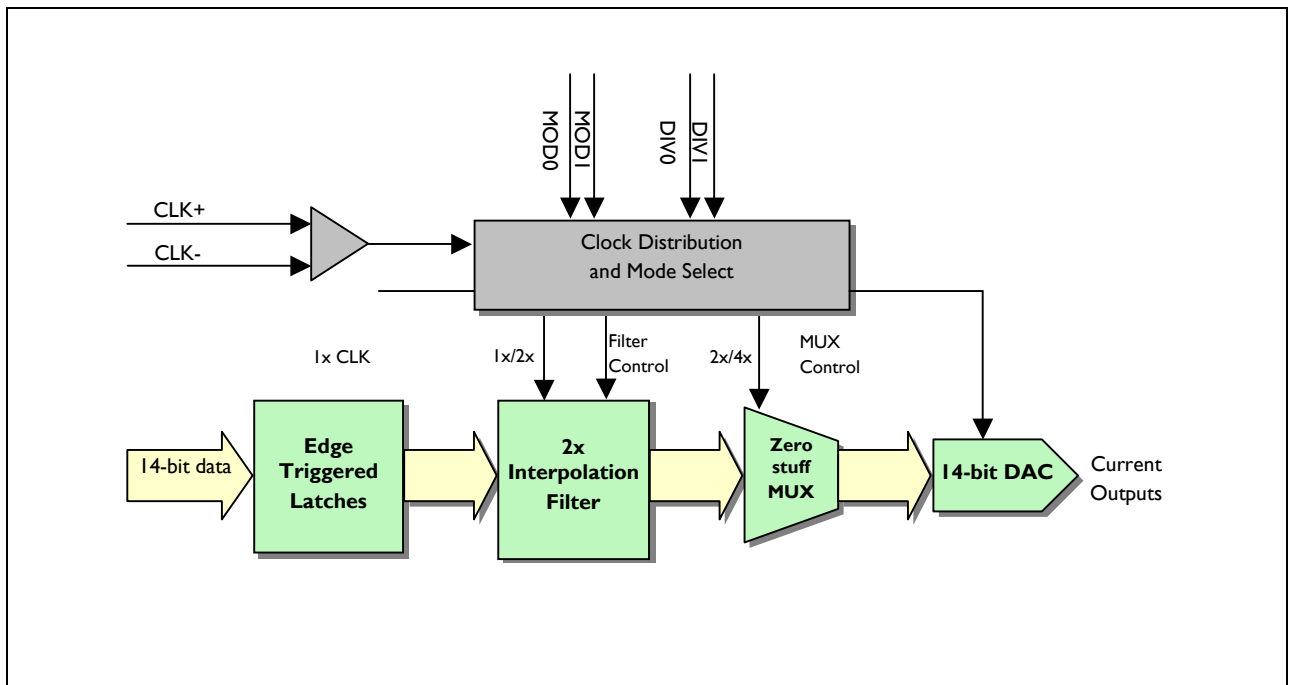


Figure 19: AD9772 Architecture

[Figure 19 on page 32](#) shows the internal architecture of the AD9772A. Initially, the user feeds 14-bits of data into the AD9772A. This data is latched into edge-triggered latches on the rising edge of the reference clock, interpolated by a factor of 2 by the digital filter, and then fed to the 14-bit DAC. The filter characteristic can be set to either low pass or high pass for baseband and IF applications respectively. The MOD0 input is used to control this function of the AD9772A.

The interpolated data can feed the DAC directly or undergo a "zero-stuffing" process, enabled using MOD1. This process involves inserting a mid-scale sample after every data sample originating from the digital filter, which improves the pass-band flatness of the DAC and also allows for the extraction of higher frequency images.

The AD9772A generates a variety of clock frequencies to operate its elements at the correct rates. To achieve these frequencies, it utilizes an internal PLL whose VCO can generate clock rates of up to 400MSPS. The AD9772A can be operated with the PLL enabled or disabled (both operations are supported on the BenADDA DIME-II module used in the Kit). The combination of the MOD and DIV input control signals determines the effective operation of the DAC devices. The hardware section which follows provides more details on the MOD and DIV pins.

Full details of the DAC (AD9772A) device are provided in the *Analog Devices AD9772A Datasheet* on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Documentation\Datasheets'.

5.2 Hardware

The BenADDA DIME-II module supports two output configurations - a single-ended DC coupled output, and a differential directly coupled output. The standard Kit is configured to have single ended DC coupled outputs from the DACs. Both these output configurations are described later in this section. The internal PLL clock multiplier of the AD9772A is also described to provide you with an insight into the internal operations of the DAC.

5.2.1 PLL Clock Multiplier

Figure 20 on page 33 illustrates how the BenADDA DIME-II module supports PLL enabled or disabled.



Note that PLL is enabled by default. Please contact Nallatech if you wish to disable the PLL function.

To supply the PLLVDD pin, you can populate a jumper to supply the pin with a 3.3v signal, or else tie the pin to ground. The supply for the PLLVDD pin is shared with the supply of the CLKVDD pin and both the CLK and PLL grounds on the chip share the same separate ground plane. When the PLL is disabled and the PLLVDD pin is tied to ground, the filter components for the LPF pin (internal loop filter for the PLL) on the DAC are not populated. This leaves the LPF with an open connection.

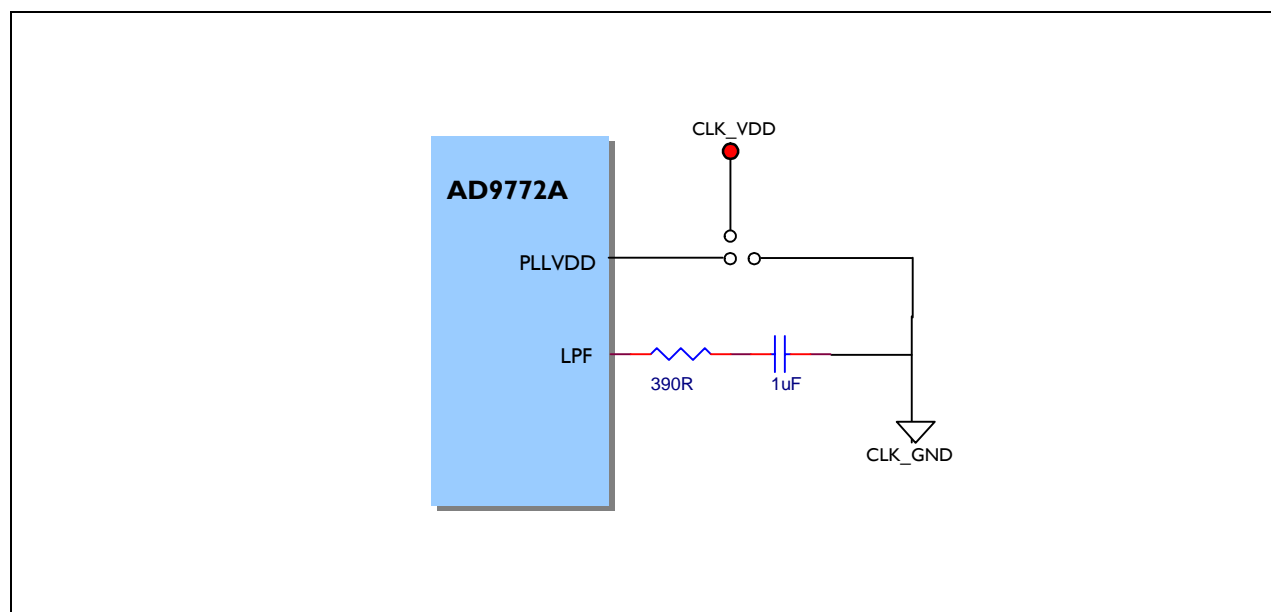


Figure 20: PLL Jumper Option

When the PLL is set to the default value enabled, the AD9772A will generate its own 2x clock from the reference clock. This allows you to transmit data at the same rate as the reference clock. The internal PLL can also generate another phase clock that allows the zero-stuffing option to be selected under the same circumstances.

If the PLL is disabled, the input data rate must be half the reference clock frequency. This is due to the interpolation filter that adds extra samples every other clock cycle. Additionally, if the zero-stuffing option is selected, the input data must be one quarter of the reference clock frequency. For example, the maximum reference clock of 160MSPS, with the PLL disabled and the zero-stuffing option selected, gives a maximum input data rate of 40MHz.

The internal PLL also deals with the phase relationship between the data and the reference clock. This means that when PLL is enabled, you do not need to use the RESET input to ensure correct alignment of clock and data.

If the PLL is disabled, consult the *Analog Devices AD9772A datasheet* provided on the *XtremeDSP Development Kit Pro CD* at the location 'CD ROM Drive\Documentation\Datasheets\'. This provides more information on how the RESET input is used to ensure correct synchronization.

Table 9 on page 34 provides a summary of the DAC input data rates.

	PLL Disabled		PLL Enabled	
	Zero-stuffing OFF	Zero-stuffing ON	Zero-stuffing OFF	Zero-stuffing ON
Input Data Rate	1/2 reference clock	1/4 reference clock	1x reference clock	1x reference clock

Table 9: DAC Input Data Rates

A PLL_LOCKED signal from each AD9772A is connected to the FPGA (see note in "PLL Clock Multiplier" on page 33) on the BenADDA DIME-II module. This signal goes high to indicate the PLL has "locked" to the input reference clock. If the PLL is not locked, due to the PLL being disabled or an unstable clock, PLL_LOCKED toggles between high and low in an asynchronous manner.

5.2.2 DAC Modes of Operation

In the introduction the MOD and DIV input control pins were highlighted. These control the way in which the DAC operates. The MOD and the DIV signals are controlled from the main User FPGA (XC2VP30-4FF1152).

As outlined earlier in this section, the interpolation filter can be set to either a low or high pass characteristic, depending on whether you wish to capture baseband or IF signals. This feature of the AD9772A, and also the zero-stuffing option, is controlled by the FPGA. The operation of the 'MOD' pins is summarized in Table 10 on page 34.

Digital Mode	MOD0	MOD1	Digital Filter	Zero-Stuffing
Baseband	0	0	LOW	NO
Baseband	0	1	LOW	YES
Direct IF	1	0	HIGH	NO
Direct IF	1	1	HIGH	YES

Table 10: Controlling Digital Modes of AD9772A

The AD9772A contains an internal Voltage Controlled Oscillator (VCO), which can operate at up to 400MSPS. To ensure the optimum phase noise and successful "locking" of the PLL, a pre-scalar stage is incorporated to allow the sampling clock to be divided down as required for slower data rates. The divide-by-ratio is selected by the DIV0 and DIV1 inputs, as shown in Table 11 on page 34.

Input Data Rate (MSPS)	MOD1	DIV1	DIV0	Zero-stuffing	Divide-by-N-ratio
48-160	0	0	0	No	1
24-100	0	0	1	No	2
12-50	0	1	0	No	4
6-25	0	1	1	No	8
24-100	1	0	0	Yes	1

Table 11: Recommended Presale Ratio Settings

Input Data Rate (MSPS)	MOD I	DIV I	DIV0	Zero-stuffing	Divide-by-N-ratio
12-50	1	0	1	Yes	2
6-25	1	1	0	Yes	4
3-12.5	1	1	1	Yes	8

Table 11: Recommended Presale Ratio Settings

5.2.3 Analog DAC Outputs

The Kit has five through-hole MCX connectors that allow interfacing to and from the module. All Analog Input and Output signals to the BenADDA DIME-II module are conducted via four MCX connectors on the top of the module. The fifth MCX connector provides an input source for an external clock. Figure 21 on page 35 outlines the positioning of these connectors.

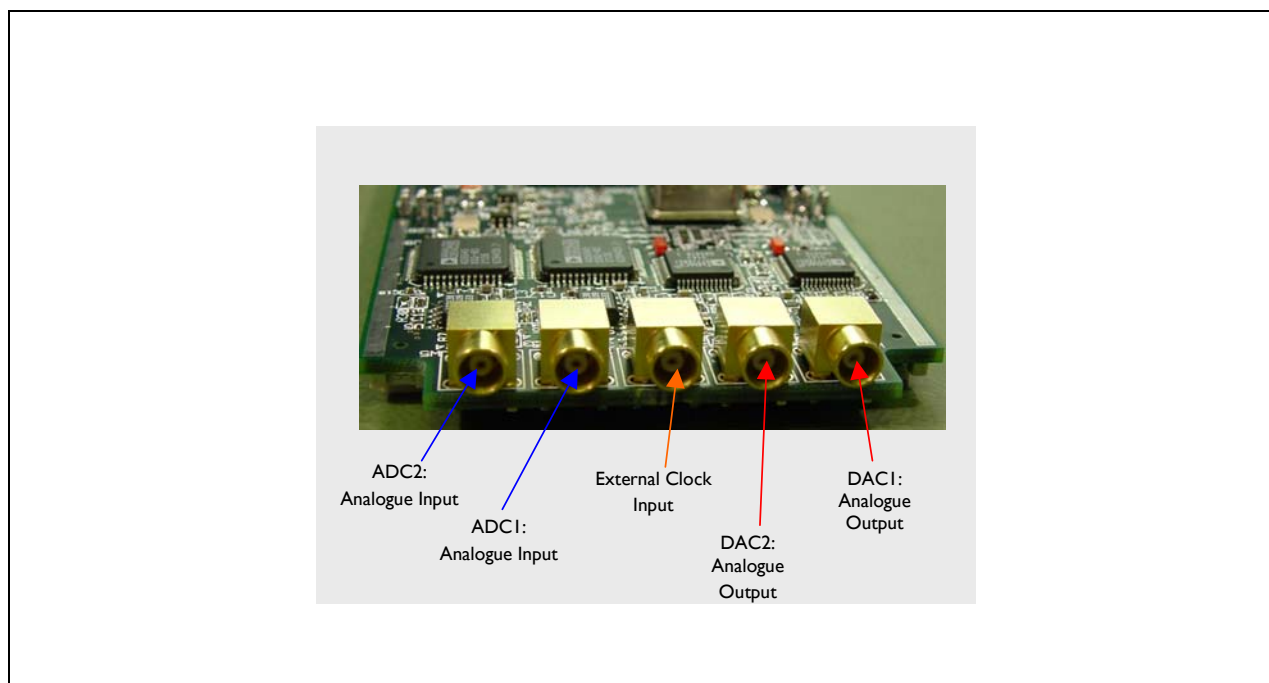


Figure 21: MCX connectors



Please note when the Kit is still in the blue board case the function of each MCX connector is labelled on the lid.

Interfacing to MCX connectors via supplied cable

The Kit is supplied with five cables that are suitable for connecting between the on board MCX connectors and a user input/output BNC connections. The cable assembly is shown in [Figure 22 on page 36](#).

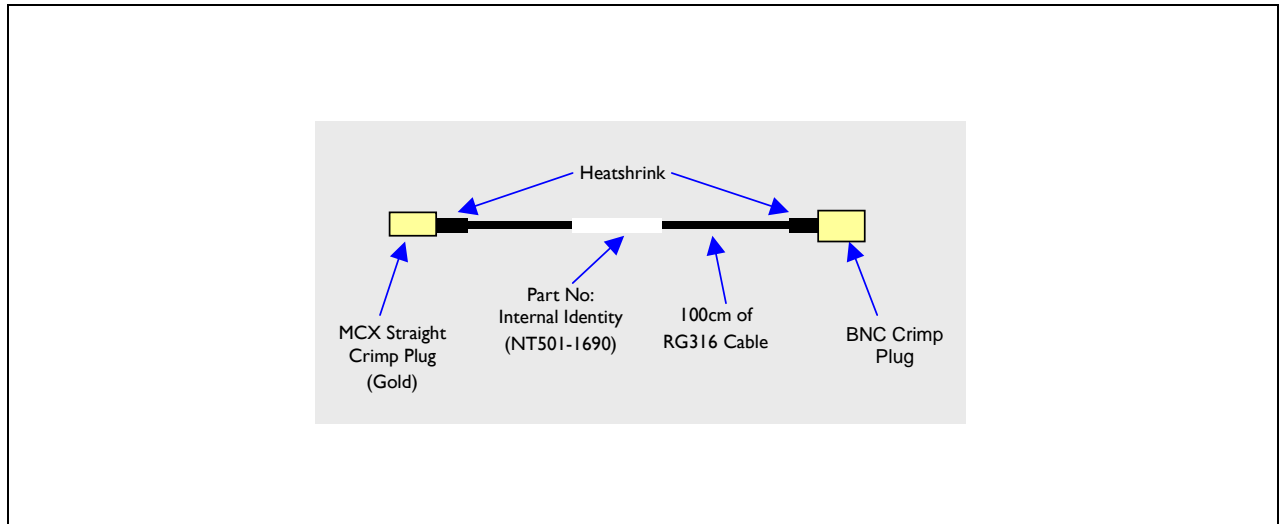


Figure 22: Supplied Cable Assembly

The MCX connectors are a "push" fit design; therefore the MCX crimp plug on the supplied cable pushes into the MCX connector on the Kit hardware. Please see the following section for details on the output configuration of the DACs.

5.2.4 Output Configurations

The AD9772A DAC supports can support two output configurations. The following are supported as build options on the BenADDA DIME-II module:

- Single-ended 50Ω output, DC-coupling using an op-amp (Kit default)
- Differential outputs using termination resistors (Custom build from Nallatech)



Please note the following specific information about the builds for different front end configurations is for reference only. Attempted repair or alteration of the goods as supplied by Nallatech immediately invalidates the warranty. Nallatech will not provide support upon alteration and cannot guarantee performance characteristics subsequently obtained.

Single-Ended DC-Coupling Using an Op-Amp (XtremeDSP Development Kit Pro Default)

The op-amp configuration is suitable for applications requiring DC coupling. Figure 23 on page 37 illustrates the set-up adopted when using an op-amp configuration at the output of the AD9772A. The full scale current output, I_{OUTFS} , from the AD9772A DAC, is 20mA. When this output current is driven through the two 50Ω resistors, a voltage of 2Vpp appears at the MAX4144 input. Since the op amp has a fixed gain of 2, the MAX4144 output is 4Vpp. The 50Ω DAC channel output impedance combined with the assumed 50Ω impedance of the connecting system at the output MCX connector ensures that the signal magnitude at the MCX connector is 2Vpp.

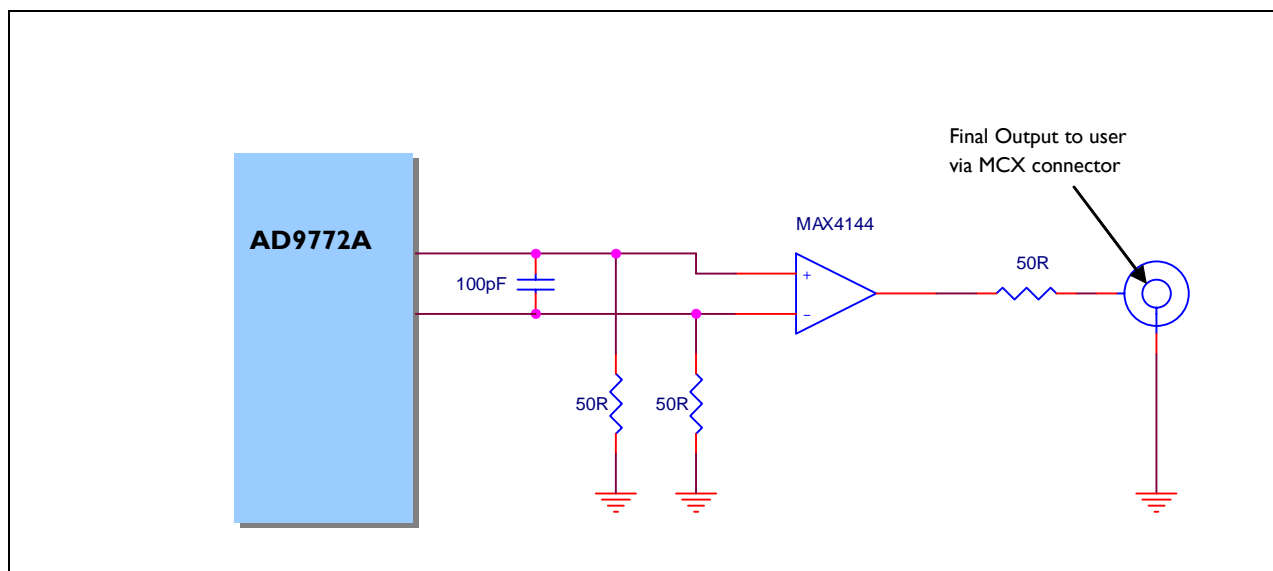


Figure 23: AD9772A Single Ended DC-coupled Output



The op-amp used in the design is an instrumentation amplifier with an inverting X2 gain from MAXIM. Therefore the output configuration will drive an output voltage of $\pm 1V$ into a 50Ω load. Please note that the output from the DAC itself is inverted due to the use of an inverting op-amp.

Differential Outputs using Termination Resistors (Nallatech Custom Build)

It is also possible to drive differential outputs using a pair of termination resistors. However, this is a specific build option from Nallatech and is not provided with the standard Kit. It is included here for completeness.

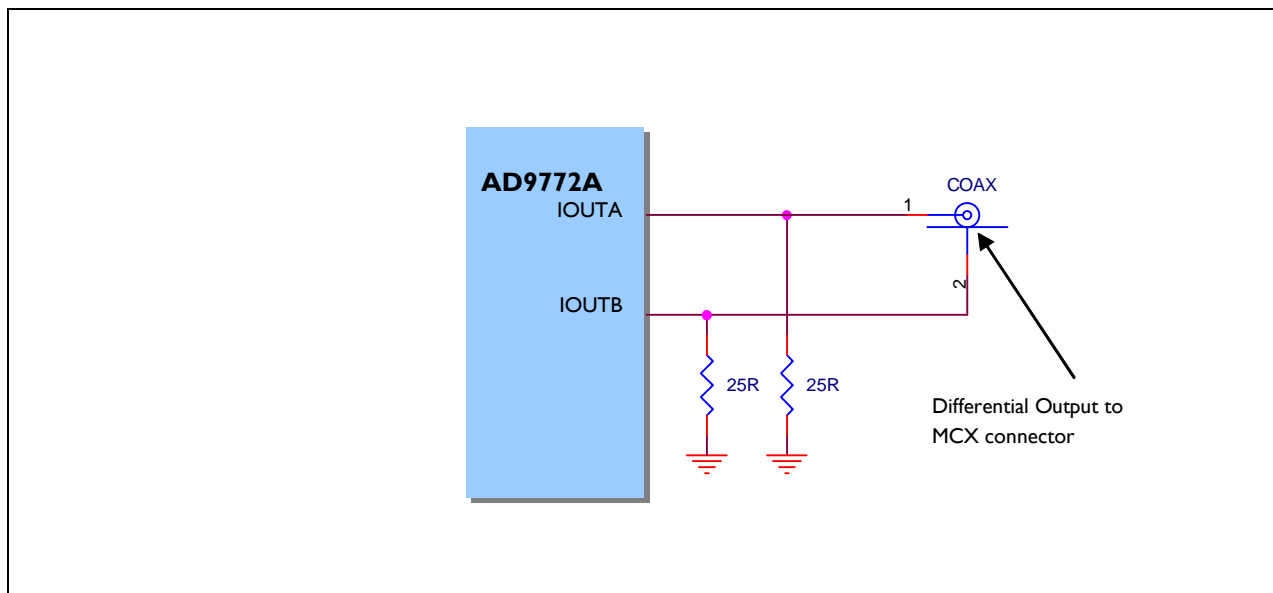


Figure 24: AD9772 Differential Directly Coupled Option

When terminated into a 50Ω load, this option provides a fully differential, 0.5Vp-p output signal swing.

5.2.5 DAC Clocking

Each DAC device is clocked directly by an independent differential, LVPECL signal. This LVPECL signal is driven from Virtex-II XC2V80-4CS144 FPGA (Clock FPGA) which is solely dedicated to managing the various methods for clocking each ADC and DAC device in the Kit. The way the DACs are clocked depends on the bitfile that is assigned to the dedicated Clock FPGA. A number of clock sources can be used through the Clock FPGA including:

- On board 105MHz crystal.
- External clock input via the middle MCX connector.
- Clocks from the programmable oscillators available in the Kit.



Please note that although the DAC devices can support a clock frequency up to 160MHz the ADC devices can only support a clock input up to 65MHz. This is important if you are creating your own specific clock configuration for the Clock FPGA.

Please refer to “Clocks” on page 65 for details on how the Clock FPGA can be used.

5.3 Firmware

The DACs are connected to the main User FPGA via a number of signals. The following signals are highlighted here for DAC1 although there are a set of corresponding signals for ADC2.

DAC1_D(13 downto 0)

This is the offset binary input to the DAC. Bit 13 is the most significant bit. This is an input to the DAC.

DAC1_MOD 0

Invokes digital high-pass filter response (i.e., "half-wave" digital mixing mode). Active High. This is an input to the DAC.

DAC1_MOD 1

Invokes "Zero-Stuffing" Mode. Active High. Note, "quarter-wave" digital mixing occurs with MOD0 also set HIGH. This is an input to the DAC.

DAC1_DIV(1 downto 0)

DIV1 along with DIV0 sets the DAC PLL's prescaler divide ration. This is an input to the DAC.

DAC1_PLLLOCK

This is the lock signal from the internal PLL of the DAC devices. Phase Lock Loop Lock Signal when PLL clock multiplier is enabled. High indicates PLL is locked to input clock. Provides 1× clock output when PLL clock multiplier is disabled. This is an input to the FPGA.

DAC1_RESET

Resets internal divider by bringing momentarily high when PLL is disabled to synchronize internal clock to the input data and/or multiple AD9772A devices. This is an input to the DAC.



Please note that the input to the DACs is offset-binary format.

The control signals should always be set. In particular it is important not to leave the DAC1_RESET and DAC2_RESET control lines floating. If the DAC_RESET controls are not being actively used these should be set low by default. **Figure 25 on page 40** shows the example code for setting fixed DAC control signals.

```

-- set low pass filter response and no zero stuffing for both DACs
DAC1_MOD0 <= '0';
DAC1_MOD1 <= '0';
DAC2_MOD0 <= '0';
DAC2_MOD1 <= '0';

-- disable resets for DACs
DAC1_RESET <= '0';
DAC2_RESET <= '0';

-- optimum settings for sampling rate
DAC1_DIVO <= '1';
DAC1_DIV1 <= '0';
DAC2_DIVO <= '1';
DAC2_DIV1 <= '0';

```

Figure 25: Code Example of Setting fixed DAC Control Signals

5.3.1 Two's Complement vs. Offset Binary Format

The DACs expect an input in offset binary format. In this format:

- An input of ALL '0's is the lowest value and would give the minimum full scale output from the DAC. For this input the DAC device AD9772A will generate an output of -IV.
- An input of ALL '1's (214-1) is the highest value and would give the maximum full scale output from the DAC. For this input the DAC device AD9772A will generate an output of +IV.

There is however an inverting op-amp on the output of each DAC, prior to the signal being output via the MCX connector. This means that the output from the DAC is inverted. Therefore in actual operation:

- An input of ALL '0's is the lowest value and would give the minimum full scale output from the DAC. For this input the DAC device AD9772A will generate an output of +IV.
- An input of ALL '1's (214-1) is the highest value and would give the maximum full scale output from the DAC. For this input the DAC device AD9772A will generate an output of -IV.

Converting from Two's Complement to Offset Binary

It is a straight forward process to convert from two's complement to Offset binary format, which involves taking a two's complement input and inverting the top most bit. [Figure 26 on page 41](#) shows the example code for this conversion.

```
-- digital output of adc to digital input of DAC
DataRegisters : process (CLK_OSC, RST1)
begin
  if RST1 = '0' then
    ADC1  <= "0000000000000000";
    ADC2  <= "0000000000000000";
    DAC1_D <= "0000000000000000";
    DAC2_D <= "0000000000000000";
  elsif CLK_OSC = '1' and CLK_OSC'event then
    ADC1  <= ADC1_D;
    ADC2  <= ADC2_D;
    DAC1_D <= not (not ADC1(13) & ADC1(12 downto 0));
    DAC2_D <= not (not ADC2(13) & ADC2(12 downto 0));
  end if;
end process;
```

Figure 26: Code from adc_to_dac_hookup Example

5.3.2 Timing Constraints

It is necessary to account for the setup and hold times required for the DAC devices. These timings are listed in full in the DAC (AD9772A) *Analog Devices AD9772A Datasheet* on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Documentation\Datasheets'. The timing requirements depend on your design. You should account for a 1ns delay for traversing the PCB in your calculations. In the 'adc_to_dac_hookup' the following example timing constraint on the data signals is used.

NET "dac1_d<*>" OFFSET = OUT 2 ns AFTER "clk1_fb";

This is a suggested constraint and the actual constraint depends on the design. Note that if the control signals are not fixed then similar timing constraints should be associated with them as well.

5.4 Software

Please refer to the *XtremeDSP Analog Capture Datasheet* on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Application_Notes\NT302-0035_XtremeDSP_Kit_Analogue_Capture\Documents'. This provides a basic example design for the DACs on the XtremeDSP Development Kit Pro.

Section 6

ZBT SRAM Memory

In this section:

- Introduction to the ZBT SRAM Memory
 - Hardware Aspects
 - Firmware Aspects
 - Software Aspects
-

6.1 Introduction

The Kit provides two independent banks of ZBT SRAM. Each bank is configured as 512K x 32. This memory can provide on board storage capabilities via a 32-bit data bus to each bank. The main features of the ZBT Memory devices include:

- Fast cycle times: 6ns, 7.5ns and 10ns
- 100% bus utilization
- Advanced control for minimum signal interface
- Single R/W (Read/Write) control pin
- Clock-controlled and registered addresses, data I/Os and control signals
- Common data inputs and data outputs
- Linear or interleaved burst modes

6.2 Hardware

The memory chips are driven exclusively by the User FPGA. Figure 27 on page 44 illustrates the inter-connect between the ZBT SRAM and the User FPGA.

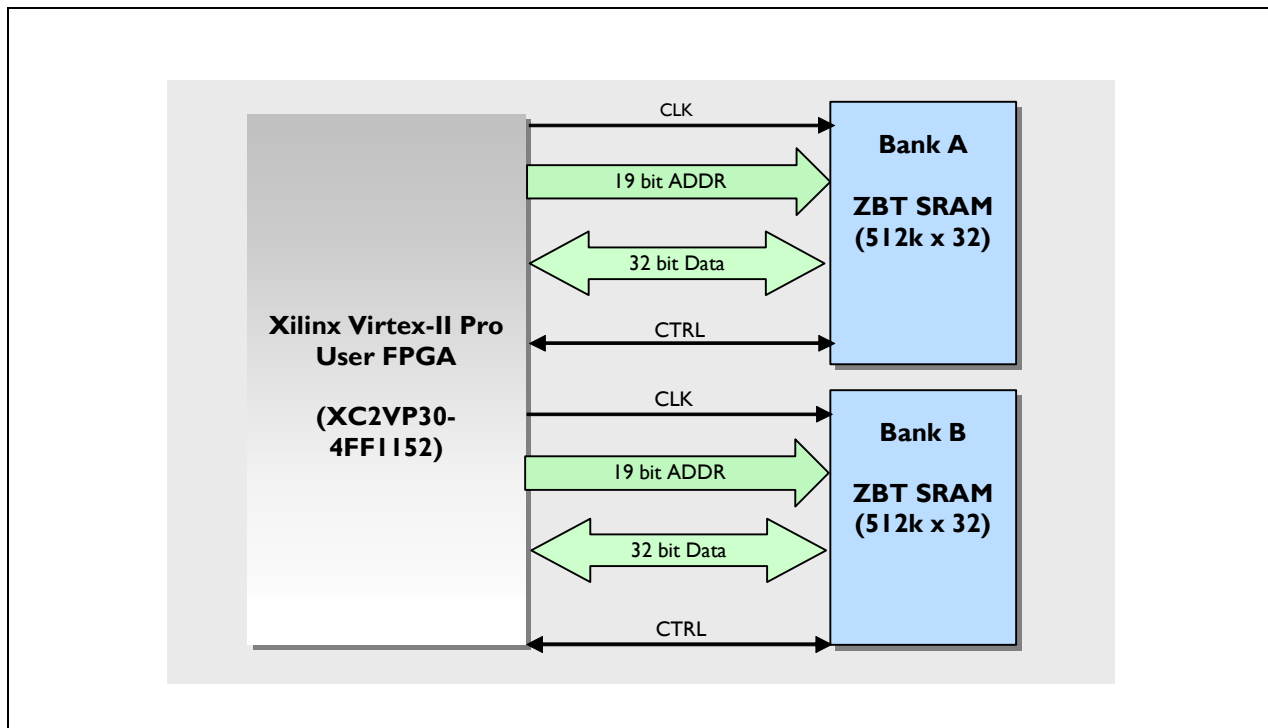


Figure 27: ZBT SRAM Interface

Each ZBT device has advanced synchronous peripheral circuitry and a 2-bit burst counter. The SRAM is optimized for 100% bus utilization, eliminating any turnaround cycles for READ to WRITE, or WRITE to READ transitions. All synchronous inputs pass through registers controlled by a positive-edge-triggered single clock input. The synchronous inputs include all addresses, all data inputs, chip enable, synchronous clock enables, write enables and Read/Write. The asynchronous inputs include the output enable, clock, and snooze enable and a burst mode that can select between interleaved and linear modes.

Clock and Control Signals for ZBT Bank A

Signal Name	User FPGA (XC2VP30-4FFI152) PIN No
ZBTA_CLK	G18
ZBTA_CLK_FB_OUT	F18
ZBTA_CLK_FB_IN	H18
ZBTA_ADV	J27
ZBTA_CKEI	F30
ZBTA_CSI<0>	J28
ZBTA_CSI<1>	E34
ZBTA_OEI	E32
ZBTA_WEI	E33

Table 12: ZBT Clock and Control Signals Pinouts Bank A

ZBT Address Signals for ZBT Bank A

Signal Name	User FPGA (XC2VP30-4FFI152) PIN No	Signal Name	User FPGA (XC2VP30-4FFI152) PIN No
ZBTA_A<0>	D29	ZBTA_A<11>	J25
ZBTA_A<1>	G25	ZBTA_A<12>	E26
ZBTA_A<2>	K23	ZBTA_A<13>	F28
ZBTA_A<3>	E28	ZBTA_A<14>	E29
ZBTA_A<4>	G22	ZBTA_A<15>	D30
ZBTA_A<5>	J23	ZBTA_A<16>	E31
ZBTA_A<6>	F26	ZBTA_A<17>	K24
ZBTA_A<7>	H22	ZBTA_A<18>	J24
ZBTA_A<8>	G26	ZBTA_A<19>	F27
ZBTA_A<9>	H26		
ZBTA_A<10>	H25		

Table 13: ZBT Address Signals Pinouts Bank A

ZBT Data Signals for ZBT Bank A

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBT_D<0>	E21
ZBT_D<1>	C22
ZBT_D<2>	E22
ZBT_D<3>	D23
ZBT_D<4>	C24
ZBT_D<5>	D24
ZBT_D<6>	D25
ZBT_D<7>	E25
ZBT_D<8>	D19
ZBT_D<9>	D20
ZBT_D<10>	C21
ZBT_D<11>	D22
ZBT_D<12>	D18
ZBT_D<13>	E18
ZBT_D<14>	E19
ZBT_D<15>	D21
ZBT_D<16>	G19
ZBT_D<17>	J19
ZBT_D<18>	F20
ZBT_D<19>	F21
ZBT_D<20>	H19
ZBT_D<21>	K18
ZBT_D<22>	K19
ZBT_D<23>	J20
ZBT_D<24>	G20
ZBT_D<25>	H21
ZBT_D<26>	F22
ZBT_D<27>	L18
ZBT_D<28>	F19
ZBT_D<29>	G21
ZBT_D<30>	K20
ZBT_D<31>	L19

Table 14: ZBT Data Signals Pinouts Bank A

Clock and Control Signals for ZBT Bank B

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTB_CLK	F17
ZBTB_CLK_FB_OUT	G17
ZBTB_CLK_FB_IN	D17
ZBTB_ADV	J7
ZBTB_CKEI	E2
ZBTB_CSI<0>	E1
ZBTB_CSI<1>	J8
ZBTB_OEI	F5
ZBTB_WEI	E3

Table 15: ZBT Clock and Control Signals Pinouts Bank B

ZBT Address Signals for Bank B

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No	Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTB_A<0>	E9	ZBTB_A<11>	E7
ZBTB_A<1>	F9	ZBTB_A<12>	F8
ZBTB_A<2>	K11	ZBTB_A<13>	H10
ZBTB_A<3>	E6	ZBTB_A<14>	H9
ZBTB_A<4>	H13	ZBTB_A<15>	E4
ZBTB_A<5>	J11	ZBTB_A<16>	F7
ZBTB_A<6>	D6	ZBTB_A<17>	G13
ZBTB_A<7>	K12	ZBTB_A<18>	J12
ZBTB_A<8>	J10	ZBTB_A<19>	D5
ZBTB_A<9>	G10		
ZBTB_A<10>	G9		

Table 16: ZBT Address Signals Pinouts Bank B

ZBT Data Signals for Bank B

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTB_D<0>	E14
ZBTB_D<1>	F15
ZBTB_D<2>	F14

Table 17: ZBT Data Signals Pinouts Bank B

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTB_D<3>	G15
ZBTB_D<4>	G14
ZBTB_D<5>	H14
ZBTB_D<6>	J15
ZBTB_D<7>	L16
ZBTB_D<8>	J17
ZBTB_D<9>	L17
ZBTB_D<10>	G16
ZBTB_D<11>	K15
ZBTB_D<12>	K17
ZBTB_D<13>	J16
ZBTB_D<14>	K16
ZBTB_D<15>	H16
ZBTB_D<16>	D16
ZBTB_D<17>	D15
ZBTB_D<18>	D14
ZBTB_D<19>	D13
ZBTB_D<20>	E17
ZBTB_D<21>	E16
ZBTB_D<22>	F16
ZBTB_D<23>	C14
ZBTB_D<24>	D12
ZBTB_D<25>	C11
ZBTB_D<26>	D11
ZBTB_D<27>	D10
ZBTB_D<28>	C13
ZBTB_D<29>	F13
ZBTB_D<30>	E13
ZBTB_D<31>	E10

Table 17: ZBT Data Signals Pinouts Bank B

6.2.1 ZBT SRAM Clocking

There is a simple setup for clocking the ZBT. A common clock is sent to both ZBT chips with a feedback net which is the same matched length as the nets going to the clock pins on the ZBT devices. This arrangement allows for de-skewing of the ZBT clock. Full details on this are provided in the Clocks feature section under **“ZBT Clocks”** on **page 71**.

6.3 Firmware

When building an interface to the ZBT you can use your own ZBT core or use an existing core provided by Nallatech. On the XtremeDSP Development Kit Pro CD an application note *NT302-0036 XtremeDSP Kit ZBT Design* is provided at the location 'CD ROM Drive\Documentation\Application Notes\'. This shows how a Nallatech specific core can be connected to the ZBT devices and also to a host interface so that data can be transferred to and from the ZBT by the host PC. Details of the specific core are given in another application note *NT302-0005 ZBT Controller* in the same folder. Please refer to both of these application notes for further details.

6.4 Software

There is no specific support for accessing the ZBT SRAM memory in the Kit from within the FUSE System Software. If you want to capture data and read it back into the host machine this functionality needs in part to be provided by the user design. Please refer to **“Part III: System Level Design”** on **page 109**, where aspects of building a host interface are discussed.

Section 7

Digital I/O

In this section:

- Introduction to the Digital I/O Headers in the Kit
 - Hardware Aspects
 - Firmware Aspects
 - Software Aspects
-

7.1 Introduction

There are several features on the Kit hardware that provide digital I/O for user designs. The following digital I/O is available:

- A 14 pin PLINK Bus Header on the motherboard. This provides 12 direct bi-directional connections to the main User FPGA with the other two pins providing GND connections.
- A 34 pin Adjacent Bus Header on the motherboard. This provides 28 direct bi-directional connections to the main User FPGA. The remainder of the pins provide a 3.3V connection, a GND connection and the remainder are 'no connects' (NC).
- A 2 pin user I/O header on the module. This provides 2 direct bi-directional connections to the main User FPGA.

Commonly the digital I/O is used for interfacing to other hardware or for the purposes of debug though the use of logic analyzers. The naming conventions of the headers may initially seem confusing but the naming conventions are carried through from those used in the DIME and DIME-II architectures. Despite the header names it should be noted that these headers are simply bi-directional user I/O that connect directly to the pins of the main User FPGA.



To see where these headers appear on the Kit please refer to **“XtremeDSP Development Kit Pro Hardware Physical Layout”** on page 17.

7.2 Hardware

7.2.1 User I/O Header - Interfacing

The BenADDA DIME-II module used in the Kit has a two-pin 0.1" pitch header that connects directly to the User FPGA. There is no assigned function for this two-pin header, which is therefore free to be used for your desired application.



The two pin header is connected directly to the Virtex-II Pro User FPGA and therefore signals applied to this MUST be within the range of 0V to +3.3V. Virtex-II Pro devices are NOT 5V tolerant.

Table 18 on page 52 contains the pinout information for the User I/O header.

Pin Name	User FPGA (XC2VP30-4FF1152) PIN No
User_IO_1	N29
User_IO_2	N30

Table 18: Pinouts of User I/O Header (J16)

7.2.2 P-Link Bus Header (J10)

This header is connected to P-Link0 on the DIME-II module slot.

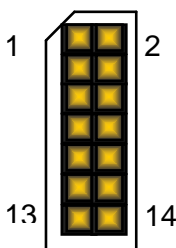
Header Pin Number	Name	User FPGA (XC2VP30-4FF1152) PIN No	
1	PP0LK<0>	U7	
2	PP0LK<1>	U8	
3	PP0LK<2>	U5	
4	PP0LK<3>	U6	
5	PP0LK<4>	U3	
6	PP0LK<5>	U4	
7	PP0LK<6>	V4	
8	PP0LK<7>	V3	
9	PP0LK<8>	V2	
10	PP0LK<9>	U2	
11	PP0LK<10>	V8	
12	PP0LK<11>	V7	
13	GND	N/A	
14	GND	N/A	

Table 19: Pinouts of P-Link Bus Header (J10)

7.2.3 Adjacent Bus Header (J8)

This header is a 28-bit general-purpose bus and is connected to the Adjacent IN Bus on the DIME-II module slot which in turn is connected to I/O on the main User FPGA.

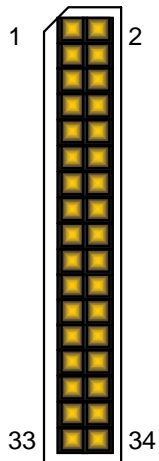
Header Pin Number	Name	User FPGA (2V3000FG676) PIN No	
1	ADJIN<12>	W6	
2	ADJIN<13>	W5	
3	ADJIN<10>	W8	
4	ADJIN<11>	W7	
5	ADJIN<8>	Y2	
6	ADJIN<9>	W2	
7	ADJIN<6>	V10	
8	ADJIN<7>	V9	
9	ADJIN<4>	AA1	
10	ADJIN<5>	Y1	
11	ADJIN<2>	W11	
12	ADJIN<3>	V11	
13	ADJIN<0>	AL17	
14	ADJIN<1>	AK17	
15	ADJIN<14>	W4	
16	ADJIN<15>	W3	
17	ADJIN<16>	Y4	
18	ADJIN<17>	Y3	
19	ADJIN<18>	W10	
20	ADJIN<19>	W9	
21	ADJIN<20>	Y10	
22	ADJIN<21>	Y9	
23	ADJIN<22>	Y7	
24	ADJIN<23>	Y6	
25	ADJIN<24>	AA6	
26	ADJIN<25>	AA5	
27	ADJIN<26>	AA4	
28	ADJIN<27>	AA3	
29	3.3V	n/c	
30	GND	n/c	
31	n/c	n/c	
32	n/c	n/c	
33	n/c	n/c	
34	n/c	n/c	

Table 20: Adjacent Bus Digital I/O Header



The 'StrathLED' is an optional module from Nallatech that can be plugged into this header to provide an array of LEDs for display purposes. For more information please contact support@nallatech.com

7.3 Firmware

The Digital I/O in your design is configured according to the specific I/O of the connected FPGA. The *Virtex-II Pro Datasheet* and *handbook* from Xilinx provide extensive information on I/O standards that can be supported by pins on the Virtex-II Pro device.

7.4 Software

There is no specific support for controlling the Digital I/O available in the Kit as there are simply connections to I/O on the main User FPGA.

Section 8

LEDs

In this section:

- Introduction to the LEDs in the Kit
 - Hardware Aspects
 - Firmware Aspects
 - Software Aspects
-

8.1 Introduction

The XtremeDSP Development Kit Pro contains a number of user-definable and status LEDs which allow you to check the operation and status of the Kit. There are LEDs provided for user designs (User LEDs) and for displaying aspects of system status such as interface configuration and power (Interface LEDs). The LEDs used in the Kit are tricolor (each LED displays a total of three different colors), meaning each LED can act as three individual LEDs. Each LED has a RED and GREEN diode inside their chip allowing for the display of RED, GREEN and YELLOW/ORANGE colors.

8.2 Hardware

8.2.1 Physical Location of LEDs

Figure 28 on page 56 shows the physical location of the LEDs on the board case lid.

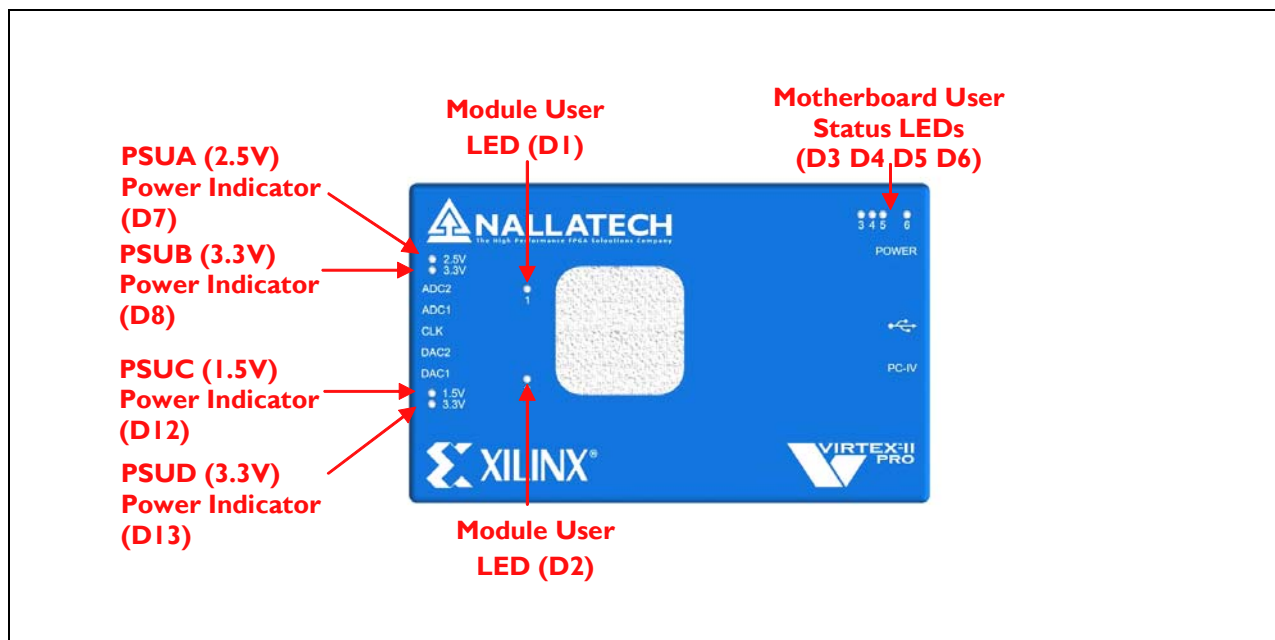


Figure 28: Board Case Lid showing LED Locations

Figure 29 on page 56 and Figure 30 on page 57 show the physical location of the Kit LEDs described in this section.

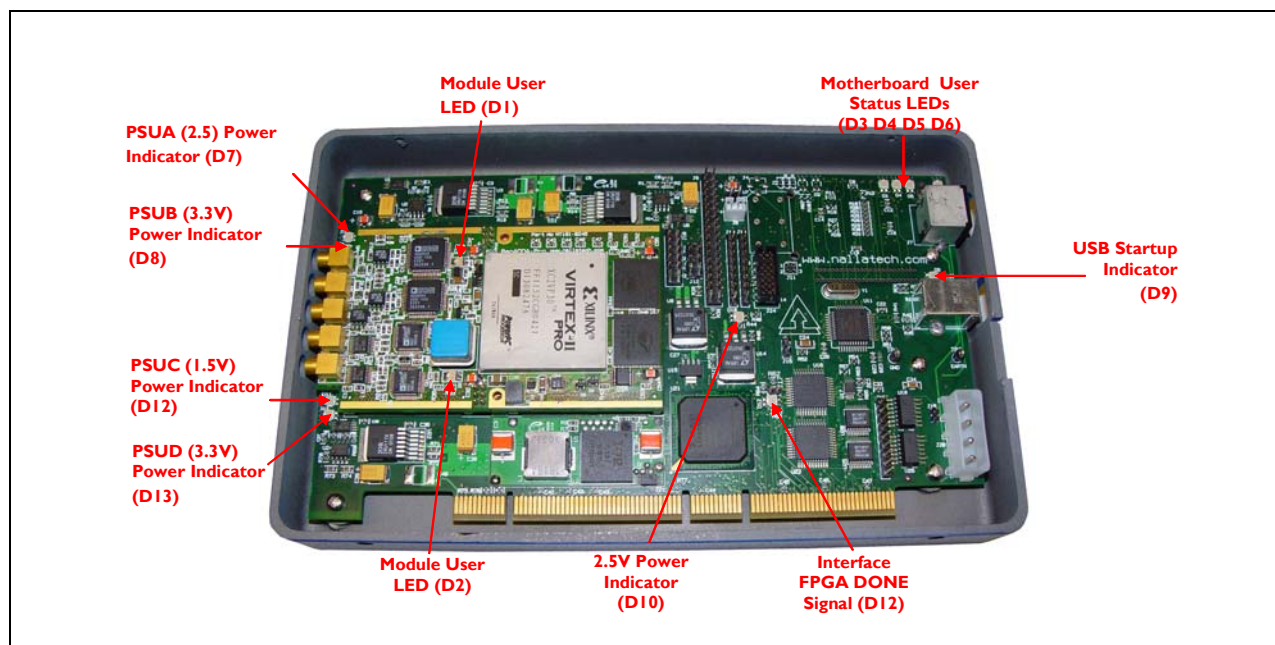


Figure 29: Front View of Physical LED Locations

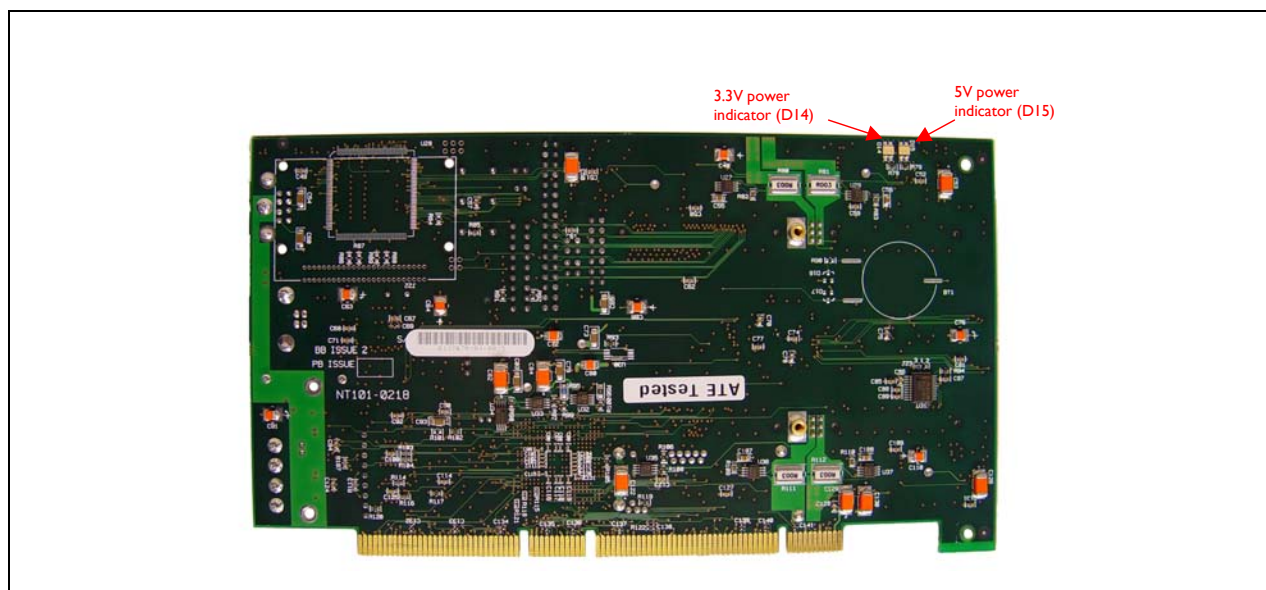


Figure 30: Back View of Physical LED Locations

8.2.2 Interface LEDs

The interface LEDs provide information on aspects of the Kit such as power good signals from the power supplies that provide the required voltages to the BenADDA DIME-II module. There is also an LED to show the output of the DONE signal for the interface FPGA to check it has successfully started up.

Interface FPGA Configuration LED

LED D12 illuminates when the interface FPGA is fully configured. If this LED is not lit shortly after power has been applied this means that the interface FPGA has failed to boot from the Xilinx XC18V02 PROMs used on the Kit hardware.

USB Physical Interface LED

The LED D9, provides information on the actual USB interface chip. This LED will be green if the device itself has started up. It should always be green when the Kit is powered up.

DIME-II Module Power Supply LEDs

The DIME-II standard requires that a number of voltages be supplied to a DIME-II module from the DIME-II motherboard it is fitted to. In the case of the Kit, four power supplies provide specific power to the module. For each of these power supplies an LED shows the power good signal when each power supply is turned on. These power supplies are termed PSUA to PSUD. The actual voltage generated for each is given in [Table 21 on page 57](#).

Silk Screen LED Identifier	Color State	State Description	State Voltage Output	Initial power on state	After card opened in FUSE
D7	Red	PSUA Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUA On	2.5V		
D8	Red	PSUB Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUB On	3.3V		

Table 21: DIME-II Module Power Supply LED States

Silk Screen LED Identifier	Color State	State Description	State Voltage Output	Initial power on state	After card opened in FUSE
D12	Red	PSUC Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUC On	1.5V		
D13	Red	PSUD Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUD On	3.3V		

Table 21: DIME-II Module Power Supply LED States

Motherboard Main Power LEDs

In addition to the power supplies for the module the Kit contains LEDs which indicate the status of the main power supplies for the motherboard itself. [Table 22 on page 58](#) defines their use.

LED	Purpose	General Operation State
D10	2.5V power indicator	GREEN
D14	3.3V power indicator	GREEN
D15	5V power indicator	GREEN

Table 22: Motherboard Main Power LEDs

8.2.3 User LEDs

Module User LEDs

The BenADDA DIME-II module has two user tricolor LEDs, which can be used for specific design purposes.

Signal Description	User LED	Signal Name	Main User FPGA (XC2VP30-4FF1152) Pin
Green Diode for LED2	D2	LED_Green2	AL29
Red Diode for LED2	D2	LED_Red2	AL30
Green Diode for LED1	D1	LED_Green1	AK6
Red Diode for LED1	D1	LED_Red1	AK7

Table 23: Module User LEDs

Motherboard User Status LEDs

The BenONE-Kit Motherboard has four software controllable LEDs. These LEDs are connected to the Interface FPGA and can only be controlled via software calls to the card and NOT via signals from the main User FPGA. Please see [“Software” on page 59](#) for details of the relevant software calls.

8.3 Firmware

The LEDs are illuminated when the corresponding pin is logic 0, otherwise the LED is not illuminated (Active LOW). Applying logic 0 to either Cathode will cause that color to illuminate and applying logic 0 to both Cathodes (green and red) will cause a third color, yellow, to illuminate.

8.4 Software

8.4.1 FUSE Software LED Function

The four PCI LEDs can be controlled from software using the FUSE API or the FUSE Probe Tool, as shown in [Figure 31 on page 59](#). In order to toggle each LED choose the LED tab (circled in red) and check the corresponding LED box. Please note that you must open and select a card in the FUSE Probe Tool before controlling the LEDs.

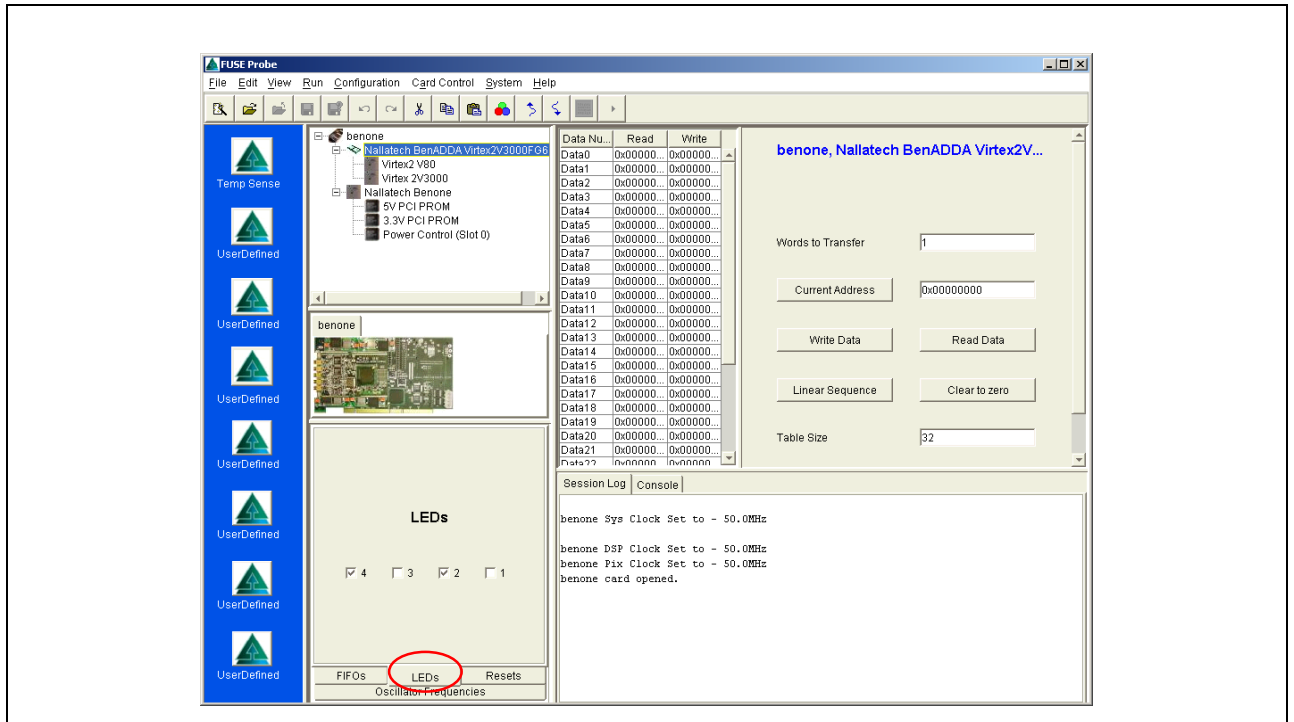


Figure 31: Reset using FUSE Probe Tool

For more information on using the FUSE Probe Tool please see the *FUSE System Software User Guide* on the supplied FUSE CD.

8.4.2 Listing of Calls in the FUSE API for LED Control

The FUSE C/C++ API provides two functions for controlling the LEDs in the Kit. These are:

- **DIME_ReadLEDs**
- **DIME_WriteLEDs**

The FUSE APIs require a data word to be written, with the bits of that word controlling the LEDs. The LEDs are illuminated when a '0' is written to the location. So for example when the four least significant bits of the data word are "1001", LEDs D3 and D4 will illuminate. The bits of the data-word that control the LED mapping are given in [Table 24 on page 60](#).

LED	Dataword Bit Number
D2	0
D3	1
D4	2
D5	3

Table 24: PCI Status LEDs

An example of how to control the LEDs using the FUSE C/C++ FUSE API is shown in [Figure 32 on page 60](#).

```
#include <dimesdl.h> //This is held in the include directory
                        within FUSE.

DIME_HANDLE hCard1;
LOCATE_HANDLE hLocate;
DWORD LEDs;
//Locate the Cards on the PCI interface
hLocate=DIME_LocateCard(d1PCI,mbtALL,NULL,dldrDEFAULT,d1DEFAULT);

//Open the first card found in the locate.
hCard1=DIME_OpenCard(hLocate,1,dccOPEN_DEFAULT);

//Change the LEDs
LEDs=DIME_ReadLEDs(hCard1);
DIME_WriteLEDs(hCard1,(LEDs-1));

//Close the card down.
DIME_CloseCard(hCard1);

//Finally close the locate down.
DIME_CloseLocate(hLocate);
```

Figure 32: FUSE API Commands for PCI LEDs

The *FUSE C/C++ API Developers Guide*, included on the FUSE CD, provides further details on both these functions.

Section 9

Resets

In this section:

- Introduction to the Reset Structure in the Kit
- Hardware Aspects
- Firmware Aspects
- Software Aspects

9.1 Introduction

There are reset signals provided from the interface FPGA (XC2S200) to the user programmable FPGAs (XC2VP30-4FF1152 and XC2V80-4CS144) in the XtremeDSP Development Kit Pro. This section details the structure of these resets and how they can be used in your designs.

9.2 Hardware

9.2.1 Reset Structure

The configuration of the resets on the Kit is shown in [Figure 33 on page 61](#).

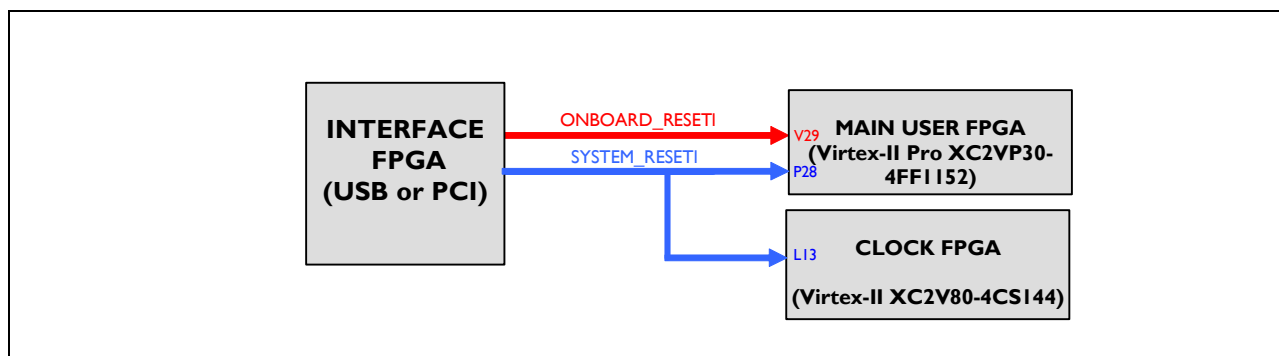


Figure 33: Reset Distribution

The Virtex-II Pro FPGA resets are controlled from the Interface FPGA, via FUSE software. Both of these resets are active-low. One or both reset signals can be used within an FPGA design.



Please note that reset signals are described here as the ONBOARD_RESETI and the SYSTEM_RESETI. The 'I' suffix indicates that these are active low signals.



Also note that the naming here has been changed slightly from that used in the DIME-II standard. This is for clarity when using the Kit.

- 'RSTI' has been renamed to 'ONBOARD_RESETI'.
- 'RESETI' has been renamed to 'SYSTEM_RESETI'.

9.3 Firmware

When designing Firmware, if you are using DCM in the Virtex-II Pro in your design, it is recommended that you use an inverted version of either the ONBOARD_RESETI or SYSTEM_RESETI to reset the DCM. Use the locked signals from DCMs to generate the reset to the rest of your design. [Figure 34 on page 62](#) provides an example of this.

```

GND <= '0';

RESET <= not RESETI;

-----clock deskew section-----
-- IBUFG Instantiation for CLK_IN
U0_IBUFG : IBUFG
  port map (
    I => CLKI_FB,
    O => CLKIN_OSC
  );
-- BUFG Instantiation for CLKFB
U0_BUFG : BUFG
  port map (
    I => CLKFB_OSC,
    O => CLK_OSC
  );
-- DCM Instantiation for internal deskew of CLK0
U0_DCM : DCM
  port map (
    CLKIN    => CLKIN_OSC,
    CLKFB    => CLK_OSC,
    DSSSEN   => GND,
    PSINCDEC => GND,
    PSEN     => GND,
    PSCLK    => GND,
    RST      => RESET,
    CLK0     => CLKFB_OSC,
    LOCKED   => RSTI
  );
-----end of clock deskew-----

```

Figure 34: Example of Reset and DCM Setup

This code snippet is taken from the 'adc_to_dac_hookup' example included on the XtremeDSP Development Kit Pro CD at the location 'CD ROM Drive\Examples\adc_to_dac_hookup'

9.4 Software

9.4.1 FUSE Software Reset Function

The Kit resets can be asserted using the FUSE Probe Tool, as shown in [Figure 35 on page 63](#). Use the check boxes for either the FPGA or SYSTEM reset assets (i.e. drives a LOW '0'). Uncheck them to leave them de-asserted (HIGH '1'). The INTERFACE RESET button simply toggles the interface FPGA to clear the data FIFOs as previously discussed. Please note that you must open and select a card in the FUSE Probe Tool prior to controlling the resets.

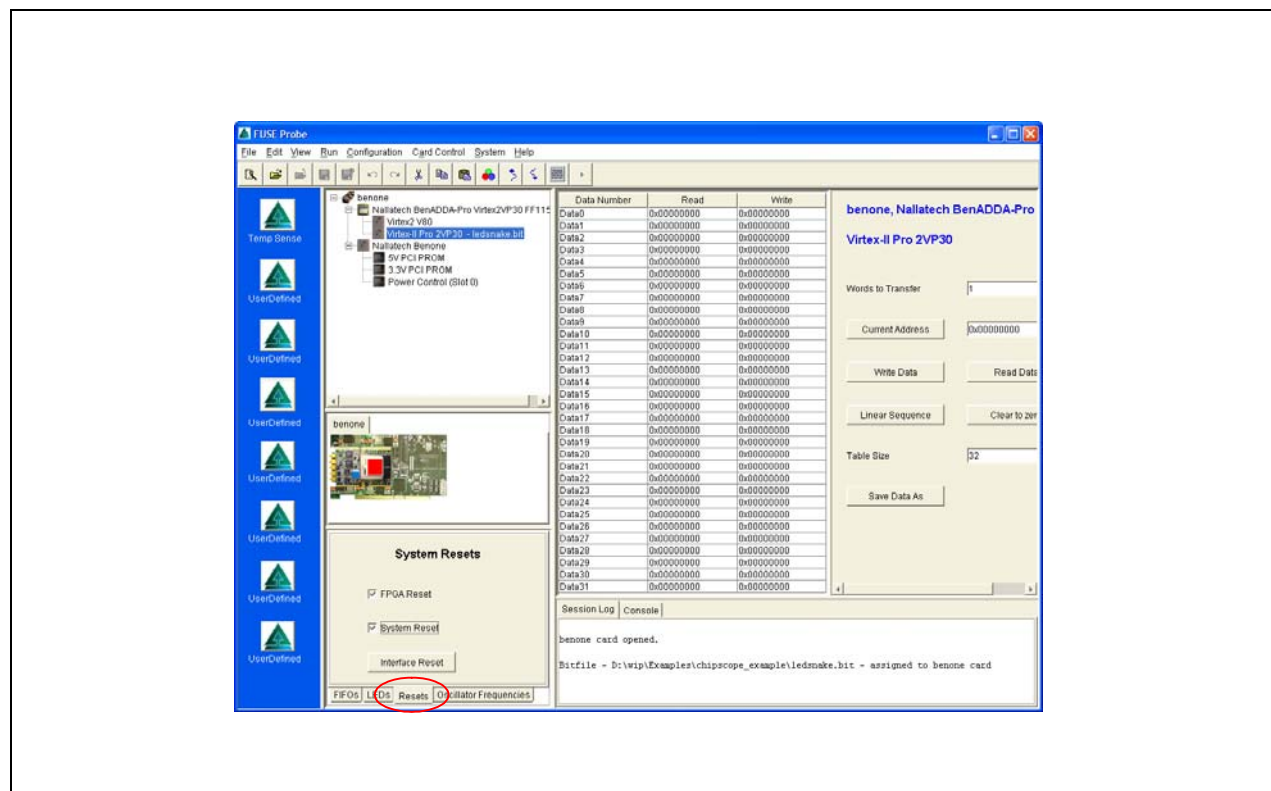


Figure 35: Reset using FUSE Probe Tool



In the FUSE Probe Tool the 'ONBOARD RESET' is referred to as 'FPGA RESET'.

For more information on resets and using the FUSE Probe Tool please see the *FUSE System Software User Guide* on the supplied FUSE CD.

9.4.2 Listing of Calls in the FUSE API for Reset Control

Both the XtremeDSP Development Kit Pro resets are controlled via the following functions:

- **DIME_CardResetControl**
- **DIME_CardResetStatus**

When a design has been downloaded into the FPGAs and the oscillators have been set to the desired frequency it is good practice to toggle all the resets within the design. Essentially, both functions consist of two parameters - a 'resetNum' and a 'cmdMode'.

- resetNum is used to select between the different resets on the card.
- cmdMode is effectively a command for the reset i.e. to enable(assert - LOW) or disable(deassert - HIGH).

Figure 36 on page 64 shows an example of the how to control the clocks via the C/C++ FUSE API.

```
//Enable the OnBoardFPGA reset.
DIME_CardResetControl(handle,drONBOARDFPGA,drENABLE,0);
//Disable the OnBoardFPGA reset.
DIME_CardResetControl(handle,drONBOARDFPGA,drDISABLE,0);
//Toggle the OnBoardFPGA reset.
DIME_CardResetControl(handle,drONBOARDFPGA,drTOGGLE,0);

// Enable the System reset.
DIME_CardResetControl(handle,drSYSTEM, drENABLE,0);
// Disable the System reset.
DIME_CardResetControl(handle,drSYSTEM, drDISABLE,0);
// Toggle the System reset.
DIME_CardResetControl(handle,drSYSTEM,drTOGGLE,0);

// Toggle the interface FPGA reset.
DIME_CardResetControl(handle,drINTERFACE,drToggle,0);
```

Figure 36: Examples of Using DIME_CardResetControl

For further details on resets please refer to the *FUSE C-C++ API Developers Guide* on the supplied FUSE CD.

Section 10

Clocks

In this section:

- Introduction to the Clocking Structure
 - Hardware Aspects
 - Firmware Aspects
 - Software Aspects
-

10.1 Introduction

The XtremeDSP Development Kit Pro has a comprehensive and flexible clock management system. The features available are as follows:

- A 105MHz crystal source on the module primarily to provide a low jitter clock source for the analog devices.
- An external clock input via one of the MCX connectors.
- Two software programmable clock sources on the motherboard which can be set to a number of frequencies. These provide two general purpose system clocks for user designs.
- Single fixed oscillator socket on the motherboard. Please note that no oscillator is provided and this socket allows for use of specific oscillators in user applications.



The clock nets on the Kit have been designed to eliminate clock skew at the FPGA destinations. This is done by using clock nets of the same length between the on board clock drivers and the DIME-II module slot.

DIME-II System Clocks

The BenADDA DIME-II module can make use of three system clocks fed from the motherboard to the User FPGA. These are called CLKA, CLKB and CLKC. The clock signals are generated on the DIME-II motherboard and routed into the module site where the BenADDA DIME-II module is placed. These clocks can be controlled by the user and are routed to Global Clock pins to provide maximum flexibility on the User FPGA. However, note that the functionality of these DIME-II clocks is determined by the motherboard. When the BenADDA DIME-II module is fitted to the BenONE-Kit Motherboard, as in the XtremeDSP Development Kit Pro configuration, the available DIME-II clocks are:

- CLKA - available programmable oscillator on the BenONE-Kit Motherboard
- CLKB - available programmable oscillator on the BenONE-Kit Motherboard
- CLKC - connected to a socket to support a crystal oscillator. Please note that no oscillator is supplied and this option on the BenONE-Kit Motherboard allows users to fit a specific crystal.

Full details of the generation of these clock signals is shown in “Clocking Configuration” on page 67. For details on how the clocks are named see “FUSE Naming Conventions” on page xvi.

10.2 Hardware

10.2.1 Physical Clock Resources

Figure 37 on page 66 shows where all the devices and inputs related to clock sources are on the actual hardware.

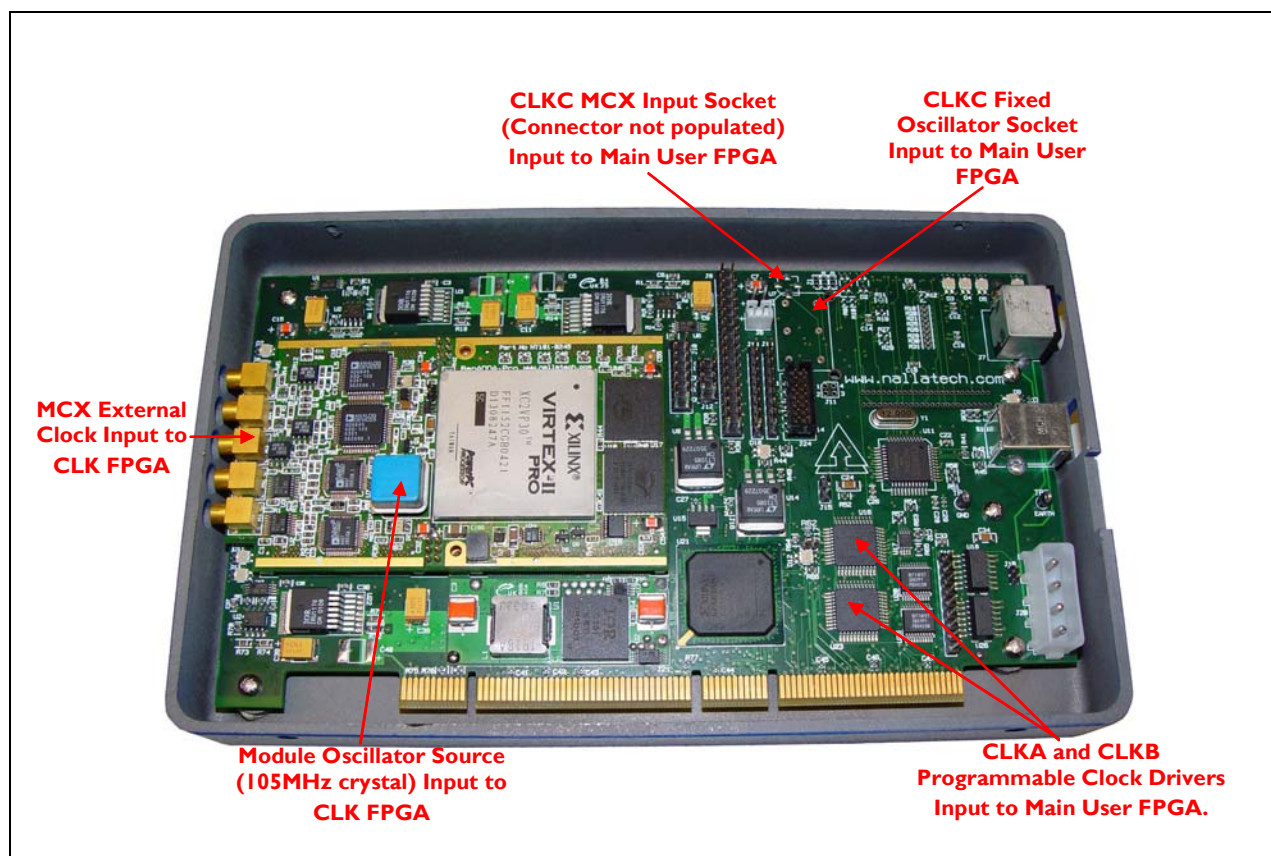


Figure 37: Physical Location of Clock Related Hardware

10.2.2 Clocking Configuration

Figure 38 on page 67 provides an overview of the XtremeDSP Development Kit Pro clock structure.

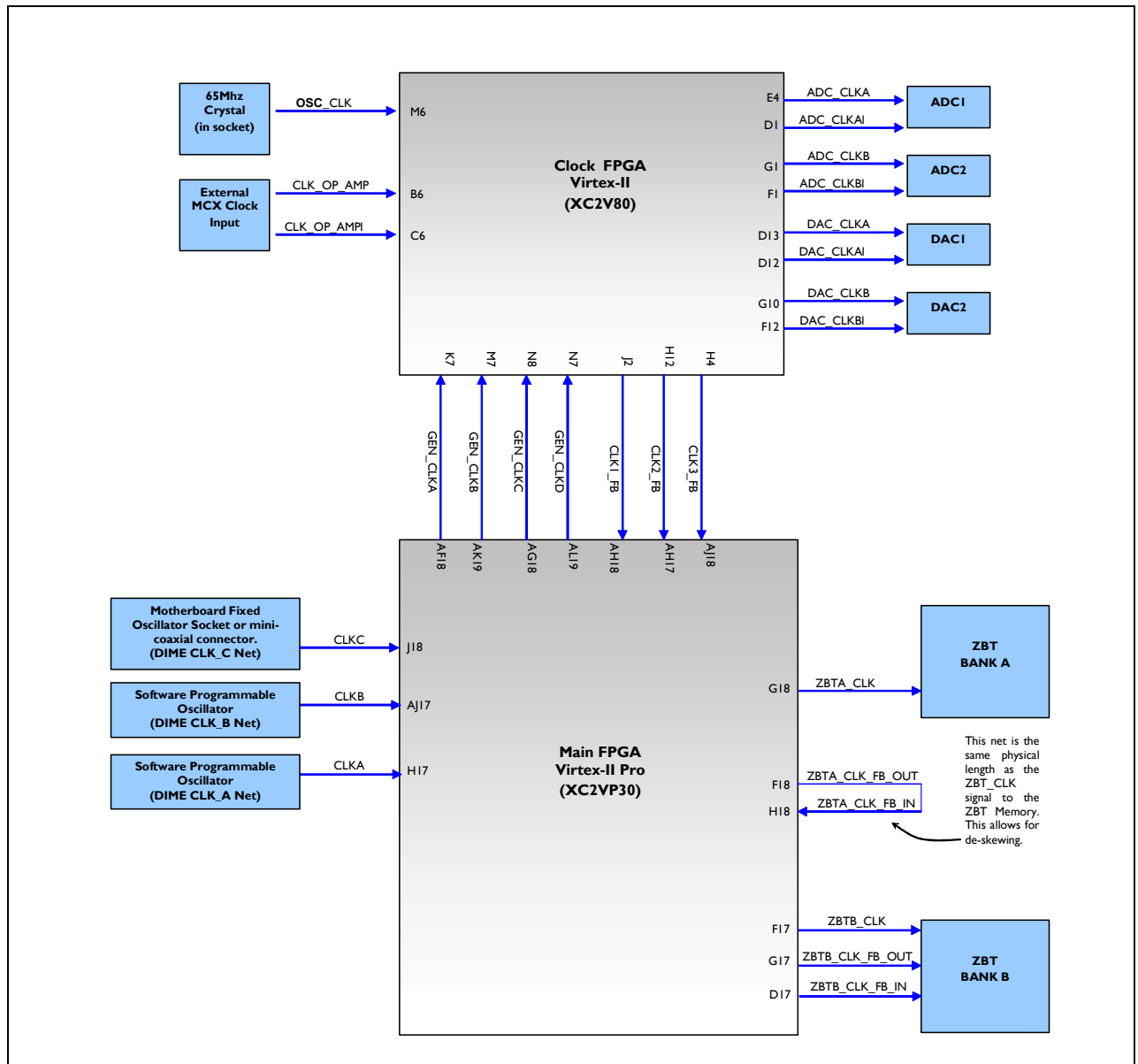


Figure 38: Clock Structure

10.2.3 Source Descriptions

Programmable Oscillators

The Programmable Oscillators are controlled via FUSE Software, through any of the available interfaces/APIs. The available operating frequencies of the programmable oscillators are as follows:

20 MHz; 25 MHz; 30 MHz; 33.33 MHz; 40 MHz; 45 MHz; 50 MHz; 60 MHz; 66.66 MHz; 70 MHz; 75 MHz; 80 MHz; 90 MHz; 100 MHz; 120 MHz.

When a frequency is requested using FUSE, which does not exactly match one of the fifteen frequencies supported by the oscillators, the firmware looks at the available frequencies and selects the one that is numerically closest to the frequency requested.

Additionally, for interfacing with the Interface FPGA (XC2S200) or where the Nallatech Interface FPGA to User FPGA Interfacing core is used, this interfacing should be clocked by the Clock B, which should be set within the range 35MHz-40MHz. One programmable oscillator drives CLKA net and the other drives CLKB net into the main User FPGA. See “Part III: System Level Design” on page 109 for more details.

Motherboard Fixed/External Oscillator

A Fixed Oscillator can be fitted to the Kit hardware, to provide a clock source which matches the user's frequency/jitter specification, for more specialized applications. The socket accepts 14-pin or 8-pin type oscillator packages, which can be plugged into the socket. The configuration of the slot is shown in Figure 39 on page 68. Alternatively, a mini-coaxial connector can be populated on the Kit hardware instead of using the fixed oscillator so that an external clock source can be brought onto the card.

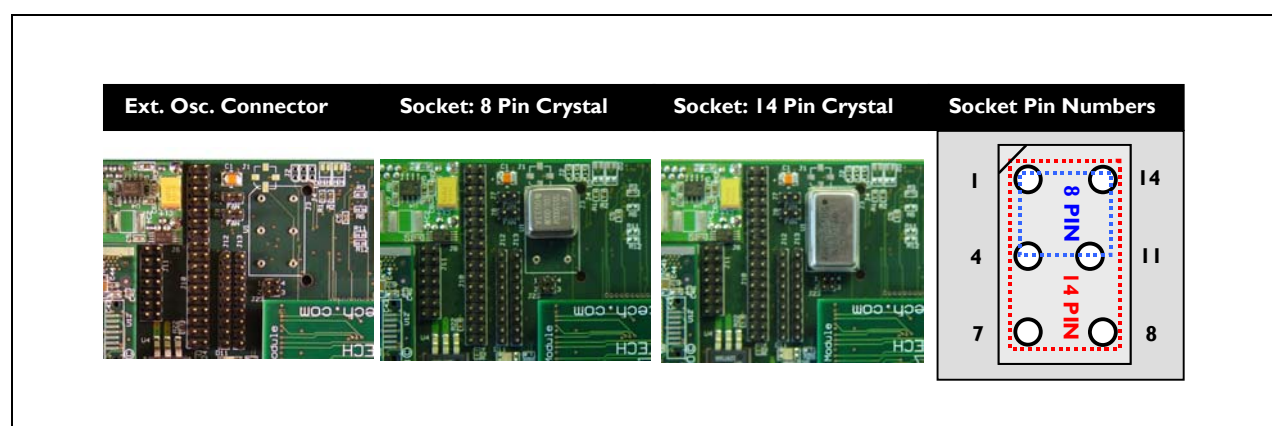


Figure 39: Oscillator Socket Configuration



The clock nets on the Kit allow either a fixed frequency oscillator to be used or an external clock, as the external clock connector is located under the fixed frequency oscillator socket.

External Clock Input via Module MCX Connection

External clock sources can be brought into the Kit for on board use. The external clock input (shown in Figure 40 on page 69) is only initially connected to the Clock. In order to meet the signal input specifications for the PGA_FPGA, an op_amp is used to provide DC biasing to level shift the input signal above 0V. The MCX clock input is setup as a 50R single ended input.

At the heart of the external clock circuit is the AD8131 Differential Driver. This converts single-ended inputs into differential outputs suitable for the Clock FPGA. The AD8131 has internal feedback with a fixed gain of 2, which allows for better thermal matching and tolerance levels. The common-mode level of the differential output is set by VO_{CM}, thereby level shifting the input signal suitable for driving the Clock FPGA.

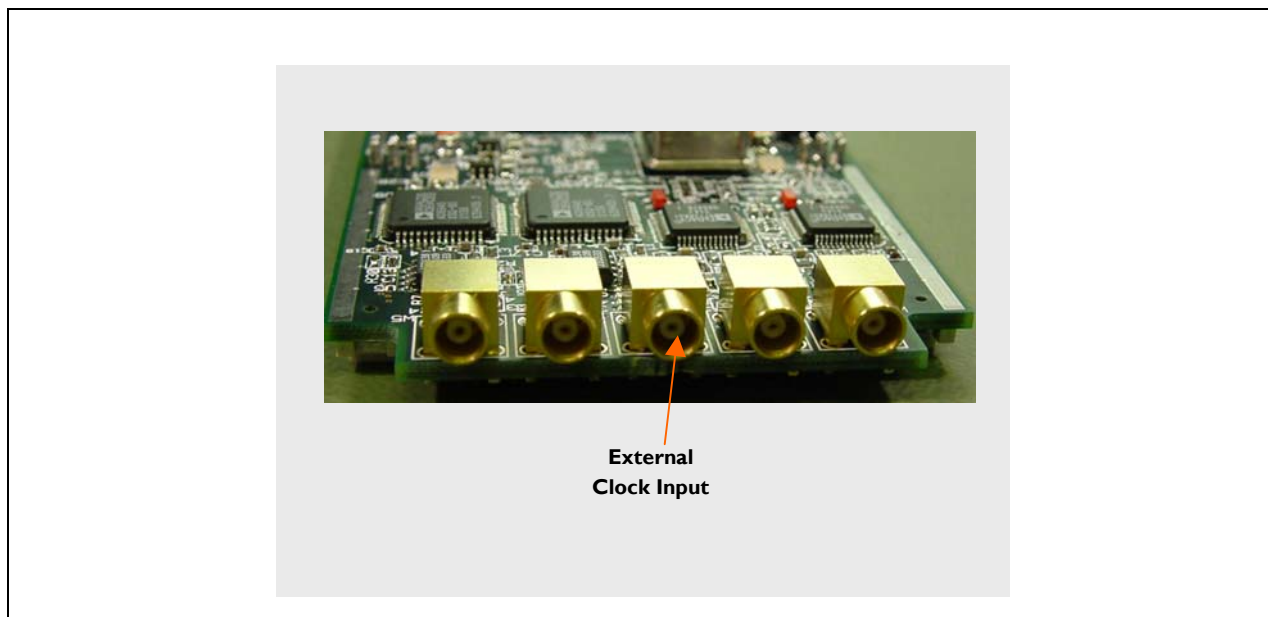


Figure 40: External MCX Clock Connector Input

VOCM is set at 1.25V to comply with the typical VICM value for the Virtex-II FPGA using LVDS voltage specifications. The Differential Driver Output Voltage for LVDS is specified as ± 250 -450mV (typ. ± 350 mV). It is therefore advised to limit the magnitude of the clock input signal at the MCX input to an unbiased sinewave input of 125mVpp-225mVpp. The maximum input signal the Clock_FPGA can accommodate is 3.3Vpp. In order to ensure that the Clock FPGA cannot be accidentally blown, the supply of the AD8131 is limited between 0V to 3.3V.

Clock Signal Description	Signal Name	CLOCK FPGA Pin
External Clock source via Op_Amp	CLK_Op_Amp	B6 (GCLK6S)
Complement of External Clock source via Op_Amp	CLK_Op_Ampl	C6 (GCLK7P)

Table 25: Pinout information for Module MCX External Clock Input

Module On board 105MHz Oscillator

An on board crystal oscillator that generates an LVTTTL clock signal is supplied with the standard build of the BenADDA DIME-II module. The LVTTTL clock signal generated by this oscillator is driven directly into the Clock FPGA. This clock signal can then be used to derive the differential clock signals used to clock both the DACs and the ADCs. The crystal oscillator supplied with the BenADDA DIME-II module has a low jitter characteristic and its speed will be matched to the sampling frequency of the ADCs. For the ADC AD6645, a 105MHz crystal oscillator is supplied.

Clock Signal Description	Signal Name	CLOCK FPGA Pin
LVTTTL Clock Oscillator	Osc_CLK	M6 (GCLK4P)

Table 26: On board Crystal Oscillator Pinout



Although the BenADDA DIME-II module is supplied with a crystal oscillator that complements the speed of the ADCs, this part can easily be replaced with an alternative exhibiting a lower speed rating. The oscillator is fixed onto the BenADDA via sockets and can simply be lifted out. Any Oscillator with similar characteristics can be brought and placed into the socket pins. The Oscillator supplied with the

BenADDA is an 8-pin DIL package from Pletronics. This oscillator is powered from 3.3V and any replacement oscillator should be the same specification. Please contact support@nallatech.com for advice on replacing the standard oscillator.

10.2.4 Inter-FPGA Clock Management

Overview

There are a number of clock nets between the main User FPGA (XC2VP30-4FF1152) and the Clock FPGA (XC2V80-4CS144). These clock nets allow for a flexible routing of clock signals between the two FPGAs to support the range of clocking structures and clock sources. They also provide facilities to deskew the clock nets passed in between the two FPGAs. There are signals for passing generated clock signals from the main FPGA to the Clock FPGA and also feedback signals from the Clock FPGA to the main FPGA.

Generated Clock Signals from User FPGA

Another method of clocking the DACs and ADCs is to use clock signals generated by the User FPGA. Within the User FPGA there are three DIME-II system clocks: CLKA, CLKB and CLKC (see “DIME-II System Clocks” on page 65 for more information. Please note that in the Kit only CLKA and CLKB clock sources are programmable and CLKC is connected to a socket for a crystal oscillator. These system clocks can be used to derive an appropriate clock frequency within the User FPGA and then driven into the Clock FPGA where they can be forwarded out to the appropriate DACs and/or ADCs.

These generated clock signals are forwarded from the User FPGA to the Clock FPGA as four single-ended signals. From the Clock FPGA, the forwarded clock signals can then be sent out to the DACs/ADCs as differential signals. [Table 27 on page 70](#) shows the generated clock signals.

Clock Signal Description	Signal Name	Clock FPGA Pin (XC2V80-4CS144)	User FPGA Pin (XC2VP30-4FF1152)
Generated Clock A	GEN_CLKA	K7 (GCLK0P)	AF18
Generated Clock C	GEN_CLKC	N8 (GCLK1S)	AK19
Generated Clock B	GEN_CLKB	M7 (GCLK6P)	AG18
Generated Clock D	GEN_CLKD	N7 (GCLK7S)	AL19

Table 27: Generated Clock Pinouts

Clock Feedbacks for De-skewing and Clock Routing

Feedback signals between the Clock FPGA and the User FPGA are necessary to allow all data going to and from the User FPGA to be clocked on the same clock edge as the data in the DACs and ADCs. There are a total of three feedback pins from the Clock FPGA to the User FPGA. These feedback signals ensure that the clock to the DACs or ADCs and the feedback pins have coincident clock edges with minimum skew.

The feedback signals from the Clock FPGA to the User FPGA are matched in physical length with the clock signals sent to the DACs and ADCs. This design ensures minimum skew between the data clocked through the ADCs/DACs and the data clocked in the User FPGA. [Table 28 on page 70](#) outlines the set-up between the Clock FPGA and User FPGA for these feedback signals. There are only three clock feedback pins due to clock pin resource constraints.

Clock Signal Description	Signal Name	Clock FPGA Pin (XC2V80-4CS144)	User FPGA Pin (XC2VP30-4FF1152)
Clock Feedback 1	CLK1_FB	J2	AH18

Table 28: Feedback Clock Pinouts

Clock Signal Description	Signal Name	Clock FPGA Pin (XC2V80-4CS144)	User FPGA Pin (XC2VP30-4FF1152)
Clock Feedback 2	CLK3_FB	H4	AH17
Clock Feedback 3	CLK2_FB	H12	AJ18

Table 28: Feedback Clock Pinouts

10.2.5 ADC and DAC Clocks

The Clock FPGA is used to directly clock each ADC and DAC device independently. The ADCs and DACs are clocked differentially from the Clock FPGA and can be clocked at various speeds. The speed at which the ADCs and DACs are clocked depends how the clocks are used which in turn depends upon the setup of the main FPGA and the clock FPGA designs.

Signal Name	Clock FPGA (XC2V80-4CS144) Pin No
ADC_CLKA	E4
ADC_CLKAI	D1
ADC_CLKB	G1
ADC_CLKBI	F1
DAC_CLKA	D13
DAC_CLKAI	D12
DAC_CLKB	G10
DAC_CLKBI	F12

Table 29: Clocking Pinouts for DACs and ADCs

10.2.6 ZBT Clocks

The two ZBT banks in the Kit are clocked by independent clocks with the addition of feedback clock signals. Each bank has a signal that can be de-skewed within the User FPGA which ensures that the clock at the ZBT SRAM Banks and the feedback pin have coincident clock edges with minimum skew. This process ensures the internal logic is clocked in phase with the data entering the ZBT chips.

Driving the ZBT SRAM clock from the FPGA ensures maximum flexibility in the clocking mechanism during system design, as it can be derived from any of the clock sources in the system. The pinouts for the various clock signals associated with the ZBTs are shown in [Table 30 on page 71](#).

Signal Name	User FPGA (2V3000FG676) PIN No
ZBTA_CLK	G18
ZBTA_CLK_FB_OUT	F18
ZBTA_CLK_FB_IN	H18
ZBTB_CLK	F17
ZBTB_CLK_FB_OUT	G17
ZBTB_CLK_FB_IN	D17

Table 30: ZBT Clock Pinouts

ZBT SRAM Clocking Example

An example of a typical clock arrangement for driving the ZBT Bank A with the input clock at the FPGA (e.g. CLKA, CLKB or CLKC) is illustrated in [Figure 41 on page 72](#).

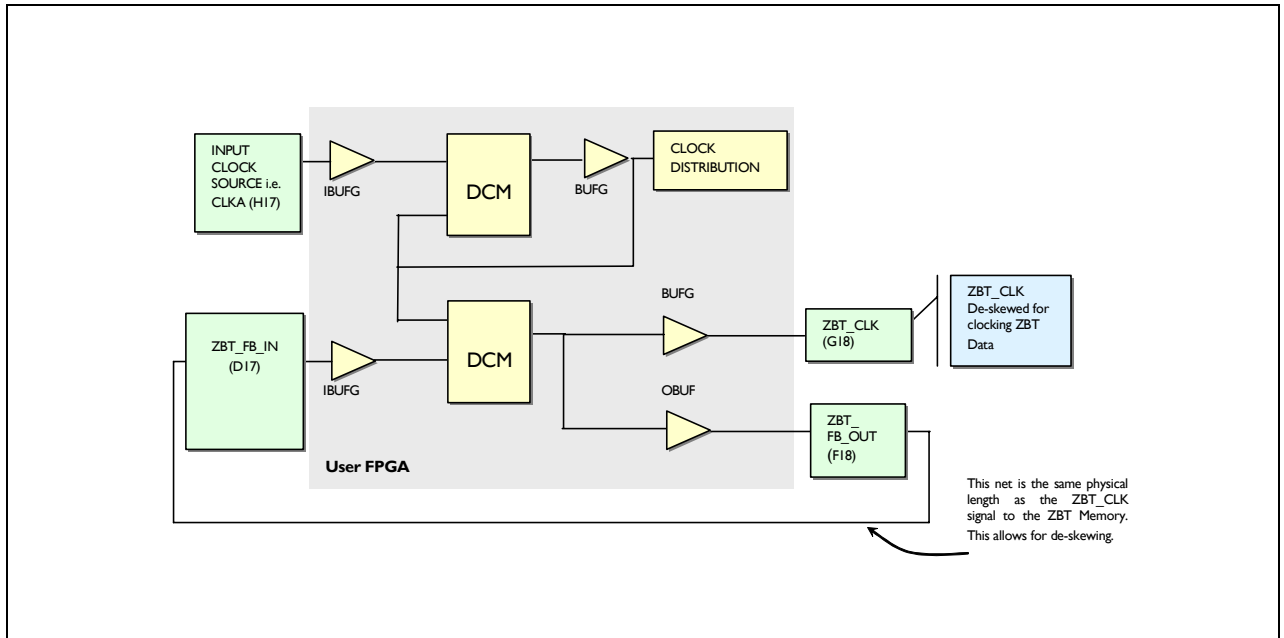


Figure 41: ZBT SRAM Clocking Example: CLKA Source



Note that the physical length of the net running from the FB_OUT to FB_IN pin is the same as the length of the ZBT_CLK nets from the pin on the User FPGA to the pin on the ZBT devices. The above arrangement ensures that the device is triggered in phase.

10.3 Firmware

After looking at the clocks from a hardware perspective, it is important to understand how to make best use of them. This Firmware section includes a brief discussion on the use of DCM (Digital Clock Managers) that are available in the Virtex-II silicon, and provides some notes and examples on using the clocks for specific purposes.

10.3.1 Using the Virtex-II Pro DCMs

Nallatech recommend use of the architecture wizard that is provided in the Xilinx ISE tools. For Virtex-II Pro targets this provides a powerful GUI-based input to create a VHDL clock module that can be added to your project. To use the Architecture wizard, from an existing project, simply go to the menu '**Project->New Source**'. The screen for a new source is shown in [Figure 42 on page 73](#). Select '**Architecture Wizard**' and then '**DCM**'.

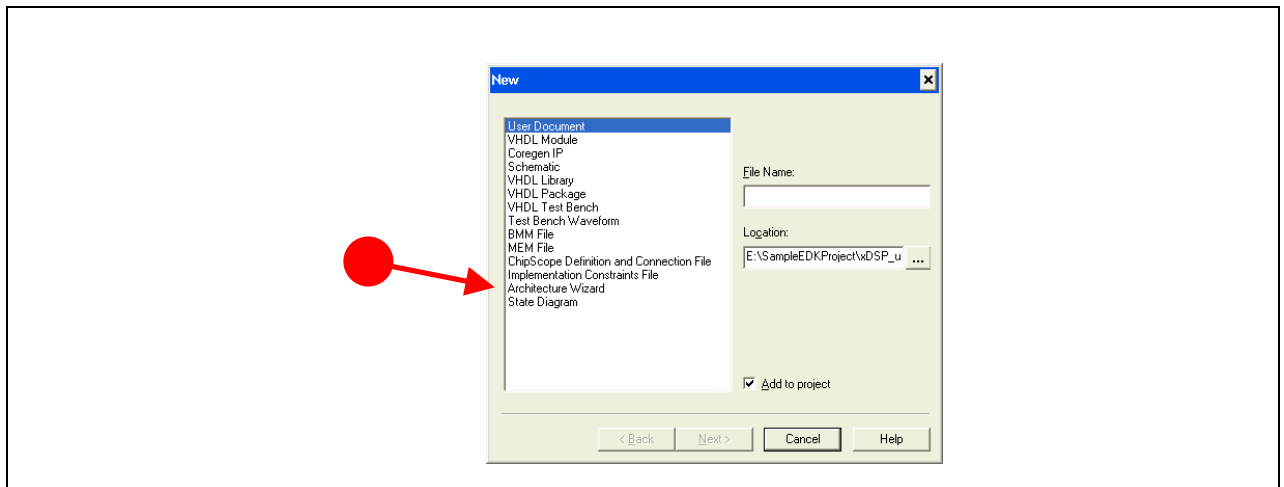


Figure 42: New Source Selection

The Xilinx DCM Wizard

In the DCM Wizard (shown in Figure 43 on page 73) select the options that are appropriate for your application.

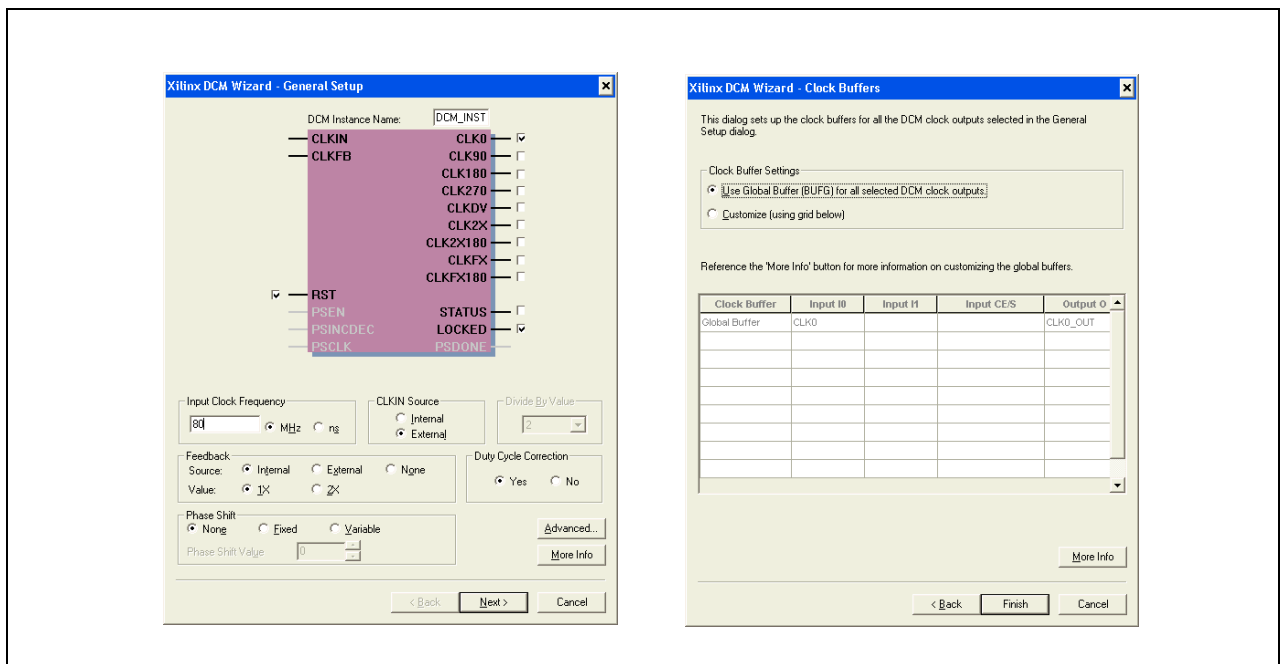


Figure 43: Virtex-II DCM Wizard

For the input frequency, it is recommended that you set the value to the desired clock frequency. This sets the low or high frequency mode of the DCM. For example, for the DIME Clocks(i.e. CLKA) you should set this to an appropriate frequency for the design. Here 80MHz has been specified as this is the frequency we intend to set the clock generator at in the software. The selection of external feedback must be set when you are using an external feedback mechanism - for example, ZBT_CLK_FB_IN net for the ZBT.

Using the DCM Locked Signal for Reset

It is good design practice to generate the reset signal from the locked signal of the DCM to the rest of your design that depends on the generated clock.

```
--Invert the active-low reset input to the FPGA to convert it to an active-high reset
RST_EXT <= not(RST1);

--Instantiate the clock module
Inst_dimeclk_module : dimeclk_module port map(
  rst_in      => RST_EXT,
  clkkin_in   => CLKB,
  locked_out  => CLKB_LOCKED,
  clk0_out    => CLKB1
);

--Create a register to register the locked signal into the clock domain before use in the rest of the system.
process(CLBK1)
begin
  if (CLKB1'event and CLKB1 = '1') then
    RST_INT1 <= CLKB_LOCKED;
  end if;
end process;
```

Figure 44: Using the DCM Locked Signal

Figure 44 on page 74 shows a section of code using a generated clock module in VHDL. The code is taken from the 'host_interface.vhd' code in the 'host_interface_basic' example that is included on the XtremeDSP Development Kit Pro CD. The example shows the active-low reset RST1 being inverted to produce an active-high reset required by the DCM. The locked signal from this DCM is then registered to produce a synchronous reset for use by the rest of the system.



Please refer to the relevant section in the *Virtex-II Pro Handbook* for detailed information on DCMs and their usage for all types of configuration.

DCM Wizard General Notes

If you check boxes for CLK_FX for example, you will be asked subsequent questions as you proceed through the wizard. This example uses a single clock input and produces a de-skewed clock output. When instantiating this module in your code, note that the reset input to the module is active high. However the reset input to the actual FPGA from the host interface is an active-low input. Therefore, inversion of the reset signal is required. For constraints you should build in some slack to your period constraints i.e. if it is a 100MHz input clock, do not set the constraint at 10ns. Rather, set it at 9.5 or 9.8ns to allow for jitter and skew on clocks and differences in the exact clock frequency.



DCMs must be reset after configuration if they make use of external feedback as is the case when using the ZBT feedback nets.

10.3.2 Clocking the ADCs and DACs

There are a number of ways in which the ADCs and DACs can be clocked and a degree of flexibility due to the inter-FPGA clocking structure between the main User FPGA (XC2VP30-4FFI152) and the Clock FPGA (XC2V80-4CS144). The options are further increased by the clock sources options described in **"Source Descriptions" on page 67**.

Using the Module 105MHz Crystal or External MCX Clock

Both these clocks are directly input to the Clock FPGA. They do not directly connect to the main User FPGA, therefore a design is required in the Clock FPGA to send the clock input down to the main User FPGA as well as to the ADCs and DACs. **Figure 45 on page 75** shows a suggested design.

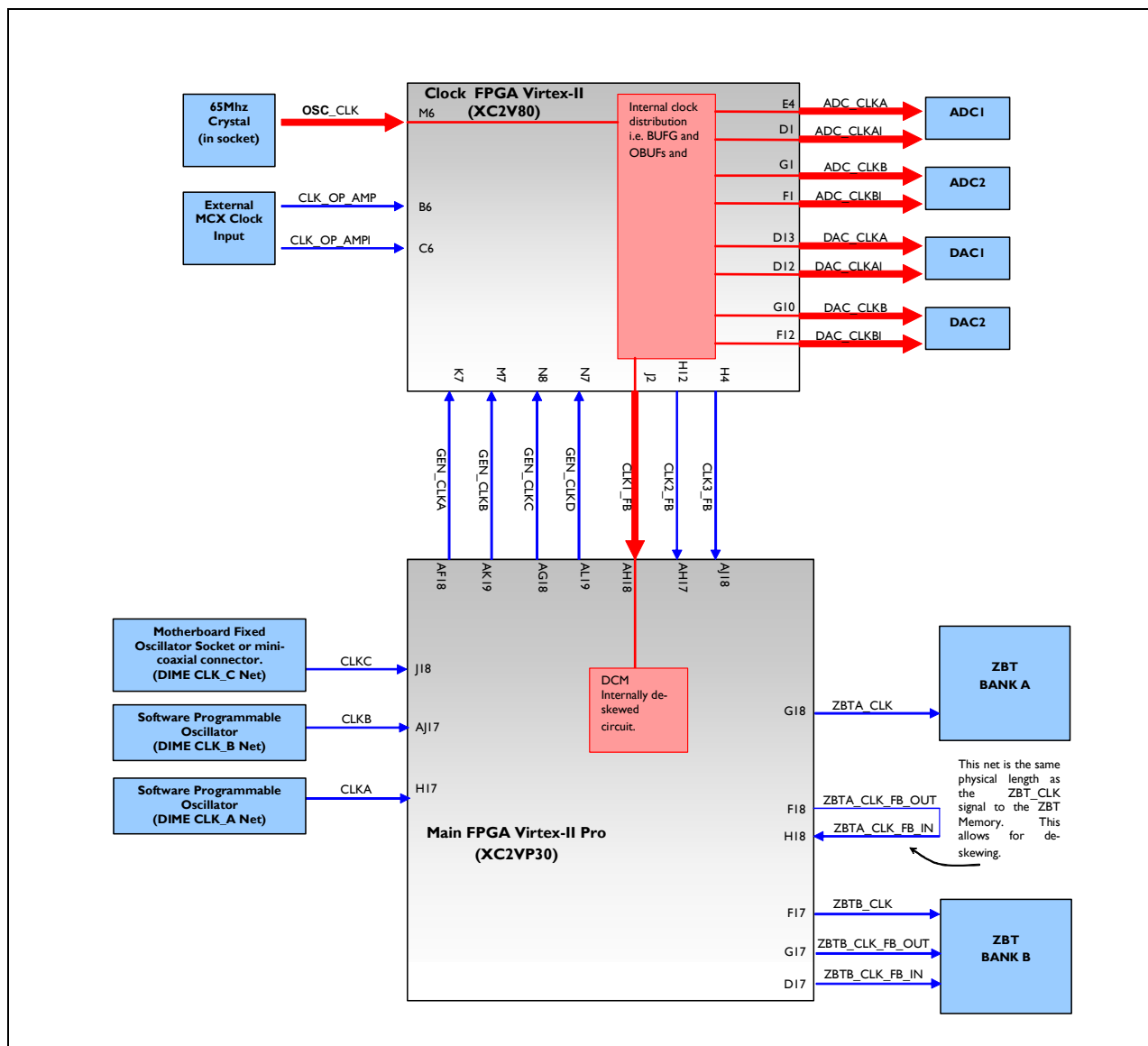


Figure 45: Using the Crystal or MCX Input to Clock the ADCs and DACs

An example of this clock circuit is provided on the XtremeDSP Development Kit Pro CD at the following location: 'CD ROM Drive\Examples\Clock_Designs'. There is VHDL source for an example showing the 105MHz crystal oscillator input, **'osc_clock.vhd'** and an example of using the external MCX clock input, **'ext_clock.vhd'**. Each of these VHDL files has an associated UCF in the same folder.

Using DIME Clocks

If you wish to make use of the DIME clocks to drive the ADC and DAC components, this must be driven up to the Clock FPGA from the User FPGA as shown in [Figure 46 on page 76](#).

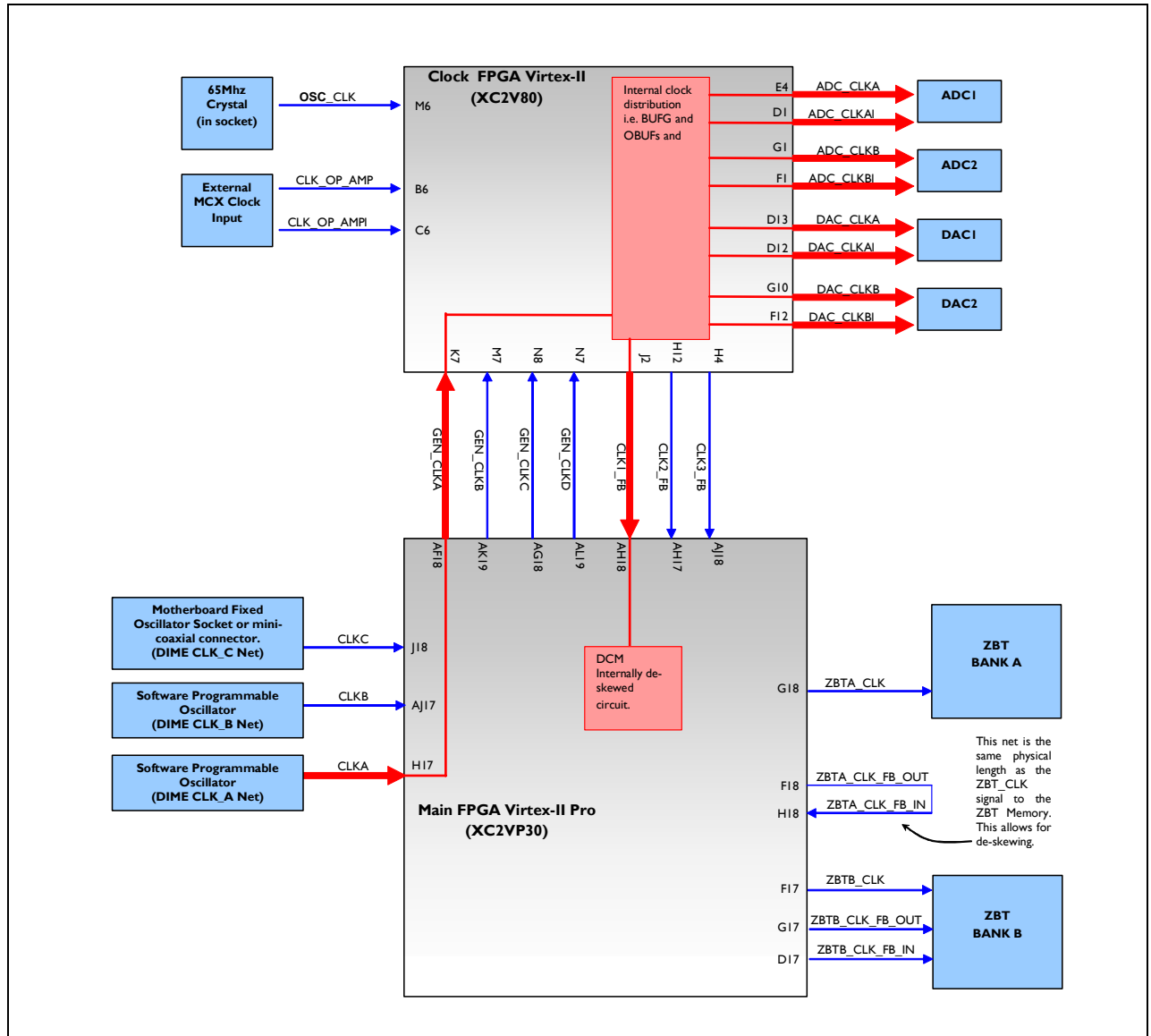


Figure 46: Using one of the DIME Clocks to Clock the ADCs and DACs

This is achieved by feeding through any one of the DIME clocks to one of the GEN_CLK nets within the main User FPGA. On the Clock FPGA, this is distributed to the ADCs, DACs and also fed back down one of the CLK_FB nets for use in designs on the main FPGA.

The CLK_FB nets and the nets to the ADCs and DACs are the same physical lengths on the board and so the skews on signals travelling down these nets will be matched. Therefore, to ensure the clocks reach the ADCs and DACs and a clock in the main User FPGA, create a DCM circuit to de-skew the internal to the main User FPGA.

Aspects of Clock Selection and Clock Jitter

With the number of available clocking options available, it is important to understand some of the factors involved when deciding how to manage the clocks in your design. One of these key factors is selecting the clock for the ADCs and DACs.

The DCMs themselves add output jitter dependent upon the input jitter and the period jitter of the DCM itself. A full discussion of calculating output jitter can be found in the Xilinx Answers Database (#13465 "Virtex/-E/-II/-II Pro - How do I calculate CLKDLL/DCM output jitter given a specific input jitter value?").

The DCMs add a small additional amount of jitter to the output clock - a factor to consider when you are clocking ADCs as this leads to aperture uncertainty which can reduce the accuracy of the ADC conversion. Therefore, a dedicated crystal is provided on the module with the additional option of fitting a dedicated crystal on the motherboard. This allows for highly accurate and low jitter clock sources.

The programmable clock sources on the board that drive the DIME clock nets (such as CLKA) are generated by Epson MG-7010SA PLL oscillators. A full *datasheet* is provided for these oscillators on the XtremeDSP Development Kit Pro CD at the following location: 'CD ROM Drive\Documentation\Datasheets'. The jitter for these clocks is 14ps rms. The *datasheet* (Pletronics Crystal.pdf) for the module crystal oscillator is also included in same folder. The jitter for this crystal is specified at 1ps rms.

10.4 Software

10.4.1 Setting the Clocks in FUSE

The FUSE Probe Tool can be used to set the programmable oscillators via the oscillator frequencies tab, highlighted in red in **Figure 47 on page 78**. Simply type the desired frequency in kHz in the box for the specific clock and hit '**Enter**' to set the clock, i.e. for a 40MHz clock type in '40000'. Please note you must open and select a card in the FUSE Probe Tool in order to control the oscillators.



Please note the following naming conventions for the clocks.

- CLKA (This is SYSCLK in the DIME-II standard naming)
- CLKB (This is DSPCLK in the DIME-II standard naming)
- CLKC (This is PIXCLK in the DIME-II standard naming)



Also note that CLKC (i.e. PIXCLK in the DIME-II standard) is not connected to a programmable oscillator and so this control is effectively redundant in the Kit hardware.

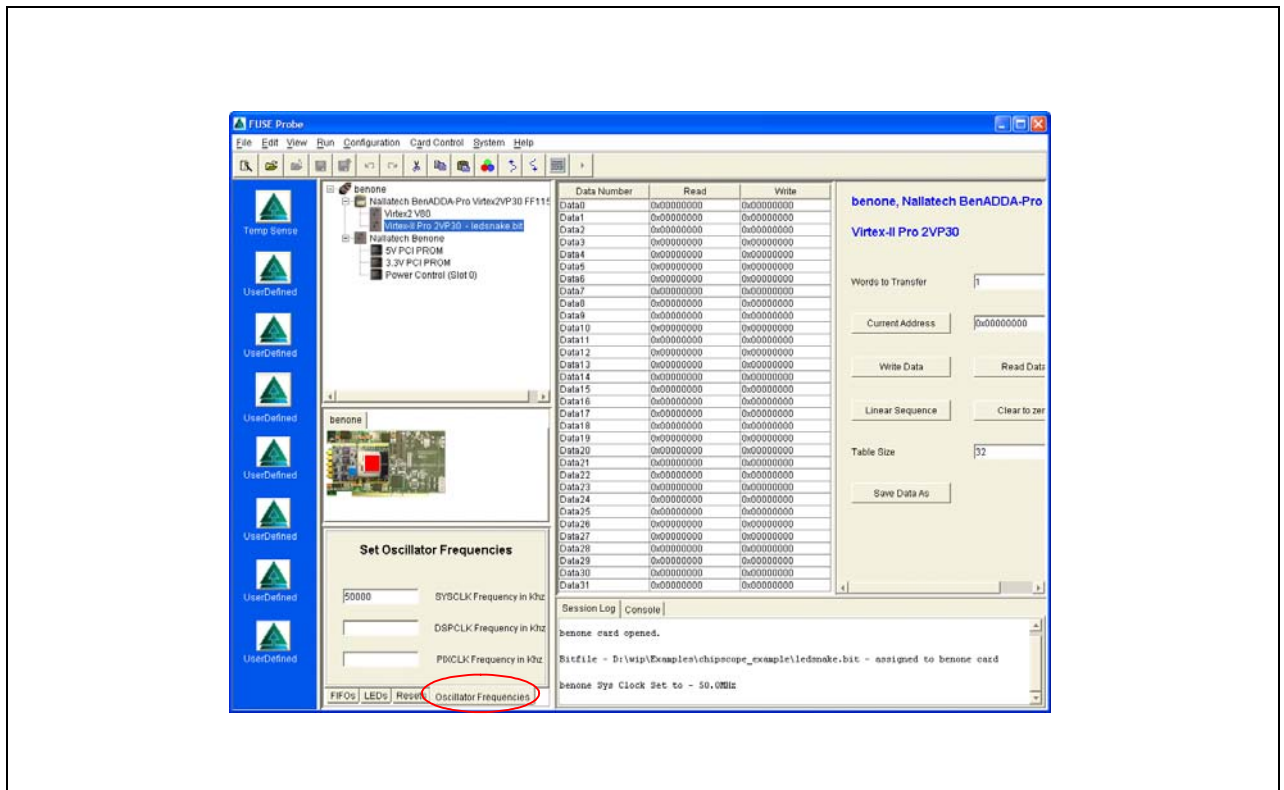


Figure 47: Setting Clock Frequencies in FUSE Probe Tool

10.4.2 Listing of Calls in the FUSE API to Set the Clocks

The FUSE C/C++ API provides a function for controlling the programmable oscillators in the Kit from the interface FPGA. These are:

- **DIME_SetOscillatorFrequency**

Essentially the command takes two parameters - a 'resetNum' and a 'cmdMode'.

- OscillatorNum determines which clock is changed where:
 - 0 = All Clocks
 - 1 = CLKA (This is SYSCLK in the DIME-II standard naming)
 - 2 = CLKB (This is DSPCLK in the DIME-II standard naming)
 - 3 = CLKC (This is PIXCLK in the DIME-II standard naming)
- DesiredFrequency is the requested frequency in MHz.

When this function is called it will try and set the selected clock to the desired frequency. If the desired frequency is not supported then the actual frequency set will be reported. This is because the programmable oscillators used in the

Kit only support a specific set of frequencies. **Figure 48 on page 79** shows an example of the how to control the clocks via the C/C++ FUSE API

```
//Change the oscillators.  
{  
double ActualFrequency;  
//Try and set oscillator 1, the system clock to 41.23456MHz  
DIME_SetOscillatorFrequency(hCard1,1,41.23456,&ActualFrequency);  
printf("Actual frequency is %f.\n",ActualFrequency);  
}
```

Figure 48: Getting Information on the Located Cards



Please note that the CLKC net is not from a programmable oscillator as it is connected to a socket to support a crystal oscillator. Calls to set CLKC may however still return a value to say the control register has been set to request a specific frequency.

The *FUSE C/C++ API Developers Guide*, included on the FUSE CD, provides further details on this function.

Section I I

Bus Structure

In this section:

- Introduction to the Bus Structures in the Kit
 - Hardware Aspects
 - Firmware Aspects
 - Software Aspects
-

II.1 Introduction

There are a number of busses used in the XtremeDSP Development Kit Pro which can be used to communicate with the host computer as well as for user-defined purposes. The Kit hardware uses a single main FPGA, therefore most of the user-specific busses are brought out to headers where possible. This section provides some discussion on the general bus structure and also a brief discussion on the naming and terminology of specific busses.

II.1.1 Bus Naming

On initial reading some of the bus names may be confusing. The particular names have been selected based on the DIME-II module standard to which the hardware has been designed. To support the large volume of communications between DIME-II modules and DIME-II motherboards, the following busses provide a robust infrastructure in the XtremeDSP Development Kit Pro:

- Adjacent In Bus
- Adjacent Out Bus
- Comm PLINK Busses
- Local Bus

In the DIME-II standard the Adjacent, Comm PLINK and Local Busses permit data to be transferred to all the various devices used in larger multi-FPGA DIME-II based systems. Below is a brief introduction to these busses.

Comms Parallel Link (PLINKs) are 12-bit bi-directional point-to-point busses, that allow the DIME-II module to communicate directly with external data.

There are two types of adjacent busses: the **Adjacent In** bus and **Adjacent Out** bus. These busses are designed to facilitate the use of pipelined architectures on multiple DIME-II module systems where the resultant data processed on one DIME-II module can be passed to the next module for further processing. Although the Adjacent busses are defined as 'Adjacent In' and 'Adjacent Out' note that both these busses connect to bi-directional I/O on the FPGA, therefore the Adjacent In and Out busses can both be considered as bi-directional despite their naming. The sectioning into an In and Out bus is a historical DIME-II naming convention where the names were used to highlight the potential dataflow in a multiple DIME-II module system.

The **Local Bus** is coupled from the Interface FPGA and the main User FPGA in the system. In the Kit this is the XC2VP30-4FF1152 main User FPGA on the module. On other Nallatech hardware this may be an FPGA on the motherboard itself. The Local Bus is intended to be used as an overall system control or broadcast bus which can typically be utilized to memory map internal registers and memory space in the FPGA into microprocessor memory space.

As the hardware supplied in the Kit is based on a single DIME-II module then most communication busses are used to provide digital I/O or host interfacing capabilities.

11.2 Hardware

DIME-II Communication Busses

Table 31 on page 82 provides details on the available I/O for each of the communication busses.

Communication Bus	XC2VP30-4FF1152	Notes
Adjacent IN	28 bits	Connects directly to the digital I/O ADJACENT HEADER (J8)
Adjacent OUT	7 bits	Provides the control and status signals for the interface to main User FPGA communication bus.
Local Bus	32 bits	All 32 bits are used as the ADIO signals in the interface to main User FPGA communication bus.
Comm Link 0	12 bits	Connects directly to the digital I/O PLINK header (J10)
Comm Link 7	4 bits	Connects to a Nallatech specific test header (RS232) (J9).

Table 31: Bus Summary

Bus Structure

Figure 49 on page 83 provides an overview of the bus structure in the Kit.

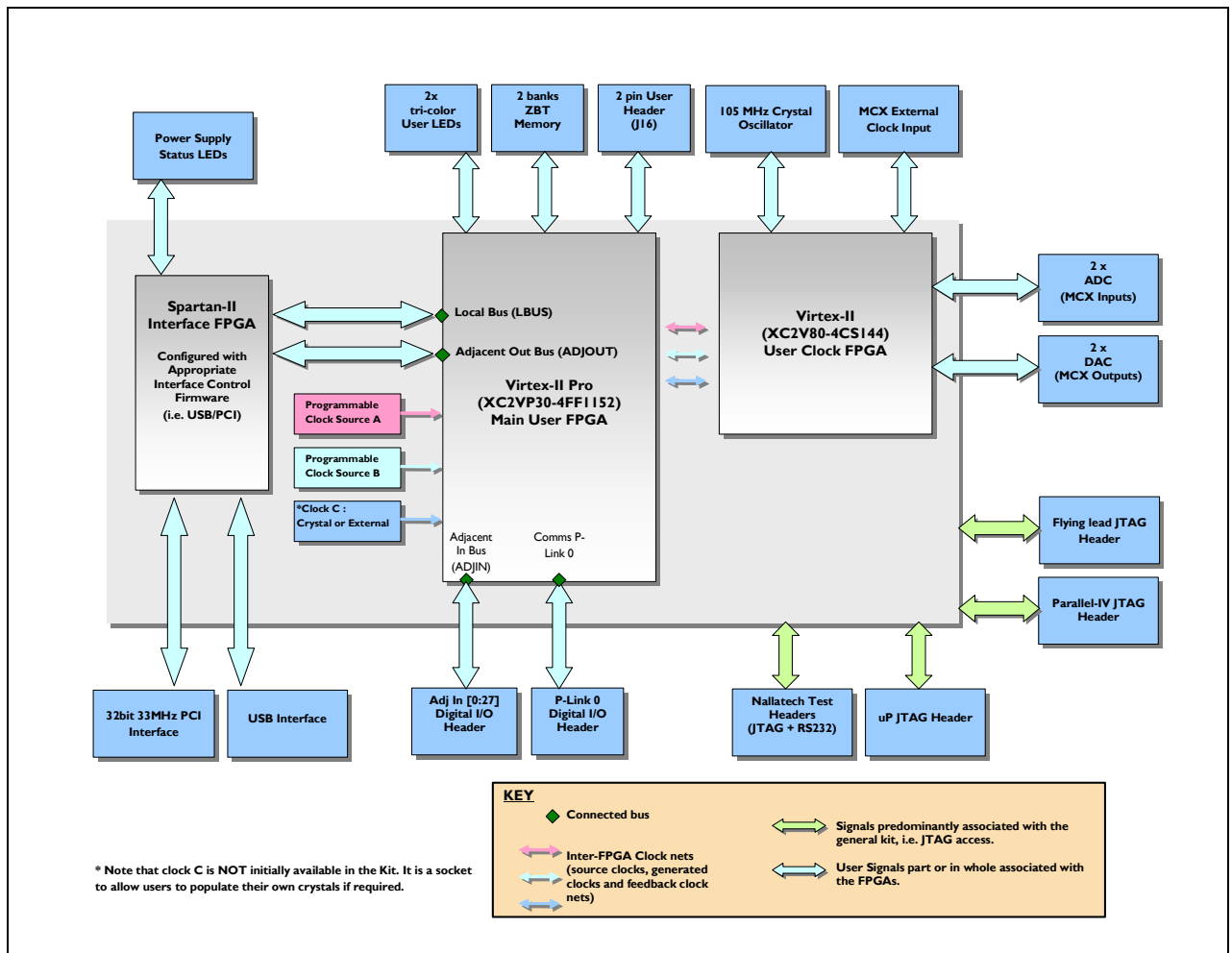


Figure 49: Bus Functional Diagram

11.3 Firmware

Nallatech provide cores for connecting the Local Bus and Adjacent Out Bus to user designs. Please see “Part III: System Level Design” on page 109 which describes how to build a host interface.

11.4 Software

There are no specific software functions in the FUSE API for controlling the actual bus functionality as this is a defined hardware architecture.

Section 12

Board and System Level Monitoring Capabilities

In this section:

- Introduction to the System Monitoring Capabilities of the Kit
 - Hardware Aspects
 - Firmware Aspects
 - Software Aspects
-

12.1 Introduction

The Kit has a number of mechanisms for monitoring and/or board level hardware control. These can be broken down into the following categories:

- Temperature monitoring
- JTAG Chain Access
- Configuration Status Monitoring

12.2 Temperature Monitoring

On board Temperature Sensor

The Kit is fitted with a User FPGA which can become very hot when running at full potential in certain environments. As a result, the Kit is fitted with a temperature device that monitors the heat levels within the main User FPGA and also the ambient temperature of the module. Figure 50 on page 86 shows the temperature sensor interface on the BenADDA DIME-II module.

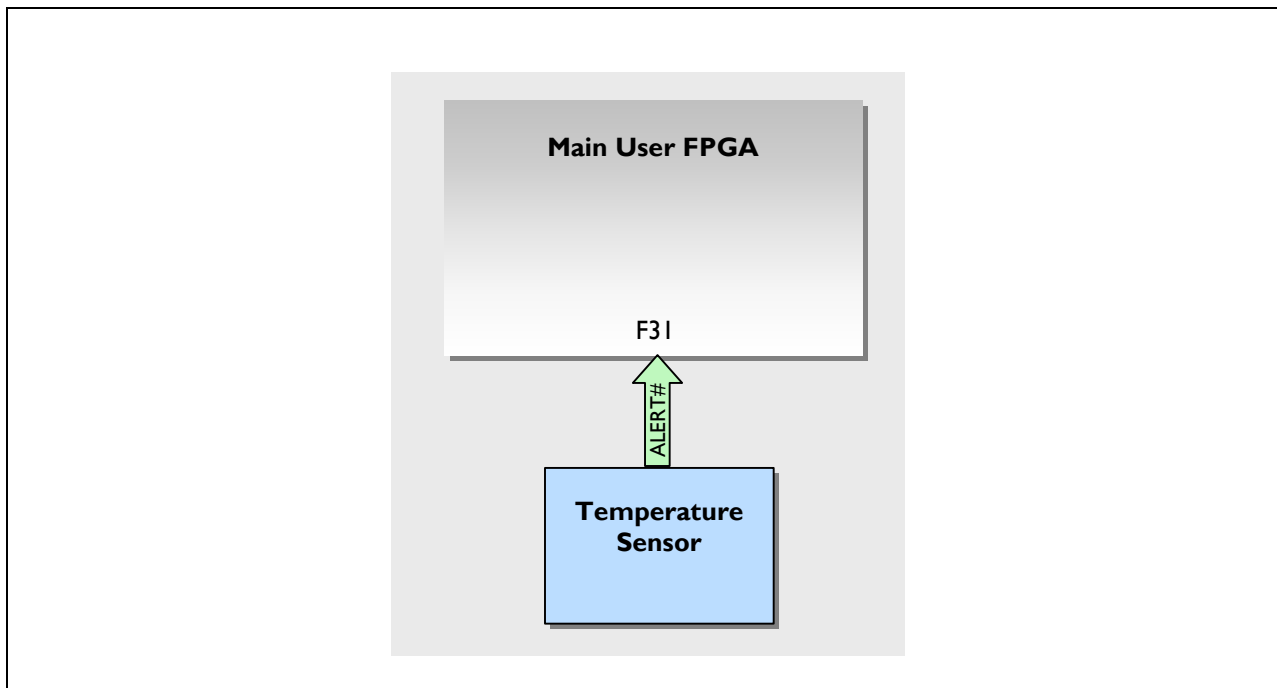


Figure 50: Temperature Sensor Interface

The Temperature Sensor used on the BenADDA DIME-II module is supplied from MAXIM (Part Number: MAX1617MEE). For full details on the specification of this device please refer to the *datasheet* MAX1617, which is supplied on the XtremeDSP Development Kit Pro CD at the following location: 'CD ROM Drive\Documentation\Datasheets'

Signal Name	User FPGA (XCVP30-4FF1152) PIN No
ALERTI	F3I

Table 32: Temperature Sensor Pinouts (2V3000)

The ALERTI signal is an active-low signal that allows the temperature sensor to signal to the main User FPGA design that a user set temperature threshold has been exceeded. This feature allows the design to automatically slow its clock or disable part of the function until the design comes back into the allowed operating range. The specific action taken is dependant on the user design.



Extreme care should be taken with the temperature levels of the FPGA. Commercial grade silicon has a junction temperature range of 0°C - 85°C, and -40°C - 100°C for Industrial grade silicon.



Incorporating the ALERTI signal into an FPGA design provides thermal protection to the FPGA. The ALERTI signal should have a pull-up instantiated in the FPGA design. It is recommend that for any high power FPGA designs a heat sink should be fitted to the FPGA.

FUSE Probe Tool Monitoring Plug-in

The XtremeDSP Development Kit Pro is supplied with a version of FUSE that provides a software GUI called the FUSE Probe Tool. This tool now has a plug-in to facilitate monitoring of the Kit's temperature and voltages.

▼ To start the FUSE Probe Tool Monitoring Plug-in use the following procedures:

- I. To open the FUSE Probe Tool either click on the Nallatech Icon on the taskbar or go to the 'Start' menu and then click on 'Progams->FUSE->Software->FUSE Probe'. This will launch the FUSE Probe Tool shown in **Figure 51** on page 87.

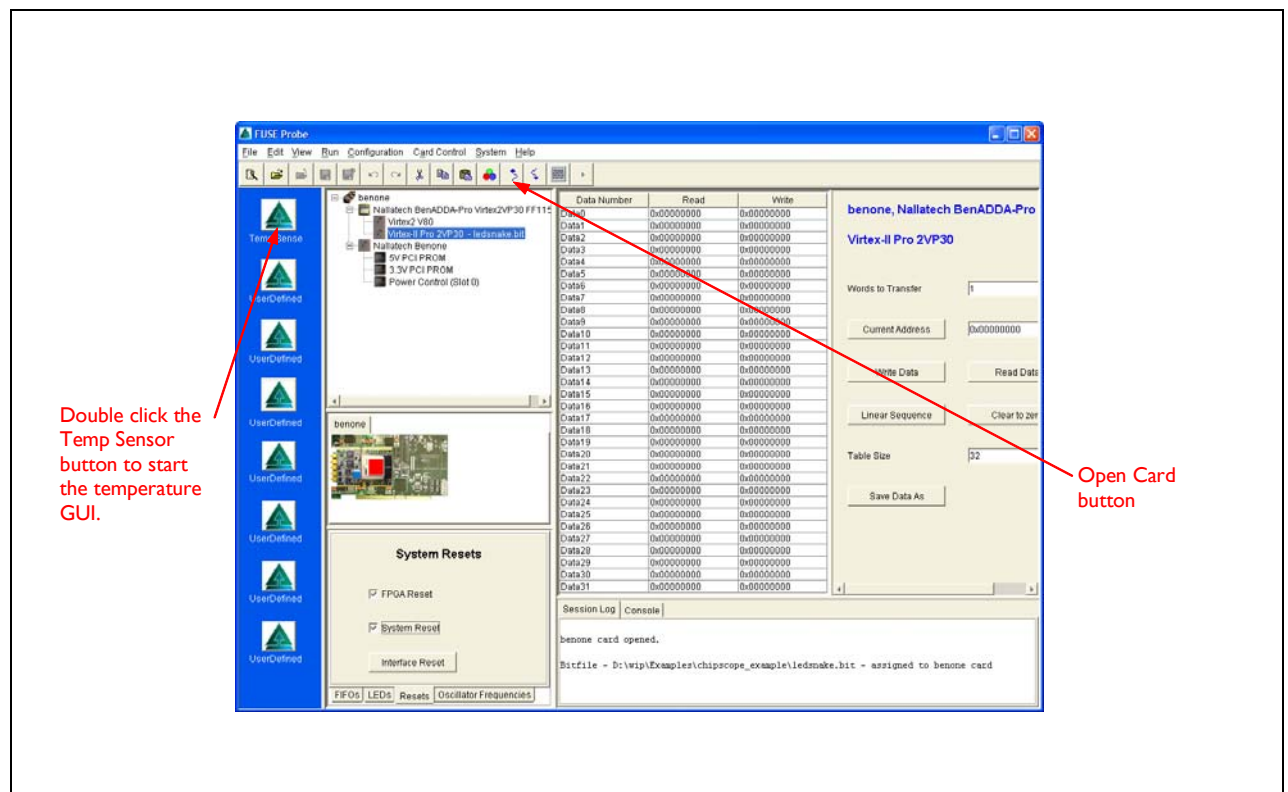


Figure 51: FUSE Probe showing Temp Sensor Plug-in

2. Double-click on the temperature sensor button in the right-hand pane of the FUSE Probe Tool to launch the Temperature GUI.

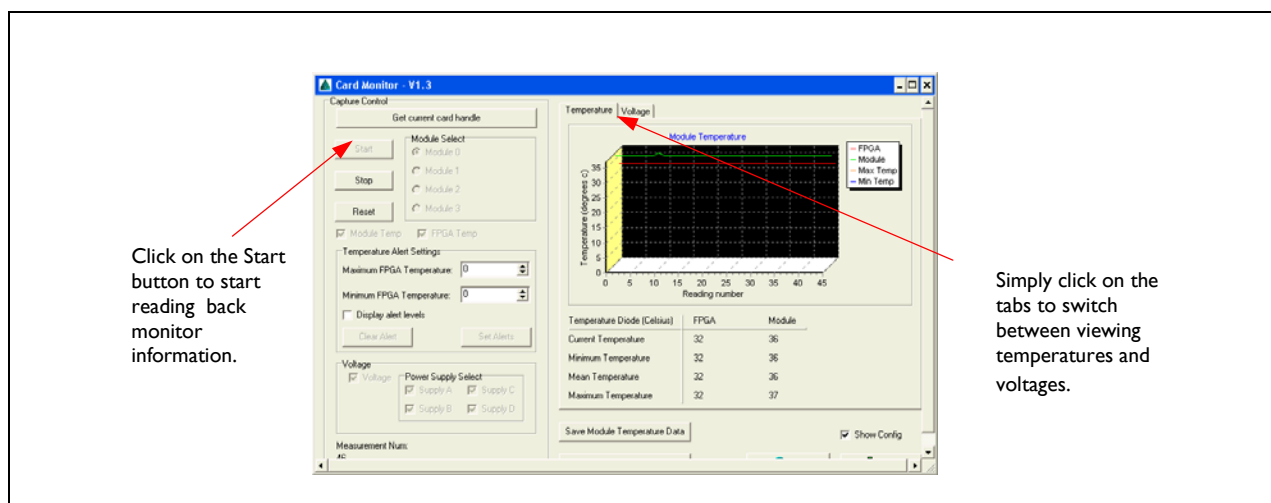


Figure 52: Temperature and Voltage Monitoring

FUSE Software API Monitoring Functions

Software functions through the FUSE API provide temperature-monitoring capability for the on board FPGA. The API functions provide the ability to readback the junction of the FPGA and the external PCB temperature. Other functions include the ability to set the minimum and maximum temperature thresholds. The temperature thresholds are linked to the ALERT1 signal - when either threshold is reached the ALERT1 signal to the on board FPGA is asserted. Voltages can also be returned for the whole module or simply for one of the individual power supplies depending upon the API calls arguments. The details of the module power supplies, i.e. PSU_A, are given in [“Power Specifications” on page 99](#).

The C/C++ API functions detailed below are taken from the *FUSE C/C++ API Developers Guide* that is provided on the FUSE CD.

DIME_ModuleControl

Syntax DWORD DIME_ModuleControl(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode, DWORD Value)

Arguments handle is a valid handle to a DIME carrier card

ModuleNum is the module that is being addressed. Note modules are numbered from 0.

CmdMode: This argument is used to specify what particular aspect of module is to be controlled.

ResetNum	Description
dinfTEMPALERTMAX	This command mode is used to set the maximum temperature level for the temperature alert signal. Once set if the FPGA die temperature exceeds this the temperature alert signal is triggered. Note that the power on default setting for this temperature is 255 degrees Celsius. Value should be the integer value that the maximum alert should be set to in degrees Celsius.

Table 33: DIME_CardResetControl ResetNum Argument Options

ResetNum	Description
dinfTEMPALERTMIN	This command mode is used to set the minimum temperature level for the temperature alert signal. Once set if the FPGA die temperature falls below this the temperature alert signal is triggered. Note that the power on default setting for this temperature is 0 degrees Celsius. Value should be the integer value that the minimum alert should be set to in degrees Celsius.
dinfTEMPALERTCLEAR	This clears the temperature alert signal if set. Note that if either the maximum or minimum temperature limits are still exceeded then the alert signal will immediately be set. Value should be set to 0.

Table 33: DIME_CardResetControl ResetNum Argument Options

Value: This argument is command mode specific.

CmdMode	Description
drDISABLE	This de-asserts the reset line for the selected reset.
drENABLE	This asserts the reset line for the selected reset.
drTOGGLE	This toggles the reset line for the selected reset.

Table 34: DIME_CardResetControl CmdMode Argument Options

The value argument is not used in this function and is only included for consistency.

Return Returns -1 on error.

Description This function is used to control certain aspects of the selected module.

Example

```

{
DWORD MaxAlert=65;
DWORD MinAlert=0;
DWORD ModuleNumber=0;
//Set the max and min alert levels
if(DIME_ModuleControl(hCard1,ModuleNumber,
                      dinfTEMPALERTMAX,MaxAlert)==0){
    printf("Maximum FPGA Temperature set to %d
    degrees.\n",MaxAlert);
}

if(DIME_ModuleControl(hCard1,ModuleNumber,
                      dinfTEMPALERTMIN,MinAlert)==0){
    printf("Minimum FPGA Temperature set to %d
    degrees.\n",MinAlert);
}

//this code should be place in your temperature alert handler
//once you've dealt with the alert and desire to clear the alert
//line to the FPGA
if(DIME_ModuleControl(hCard1,ModuleNumber,
                      dinfTEMPALERTCLEAR,0)==0){
    printf("The temperature alert line for module %d has been
    cleared.\n",ModuleNumber);
}
}

```

Figure 53: Setting Maximum and Minimum Temperature Alert Limits

DIME_ModuleStatus

Syntax DWORD DIME_ModuleStatus(DIME_HANDLE handle, DWORD ModuleNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card

ModuleNum is the module that is being addressed. Note modules are numbered from 0.

CmdMode: This argument is used to specify what particular aspect of module status information is to be returned. The table below gives details of the available command modes.

ResetNum	Description
dinfFPGATEMP	This command mode returns the die temperature of the FPGA in degrees Celsius. Temperatures are accurate to +/- 1 degree
dinfMODULETEMP	This command mode returns the temperature of the module in degrees Celsius. Temperatures are accurate to +/- 1 degree. Note that this temperature is measured next to the User FPGA and hence usually follows the FPGA temperature. It shows the temperature of the module as a whole but not one specific device.

Table 35: DIME_ModuleStatus CmdMode Argument Options

ResetNum	Description
dinfTEMPALERTMAX	This command mode is used to set the maximum temperature level for the temperature alert signal. Once set if the FPGA die temperature exceeds this temperature the then the temperature alert signal is triggered. Note that the power on default setting for this temperature is 255 degrees Celsius.
dinfTEMPALERTMIN	This command mode is used to set the minimum temperature level for the temperature alert signal. Once set if the FPGA die temperature falls below this the temperature alert signal is triggered. Note that the power on default setting for this temperature is 0 degrees Celsius.

Table 35: DIME_ModuleStatus CmdMode Argument Options

Return The return value is dependant upon the command mode. Returns -1 on error.

Description This function returns module status information.

Example

```
//read the temperature alert levels and both the module and FPGA
//temperatures
{
    DWORD ModuleNumber=0;
    DWORD FPGATemp,ModuleTemp,MaxAlert,MinAlert;
    //Read the FPGA temperature (degrees c)
    FPGATemp=DIME_ModuleStatus(hCard1,ModuleNumber,dinfFPGATEMP);
    //Read the module temperature (degrees c)
    ModuleTemp=DIME_ModuleStatus(hCard1,ModuleNumber,dinfMODULETEMP);
    //Read the maximum alert threshold temperature (degrees c)
    MaxAlert=DIME_ModuleStatus(hCard1,ModuleNumber,dinfTEMPALERTMAX);
    //Read the minimum alert threshold temperature (degrees c)
    MinAlert=DIME_ModuleStatus(hCard1,ModuleNumber,dinfTEMPALERTMIN);
}
```

Figure 54: Reading Temperature Levels

DIME_PPSSStatus

Syntax DIME_PPSSStatus(DIME_HANDLE handle, DWORD ModuleNum, DWORD SupplyNum, DWORD CmdMode)

Arguments handle is a valid handle to a DIME carrier card

ModuleNum: This is the module number.

CSupplyNum: This argument is used to specify the targeted power supply. Valid supply numbers are given below.

SupplyNum	Description
dppsSUPPLYA	Power Supply A is selected.
dppsSUPPLYB	Power Supply B is selected.
dppsSUPPLYC	Power Supply C is selected.
dppsSUPPLYD	Power Supply D is selected.
dppsALLSUPPLYS	All supplies are selected.

Table 36: DIME_PPSSStatus Supply Number Options

CmdMode: This argument is used to specify what particular aspect of programmable power supplies information is required.

Cmd Mode	Description
dppsVOLTAGE	This command mode selects that only voltage information is returned. The voltage returned is given in millivolts and has an error of +/- 100millivolts.

Table 37: DIME_PPSSStatus Command Mode Options

Return The return is dependant upon the selected command mode.

Description Returns status information for the programmable power supplies.

Note The voltage capabilities are only applicable to DIME II systems.

Example

```
//Get the Voltage for module 0 power supply C.
{
DWORD Voltage;
Voltage=DIME_PPSStatus(hCard1,dppsMODULE0,
                        dppsSUPPLYC, dppsVOLTAGE);
printf("Core Voltage is %d millivolts.\n", Voltage);

//Get the Voltage for all of module 0.
Voltage=DIME_PPSStatus(hCard1,dppsMODULE0,
                        dppsALLSUPPLIES, dppsVOLTAGE);
printf("The total Voltage for Module 0 is %d.\n", Voltage);
}
```

Figure 55: Getting Information on Power Supply Voltages

12.3 JTAG Chain Access

The JTAG chain is used for test and configuration purposes. All DIME-II modules, such as the BenADDA DIME-II module used in the Kit, have a JTAG based Plug and Play (PnP) facility to enable auto-detection of the modules present in a system. Each DIME-II module has a unique ID number. The BenADDA ID is listed in [Table 38 on page 94](#):

ID Number (Hex)	Description
32983033	User FPGA: XCVP30-4FF1152, Clock FPGA: XC2V80-4CS144

Table 38: BenADDA Assigned MDF Code Listing

The physical order of the devices in the JTAG chain, illustrated in [Figure 56 on page 94](#), is:

1. User FPGA
2. Clock FPGA

To establish the module order in the JTAG chain, FUSE is deployed. The software initially scans the chain to identify and index the devices on the BenADDA DIME-II module. The device nearest the TDO output of the module is identified as device 0 on that module. For each device upstream on the JTAG chain, the index is incremented. [Figure 56 on page 94](#) shows the device numbers that are assigned for the BenADDA DIME-II module used in the Kit.

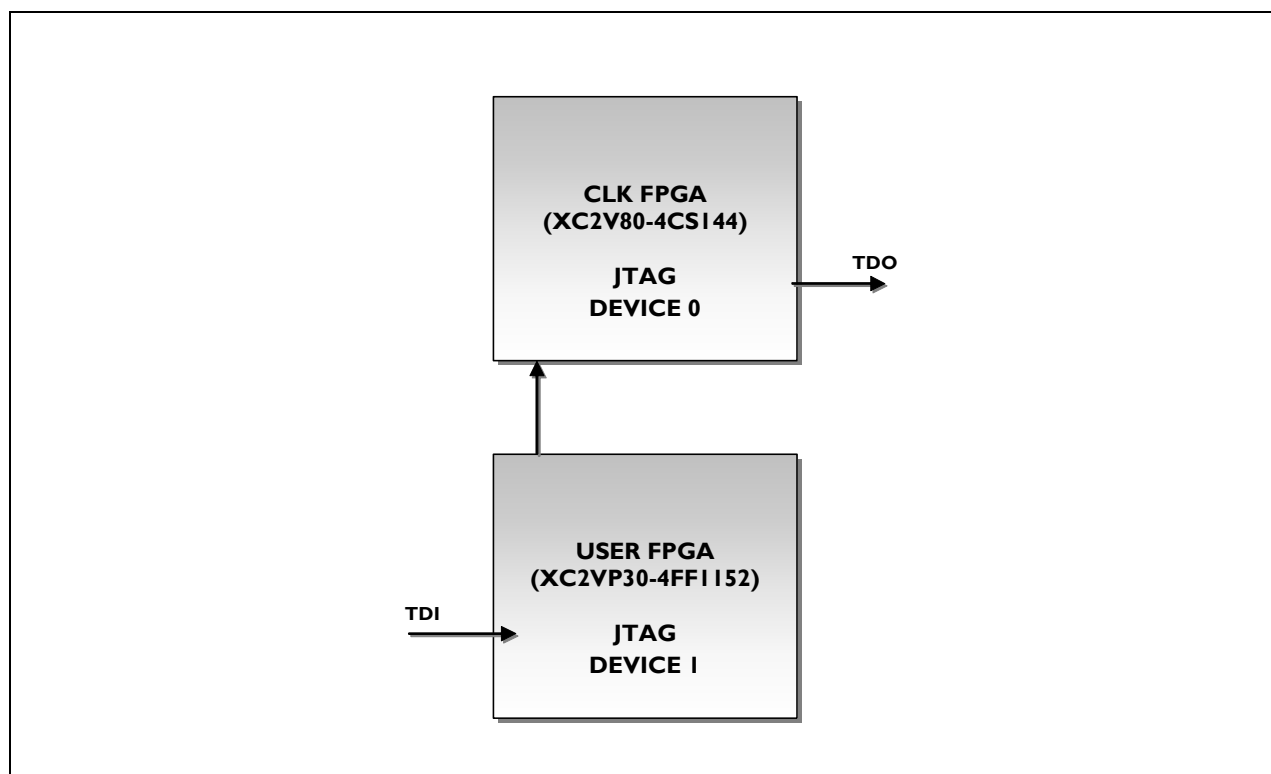


Figure 56: JTAG Device Indexing

There are a number of headers on the board that allow access to the JTAG chain. The location of each of these headers is shown in [“Front View of Board Physical Layout” on page 17](#). Details of configuring FPGAs through the JTAG chain is provided in [“FPGA Configuration using General JTAG Chain” on page 126](#). Further details on using the headers with specific software such as Chipscope, EDK and Impact are also provided later in the User Guide.

12.3.1 General JTAG Header

The General JTAG header (Figure 57 on page 95) connects to the chain that runs through the standard JTAG pins on the User FPGA devices in the Kit. It also connects to some of the PROMs and CPLDs that are used. It is a standard 0.1" pitch header which supports flying lead connections for, amongst others, the Xilinx Parallel-III or Parallel-IV pods. The header pinouts are shown in Table 39 on page 95

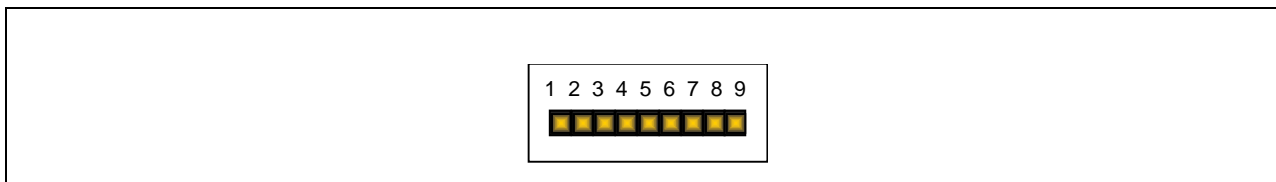


Figure 57: General JTAG Connector J14

Pin #	Name	Description
1	3.3V	3.3 Volts Supply
2	GND	Signal Ground
3	N/C	Not connected - do not use
4	TCK	ALT JTAG TCK Signal
5	N/C	Not connected - do not use
6	TDO	ALT JTAG TDO Signal
7	TDI	ALT JTAG TDI Signal
8	TRST#	ALT JTAG TRST# Signal
9	TMS	ALT JTAG TMS Signal

Table 39: General JTAG Header J14 Pinouts

12.3.2 Parallel-IV JTAG Header

This is a JTAG header that connects to the chain running through the standard JTAG pins on the User FPGA devices in the Kit and also some of the PROMs and CPLDs that are used. The header (J24 on the motherboard) allows a Xilinx Parallel-IV cable to be plugged in. The connector is keyed and follows the pinout on the datasheet supplied with the Parallel-IV cable from Xilinx.



Please note that the Kit is not supplied with a Parallel-IV download cable and that this section provides information on how to connect the cable only if one is available.



This is the recommended header to use if you are using the Xilinx JTAG Cosimulation option in the Xilinx System Generator tool.

When the board is used inside the blue board case, the Parallel-IV ribbon cable is brought out through the small gap marked P-IV in the side of the case. This case should be opened, the Parallel-IV ribbon cable fitted and then the case closed again.

Fitting the cable

To fit the cable remove the four screws round the sides of the case and remove the lid. Then plug one end of the Parallel-IV ribbon cable into the keyed header and bring the cable out to the small opening in the base of the case as shown in [Figure 58 on page 96](#).

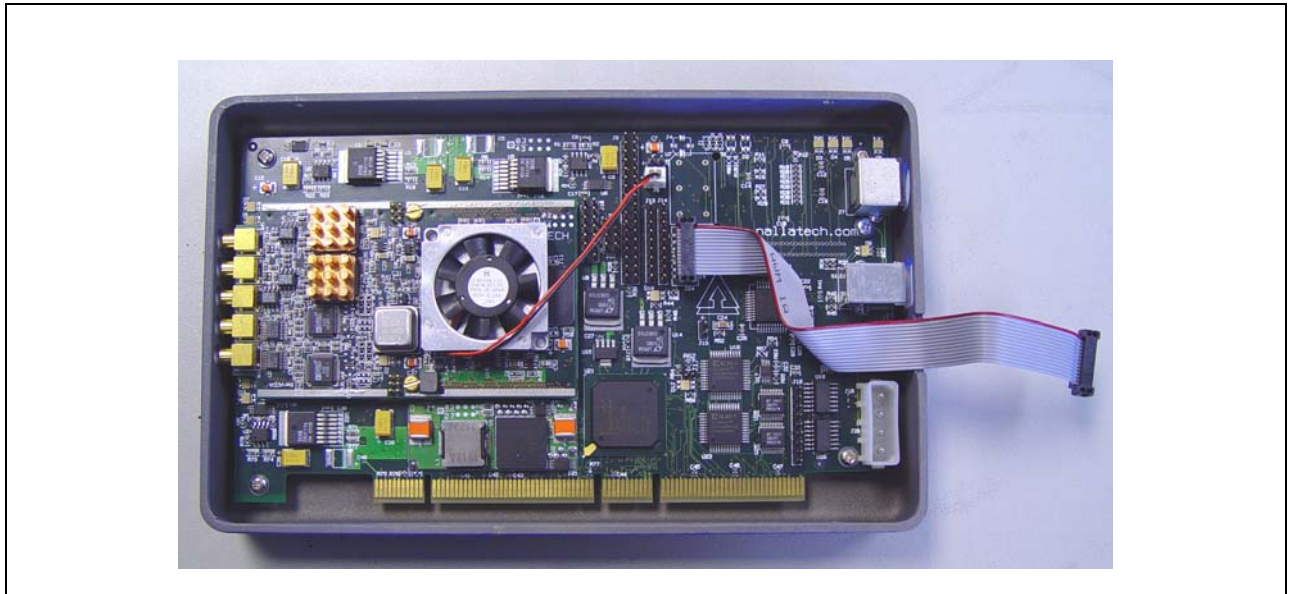


Figure 58: Routing Cable Out from Header

Then fit the lid back onto the case, refitting the screws if necessary, and ensure that the ribbon cable is not caught between the top and bottom of the blue case. Finally, connect the Parallel-IV pod to the exposed end of the ribbon cable as shown in [Figure 59 on page 96](#). Both the connector on the cable and the socket on the pod are keyed for ease of use.

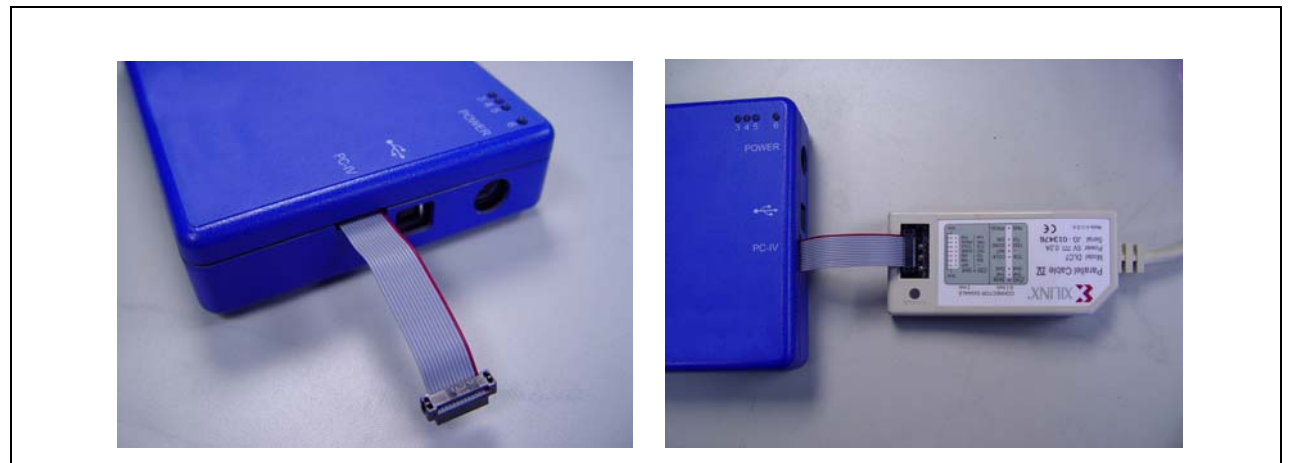


Figure 59: Re-fitting the case lid (first), Connecting the Parallel-IV pod (second)

12.3.3 Configuration Status Monitoring

The BenADDA DIME-II module has a mechanism for detecting the configuration status of the module. This signal is designed for larger systems where there are multiple modules present. There is the provision of a signal called 'CONFIG_DONE' which is related to the configuration of the on board FPGA. The DIME-II standard allows a design to be implemented based on the status of the entire system. The 'CONFIG_DONE' signal provides built-in control that can be used by the system designer. This feature allows a designer to synchronize all aspects of the configuration startup of complete system.

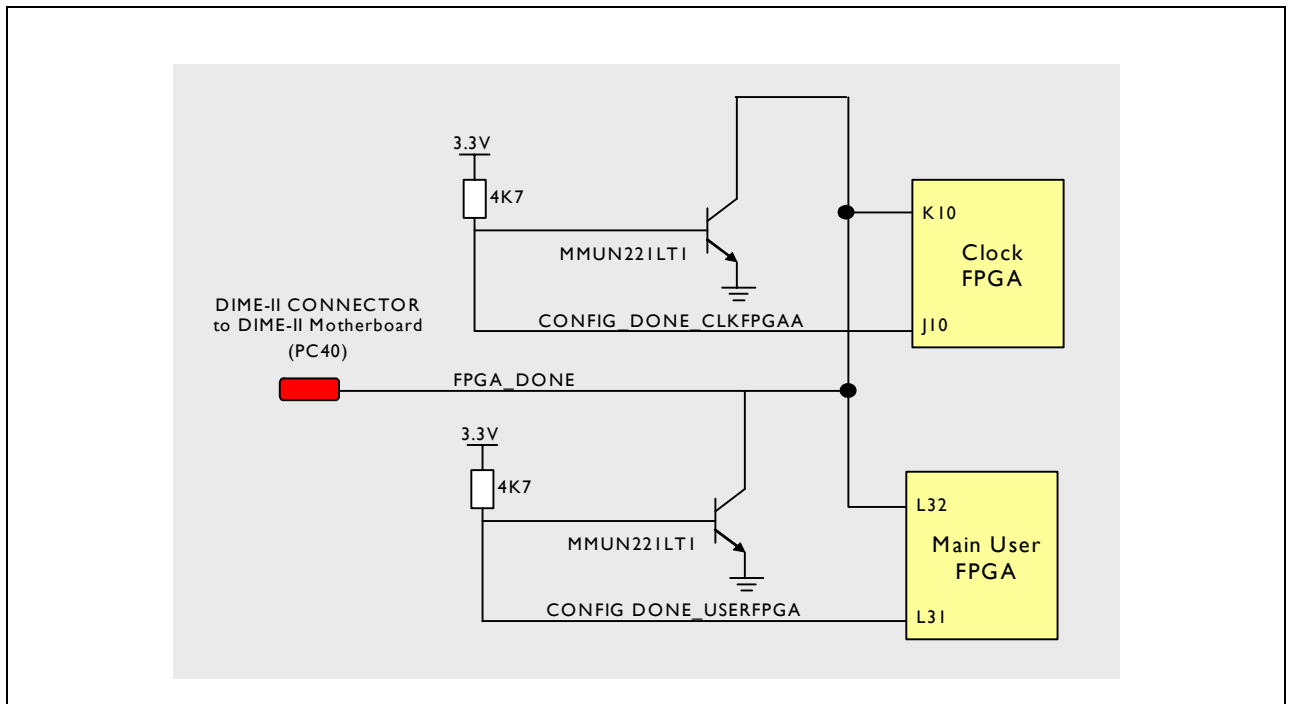


Figure 60: CONFIG DONE Circuit

Figure 60 on page 97 shows the set-up of the 'CONFIG_DONE' signal on the BenADDA DIME-II module. The FPGA_DONE signal to the two on board FPGAs determines the status of the FPGAs on the module. The FPGA_DONE signal is connected directly to the carrier motherboard where it is then connected to the Interface FPGA. This signal has a weak pull-up applied to it on the motherboard.

When the FPGAs on the BenADDA DIME-II module are NOT configured, the base of the two transistors (MMUN221LT1) will be switched on by the 3.3V pull-up. With the base of the transistors switched on, the FPGA_DONE signal (connected to the motherboard via the DIME-II connector) will be pulled LOW through the transistor.

Once the FPGA has been configured, the user should send out a LOW signal on the appropriate CONFIG_DONE pin (i.e. a LOW would be driven out of 'IOB B' on the FPGA). This turns the base of the transistor off; and the FPGA_DONE value is now subject to the status of the complete system. Once all other FPGAs in the system are configured, the FPGA_DONE signal will be HIGH via the pull-up on the motherboard. However, if one FPGA is not configured, the FPGA_DONE signal from that device will still be pulled LOW, meaning that FPGA_DONE for the entire system would be LOW. The system designer will therefore be able to read the value of FPGA_DONE, via FUSE software, at the 'IOB A' input on the various FPGAs to determine the overall state of the system.

If the 'Config DONE' signal is to be utilized in a system, the user should ensure that IOB B is driven low once the FPGA has been successfully configured. Alternatively, if a system initialisation sequence is required then IOB B can be driven low after this. The FPGA then polls IOB A to see that all other FPGAs in the system have been configured.

Section 13

Power Specifications

In this section:

- Introduction to the Power Specification for the Kit
- Hardware
- Enabling Power Supplies at Power-On

13.1 Introduction

The XtremeDSP Development Kit Pro can be powered using either the standalone power supply provided with the Kit, **OR** a PCI Slot. This section details the overall power specification with a breakdown of power capabilities for individual power supplies. The actual power requirements of a User FPGA design vary depending upon the design in terms of clock frequency, toggle rates and device utilization.

13.1.1 External Standalone Power Supply Specification

A triple voltage +12V, -12V, +5V 45W external power supply is used. The unit is capable of the following maximums:

- 5A @ +5V
- 2A @ +12V
- 0.8A @ -12V

Note that the 1.5V and 3.3V supplies are generated from the +5V supply rail. Therefore the absolute maximums are provided in [Table 40 on page 99](#).

Component	Effective 5V power draw	Generation
Kit Board Additional Components	1.5A	Linear Regulators
Clock FPGA XC2V80-4CS144	0.5A	Linear Regulators
ZBT Components	0.7A	Linear Regulators
FPGA Sidebank I/O	0.8A	Linear Regulators
ADC/DAC Digital I/O	0.25A	Linear Regulators
Main User FPGA Core	1.25A (4A@1.5V converted from 5V to 1.5V)	via 96% efficiency DC to DC converter (PPS see later in this section)
TOTAL	5A @ 5V abs limit of external supply	

Table 40: Power Budget with External Power Supply

13.1.2 PCI Power Specification

The Kit hardware can be used in a PCI slot which means the external power supply is not used to power the board. Therefore the power specification is determined by regulators on the Kit hardware and also by the PCI specification.



If you are using a high power design you should, where possible, make use of a PCI slot due to the additional power capabilities.

The recommended specification in **Table 41 on page 100** is made for a Kit connected to a PCI slot.

Power Rail(Volts)	Current Rating(Amps)	Description
1.5	5	Virtex-II Main FPGA Core and Clock FPGA Core
3.3	1.2	Virtex-II FPGA I/O
3.3	1.2	Virtex-II FPGA I/O and ZBT devices
3.3	1.2	Spartan-II FPGA I/O
2.5	1.2	Spartan-II FPGA Core

Table 41: On board Power Supplies



The recommended power max for the Kit is 25Watts (5Volts @ 5Amps) as defined in the PCI specification. The Kit requires 5Volts and +12Volts. If you wish to exceed this level please consider the cooling requirements for the system. For more information on system cooling please see **“Cooling” on page 104**.



If the Kit is connected to PCI slot and you are developing large designs that may require high current ratings (i.e. over 25W), Nallatech recommend the use of the disk drive connector to increase the power rating. The disk drive connector can **ONLY** be used when the Kit is inserted in a PCI slot. The power source for the disk drive connector **MUST** be from the same power source as the PCI Slot.

The power requirements of the Virtex-II Pro FPGAs depend upon the density and speed of the application designs running in them. The Xilinx Power Estimator can be used to estimate the power requirements of application designs. For more information on the Xilinx Power Estimator use the following link: http://support.xilinx.com/ise/power_tools/index.htm

13.2 Hardware

The Kit has a number of devices that generate specific voltages required in the hardware. As the Kit is based on the DIME-II module standard, it follows the standard approach in DIME-II for providing voltages required by the module and also those required by the motherboard itself.

Standard DIME-II hardware makes use of modular power supplies to provide the voltages required by specific modules. There are two types of power modules: the **Fixed Power Supply (FPS)** unit and the **Programmable Power Supply (PPS)** unit. Details of the two types of modules are provided later in this section. In addition to the power modules, the Kit hardware uses a number of linear regulators to generate the voltages required by some of the devices on the motherboard itself. Figure 61 on page 101 shows the voltage generation and use in the Kit hardware.

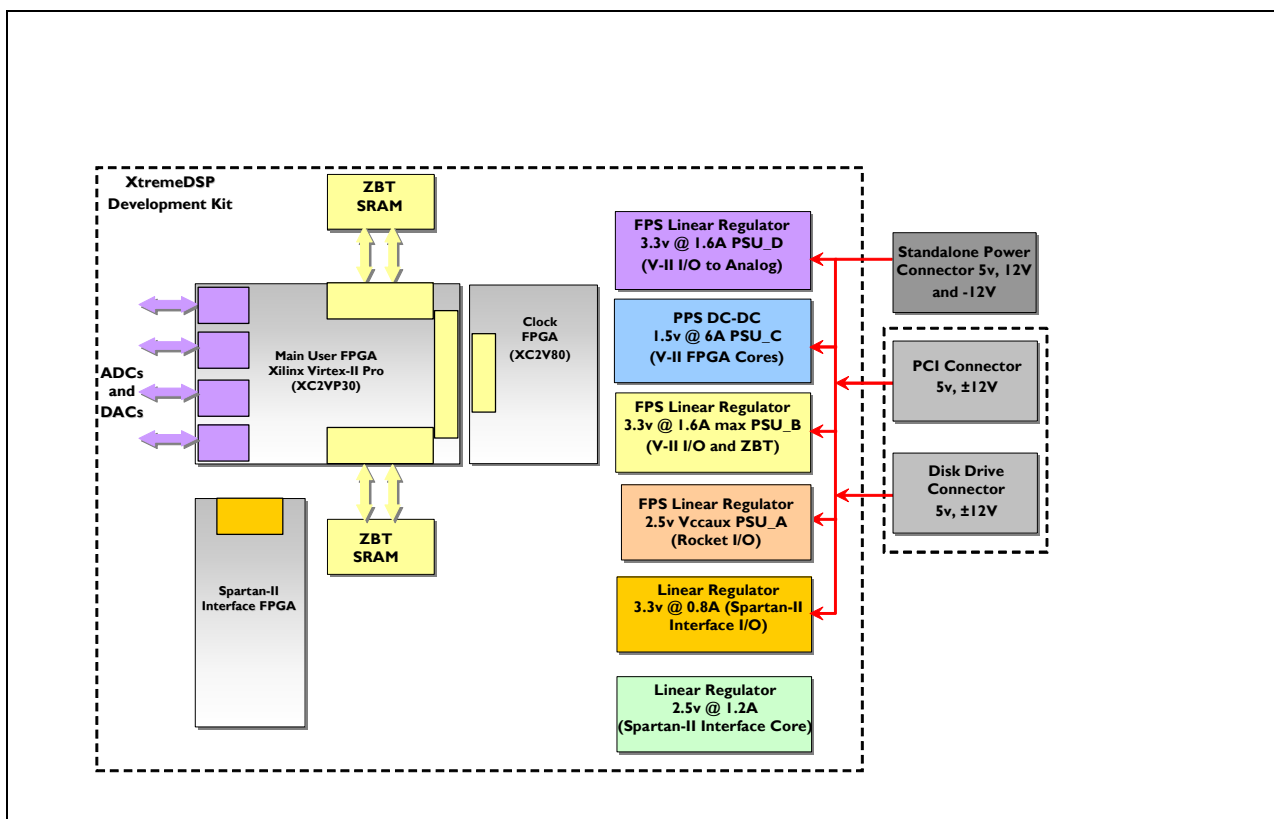


Figure 61: On board Power Set-up

There are three FPS units on the Kit hardware. The first FPS (Power Supply B - PSU_B) provides 3.3V to the side banks (i.e. the DIME-II Local Bus) of the main User FPGA and the two ZBT devices on the BenADDA DIME-II module. The FPS unit is capable of providing 1.6Amps @ 3.3V and therefore there is 800mA min available for the side bank FPGA I/O.

The second FPS (power supply D - PSU_D) is used to power the Digital I/O of the ADC and DACs and the corresponding I/O of the main User FPGA. Again this FPS is capable of 1.6Amps @ 3.3V which allows for dependent I/O. The third FPS (power supply A - PSU_A) is used to power Vccout of the Virtex-II Pro.

There is one PPS (power supply C - PSU_C) used on the Kit hardware to provide the core voltage on 1.5V for the User FPGA on the BenADDA DIME-II module. The PPS can supply up to 15Amps, but for the purpose of the Kit and due to limits of the external power supply, the recommended max is: ~4A @ 1.5V as described in the previous specifications. When using the Kit in a PCI slot the limits that apply are the 15A rating of the PPS unit and the thermal limit of the Virtex-II Pro silicon, but in practice the recommend max is 5Amps max when using the PCI slot. There are several linear regulators supplying 3.3V and 2.5V power for the interface FPGA but these are dedicated to the interface and do not affect user designs.

Modular Programmable Power Supply (PPS)

It is possible to dynamically adjust the voltages applied to the module slot, using the on board DC-DC converters. Each supply is capable of supplying voltages in the range 1V to 3.3V at a maximum current of 15A(abs. max). Note that no user intervention is required to set up the PPSs - these are automatically configured by the system.

Modular Fixed Power Supply (FPS)

Nallatech offer the fixed power supply option, which has a reduced output current capability and produces a fixed voltage output. This option reduces the flexibility of the BenONE-Kit Motherboard, as only DIME-II modules with the same power supply requirements will function. Power supplies are configured at the time of manufacture, voltages in the range of 1V to 3.3V are available with a maximum output current of 7A(abs. max). Although the unit is capable of supplying 7A(abs. max), the actual output voltage, and more importantly the drop from the input to the output, limits the operating range of the unit due to its ability to dissipate the heat generated by the conversion. The following calculations can be used to work out the max power that can be drawn.

For a FPS supplying 1.5V to the system, with an ambient temperature (T_A) of 25°C and a thermal coefficient of 35°C/W:

$$P_{MAX} = \frac{T_J - T_A}{\theta_{JA}}$$

$$\Rightarrow T_J = T_A + (\theta_{JA} \times P_{MAX}), \text{ where } \theta_{JA} = 35^\circ \text{C/W}, T_J = 125^\circ \text{C(max)}, T_A = 25^\circ \text{C}$$

$$\Rightarrow 125 = 25 + (35 \times P_{MAX})$$

$$\Rightarrow \underline{\underline{P_{MAX} = 2.857W}}$$

$$P_{MAX} = I_{out_{MAX}} \times (V_{IN} - V_{OUT})$$

$$2.857 = I_{out_{MAX}} \times (5 - 1.5)$$

$$\Rightarrow \underline{\underline{I_{out_{MAX}} = 816.33mA}}$$

For a FPS that is supplying 3.3V to the system:

$$P_{MAX} = \frac{T_J - T_A}{\theta_{JA}}$$

$$\Rightarrow T_J = T_A + (\theta_{JA} \times P_{MAX}), \text{ where } \theta_{JA} = 35^\circ \text{C/W}, T_J = 125^\circ \text{C(max)}, T_A = 25^\circ \text{C}$$

$$\Rightarrow 125 = 25 + (35 \times P_{MAX})$$

$$\Rightarrow \underline{\underline{P_{MAX} = 2.857W}}$$

$$P_{MAX} = I_{out_{MAX}} \times (V_{IN} - V_{OUT})$$

$$2.857 = I_{out_{MAX}} \times (5 - 3.3)$$

$$\Rightarrow \underline{\underline{I_{out_{MAX}} = 1.681A}}$$

For a FPS that is supplying 2.5V to the system:

$$P_{MAX} = \frac{T_J - T_A}{\theta_{JA}}$$

$$\Rightarrow T_J = T_A + (\theta_{JA} \times P_{MAX}), \text{ where } \theta_{JA} = 35^\circ \text{C/W}, T_J = 125^\circ \text{C(max)}, T_A = 25^\circ \text{C}$$

$$\Rightarrow 125 = 25 + (35 \times P_{MAX})$$

$$\Rightarrow \underline{P_{MAX} = 2.857 \text{W}}$$

$$P_{MAX} = I_{out_MAX} \times (V_{IN} - V_{OUT})$$

$$2.857 = I_{out_MAX} \times (5 - 2.5)$$

$$\Rightarrow I_{out_MAX} = 1.143 \text{A}$$

13.2.1 Power Supply LEDs

The feature section “LEDs” on page 55 provides details on the specific power supply LEDs on the card. This information is repeated here for reference only.

DIME-II Module Power Supply LEDs

The DIME-II standard requires that a number of voltages be supplied to a DIME-II module from the DIME-II motherboard to which it is fitted. In the case of the XtremeDSP Development Kit Pro, four power supplies provide power to the module. For each of these power supplies a corresponding LED shows when each power supply is turned on. These power supplies are termed PSUA to PSUD. The actual voltage generated for each is given in Table 42 on page 103.

Silk Screen LED Identifier	Color State	State Description	State Voltage Output	Initial power on state	After card opened in FUSE
D7	Red	PSUA Off	0V	Off i.e. RED	Off i.e. GREEN
	Green	PSUA On	2.5V		
D8	Red	PSUB Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUB On	3.3V		
D12	Red	PSUC Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUC On	1.5V		
D13	Red	PSUD Off	0V	Off i.e. RED	On i.e. GREEN
	Green	PSUD On	3.3V		

Table 42: DIME-II Module Power Supply LED States

Motherboard Main Power LEDs

In addition to the power supplies for the module there are several LEDs which indicate the status of the main power supplies for the motherboard itself. Table 43 on page 103 defines their use.

LED	Purpose	General Operation State
D10	2.5V power indicator	GREEN
D14	3.3V power indicator	GREEN

Table 43: Motherboard Main Power LEDs

LED	Purpose	General Operation State
D15	5V power indicator	GREEN

Table 43: Motherboard Main Power LEDs

13.2.2 Cooling

Main FPGA

A single fan is attached to the main User FPGA. [Figure 62 on page 105](#) shows the range of experimental results for temperature vs. power for the XC2VP30-4FF1152 FPGA when fitted with a fan. The limits of the cooling depend on whether the board is used inside the blue board case with the lid fitted, used with the lid removed or used with the board fitted in a PCI slot.



Where possible it is recommended that the temperature measurement facilities described in [“Temperature Monitoring” on page 86](#) are used in high power designs to ensure the main FPGA operates within its allowable temperature range. In addition to this, Xilinx provide the XPower tool which is a useful way of approximating the power requirements of designs.

Limits when Used in the Board Case with the Lid Fitted

The default setup is to use the hardware fitted inside the blue board case. In this configuration there is slightly reduced airflow. Therefore, when the lid is fitted, it is recommended that designs use up to 2.4 Amps on the 1.5V core FPGA supply. The 2.4 Amp limit will cover almost all designs - for example a design that uses 100% of the flip-flops in the with a 50% toggle rate draws 1.893 Amps when run at 100MHz.



Beyond the 2.4 Amps you should remove the lid from the board case to increase the airflow and allow the FPGA to operate up to the limits documented in [page 100](#).

Limits with Lid Removed or when Fitted in a PCI Slot

The power and current readings are for the 1.5V supply only i.e. the FPGA core power supply. The results (shown in [Figure 62 on page 105](#)) indicate that, even when the Kit is fitted to a PCI slot, the thermal limits of the Virtex-II Pro are a key factor. The figures also indicate that the max draw on 1.5V is 8W, or 5.3 Amps. The recommended limit is therefore 5 Amps when using PCI.

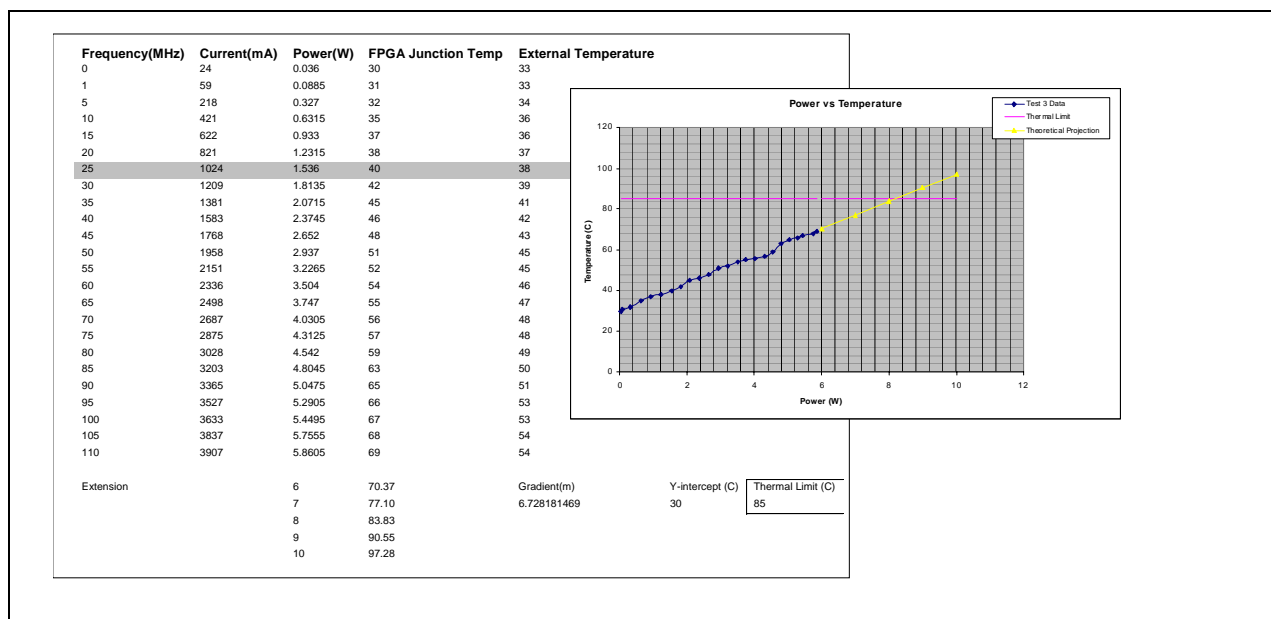


Figure 62: Power vs. Temperature with Lid Off Case or in PCI slot



The recommended power max for the Kit is 25Watts (5Volts @ 5 Amps) as defined in the PCI specification. The Kit requires 5Volts and +12Volts. If you wish to exceed this level please consider the cooling requirements for the system.

Table 44 on page 105 lists all the necessary jumper configurations for each fan. Please note that each fan jumper is a 2-pin header - one pin supplies 5 volts to the fan and the other provides a ground. One of these headers is fitted with a lockable fan connector whilst the other is simply a 2 pin 0.1" pitch header. This is intended to provide flexibility for the user to fit other fans, if required, for their own applications. The silkscreen on the PCB for each fan jumper is a white box with a corner missing. Pin 1 is always nearest the missing corner.



Fan Jumper Name	Description
J5 	Supplies power to a 5 Volt Cooling Fan Pin 1: Ground Pin 2: + 5 Volts
J6 	Supplies power to a 5 Volt Cooling Fan Pin 1: Ground Pin 2: + 5 Volts

Table 44: Fan Jumpers

Note that one of these connectors is already allocated to the fan fitted above the main User FPGA.

ADC Heatsinks

Each ADC device (AD6645) is fitted with a copper heatsink that has an optimal pin configuration for low airflow situations.

13.2.3 External Connectors

Standalone Power Connector

This is a standard 8 way mini DIN connector.

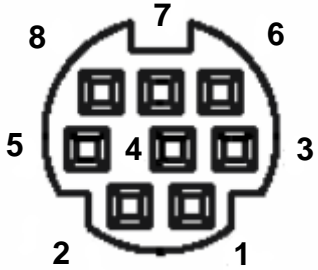
Supply Voltage	Pin#	Looking at connector on BenONE-Kit Motherboard
+5V	1	
+5V	2	
+5V	3	
+12V	4	
-12V	5	
RETURN	6	
RETURN	7	
RETURN	8	

Table 45: Mini DIN Pinout

Additional Power Connector (J20)

If you are developing large designs with the board plugged in via a PCI connection, you may require high current ratings. Nallatech therefore advise the use of this additional power input connector to increase the power rating. You will see however that this connector is limited to supplying only the positive supplies.

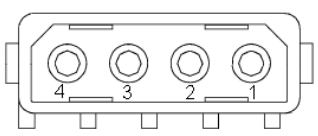
Supply Voltage	Pin#	Looking at connector
+12V	1	
RETURN	2	
RETURN	3	
+5V	4	

Table 46: Additional Power Connector (Disk Drive Style)



If the Kit is connected to a PCI slot this connector, if used, must be connected to the same ATX power supply that is powering the slot. Alternatively if the Kit is operating as a standalone unit this can be connected to any external source. However, ensure that if a power supply is also connected to the Mini DIN connector it is not attempting to power the +5V and +12V rails.

13.3 Enabling Power Supplies at Power-On (using External Power Supply and JTAG connectors)

The USB firmware has the option of automatically turning on the power supplies to the module when the main power is applied from the external power supply. To enable this feature, fit a jumper (default) on J17. This is useful when working with tools such as System Generator or Impact and you do not wish to open the card via USB to enable the power supplies to complete the JTAG chain.

Section 14

Environmental Specifications

In this section:

- Outline of the Environmental Specification of the Kit
-

14.1 Specifications

14.1.1 Operating Temperatures

The Kit has been tested under operation within the following temperature ranges: 0 to +30°C

14.1.2 Storage Temperatures

The Kit may be stored within the following temperature ranges: -20°C to +80°C

14.1.3 Relative Humidity

The Kit has been tested in the following conditions: 20 to 95% (non-condensing)



NALLATECH
The High Performance FPGA Solutions Company

Part III: System Level Design

This part of the User Guide provides information on how to create a complete system design based on the XtremeDSP Development Kit Pro, and includes details on design partitioning, inter-FPGA communications and building a host interface.

Section 15

Building a Host Interface

In this section:

- Creating a Hardware/Software interface for Designs on the Kit.
- Aspects of Hardware, Firmware and Software Design

15.1 Introduction

The Kit often requires an interface to be built between a design running in the FPGA and the PC host. This section provides details on how such an interface can be created. It provides an introduction to a core called the 'Interface FPGA to User FPGA' that aids in building the link between the hardware and software.

15.2 Hardware

15.2.1 Design Partitioning

The Kit allows the user to partition the functionality of their application between software and hardware easily and effectively. The design partitioning of the Kit is shown in [Figure 63 on page 111](#).

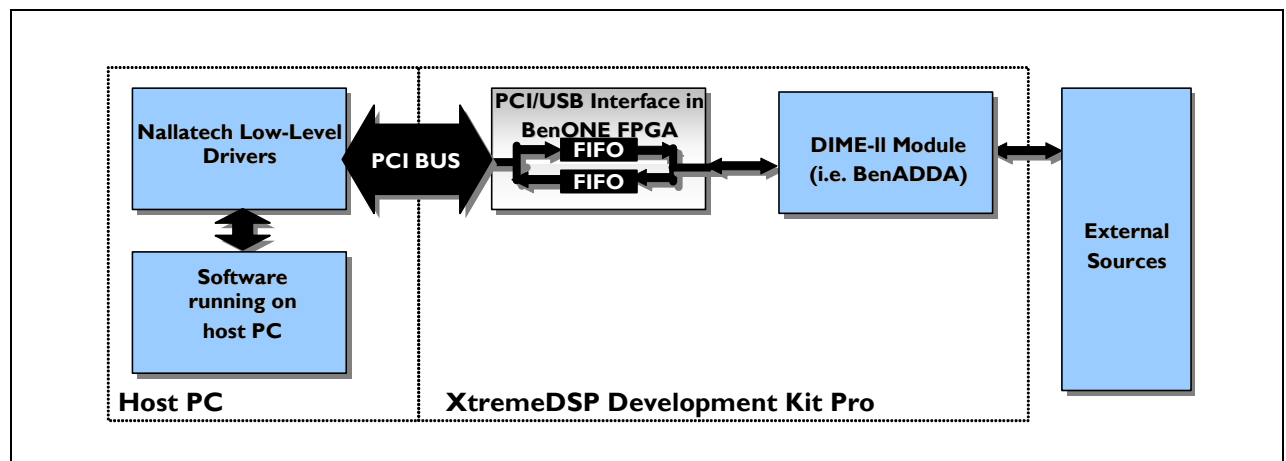


Figure 63: Design Partitioning

The green blocks do not require any further design from a user perspective. The interface is pre-configured with either PCI or USB and the external sources are assumed to be in place. The user needs to design application designs for any FPGAs on the hosted DIME-II module - the BenADDA in this case. The software running on the host PC is available as a pre-designed GUI. For users who require additional functionality and wish to have their own software front-end, the FUSE Software Library provides functions for use in application programs. These facilitate functionality such as FPGA configuration/reset, clock speed setting, data transfer.

In order to make use of these software-interfacing functions for specific data transfer to their designs, certain signals must be connected into the design. These signals are listed in the 'Interface COMM Signal' column in [Table 47 on page 112](#) and [Table 48 on page 113](#). A core for use in the User FPGA is provided to aid the process of integration and communication with these signals.

For further details please see the *PCI to User FPGA Interface Application Note* provided on the FUSE CD at the location 'CD ROM Drive\Application Notes\NT302-0000 Spartan to Virtex Interface\'

Interface COMM Signal	General Bus Signal Name	DIME-II Connector PIN No	User FPGA(XC2VP30-4FF1152)Pin No
ADIO<0>	LBUS<0>	PB1	F4
ADIO<1>	LBUS<1>	PB2	H2
ADIO<2>	LBUS<2>	PB3	H1
ADIO<3>	LBUS<3>	PB4	K5
ADIO<4>	LBUS<4>	PB6	K4
ADIO<5>	LBUS<5>	PB7	J2
ADIO<6>	LBUS<6>	PB8	K2
ADIO<7>	LBUS<7>	PB9	K1
ADIO<8>	LBUS<8>	PB10	L8
ADIO<9>	LBUS<9>	PB11	L7
ADIO<10>	LBUS<10>	PB12	L6
ADIO<11>	LBUS<11>	PB13	M10
ADIO<12>	LBUS<12>	PB15	N10
ADIO<13>	LBUS<13>	PB16	M9
ADIO<14>	LBUS<14>	PB17	N9
ADIO<15>	LBUS<15>	PB18	N8
ADIO<16>	LBUS<16>	PB19	M7
ADIO<17>	LBUS<17>	PB20	N7
ADIO<18>	LBUS<18>	PB21	M6
ADIO<19>	LBUS<19>	PB22	L3
ADIO<20>	LBUS<20>	PB24	L2
ADIO<21>	LBUS<21>	PB25	L1
ADIO<22>	LBUS<22>	PB26	L5
ADIO<23>	LBUS<23>	PB27	N4
ADIO<24>	LBUS<24>	PB28	L4
ADIO<25>	LBUS<25>	PB29	M2
ADIO<26>	LBUS<26>	PB30	P10
ADIO<27>	LBUS<27>	PB31	P9
ADIO<28>	LBUS<28>	PB33	M4
ADIO<29>	LBUS<29>	PB34	N6
ADIO<30>	LBUS<30>	PB35	N5

Table 47: Interface to User FPGA Comms Signals

Interface COMM Signal	General Bus Signal Name	DIME-II Connector PIN No	User FPGA(XC2VP30-4FF1152)Pin No
ADIO<31>	LBUS<31>	PB36	P2
BUSY	ADJOUT<0>	PD29	U28
EMPTY	ADJOUT<1>	PD30	U27
RDI_WR	ADJOUT<2>	PD31	V33
AS_DSI	ADJOUT<3>	PD32	U33
RENI_WENI	ADJOUT<4>	PD33	U32
INTI	ADJOUT<5>	PD34	U31
RSTI	ADJOUT<6>	PD35	V29

Table 47: Interface to User FPGA Comms Signals


Also DSP_CLK should be connected to CLKI (sometimes referred to as CLKB).

Interface COMMSignal	General Bus Signal Name	DIME-II Connector PIN No	User FPGA(XC2VP30-4FF1152)Pin No
DSP_CLK	CLKI	PC31	AJ17

Table 48: Interface to User FPGA Clock Requirements

15.2.2 Interface Communications Bus

The Interface Communications Bus is an important communications channel, as it provides a path for data communication between the User FPGA, the Interface (PCI or USB) FPGA and onto the host PC.

This bus has a pre-defined communications protocol to facilitate communications to the Interface FPGA. In order to communicate with the Interface FPGA from the User FPGA, the User FPGA application design must incorporate a mechanism to communicate over this bus. This communications mechanism can be implemented directly by the user in the design, or by using Nallatech's drop-in IP core - the PCI to User FPGA Interface Core. This core implements the Interface to User FPGA Comms communications mechanism and offers the user a simplified interface to which they can connect their own designs.

15.3 Firmware

15.3.1 Interface FPGA to User FPGA Interface Core

The Interface to User FPGA Interface Core is a drop-in IP core, which can be incorporated into the User FPGA Application Design. This core implements a mechanism, which deals with the protocol to communicate over the Interface Communications Bus. This abstracts the complexities of the protocol and provides a simplified user interface, offering a memory-mapped address space for registers/peripherals and DMA channels for high-speed data transfer. A block diagram for the implementation of this core is shown in [Figure 64 on page 114](#).

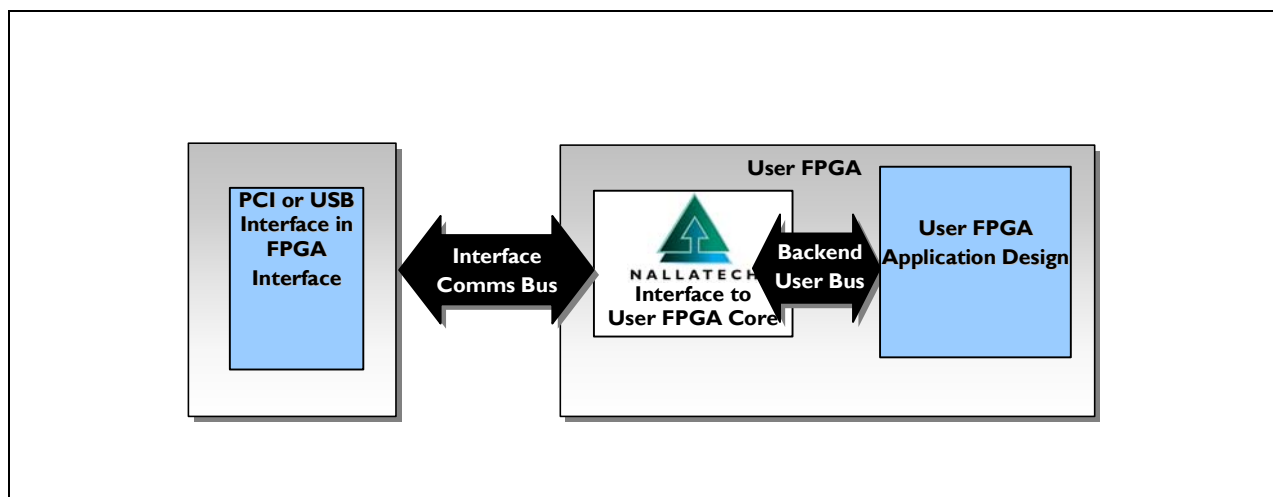


Figure 64: Implementation with PCI or USB to User FPGA Interface Core

For full details of the Interface to User FPGA Interface Core, please refer again to *PCI to User FPGA Interface Application Note* provided on the FUSE CD at the location 'CD ROM Drive\Application Notes\NT302-0000 Spartan to Virtex Interface\'. The necessary VHDL code and an EDIF file for the core are also provided on the CD. Please note that this is a generic Application Note whether PCI or USB interfacing is used.

15.3.2 Implementing the Communications Mechanism

Instead of communicating over the Interface Comms bus using the Interface to User FPGA Interface Core, a mechanism to communicate over this bus using the appropriate protocol can be implemented directly by the user in their design. A block diagram for the implementation of this core is shown in [Figure 65 on page 114](#).

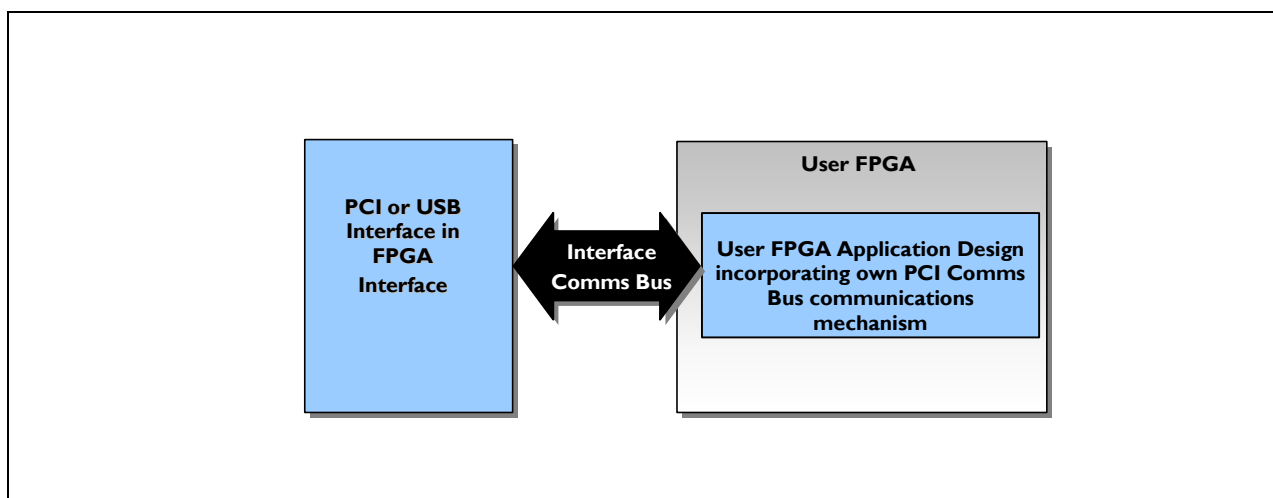


Figure 65: Implementation with Own Communications Mechanism

15.3.3 Communications Bus Protocol

Data is transferred between the User FPGA and the Interface FPGA with the use of 5 control signals. Three of these signals are driven by the PCI or USB interface. These are AS/DSL, EMPTY and BUSY.

AS/DSI: This is an address strobe/data strobe signal. When this is HIGH, the data being transferred to the User FPGA is an address and when this signal is LOW, the data being transferred to the User FPGA is data from/to the last address given. Generally, addresses are only sent from the PCI FPGA to the User FPGA.

This signal is always in sync with the data passed through the internal FIFOs of the PCI FPGA. The internal FIFO can have a mixture of actual data and addresses and this AS/DS# line will automatically indicate the true type of data.

EMPTY: This signal indicates that there is data waiting to be written from the PCI or USB FPGA interface to the User FPGA. This signal will go HIGH when there is no more data to be written to the User FPGA.

BUSY: This signal indicates that the PCI or USB FPGA interface can receive data from the User FPGA. When this signal goes HIGH, no more data should be written to the Interface FPGA.

The User FPGA drives the two remaining control signals. These are RDL_WR and RENL_WENL.

RENI_WENI: This signal is a read/write enable signal. When this signal is LOW, if the R#/W signal is HIGH, data is on the bus ready to be written to the PCI FPGA. If the signal is LOW and R#/W is LOW, then data will be driven onto the data bus from the PCI FPGA on the next clock edge. When this signal is HIGH, there should be no data on the bus.

ADIO: This is the data bus that is used to transfer data between the PCI FPGA and the User FPGA. It is a 32-bit bidirectional bus that can be driven by both the PCI FPGA and the User FPGA.

The general functionality is similar to that of FIFOs. The EMPTY and BUSY signals act similarly to FIFO_EMPTY and FIFO_FULL signals. The RDI_WR and RENI_WENI signals combine to give the RENI and WENI signals of a FIFO.

RDI_WR: This signal determines the direction of the data transferred between the Interface FPGA and the User FPGA. If this signal is LOW, data is being read from the Interface FPGA and so the Interface FPGA drives the data bus. If the signal is HIGH, data is being written to the Interface FPGA and so the User FPGA drives the data bus.

The clock used for the Interface FPGA to User FPGA communications is always DSPCLK.

Reading from Interface FPGA to User FPGA

Reading from the Interface FPGA is similar to reading from a FIFO. The EMPTY signal goes LOW to indicate that there is data to be read. The FIFO, whose data is read from on the Interface FPGA is a First-Word-Fall-Through with a latency of one clock cycle. When reading from the Interface FPGA the user must ensure that the read enable is not active until at least one clock after the EMPTY signal goes LOW. The read enable should go inactive immediately after the EMPTY signal goes HIGH, although no data will be read if EMPTY is HIGH and the read enable is active.

The diagram in [Figure 66 on page 116](#) shows a functional representation of data reads in operation. These are single word transfers.

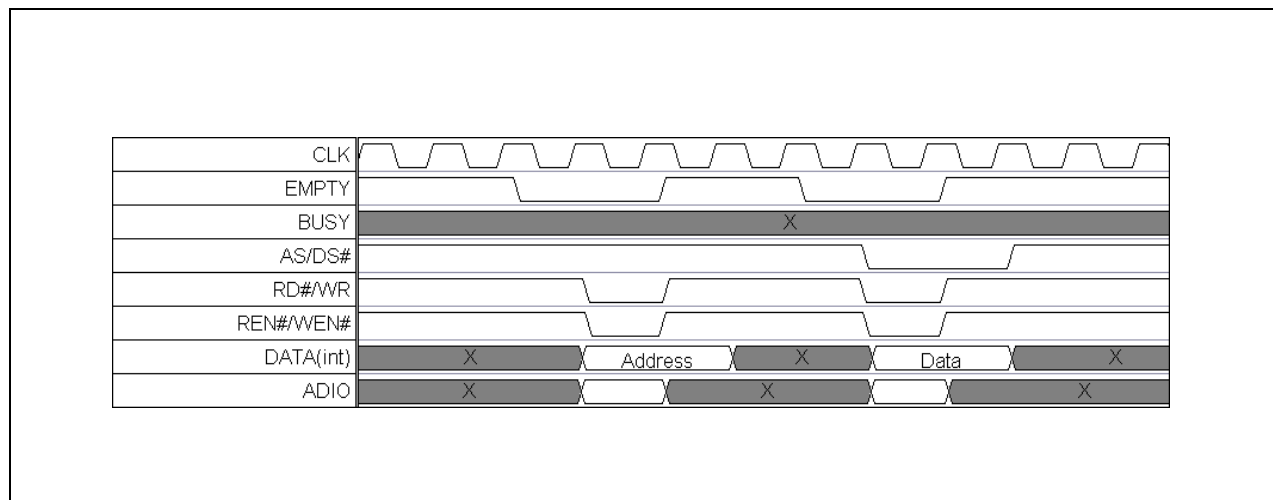


Figure 66: Single Read Transfer

The first read sends an address from the Interface FPGA to the User FPGA whilst the second read sends the data across. The DATA(int) bus shows the internal data waiting to be driven onto the ADIO bus. The diagram in [Figure 67 on page 116](#) shows a burst-read in operation.



Figure 67: Burst Read Transfer

This example shows a single address being sent from the Interface FPGA to the User FPGA followed by a burst of data. Again the DATA(int) bus shows the internal data waiting to be driven onto the ADIO bus. It should be noted that after the first read, the next set of data will not be available until one clock cycle after the read. So long as the reads are continuous, data will then follow every clock cycle.

Writing to Interface FPGA from User FPGA

Writing to the Interface FPGA is similar to writing to a FIFO. If the Interface FPGA can receive more data, the BUSY signal is LOW and when the Interface FPGA cannot receive any more data, the BUSY signal is HIGH. To help meet timing specifications the User FPGA application is allowed to over-run by two further data samples. In other words, after the BUSY signal is asserted two further data samples can be written to the Interface FPGA.

The diagram in [Figure 68 on page 117](#) shows a Burst-Write function in operation and also demonstrates the maximum data over-run of 2.

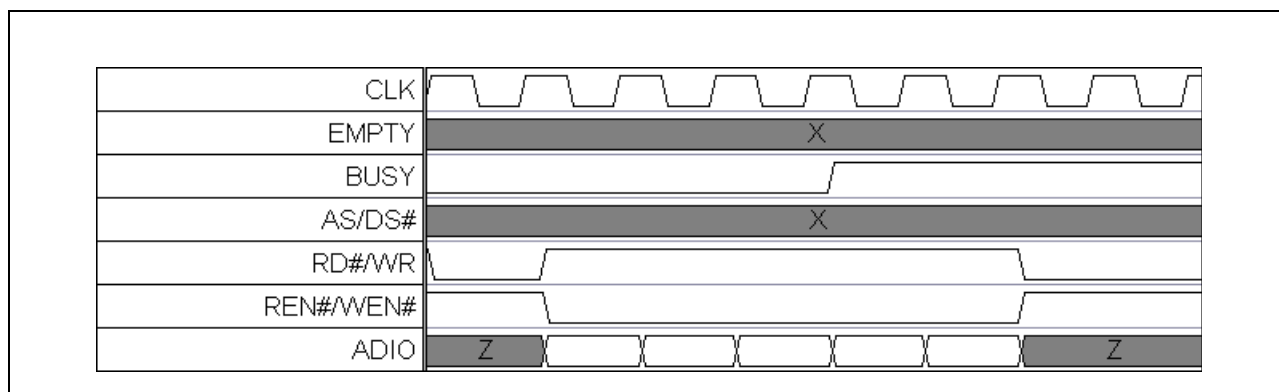


Figure 68: Burst Write Transfer

Timing Information

The information in [Table 49 on page 117](#) provides the timing information required to write the *User Constraints File (UCF)* for implementing a design into an FPGA. These are recommended constraints.

Signal	In/Out	Details
EMPTY	In	9ns before clock
BUSY	In	8ns before clock
AS_DSI	In	18ns after clock
RDI_WR	Out	11ns after clock
RENI_WENI	Out	11ns after clock
ADIO	In	5ns before clock
ADIO	Out	12.5ns after clock

Table 49: Timing Information

Timing specifications should be written into the UCF as a NET <name> OFFSET = <spec> specification (in other words NET EMPTY OFFSET = 9ns BEFORE CLK;).

Other signals

There are two other signals between the User FPGA and the Interface FPGA. These are RSTI and INTI.

RSTI: This signal is driven by the PCI or USB FPGA Interface and can be used as a global reset within the User FPGA device.

INTI: This signal is driven by the User FPGA and can be used to signal an interrupt to the Interface FPGA and cause a PCI interrupt.

15.3.4 Further Information on Interface Core

Further information on the interface core is available in the following application notes:

- *PCI to User FPGA Interface Application Note* available on the FUSE CD.
- *XtremeDSP Kit Ping Example Application Note* available on the XtremeDSP Development Kit Pro CD.



Please note that the following software section may provide more up to date support library information than the PCI to User FPGA Interface Application Note for the core.

15.4 Software

15.4.1 FUSE API Interlinks

The C/C++ FUSE API provides a number of low level functions for sending and receiving data across the PCI or USB interface. In order to simplify the software interface when using the Interface FPGA to User FPGA interface core an additional library has been created called 'vidime'. This is provided as a header file 'vidime.h' with associated library files ('vidime.lib' is the COFF format library suitable for Microsoft Visual Studio and other tools that use the COFF format. 'vidimomf.lib' is the OMF equivalent library for use with tools such as Borland that use the OMF object model format.) All these files are located in the 'include' sub-folder where FUSE was installed.

```
typedef void * VIDIME_HANDLE;

CMANGLE NALLAEXPORT DWORD viDIMEError;
CMANGLE NALLAEXPORT char viDIMEErrorText[200];

#define viDMACSRREG      0
#define viDMACOUNTREG   1

#define viDMAISABLE      0
#define viDMAENABLE      1
#define viDMAREADDIRECTION 0x2
#define viDMAWRITEDIRECTION 0

CMANGLE NALLAEXPORT VIDIME_HANDLE viDIME_Open(DIME_HANDLE DHandle, DWORD flags);
CMANGLE NALLAEXPORT void viDIME_Close(VIDIME_HANDLE handle);
CMANGLE NALLAEXPORT DWORD viDIME_WriteRegister(VIDIME_HANDLE handle, DWORD Address, DWORD Data, DWORD Timeout);
CMANGLE NALLAEXPORT DWORD viDIME_ReadRegister(VIDIME_HANDLE handle, DWORD Address, DWORD Timeout);
CMANGLE NALLAEXPORT DWORD viDIME_DMAAbort(VIDIME_HANDLE handle);
CMANGLE NALLAEXPORT DWORD viDIME_DMARead(VIDIME_HANDLE handle, DWORD *Data, DWORD WordCount, DWORD DMAChannel,
volatile DWORD *Terminate, DWORD *Currcount, DWORD Timeout);
CMANGLE NALLAEXPORT DWORD viDIME_DMAWrite(VIDIME_HANDLE handle, DWORD *Data, DWORD WordCount, DWORD DMAChannel,
volatile DWORD *Terminate, DWORD *Currcount, DWORD Timeout);
CMANGLE NALLAEXPORT DWORD viDIME_GetVersionNumber(void);
```

Figure 69: Section of vidime.h

After you have obtained a handle to the card, the **FUSE API DIME_LocateCard** and **DIME_OpenCard** functions, allow access to the interface in the design. Full details of these functions are available in the *FUSE C/C++ API Developers Guide* included on the FUSE CD.

▼ To use the FUSE API to send and receive data across the PCI or USB interface:

1. Obtain a 'Locate Handle' to the detected hardware in the system using the DIME_LocateCard Function.
2. Obtain a 'Card Handle' for a specific card in the system using the DIME_OpenCard function. Use the 'Locate Handle' as a parameter.
3. Then use the Card Handle with the functions to control the resets, clocks and configure the FPGAs.

4. Now obtain a handle for talking to the vidime interface. To do this use the 'viDIME_Open' function. This function locks down required memory and other initialization functions for the other functions in the vidime.h library.
5. Now you can use this handle to the vidime interface in the other functions such as 'viDIME_DMAWrite' and 'viDIME_DMARead'.
6. Once finished close the handle to the vidime interface using viDIME_Close. This frees up memory allocated.
7. Finally, close the card handle and locate handles using the 'DIME_CloseCard' and 'DIME_CloseLocate' functions.

An example of this is provided on the XtremeDSP Development Kit Pro CD at the following location: 'CD ROM Drive\Examples\host_interface_basic'. The main file '**OpeningASingleCard.c**' is in the subfolder '**CCodeTester**'. The actual VHDL source for this example is also included.

15.4.2 Interface Performance

Interface performance is largely affected by the type of interface used - USB or PCI - and PC performance. This section provides performance figures recorded on a standard Nallatech PC configuration meeting minimum requirements for the XtremeDSP Development Kit Pro.

USB 1.1 Interface

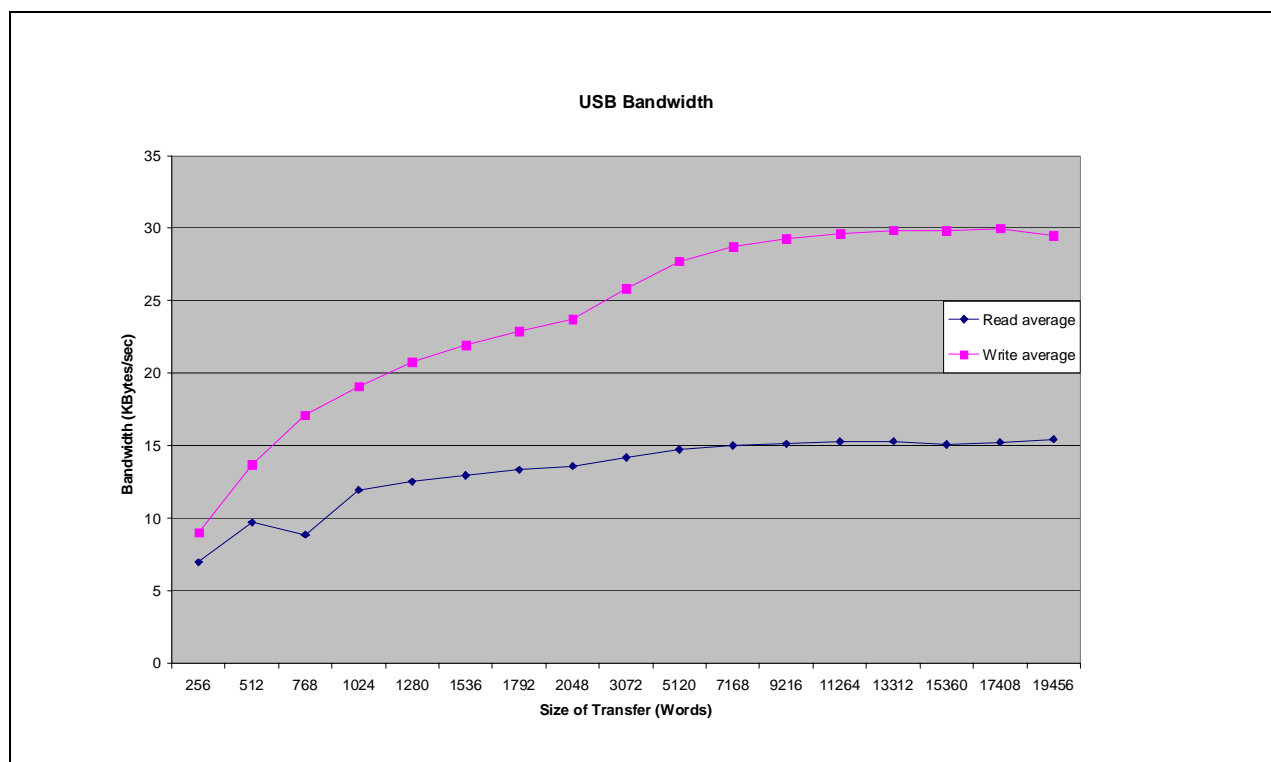


Figure 70: USB Interface Performance

PCI Interface

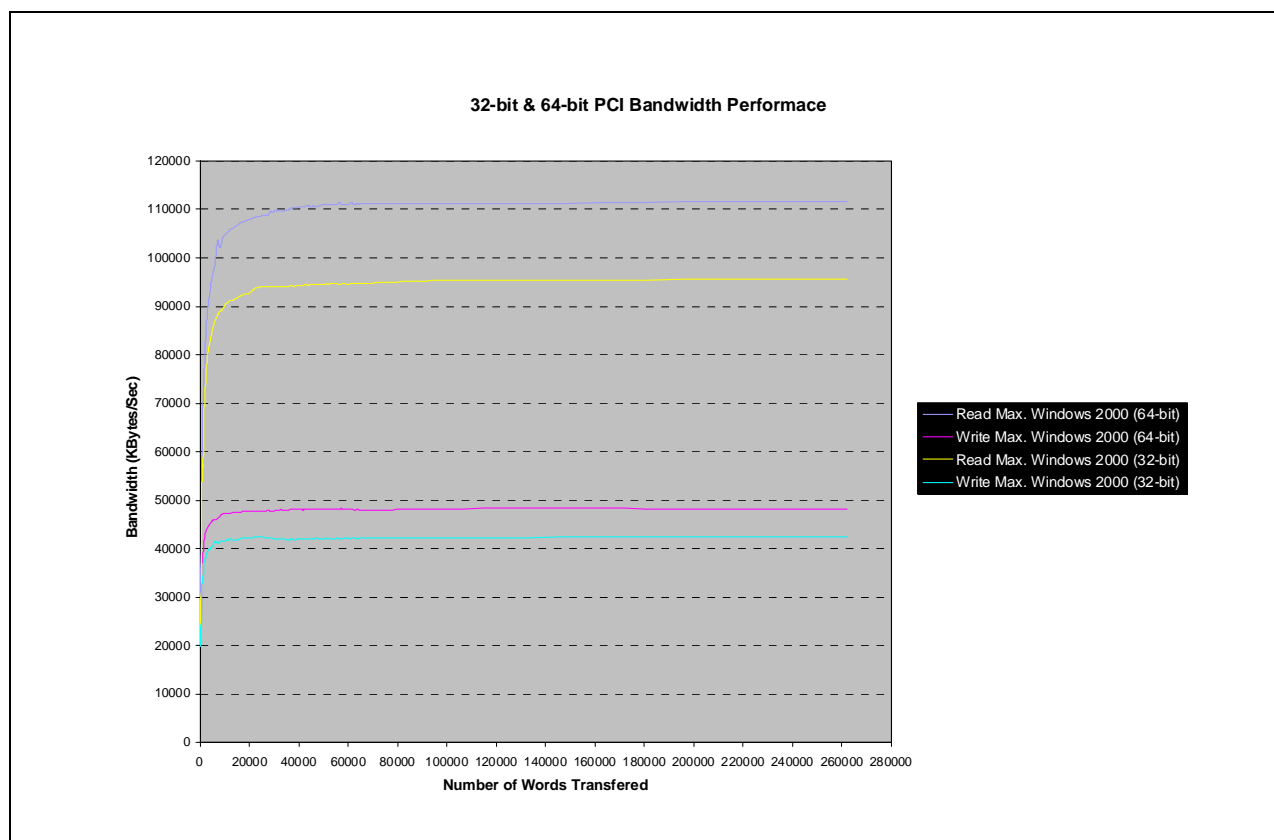


Figure 71: PCI Interface Performance



Please note that the 32-bit PCI interface is supplied in the Kit.



NALLATECH
The High Performance FPGA Solutions Company

Part IV:FPGA Configuration

This part of the User Guide provides information for configuration of the FPGAs in the XtremeDSP Development Kit Pro.

Section 16

FPGA Configuration

In this section:

- FPGA Configuration Options
 - Key Steps for FPGA Configuration
 - FPGA configuration using FUSE Probe Tool
 - FPGA configuration using FUSE APIs
 - FPGA configuration using DIMEScript
 - FPGA configuration using General JTAG Chain
-

16.1 Configuration Options

The User FPGAs (XC2V80-4CS144 and XC2VP30-4FF1152) in the Kit can be configured via a variety of methods:

- Using the FUSE Probe Tool
- Using DIMEScript
- Using FUSE Software APIs
- Using an external JTAG programmer, via the General JTAG chain and pin header

16.1.1 Module and Device Numbering within FUSE

When configuring FPGAs using FUSE, it is necessary to use Module ID and Device ID numbers to target the correct FPGA for configuration, as the Kit has multiple FPGAs. The only exception to this is when using the FUSE Probe Tool, where the software allows FPGAs to be targeted graphically.

Each FPGA must be uniquely identified, in order that the user can target the correct bitfile to each device. The system used for identifying FPGAs (and also any PROMs in the general JTAG chain) consists of a module ID, which identifies the module the FPGA is on and a device ID, which identifies the device within that module.

The module ID and device ID are determined by the order in which the devices are detected and how many modules and devices are present. The module ID always starts at 0 and increments for each additional module to be configured, therefore the maximum module ID depends upon the number of modules fitted. In the case of the XtremeDSP Development Kit Pro there are effectively two modules. Module 0 is the BenADDA DIME-II module and Module 1 consists of the other components in the JTAG chain that reside on the BenONE-Kit Motherboard.

The device ID works in a similar way to the module ID - it always starts at 0 and increments for each additional device to be configured within a module or virtual module, so the maximum device ID depends upon the number of device present. The numbering for the hardware as supplied in the Kit is shown in [Figure 72 on page 124](#).

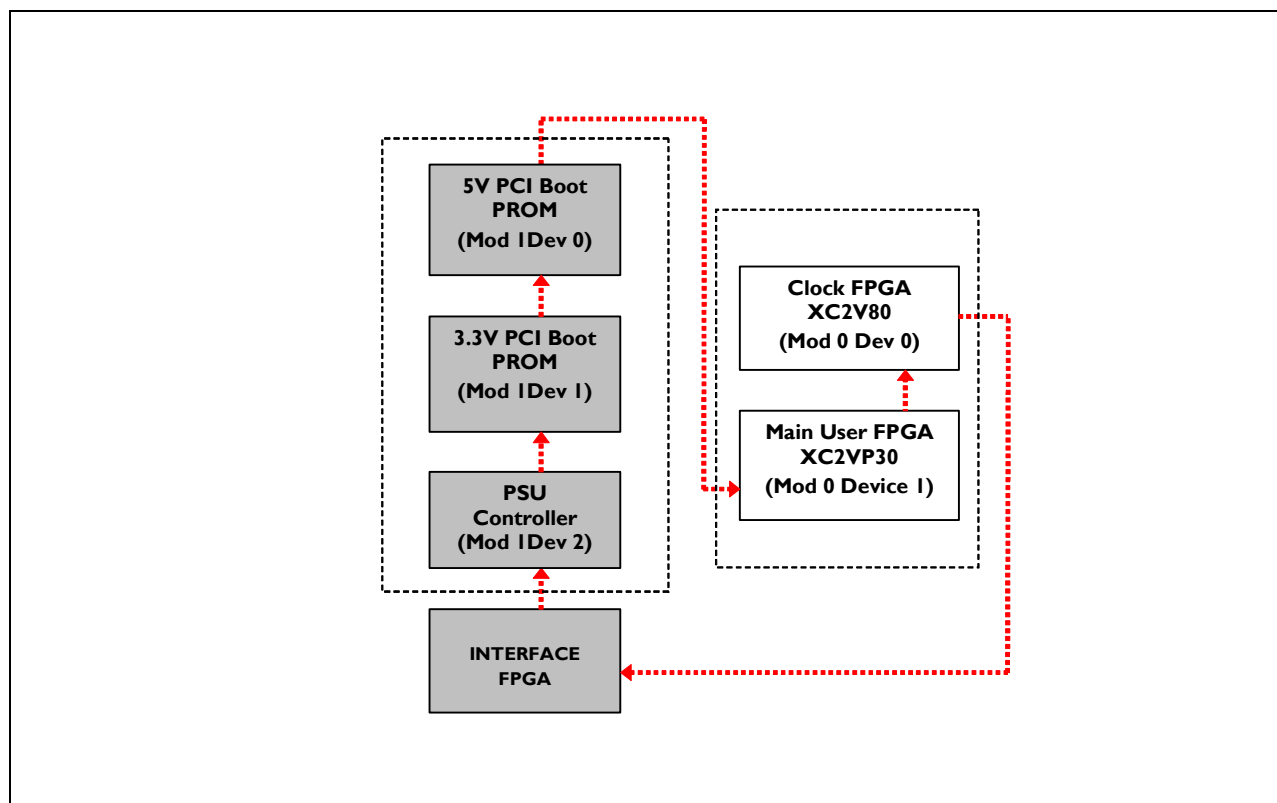


Figure 72: Module and Device Numbering in the XtremeDSP Development Kit Pro

16.2 Key Steps for FPGA Configuration

16.2.1 Using the FUSE APIs or DIMEScript

When developing applications incorporating FPGA configuration using FUSE APIs or DIMEScript, there are a number of key steps that need to be performed. As an example, the FUSE C/C++ API is used to show these key steps, although the principle is the same for any of the FUSE APIs, or DIMEScript.

To communicate with the Kit, the software must first find the card in the host system. This can be achieved by using the **DIME_LocateCard**(int LocateType, DWORD MBType, void* LocateTypeArgs, DWORD DriverVersion, DWORD Flags) function. Various arguments are required by the function to locate the card. For example, LocateType and MBType respectively determine which interface and motherboard type should be searched for.

Once the software has found the Kit, the next step is to open the card using **DIME_OpenCard**(LOCATE_HANDLE LocateHandle, int CardNumber, DWORD Flags). This is required to open the card and perform all the necessary set-up procedures in order to interface to the BenONE-Kit Motherboard. This function requires several arguments to open the card: LocateHandle is the handle returned by the DIME_LocateCard function, CardNumber is the index of the card within the locate handle that the user wishes to open, while Flags is a parameter which allows users to customize the card opening process. The DIME_OpenCard handle is passed to later functions in order to allow these functions to communicate with the card.

Once the board is open, a number of software functions can then be called to perform operations on the re-programmable devices in the chain. This stage covers functions such as configuration of individual devices, setting bitfile names, resetting FPGA devices and other functions.

Once the application has finished using the Kit hardware, the handle returned from DIME_OpenCard should be closed in order to free all the resources used to interface to the card on the host system. This can be accomplished by using

the **DIME_CloseCard**(DIME_HANDLE CardHandle) function. Additionally, the handle returned from **DIME_LocateCard** should also be closed. This can be achieved using the **DIME_CloseLocate**(LOCATE_HANDLE LocateHandle) function.

Please consult the *FUSE C-C ++ API Developers Guide* on the FUSE CD for further details on all the available DIME software functions.

16.2.2 Access through JTAG Headers

When using JTAG headers it is important to understand the operation of power supplies for the DIME-II module used in the Kit. When the hardware is initially powered on the power supplies (PSU_A to PSU_D) are disabled. This allows the DIME-II hardware to check the power supply levels requested by the module to see if they can be supplied by the motherboard, prior to switching the power supplies on. The easiest way to enable the power supplies to the module is to open the card within FUSE. This can be in the FUSE Probe Tool, one of the FUSE APIs or DIMEscript. The power supplies will remain on after the card is closed down in FUSE.



If you are using a JTAG header then these module power supplies need to be enabled in order to access all the devices in the JTAG chain. Without this the JTAG chain will not be complete and you will receive errors.

16.3 FPGA Configuration using the FUSE Probe Tool

The FUSE Probe Tool is an easy to use software interface, which allows users to access a subset of the functionality provided by FUSE.

- Full instructions on how to use the tool are provided in the *FUSE System Software User Guide* on the FUSE CD.

16.4 FPGA Configuration using the FUSE APIs

The FUSE Software development API enables users to call functions to control DIME hardware in their own programs. This allows users to develop software applications to complement the FPGA application designs running on DIME hardware. An example of FPGA configuration using the FUSE API is provided in [Figure 73 on page 126](#).

- FUSE APIs are available to support a number of development languages, including C, C++, MATLAB and Java. Full instructions on how to use the APIs are provided in the relevant *FUSE Developers Guide* on the FUSE CD.

```

#include "dimesdl.h" //contains the API functions

//Declare variables
DWORD Status1, Status2, Status3, Status4;
DIME_HANDLE hBenONE;
LOCATE_HANDLE hLocate;

if ((hLocate =
DIME_LocateCard(dlUSB,mbtTHEBENONE,NULL,dldrDEFAULT,dlDEFAULT))==NULL)
{
    //Error: Could not locate the BenONE - PCI
    return (1);    //Exit the app
}

if ((hBenONE=DIME_OpenCard(hLocate,1,dccOPEN_DEFAULT))==NULL)
{
    //Error: Could not open the BenONE - PCI
    return (1);    //Exit the app
}

//Boot the clock FPGA (device 0) on Module 0with the bit file "BitfileC.bit"
Status1 = DIME_ConfigDevice(hBenONE,"BitfileC.bit",0,0,NULL,0);

//Boot the main user FPGA (device 1) on Module 0 with the bit file "Bitfile.bit"
Status2 = DIME_ConfigDevice(hBenONE,"Bitfile.bit",0,1,NULL,0);

DIME_CloseCard(hBenONE);
DIME_CloseLocate(hLocate);

```

Figure 73: Example C code to configure FPGAs with FUSE C/C++ API

16.5 FPGA Configuration using DIMEscript

DIMEscript is a high-level scripting language, which provides users with an easy-to-use language for the configuration and control of DIME systems. DIMEscript uses a simple command set, eliminating the need for developers to use complicated programming interfaces to control and communicate with application designs running in FPGAs. DIMEscript also offers platform portability through ASCII based scripts, allowing users to use DIMEscript on both Windows and Linux installations.

DIMEscript can be used either to write script files, which can then be executed as a single process, or can be used from a command line interface, with the user executing commands as required.

- Full instructions on how to use DIMEscript are provided in the *DIMEscript User Guide*, which is on the FUSE CD at the location 'CD ROM Drive\Documentation\FUSE'

16.6 FPGA Configuration using General JTAG Chain

The General JTAG chain is the principal JTAG chain in the Kit and facilitates the configuration of the FPGAs on the DIME-II module hosted on the BenONE-Kit Motherboard. The JTAG Chain is driven by the interface FPGA, or can be driven from the General JTAG pin header on the motherboard, via a JTAG programmer, such as the Xilinx Parallel-III programmer. Therefore this is where to connect the Parallel-III or IV cables in order to use products such as ChipScope ILA.

The General JTAG chain has built-in switches, which only switch a DIME-II module site into the chain, if a module is populated. Therefore, if a module is not populated in a slot, the chain skips that slot.



Please note that specific information about builds for different front-end configurations is for reference only. Attempted repair or alteration of the goods as supplied by Nallatech immediately invalidates the

warranty. Nallatech will not provide support upon alteration and cannot guarantee performance characteristics subsequently obtained.



If you are using a JTAG header then these module power supplies need to be enabled in order to access all the devices in the JTAG chain. Without this the JTAG chain will not be complete and you will receive errors. See “[Enabling Power Supplies at Power-On \(using External Power Supply and JTAG connectors\)](#)” on [page 106](#) for information on enabling module power supplies at power-on.

The General JTAG chain configuration is shown in [Figure 74 on page 127](#) along with the header to access the chain. The header pinout is detailed in [Table 50 on page 127](#).

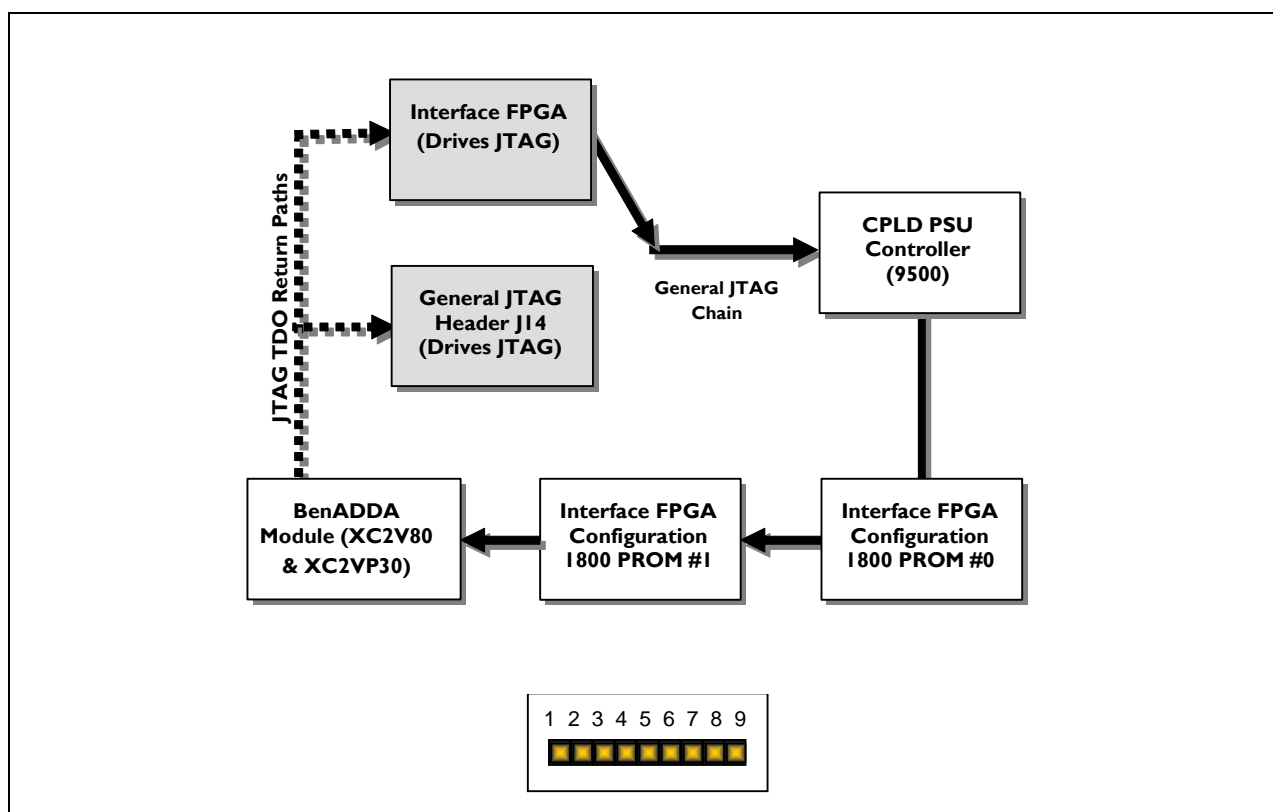


Figure 74: General JTAG Chain (top) and General JTAG Connector J14 (bottom)

Pin #	Name	Description
1	3.3V	3.3 Volts Supply
2	GND	Signal Ground
3	N/C	Not connected - do not use
4	TCK	ALT JTAG TCK Signal
5	N/C	Not connected - do not use
6	TDO	ALT JTAG TDO Signal
7	TDI	ALT JTAG TDI Signal

Table 50: General JTAG Header Pinouts

Pin #	Name	Description
8	TRST#	ALT JTAG TRST# Signal
9	TMS	ALT JTAG TMS Signal

Table 50: General JTAG Header Pinouts

Care must be taken when using this method to program a module. Inadvertent programming of either the PSU Controller or Interface Boot prompts on your Kit could render it inoperable or in extreme cases damage could occur.



NALLATECH
The High Performance FPGA Solutions Company

Part V:Hardware Examples

This part of the User Guide provides hardware example applications which enable you to start using the XtremeDSP Development Kit Pro.

Section 17

Feature Examples

In this section:

- Introduction
 - Simple ADC Hookup Example
 - ADC to DAC Example
 - Host Interface Example
 - Relevant Application Notes
-

17.1 Introduction

This section details a number of examples that are included with the Kit. It also makes reference to some of the application notes on either the FUSE CD or the XtremeDSP Development Kit Pro CD. Apart from these examples related to specific features on the hardware, there are some additional examples later in the User Guide that deal with support for specific tools such as Xilinx Impact. In each example the top level source is included with notes added to explain the function in each example.

17.2 Simple ADC Hookup Example

17.2.1 Example Overview

The purpose of this example is to show how to connect up to the ADCs on the hardware. The code brings in the data from each ADC and registers it, then uses some of the bits to drive the LEDs on the Kit hardware. Remember that the LEDs are active-low (i.e. a low will turn them on).

17.2.2 Source

The source files are provided in the folder 'CD ROM Drive\Examples\simple_adc_hookup\source' on the XtremeDSP Development Kit Pro CD.

17.2.3 Implementation

A UCF is also provided in the source folder that includes all pin LOC and timing constraints. Apart from the source files a Xilinx ISE project has been included on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Examples\simple_adc_hookup\ise\simple_adc_hookup' or is installed to the location where you installed the Kit CD. This project has been created in Xilinx ISE 6.3i. A UCF is also provided in the source folder that includes all pin LOC and timing constraints.

17.2.4 Running the Example

Please note that although this example can be run it is primarily intended as a simple introduction to ADC connection. The bitfiles for the example have already been generated.

▼ **To run the ADC Hookup example use the following procedures:**

1. Start the FUSE Probe Tool and assign the following bitfiles:
 - Osc_clk_2v80.bit' to the Clock FPGA.
 - Simple_adc_hookup.bit' to the main User FPGA.
2. Toggle the resets:
 - Check 'FPGA Reset' to enable the reset to the main User FPGA.
 - Check 'System Reset' to enable the reset common to both FPGAs.
 - Click the button for 'Interface Reset' to reset the interface core, i.e. reset the Interface FIFOs.
 - Uncheck 'FPGA Reset' to disable the reset to the main User FPGA.
 - Uncheck 'System Reset' to disable the reset common to both FPGAs.
3. You can then connect a suitable input signal, as specified in **“Analog ADC Inputs” on page 21**, with the MCX inputs corresponding to the ADCs. The LEDs will change depending on the levels selected in the code.

Listing (simple_adc_hookup.vhd)

```

-- Title   : simple_adc_hookup
-- Project :

-- File    : simple_adc_hookup.vhd
-- Author  : <derekstark@DELL2000AXP>
-- Company :
-- Created : 2003-11-11
-- Last update: 2003-11-11
-- Platform :
-- Standard : VHDL'87

-- Description: Showing a simple ADC hookup in the XtremeDSP kit. This code
-- simply brings in the data from each ADC and registers it. It then simply
-- uses some of the bits from them to drive the LEDs on the
-- kit hardware. Remember that the LEDs are active-low, i.e. a low will turn
-- them on.

-- Copyright (c) 2003

-- Revisions :
-- Date      Version Author Description
-- 2003-11-11 1.0   derekstark Created

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity simple_adc_hookup is
  port (
    -- main clock input from oscillator
    CLK1_FB : in std_logic;
    -- main reset input from mb
    RESET1 : in std_logic;
    -- configuration done signal
    CONFIG_DONE : out std_logic;
    -- adc 14 bit data inputs
    ADC1_D : in std_logic_vector(13 downto 0);
    ADC2_D : in std_logic_vector(13 downto 0);
    -- adc dry signals
    ADC1_DRY : in std_logic;
    ADC2_DRY : in std_logic;
    -- led flash signals used to give some indications.
    LED_Red1 : out std_logic;
    LED_Red2 : out std_logic;
    LED_Green1 : out std_logic;
    LED_Green2 : out std_logic
  );
end simple_adc_hookup;

architecture structural of simple_adc_hookup is

  signal ADC1_Di, ADC2_Di: std_logic_vector(13 downto 0); -- internal
  versions of the ADC data signals

begin

  -- module configured
  CONFIG_DONE <= '0';

  -- register the adc inputs

```

Top level entity. In this example CLK1_FB is fed down from the Clock FPGA to the User FPGA. We will be using the standard clock bitfile osc_clk_2v80.bit that simply takes the 65MHz module crystal clock source and sends it to the ADCs and DACs as well as sending it to the main User FPGA.

```

adcDataRegisters : process (CLK1_FB, RESET1)
begin
  if RESET1 = '0' then
    ADC1_Di <= (others => '0');
    ADC2_Di <= (others => '0');
  elsif CLK1_FB = '1' and CLK1_FB'event then
    ADC1_Di <= ADC1_D;
    ADC2_Di <= ADC2_D;
  end if;
end process;

-- led flasher section

process (CLK1_FB, RESET1)
begin
  if RESET1 = '0' then
    -- led assignments
    LED_Red1 <= '1';
    LED_Red2 <= '1';
    LED_Green1 <= '1';
    LED_Green2 <= '1';
  elsif CLK1_FB'event and CLK1_FB = '1' then

    -- led assignments inverted as leds are active low
    if (ADC1_Di(6 downto 0) = "111111") then
      LED_Red1 <= '0';
    else
      LED_Red1 <= '1';
    end if;

    if (ADC1_Di(13 downto 7) = "111111") then
      LED_Green1 <= '0';
    else
      LED_Green1 <= '1';
    end if;

    if (ADC2_Di(6 downto 0) = "111111") then
      LED_Red2 <= '0';
    else
      LED_Red2 <= '1';
    end if;

    if (ADC2_Di(13 downto 7) = "111111") then
      LED_Green2 <= '0';
    else
      LED_Green2 <= '1';
    end if;
  end if;
end process;
end structural;

```

Registering the ADC inputs is important to ensure that we properly capture the data into the clock domain that we are using. This is important so that we meet timing requirements.

This process is a very simply setup to look for specific values on the ADC inputs. Remember that the ADC inputs are 2s complement format and so the current default values used here to match are for an overall value of -1.

The process will simply illuminate LEDs if any of the values match. Note that to get the LEDs to light they require a LOW i.e. '0' value to be driven into them.

For the sake of experimentation these match values or even the slicing of the ADC data buses can be changed if so desired.

17.3 ADC to DAC Example

17.3.1 Example Overview

The purpose of this example is to demonstrate a simple ADC to DAC hookup with a count driven out of the LEDs. The design captures data from the ADCs (registering it) and then outputs this on the DACs with a slight conversion to change from Two's complement (ADCs) to offset binary (DACs).

17.3.2 Source

The source files are provided on the XtremeDSP Development Kit Pro CD in the folder 'CD ROM Drive\Examples\adc_to_dac_hookup\source' or is installed to the location where you installed the Kit CD.

17.3.3 Implementation

Apart from the source files a Xilinx ISE project has been included on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Examples\adc_to_dac_hookup\ise\adc_to_dac_hookup' or is installed to the location where you installed the Kit CD. This project has been created in Xilinx ISE 6.3i. A UCF is also provided in the source folder that includes all pin LOC and timing constraints.

17.3.4 Running the Example

Please note that although this example can be run it is primarily intended as a simple introduction to creating an ADC to DAC link. The primary purpose is to highlight the differences in data format between the ADCs and DACs and how to convert between them. The bitfiles for the example have already been generated.

▼ **To run the ADC to DAC example use the following procedures:**

1. Start the FUSE Probe Tool and assign the following bitfiles:
 - Osc_clk_2v80.bit' to the Clock FPGA.
 - Adc_to_dac_hookup.bit' to the main User FPGA.
2. Finally toggle the resets by doing the following:
 - Check 'FPGA Reset' to enable the reset to the main User FPGA.
 - Check 'System Reset' to enable the reset common to both FPGAs.
 - Click the button for 'Interface Reset' to reset the interface core, i.e. reset the Interface FIFOs.
 - Uncheck 'FPGA Reset' to disable the reset to the main User FPGA.
 - Uncheck 'System Reset' to disable the reset common to both FPGAs.

3. You can then connect a suitable input signal, as specified in “Analog ADC Inputs” on page 21, with the MCX inputs corresponding to the ADCs. Also the DACs can be connected to a suitable display such as an Oscilloscope. The LEDs will move through a count and the input signal is sent through to the output.

Listing (adc_to_dac_hookup.vhd)

```

-- Title   : adc_to_dac_hookup
-- Project : adc_to_dac_hookup

-- File    : adc_to_dac_hookup.vhd
-- Author  : <derekstark@DELL2000AXP>
-- Company : Nallatech Ltd
-- Created : 2003-11-12
-- Last update: 2003-11-12
-- Platform : DIME-II
-- Standard : VHDL'87

-- Description: Simple adc to dac hookup with a count being driven out the
-- LEDs. The design simply captures data from the ADCs (registering it) and
-- then outputs this on the DACs with a slight conversion to change from 2s
-- complement to offset binary.

-- Copyright (c) 2003

-- Revisions :
-- Date      Version Author Description
-- 2003-11-12 1.0    derekstark Created

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Toplevel is
  port (
    CLK1_FB : in std_logic; -- main clock input from oscillator
    RESET1 : in std_logic; -- main reset input from mb
    CONFIG_DONE : out std_logic; -- configuration done signal
    -- dac 14 bit data outputs
    DAC1_D : out std_logic_vector(13 downto 0);
    DAC2_D : out std_logic_vector(13 downto 0);
    -- adc 14 bit data inputs
    ADC1_D : in std_logic_vector(13 downto 0);
    ADC2_D : in std_logic_vector(13 downto 0);
    -- dac reset signals
    DAC1_RESET : out std_logic;
    DAC2_RESET : out std_logic;
    -- dac setup
    DAC1_MOD0 : out std_logic;
    DAC1_MOD1 : out std_logic;
    DAC2_MOD0 : out std_logic;
    DAC2_MOD1 : out std_logic;
    -- dac clock divider setup
    DAC1_DIV0 : out std_logic;
    DAC1_DIV1 : out std_logic;
    DAC2_DIV0 : out std_logic;
    DAC2_DIV1 : out std_logic;
    -- led flash signals
    LED1_Red : out std_logic;
    LED2_Red : out std_logic;
    LED1_Green : out std_logic;
    LED2_Green : out std_logic;
  );
end Toplevel;

```

Top level entity. In this example CLK1_FB is fed down from the Clock FPGA to the User FPGA. We will be using the standard clock bitfile osc_clk_2v80.bit that simply takes the 65MHz module crystal clock source and sends it to the ADCs and DACs as well as sending it to the main User FPGA.

In this example the RESET1 is the active-low System Reset signal that can be controlled via FUSE.

It connects the data buses from the ADCs and DACs but also has pins defined to control the operational mode of the DACs. These pins are DAC1_MOD and DAC1_DIV etc.

The other main point to note here is the inclusion of the CONFIG_DONE output signal that is a pin that should be driven low when the design configures to indicate to the reset of the system that it has configured.

architecture Behavioral of Toplevel is

```

-- clock components
component BUFG
port (
  I : in std_logic;
  O : out std_logic
);
end component;
component IBUFG
port (
  I : in std_logic;
  O : out std_logic
);
end component;
component DCM
generic (
  DLL_FREQUENCY_MODE : string := "LOW";
  DUTY_CYCLE_CORRECTION : string := "TRUE";
  STARTUP_WAIT : string := "FALSE"
);
port (
  CLKIN : in std_logic;
  CLKFB : in std_logic;
  DSSEN : in std_logic;
  PSINCDEC : in std_logic;
  PSEN : in std_logic;
  PCLK : in std_logic;
  RST : in std_logic;
  CLK0 : out std_logic;
  CLK90 : out std_logic;
  CLK180 : out std_logic;
  CLK270 : out std_logic;
  CLK2X : out std_logic;
  CLK2X180 : out std_logic;
  CLKDV : out std_logic;
  CLKFX : out std_logic;
  CLKFX180 : out std_logic;
  LOCKED : out std_logic;
  PSDONE : out std_logic;
  STATUS : out std_logic_vector(7 downto 0)
);
end component;
-- end of clock components

-- internal clock and reset signals
signal CLKIN_OSC, CLKFB_OSC, CLK_OSC, RESET, RST1 : std_logic;

-- temporary registers
signal ADC1, ADC2 : std_logic_vector(13 downto 0);

-- common ground
signal GND : std_logic;

begin

GND <= '0';

RESET <= not RESETI;

-- clock deskew section --
-- IBUFG Instantiation for CLK_IN
U0_IBUFG : IBUFG
port map (
  I => CLKI_FB,
  O => CLKIN_OSC
);
-- BUFG Instantiation for CLKFB
U0_BUFG : BUFG
port map (
  I => CLKFB_OSC,
  O => CLK_OSC
);
-- DCM Instantiation for internal deskew of CLK0
U0_DCM : DCM
port map (
  CLKIN => CLKIN_OSC,
  CLKFB => CLK_OSC,
  DSSEN => GND,
  PSINCDEC => GND,
  PSEN => GND,
  PCLK => GND,
  RST => RESET,
  CLK0 => CLKFB_OSC,
  LOCKED => RST1
);
-- end of clock deskew --

-- module configured
CONFIG_DONE <= '0';

-- set low pass filter response and no zero stuffing for both DACs
DAC1_MOD0 <= '0';
DAC1_MOD1 <= '0';
DAC2_MOD0 <= '0';
DAC2_MOD1 <= '0';

-- disable resets for DACs
DAC1_RESET <= '0';
DAC2_RESET <= '0';

-- optimum settings for sampling rate
DAC1_DIV0 <= '1';
DAC1_DIV1 <= '0';
DAC2_DIV0 <= '1';
DAC2_DIV1 <= '0';

```

In this example the DCM and clock circuitry are manually included at this level of the VHDL rather than using a generated clock module. This has been done simply to show how it is done manually in the code.

Note that the DCM has been declared to be in the low frequency mode. This is important and is suitable for this example as the 105MHz clock input is within the LOW frequency operating mode of the DCM.

GND is declared as a signal so it can be assigned a value to improve the readability of the code further down.

GND is then assigned to be a single bit and to be LOW.

The reset signal is active-Low when it is input to the chip. However, the reset signal needs to reset a DCM which has an active high reset input. Hence the need for the inversion of the input active-low reset signal.

Following this the IBUFG, BUFG and DCM components are instantiated to create a typical CLK0 DCM circuit.

The CONFIG_DONE signal is driven low as this allows the reset of the system to determine that the main User FPGA has successfully configured.

The DAC mode control signals are set to specific values for operation. In this example these are fixed in the design but these could equally be set by a value in a software controllable register in a user design.

It is important to not leave the reset signals to the DACs floating. In this example these are simply tied to 0.

```

-- digital output of adc to digital input of DAC
DataRegisters : process (CLK_OSC, RSTI)
begin
  if RSTI = '0' then
    ADC1 <= "0000000000000000";
    ADC2 <= "0000000000000000";
    DAC1_D <= "0000000000000000";
    DAC2_D <= "0000000000000000";
  elsif CLK_OSC = '1' and CLK_OSC'event then
    ADC1 <= ADC1_D;
    ADC2 <= ADC2_D;
    DAC1_D <= not (not ADC1(13) & ADC1(12 downto 0));
    DAC2_D <= not (not ADC2(13) & ADC2(12 downto 0));
  end if;
end process;

-- led flasher section
-- led flash counter
process (CLK_OSC, RSTI)
variable COUNT : std_logic_vector(26 downto 0);
begin
  if RSTI = '0' then
    COUNT := (others => '0');
    -- led assignments
    LED1_Red <= '0';
    LED2_Red <= '0';
    LED1_Green <= '0';
    LED2_Green <= '0';
  elsif CLK_OSC = '1' and CLK_OSC'event then
    COUNT := COUNT + 1;
    -- led assignments
    LED1_Red <= COUNT(26);
    LED2_Red <= COUNT(25);
    LED1_Green <= COUNT(25);
    LED2_Green <= COUNT(26);
  end if;
end process;
-- end of led flasher

```

This process is used to create a simple set of registers for the data captured via the ADCs before it is sent to the DACs.

The data to the DACs is simply the 2s complement ADC data converted to offset binary for the DACs. This is converted by simply inverting the top bit of the ADC data. The reason for the additional inversion of the whole value is due to the fact that the op-amp that drives out of the final MCX connector is an inverting op-amp.

Finally a counter is inferred and the more significant bits are connected up to the LEDs on the module.

17.4 Host Interface Example

17.4.1 Example Overview

The XtremeDSP Development Kit Pro CD and installation also includes an example of a host interface that makes use of the 'Interface FPGA to User FPGA Interface Core'. This example is based on the information included in the applications notes:

- *PCI to User FPGA Interface Core* (Included on the FUSE CD).
- *XtremeDSP Kit Ping Example* (Included on the XtremeDSP Development Kit Pro CD).

This example is intended to provide additional information and insight on the software code, to allow for a greater understanding on building a host interface to your Kit designs.

17.4.2 Source

In this example there is source for both VHDL and for C.

- The VHDL source files are provided on the XtremeDSP Development Kit Pro CD in the location 'CD ROM Drive\Examples\host_interface_basic\source'
- The C source files are provided on the XtremeDSP Development Kit Pro CD in the folder 'CD ROM Drive\Examples\host_interface_basic\CCodeTester'

17.4.3 Implementation

Apart from the source files a Xilinx ISE project has been included on the XtremeDSP Development Kit Pro CD at the location: 'CD ROM Drive\Examples\host_interface_basixise\host_interface' or is installed to the location where you installed the Kit CD. This project has been created in Xilinx ISE 6.3i. A UCF is also provided in the source folder that includes all pin LOC and timing constraints.

In addition to this a Microsoft Visual Studio project and workspace are available. The project files are in 'CD ROM Drive\Examples\host_interface_basic\CCodeTester'.

17.4.4 Running the Example

To help run the example a Microsoft Visual Studio project has already been compiled into both a debug and release executable. Running either of these files. Note however that the card interface type should be changed in the code to match the Kit hardware interface (PCI or USB). The executables are located in:

- 'CD ROM\Examples\host_interface_basic\CCodeTester\release'
- or
- 'CD ROM\Examples\host_interface_basic\CCodeTester\debug'

VHDL Listing (host_interface.vhd)

```

library IEEE;
use IEEE.std_logic_1164.all;

entity host_interface is
  port (
    -- Interface clock.
    CLKB      : in  std_logic;
    -- Global reset.
    RSTI      : in  std_logic;
    -- Indicates whether Spartan can receive data.
    BUSY      : in  std_logic;
    -- Indicates whether Spartan has data to send.
    EMPTY     : in  std_logic;
    -- Indicates that ADIO is address or data.
    AS_DSI    : in  std_logic;
    -- Read/Write enable.
    RENI_WENI : out std_logic;
    -- Indicates if Spartan is being read or being written to.
    RDI_WR    : out std_logic;
    -- Interrupt to Spartan.
    INTI      : out std_logic;
    -- Test LEDs.
    LEDS      : out std_logic_vector (7 downto 0);
    -- Data IO between Spartan and Virtex.
    ADIO      : inout std_logic_vector (31 downto 0);
    DUMMSIGNAL : out std_logic
  );
end host_interface;

architecture host_interface_arch of host_interface is

  component SV_IFACE
    generic (
      -- Number of memory blocks in memory map in bit size.
      NUM_BLOCK_Sg : integer range 1 to 29; -- := 4;
      -- Block size.
      BLOCK_SIZEg : integer range 1 to 29; -- := 8;
      -- Number of registers.
      NUM_REG_Sg : integer range 1 to 29 -- := 4
    );
    port (
      -- Interface clock.
      CLK      : in  std_logic;
      -- Global reset.
      RSTI     : in  std_logic;
      -- Indicates whether Spartan can receive data.
      BUSY     : in  std_logic;
      -- Indicates whether Spartan has data to send.
      EMPTY    : in  std_logic;
      -- Indicates that ADIO is address or data.
      AS_DSI   : in  std_logic;
      -- DMA engine is writing data out.
      DMA_WEN  : in  std_logic;
      -- DMA engine is reading data in.
      DMA_REN  : in  std_logic;
      -- Interrupt.
      INT      : in  std_logic;
      -- Read/Write enable.
      RENI_WENI : out std_logic;
      -- Indicates if Spartan is being read or being written to.
      RDI_WR   : out std_logic;
      -- Interrupt to Spartan.
      INTI     : out std_logic;
      -- Current active address.
      ADDRESS  : out std_logic_vector (30 downto 0);
      -- Write data into register.
      WRITE_STROBE : out std_logic;
      -- Read data from register.
      READ_STROBE  : out std_logic;
      -- Current DMA count.
      COUNT     : out std_logic_vector (31 downto 0);
      -- Enable DMA engine.
      DMA_ENABLE : out std_logic;
      -- DMA direction.
      DMA_DIRECTION : out std_logic;
      -- DMA select.
      DMA_SEL   : out std_logic_vector (3 downto 0);
      -- DMA control is ready for DMA engine to send data.
      DMA_RDY   : out std_logic;
      -- DMA control has data for DMA engine to read.
      DMA_DATA_AVAILABLE : out std_logic;
      -- Reset to rest of Virtex.
      RST       : out std_logic;
      -- Synchronous reset.
      SYNC_RESET : out std_logic;
      -- DMA is reset.
      DMA_RESET : out std_logic;
      -- Data IO between Spartan and Virtex.
      ADIO      : inout std_logic_vector (31 downto 0);
      -- Internal data bus.
      DATA     : inout std_logic_vector (31 downto 0);
      -- Internal DMA data bus.
      DMA_DATA  : inout std_logic_vector (31 downto 0)
    );
  end component;

```

The top level entity makes use of the CLKB DIME Clock as the primary clock. In the documentation this is sometimes referred to as CLK1 or even DSP_CLK.

The entity has signals on it related to the interface to main User FPGA communications bus.

This is the declaration of the main component for the interface to User FPGA interface core described in application note NT302-0000 Spartan to Virtex Interface.

```

component SYNCFIFO
port (RCLK      : in std_logic;
      WCLK      : in std_logic;
      READ_EN   : in std_logic;
      WRITE_EN  : in std_logic;
      FIFO_RST  : in std_logic;
      D         : in std_logic_vector (31 downto 0);
      Q         : out std_logic_vector (31 downto 0);
      FIFO_STATUS : out std_logic_vector (4 downto 0);
      FIFO_FULL  : out std_logic;
      FIFO_EMPTY : out std_logic;
      );
end component;

```

–Declare the clock module that contains the DCM

```

component dimeclk_module
port(
  rst_in   : in std_logic;
  clkIn_in : in std_logic;
  locked_out : out std_logic;
  clk0_out  : out std_logic;
  );
end component;

```

–Declare a signal for the internal clock signal produced by the DCM in the clock module.

```

signal CLKBi : std_logic;

```

–Declare a signal for the locked signal from the clock module.

```

signal CLKB_LOCKED : std_logic;

```

–Declare delayed versions of some of the host interface control signals.

```

signal EMPTYd1, EMPTYd2 : std_logic;
signal AS_DSId1, AS_DSId2 : std_logic;

```

– DMA control.

```

signal DMA_WEN : std_logic;
signal DMA_REN : std_logic;

```

– DMA control.

```

signal DMA_ENABLE : std_logic;
signal DMA_DIRECTION : std_logic;
signal DMA_RDY : std_logic;
signal DMA_DATA_AVAILABLE : std_logic;
signal DMA_DATA : std_logic_vector (31 downto 0);

```

– Register control.

```

signal ADDRESS : std_logic_vector (30 downto 0);
signal WRITE_STROBE : std_logic;
signal READ_STROBE : std_logic;
signal DATA : std_logic_vector (31 downto 0);

```

–Declare an internal version of the interrupt signal to the host interface

```

signal INT : std_logic;

```

–Declare the output reset signal from the host interface.

```

signal RST : std_logic;

```

–Declare the inverted version of the external reset signal. This is the active-high version.

```

signal RST_EXT : std_logic;

```

–Declare and internal reset signal - this is based on the locked signal from DCM in clock module.

```

signal RST_INTI : std_logic;

```

– FIFO data/status/control.

```

signal FIFO_AFULL : std_logic;
signal FIFO_EMPTY : std_logic;
signal FIFO_STATUS : std_logic_vector (4 downto 0);
signal FIFO_VWEN : std_logic;
signal FIFO_REN : std_logic;
signal FIFO_IN : std_logic_vector (31 downto 0);
signal FIFO_OUT : std_logic_vector (31 downto 0);

```

–Define signals for the registers in the design.

```

signal REG1 : std_logic_vector (31 downto 0);
signal REG2 : std_logic_vector (31 downto 0);
signal REG1_WR : std_logic;
signal REG2_WR : std_logic;
signal REG1_RD : std_logic;
signal REG2_RD : std_logic;

```

begin

–Invert the active-low reset input to the FPGA to convert it to an active-high reset

```

RST_EXT <= not(RSTI);

```

–Instantiate the clock module

```

Inst_dimeclk_module : dimeclk_module port map(
  rst_in   => RST_EXT,
  clkIn_in => CLKB,
  locked_out => CLKB_LOCKED,
  clk0_out  => CLKBi
);

```

–Create a register to register the locked signal into the clock domain before use in the rest of the system.

```

process(CLKBi)
begin
  if (CLKBi'event and CLKBi = '1') then
    RST_INTI <= CLKB_LOCKED;
  end if;
end process;

```

This declaration is simply for a 511x32 FIFO which is included to simply provide a target to which data can be burst to or from for read and writes from the host PC.

In this design we make use of a generated clock module that has been produced using the DCM Architecture Wizard in the Xilinx ISE tools.

Various internal signals are declared.

A number of signals are declared that will represent the registers in the design. Corresponding read and write control signals are also declared here.

The external active-low reset is inverted here to provide an active-high reset that will be required by a number of components such as the DCM components in the clock module.

The locked signal from the DCM in the clock module is then used, after a register, as the reset for the rest of the system.

```

-- Main interface host interface component.
H1_IFACE : SV_IFACE
  generic map (NUM_BLOCKSg => 1,
    BLOCK_SIZEg => 4,
    NUM_REGSg => 3
  )
  port map (CLK          => CLKBi,
    RSTi                 => RST_INTI,
    BUSY                  => BUSY,
    EMPTY                 => EMPTY,
    AS_DSI                => AS_DSI,
    DMA_WEN               => DMA_WENi,
    DMA_REN               => DMA_RENi,
    INT                   => INTi,
    RENi_WENi             => RENi_WENi,
    RDI_WVR               => RDI_WVRi,
    INTi                  => INTi,
    ADDRESS               => ADDRESS,
    WRITE_STROBE          => WRITE_STROBE,
    READ_STROBE           => READ_STROBE,
    COUNT                 => open,
    DMA_ENABLE            => DMA_ENABLE,
    DMA_DIRECTION         => DMA_DIRECTION,
    DMA_SEL               => open,
    DMA_RDY               => DMA_RDY,
    DMA_DATA_AVAILABLE => DMA_DATA_AVAILABLE,
    RST                   => open,
    SYNC_RESET            => RST,
    DMA_RESET             => open,
    ADIO                  => ADIO,
    DATA                 => DATA,
    DMA_DATA              => DMA_DATA
  );

-- test FIFO to allow demonstration of DMA interface of the host interface.
-- This could be any component capable of receiving bursts of data. In this case
it is
-- simply a 51 x 32 FIFO.
H2_FIFO : SYNCFIFO port map (RCLK          => CLKBi,
  WCLK          => CLKBi,
  READ_EN       => FIFO_REN,
  WRITE_EN      => FIFO_WEN,
  FIFO_RST      => RST,
  D             => FIFO_IN,
  Q             => FIFO_OUT,
  FIFO_STATUS   => FIFO_STATUS,
  FIFO_FULL     => open,
  FIFO_EMPTY    => FIFO_EMPTY
);

-- FIFO data.
FIFO_IN <= DMA_DATA;
DMA_DATA <= FIFO_OUT when DMA_DIRECTION = 'I' else
(others => 'Z');

-- Create a fifo almost full signal.
FIFO_AFULL <= FIFO_STATUS(3);

-- FIFO and DMA control.
FIFO_REN <= 'I' when DMA_ENABLE = 'I' and DMA_DIRECTION =
'I' and DMA_RDY = 'I' and FIFO_EMPTY = '0' else '0';
FIFO_WEN <= 'I' when DMA_ENABLE = 'I' and DMA_DIRECTION =
'0' and DMA_DATA_AVAILABLE = 'I' and FIFO_AFULL = '0' else '0';
DMA_REN <= FIFO_WEN;

-- Create a registered version of some of the control signals.
process (CLKBi, RST)
begin
  if RST = 'I' then
    DMA_WEN <= '0';
  elsif CLKBi'event and CLKBi = 'I' then
    DMA_WEN <= FIFO_REN;
  end if;
end process;

-- Two registers control.
REG1_WR <= 'I' when WRITE_STROBE = 'I' and ADDRESS(3
downto 0) = "0010" else '0';
REG2_WR <= 'I' when WRITE_STROBE = 'I' and ADDRESS(3
downto 0) = "0011" else '0';
REG1_RD <= 'I' when READ_STROBE = 'I' and ADDRESS(3
downto 0) = "0010" else '0';
REG2_RD <= 'I' when READ_STROBE = 'I' and ADDRESS(3
downto 0) = "0011" else '0';

-- Map to the DATAbus when reading either register depending upon selection.
DATA <= REG1 when REG1_RD = 'I' else (others => 'Z');
DATA <= REG2 when REG2_RD = 'I' else (others => 'Z');

-- Infer two registers.
process (RST, CLKBi)
begin
  if RST = 'I' then
    REG1 <= (others => '0');
    REG2 <= (others => '1');
  elsif CLKBi'event and CLKBi = 'I' then
    if REG1_WR = 'I' then
      REG1 <= DATA(31 downto 0);
    end if;
    if REG2_WR = 'I' then
      REG2 <= DATA(31 downto 0);
    end if;
  end if;
end process;

```

The interface to user FPGA interface core is now instantiated with generics set appropriately. Please see application note NT302-0000 Spartan to Virtex Interface for more details on these generics.

Essentially though this core is instantiated and then the interface side signal are connected to top level ports. The user side signals are assigned to internal nets.

Note that DMA_Sel is not used in this example as we are assuming that we only want a single DMA channel. Not that the DMA channels are set by the DMA_Sel and are purely a way of decoding where on the card the burst data is for.

Instantiate the FIFO that will be used to store burst of data. This FIFO is simply for example purposes of how to connect to the sv_iface component. In user designs the FIFO may be any other specific component that can receive data.

The FIFO data ports are connected the DMA_DATA port on the sv_iface core.

It is important for the sake of interfacing that some measure of FIFO contents are fed back to the sv_iface component to indicate when the FIFO is almost full.

Using the FIFO status, along with the control signals from the interface core, FIFO read and write enable signals are generated.

Some of tesecontrol signals require to be registered to allow for safe usage.

Now some decoding of the address and control signals are required in order to create read and write control signals for the individual registers in the design.

When the registers are selected for readback to the host their values are put onto the bi-directional databus to the interface core.

Infer the actual two registers.

```

--Simply map part of each of the two registers to the output LEDs for visual
feedback.
LEDS(3 downto 0) <= REG1(3 downto 0);
LEDS(7 downto 4) <= REG2(3 downto 0);

--Create a dummy signal to stop the tool optimising out the rest of the registers
REG1 and REG2 that are not used as outputs.
process (RST, CLKBi)
begin
  if RST = '1' then
    DUMMY SIGNAL <= '0';
  elsif (REG1 = X"00000000" and REG2 = X"00000000") then
    DUMMY SIGNAL <= '1';
  end if;
end process;

--Simply tie the interrupt line low in this example as it is not used.
INT <= '0';

end host_interface_arch;

```

Map part of the two register to the LEDs on the hardware. Note that here 8 LEDs seem to be specified but in the actual implementation only 4 are actually wired to physical LEDs.

The dummy signal is used here to stop the synthesis tool optimising out parts of the registers that are not specifically being used. In real designs the outputs of the registers will actually provide a specific purpose. In this case, as we only write and read from them with no part of the following circuit being dependent upon them, the synthesis tool will remove the registers in part. The inclusion of the dependency of the dummy signal on the register values ensure that for the sake of this example the registers are not optimised out.

C Code Listing (OpeningASingleCard.c)

```
// Opening Cards Demo
// This is a simple program to show how to locate and open a card.
//
//
/*
** $Id$
*/
//Note that in this project settings dimesd.lib is included.
//dimesd.lib is required to gain access to the FUSE API functions.
//dimesd.lib and dimesd.lib can both be found in the FUSE\include directory.
#include "dimesd.h"
#include "vidime.h"
#include <stdio.h>

DWORD ModuleNumber=0; //THIS NUMBER NEEDS TO BE
CHANGED TO THE DESIRED MODULE NUMBER.

DWORD PrimaryFPGADeviceNum=1; //THIS NUMBER NEEDS TO
BE CHANGED TO THE MODULE DEVICE NUMBER.

DWORD SecondaryFPGADeviceNum=0; //THIS NUMBER NEEDS
TO BE CHANGED TO THE MODULE DEVICE NUMBER.

char Filename1 []="host_interface.bit";
char Filename2 []="osc_clock_2v80.bit";

int main(int argc, char* argv[])
{
    LOCATE_HANDLE hLocate = NULL;
    DWORD ErrorNum,NumOfCards,LoopCnt,LEDs;
    char ErrorString[1000];
    DIME_HANDLE hCard1;
    double ActualFrequency;
    VIDIME_HANDLE viHandle;

    DWORD Result, error=0;
    DWORD j=0;

    DWORD ReadData[2048];
    DWORD WriteData[2048];

    for(j=0;j<2048;j++)
    {
        WriteData[j]=+1;
        ReadData[j]=0;
    }

    printf("BIST Embedded Test - Test!\n");

    //Call the function to locate all Nallatech cards on the PCI interface.
    if( hLocate =
    DIME_LocateCard(dIPCI,mbtALL,NULL,dldrDEFAULT,dIDEFAULT) ==
    NULL)
    {
        //Error hLocate NULL
        //Print the error then terminate the program
        DIME_GetError(NULL,&ErrorNum,ErrorString);
        printf("Error Number %d\n",ErrorNum);
        printf("%s\n",ErrorString);
        exit(1);
    }

    //Determine how many Nallatech cards have been found.
    NumOfCards = DIME_LocateStatus(hLocate,0,dINUMCARDS);
    printf("%d Nallatech card(s) found.\n",NumOfCards);
}
```

Header files are included here.

Dimesd.h contains the main FUSE header for the C/C++ API. This contains a number of declarations that are used in the following code.

Vidime.h contains the header for the functions used to simplify the software interface to the interface FPGA to User FPGA core.

A few variables are defined here for ease. They have all be pre-set for the XtremeDSP Kit and does not need changed.

Filenames are then specified for the purposes of the example.

The main function includes more declarations for variables in the system. Also some data areas are reserved statically, i.e. ReadData and WriteData. These will be used to demonstrate data transfer to and from the interface.

A simple loop is carried out to create some test data in the WriteData array.

The first step is to obtain a handle to the cards detected in the system by FUSE. This is done using the DIME_LocateCard function. It will return a handle. It is important to note that in this example the PCI interface type has been selected 'dIPCI'. If you wish to detect cards on the USB interface this should simply be changed to 'dIUSB'.

```
//Get the details for each card detected.
for (LoopCnt=1; LoopCnt<=NumOfCards; LoopCnt++)
{
    printf("Details of card number %d, of %d:

    printf("    The card driver for this card is a %s.
    printf("    The cards motherboard type is %d.
}

//At this stage we now have all the information we need to open a card up.

//Open up the first card found. To open the nth card found simple change the
second argument to n.
hCard1 =
DIME_OpenCard(hLocate,1,dccOPEN_NO_OSCILLATOR_SETUP); //
opens up card 1 with default flags
if (hCard1 == NULL) //check to see if the open worked.
{
    printf("Card Number One failed to open.

    DIME_CloseLocate(hLocate);
    printf("

    getchar();
    return(1);
}

//Change the LEDs
LEDs = DIME_ReadLEDs(hCard1);
DIME_WriteLEDs(hCard1, (LEDs - 1));
printf("LEDs now changed on the card.

//Enable the resets

//Reset the circuit and clear the PCI FIFOs for the card
DIME_CardResetControl(hCard1,drONBOARDFPGA, drENABLE,0);
DIME_CardResetControl(hCard1,drSYSTEM, drENABLE,0);
DIME_CardResetControl(hCard1,drINTERFACE, drTOGGLE,0);

//Set the clocks
DIME_SetOscillatorFrequency(hCard1,1,80.000,&ActualFrequency);
printf("ClkA: Actual frequency is %.

DIME_SetOscillatorFrequency(hCard1,2,40.000,&ActualFrequency);
printf("ClkB: Actual frequency is %.

DIME_SetOscillatorFrequency(hCard1,3,50.000,&ActualFrequency);
printf("ClkB: Actual frequency is %.

//Configure the FPGAs
//Now configure the modules primary FPGA with the bif file.
DIME_ConfigDevice(hCard1,Filename2,ModuleNumber,SecondaryFPGA
DeviceNum,0,0);
//Now configure the modules primary FPGA with the bif file.
DIME_ConfigDevice(hCard1,Filename1,ModuleNumber,PrimaryFPGADe
viceNum,0,0);

//Disable the resets
DIME_CardResetControl(hCard1,drONBOARDFPGA, drDISABLE,0);
DIME_CardResetControl(hCard1,drSYSTEM, drDISABLE,0);

Sleep(500);
//Enable the resets
DIME_CardResetControl(hCard1,drONBOARDFPGA, drENABLE,0);
DIME_CardResetControl(hCard1,drSYSTEM, drENABLE,0);
DIME_CardResetControl(hCard1,drINTERFACE, drTOGGLE,0);
Sleep(500);
//Disable the resets
DIME_CardResetControl(hCard1,drONBOARDFPGA, drDISABLE,0);
DIME_CardResetControl(hCard1,drSYSTEM, drDISABLE,0);
Sleep(500); //Give it a couple second for the embedded proc to sort out the
DIME talk network i.e. reset control etc.

//Open the handle to vidime
viHandle = viDIME_Open(hCard1, 0);
```

The locate handle can then be used to fund out information on the cards that have been detected in the system. The DIME_LocateStatus function uses the locate handle and can retrieve information about the hardware such as motherboard types etc.

The next key step is to actually obtain a handle to a specific card in the system. This is done using the DIME_OpenCard function. Again this uses the locate handle but now returns a specific card handle.

Again status information can be obtained about a specific card given a specific card handle. The DIME_CardStatus function can be used for this.

Simple test of the host PC communication to the PCI or USB interface on the Kit hardware is to simply change the user status FPGAs on the motherboard.

Now enable the resets, set the clocks to the required frequencies and then configure each of the FPGAs whilst the system is in reset. Once the configuration is complete a standard reset sequence is followed to make sure the device and design properly start.

Now a handle can be obtained for the actual interface to the interface to user FPGA core. This is done using the viDIME_Open function.

<pre> Result = viDIME_DMAWrite(viHandle,WriteData,256,0, NULL, NULL, 5000); Result = viDIME_DMARead(viHandle,ReadData,256,0, NULL, NULL, 5000); error=0; for(j=0;j<256;j++) { if(WriteData[j] != ReadData[j]) { printf(" error = 1; } } if(error==0) printf(" else printf(" //Close the card viDIME_Close(viHandle); DIME_CloseCard(hCard1);//Closes down the card. //Finally the last thing that should be done is to close down the locate. DIME_CloseLocate(hLocate); printf (" getchar(); return 0; </pre>	<p><i>With this handle data can then be transferred and then read back to the host using functions such as viDIME_DMAWrite.</i></p> <p><i>Finally a simple check is carried out to check that the readback data matches the written data.</i></p> <p><i>Close down the viDIME handle to free resources on the host.</i></p> <p><i>Then close the card handle to free allocated resources.</i></p>
---	---

17.5 Relevant Application Notes

There are a number of relevant application notes on the supplied CDs, with additional examples. This section lists the application notes that may be of interest, with a brief description of their contents.

- XtremeDSP Kit Ping Example
 - Provides another example of building a host interface to a design on the main User FPGA.
- XtremeDSP Kit Analog Capture
 - Provides an example of loading data to memory in the design where stimulus from the DACs is captured via the ADCs and read back to the host. This example also makes use of the FUSE Toolbox for MATLAB® which allows the FUSE API to be used directly within MATLAB scripts.
- ZBT Controller
 - Gives details of a Nallatech ZBT core.
- XtremeDSP Kit ZBT Design
 - Gives details on using the Nallatech ZBT controller core to interface to the ZBT used in the Kit.



NALLATECH
The High Performance FPGA Solutions Company

Part VI: Xilinx ISE Support

This part of the User Guide provides information on common Xilinx ISE settings for FPGA designs running on the XtremeDSP Development Kit Pro hardware.

Section 18

Common ISE Settings

In this section:

- Synthesis and Implementation Settings
-

18.1 Synthesis and Implementation Settings

This section details the synthesis and implementation settings, which should be used for the development of FPGA designs to run on Nallatech hardware.

18.1.1 Synthesis Options

When developing FPGA designs to run on Nallatech hardware, it is not necessary to select any specific settings for the synthesis of HDL code for FPGA designs.

18.1.2 Implementation Options

Developing FPGA designs to run on Nallatech hardware with on board Xilinx FPGAs will ultimately require use of the Xilinx Implementation tools to take the synthesized design to the hardware device. These tools are available in a variety of formats, including Alliance, Foundation and ISE.

When performing the implementation stage of the design process, some settings are mandatory and need to be specified for the design to configure and run on Nallatech hardware. This section details these settings.

Necessary Settings

1. Enable Readback and Reconfiguration
2. Select the JTAG Startup Clock (selected configuration clock)

These settings are easily accessed and set in the Xilinx ISE Foundation tools. The following screen captures are taken from the ISE 6.3i release.

If you right click on the process for 'Generate Programming File', go to properties on the pop-up menu and then select the tab for 'Startup Options'. You will see the options shown in [Figure 75 on page 150](#).

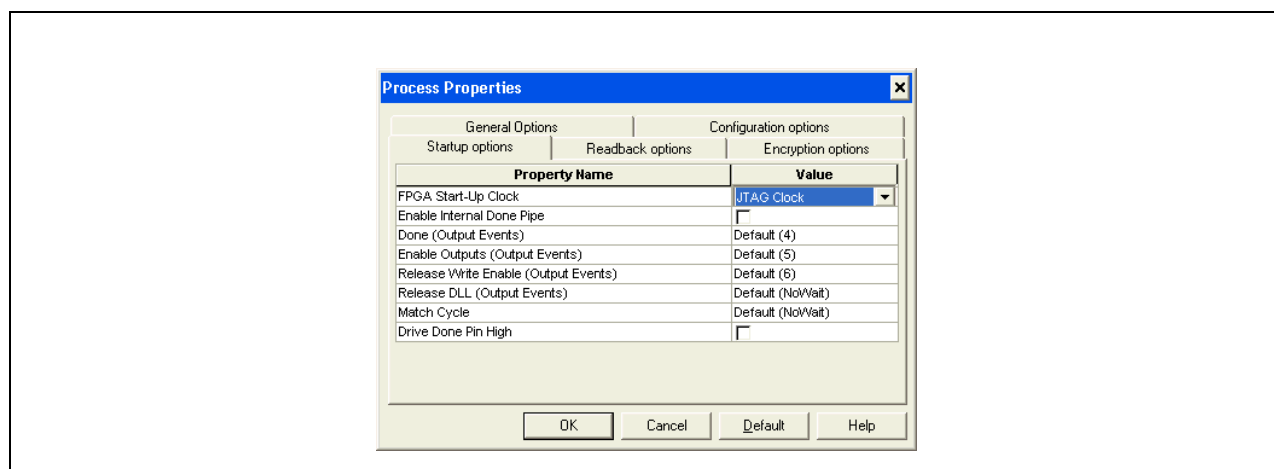


Figure 75: Startup Options

Simply click on the pull-down menu for the 'FPGA Startup Clock' and select '**JTAG Clock**'. Now click on the 'Readback options' tab. You will see the options shown in [Figure 76 on page 150](#).

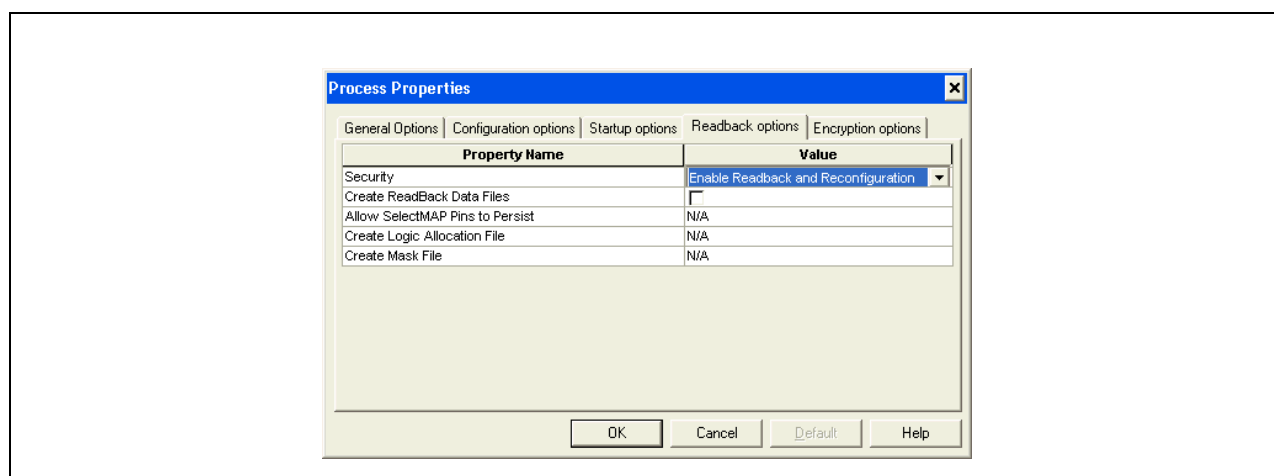


Figure 76: Startup Options

If it is not already selected, click on the pull-down menu for 'Security' and select '**Enable Readback and Reconfiguration**'.



Failure to enable readback and reconfiguration will result in the value being returned from the Virtex-II Pro status register of 0x0 during the configuration sequence which will be flagged as an error. This will manifest itself in the configuration software reporting an error of DONE LOW, INIT LOW, No CRC errors.



If reconfiguration has not been enabled and you configure once, it is necessary to cycle the power to the FPGAs in order to clear the security protection on the FPGA.



NALLATECH
The High Performance FPGA Solutions Company

Part VII: Xilinx Impact Support

This part of the User Guide provides details on how to use the Xilinx Impact Support Tool in conjunction with the XtremeDSP Development Kit Pro.

Section 19

Impact Support

In this section:

- Support for the Xilinx Impact Tool.
- How to Establish a Connection
- Program User FPGAs via a Download Cable

19.1 Introduction

The Xilinx Impact tool can be used to configure the User FPGAs in the XtremeDSP Development Kit Pro. This section details how the Impact tool can be used with the Kit.

19.2 Set up a Connection

19.2.1 Connect a JTAG Download Cable

A download cable is required in order to connect to the general JTAG chain through which the User FPGAs are configured. Supported cables are the Parallel-III or Parallel-IV cables from Xilinx, which can be connected to the two headers on the board. One header supports flying lead connections from either pod and the other supports the faster ribbon. Details of these headers are included on [page 95](#).

19.2.2 Open the Card to Enable Power Supplies

In order to detect all the devices in the JTAG chain the module power supplies must be enabled. The power supplies for the module are on when the power good LEDs for PSUB to D are green rather than red. Please refer to [page 100](#) for more details on the module power supplies.

If using the Kit standalone with the external power supply

To enable the power supplies when using the external power supply, jumper (J17) must be fitted. This allows the module power supplies to come on as soon as power is supplied to the board. It also sets both CLK A and CLK B to 40MHz.

If using the Kit in a PCI slot

To enable the power supplies when using the Kit in a PCI slot, open the Kit in the FUSE Probe Tool as shown in [Figure 77 on page 156](#).



Using the Kit in a PCI slot provides greater bandwidth and therefore faster cosimulation for the Kit.

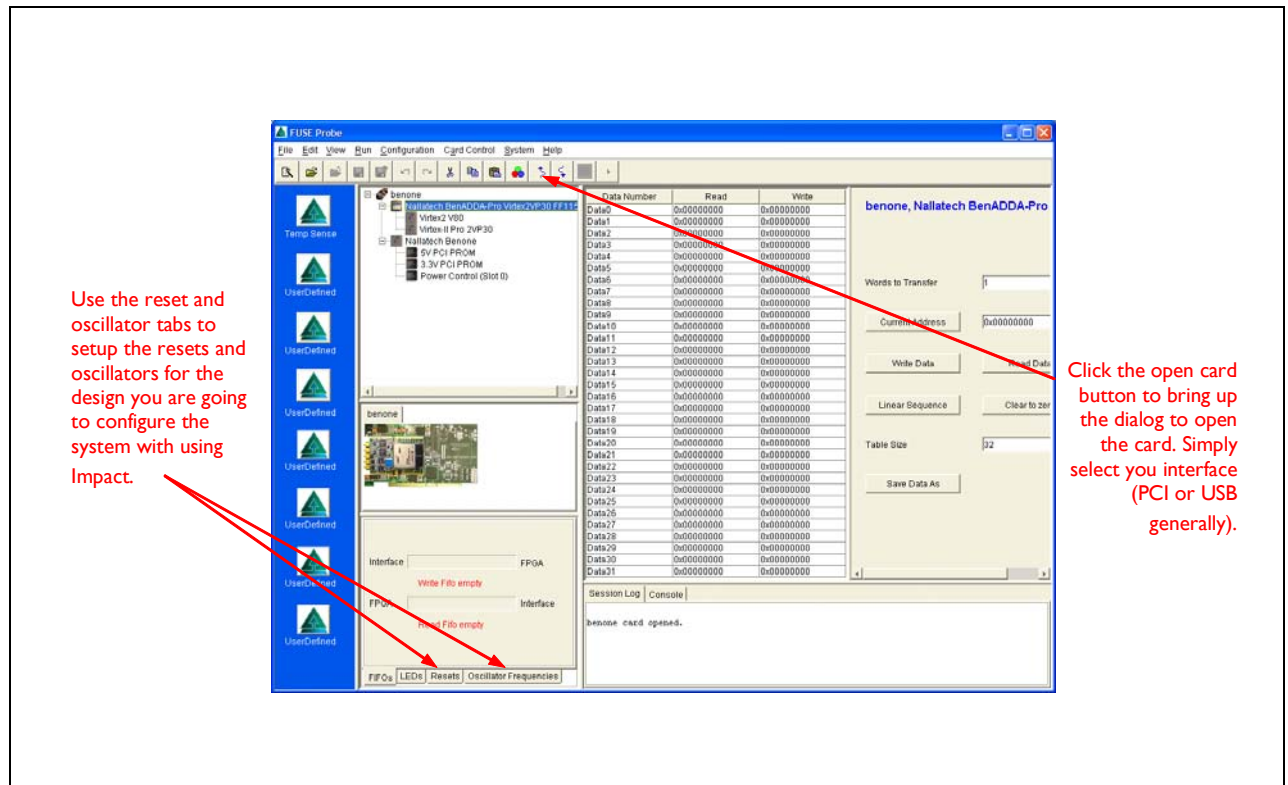


Figure 77: Open Card Enabling Module Power Supplies

▼ **After opening the card in FUSE carry out the following procedures:**

1. Enable the resets to the cards by checking the reset box. You should enable the resets during configuration and release them once all devices have been configured.
2. Set the oscillators to the required frequencies before configuring your design.

19.2.3 Alternative Method of Enabling Power Supplies

To enable the power supplies when using the external power supply jumper (J17) must be fitted. This allows the module power supplies to come on as soon as power is supplied to the board.



The power supplies for the module are on when the power good LEDs for PSUB to D are green rather than red. Please refer to [page 102](#) for more details on the module power supplies.



Note that the power supplies are not closed down when the card is closed in FUSE. Therefore, you only need open the card once during initial mains power on to enable the module power supplies. However, you still need to open a card in FUSE to control the oscillators and the resets.

19.2.4 Launch Impact

The Impact tool can be launched from the Xilinx ISE->Accessories program group. On startup you are presented with a selection of the operation mode as shown in [Figure 78 on page 157](#).

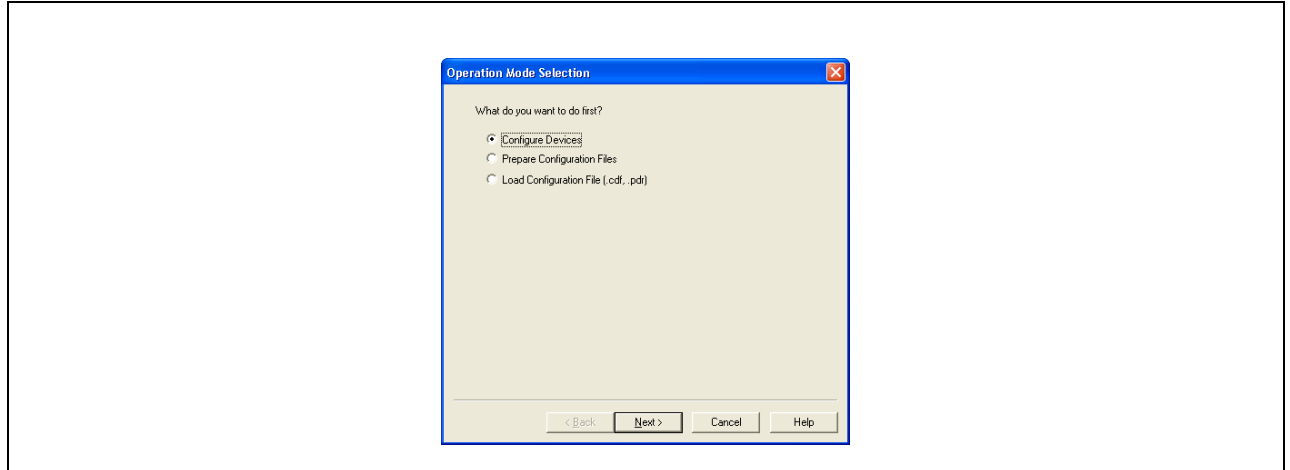


Figure 78: Operation Mode Selection

To configure devices in the general JTAG chain, select '**Configure Devices**'. Then click '**Next**' to continue.

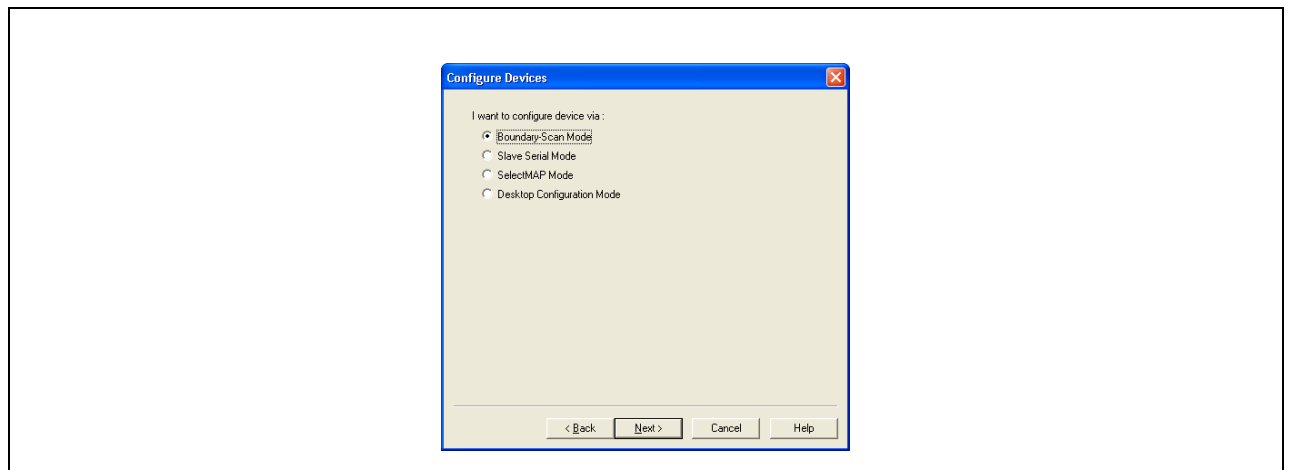


Figure 79: Configure Devices Dialog

Select '**Boundary Scan Mode**' as we are using JTAG to configure the devices on the hardware. Click on '**Next**' to continue.

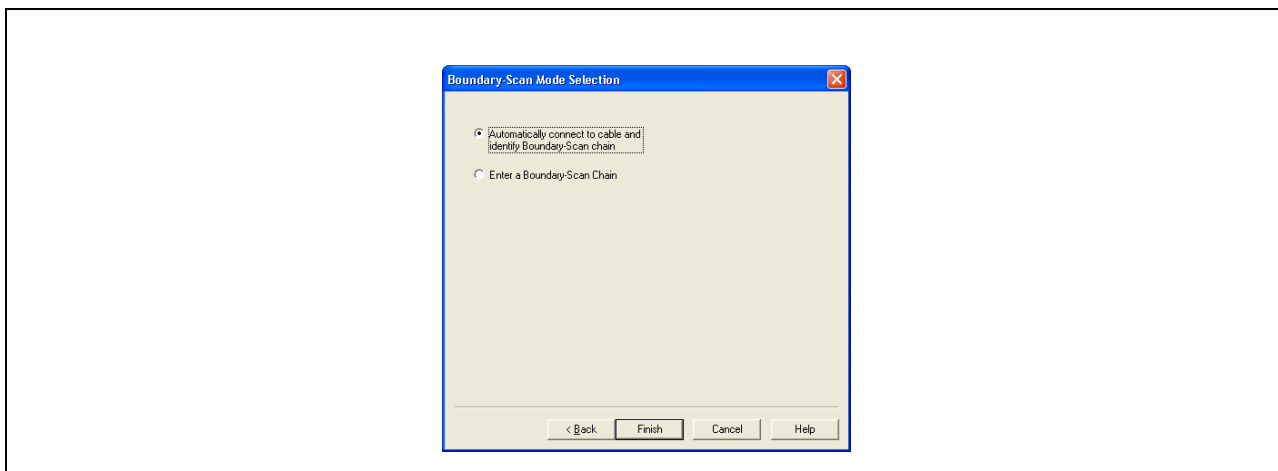


Figure 80: Boundary Scan Mode Selection

Select to automatically connect to and identify boundary scan chain. Click '**Next**' to continue. You will then see the scan of the JTAG devices running, as shown in **Figure 81** on page 158..

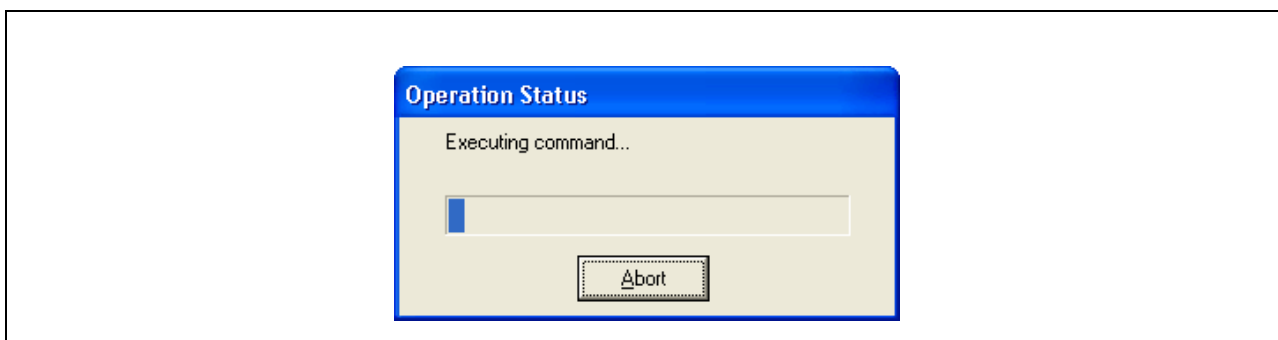


Figure 81: Scanning for Devices

Once this scan completes, five devices should be shown in the chain..

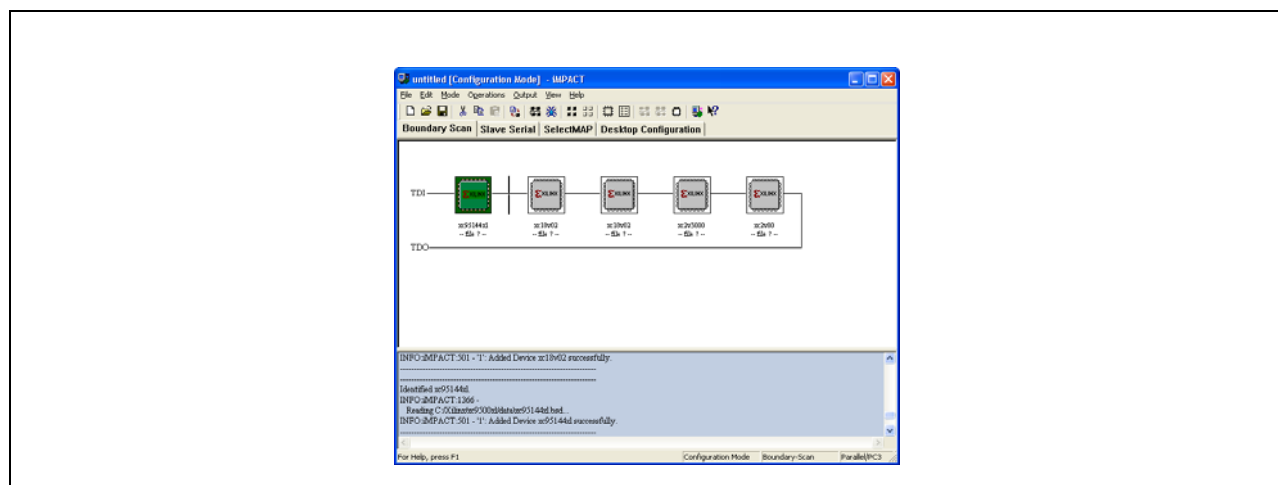


Figure 82: Showing Detected Devices

Impact will then prompt for configuration files for each device in the chain. It is not necessary to assign configuration files at this time as this can be carried out manually.



Please note that the XC95144XL CPLD should not be programmed and nor should XC18V02 PROM unless a manual firmware update is being carried out.

Setting Cable Speeds

One aspect to the support for different cables is that different configuration rates are supported by different cable leads. The Parallel-IV pod can support a ribbon cable header or a flying lead header. The ribbon cable, due to it's nature, can operate at a higher rate than the flying lead connections. Therefore, if you are using flying leads ensure that a suitable rate is being selected.

Go to the main menu in Impact, '**Edit->Preferences**'. This brings up the dialog shown in **Figure 83 on page 159**.

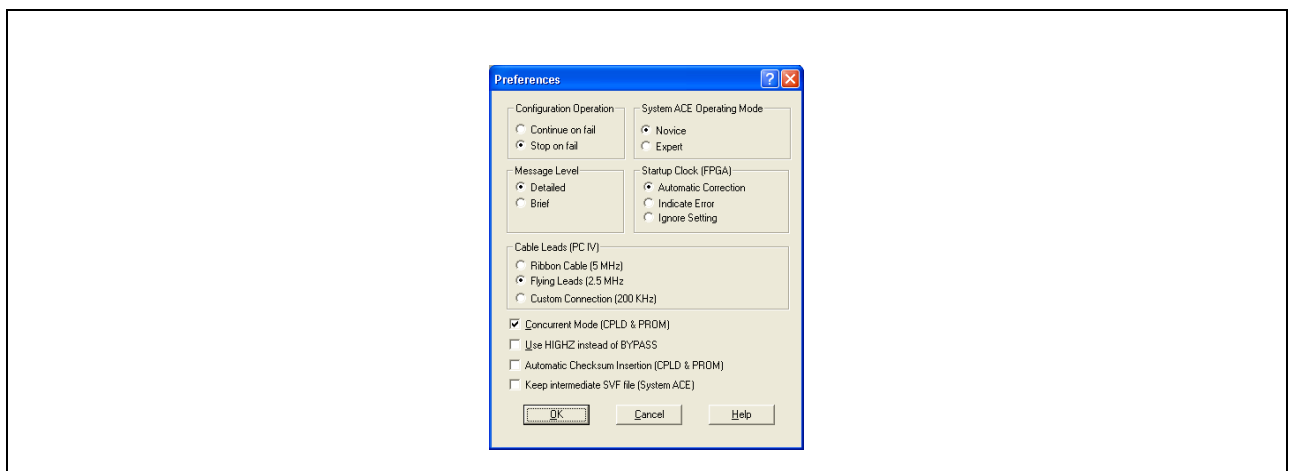


Figure 83: Setting Cable Preferences

There is a section for 'Cable Leads (PC-IV)'. If you are using the flying leads this should be set to 2.5MHz, if you are using the ribbon cable this may be set to a higher setting.

Assigning Bitfiles and Configuring

To assign bitfiles to the Clock FPGA and the main User FPGA, right click on each device and select '**Assign New Configuration File**'. This brings up a dialog to browse to the bitfile of your choice. Please note that you may wish to change the 'Files of type' to *.bit unless already selected.

After assigning bitfiles you should now configure the devices. To do this right click on each device in turn and select **'Program'**. This will bring up the programming dialog shown in **Figure 84 on page 160**.

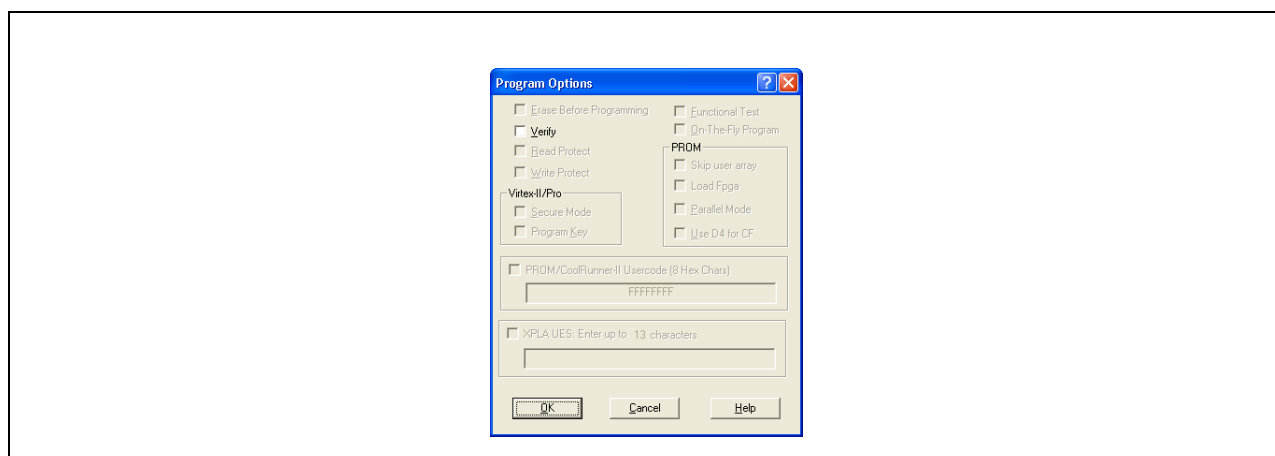


Figure 84: Programming Dialog

Click **'OK'** to start the programming. You will see the operation being executed.



NALLATECH
The High Performance FPGA Solutions Company

Part VIII:Chipscope ILA Support

This part of the User Guide provides details on how to use the Xilinx Chipscope ILA Tool in conjunction with the XtremeDSP Development Kit Pro.

Section 20

Chipscope ILA Support

In this section:

- Introduction
 - Support for the Xilinx Chipscope ILA Tool
 - How to Establish a Connection
 - Using Chipscope with the XtremeDSP Development Kit Pro
-

20.1 Introduction

Xilinx supply an optional tool called Chipscope Pro which is not included in the Kit. However details of its use are included here as some users may have access to Chipscope and may wish to use it on the Kit. This section provides information on using Chipscope through a worked example that is included on the XtremeDSP Development Kit Pro CD in the folder: 'CD ROM Drive\Examples\chipscope_example'.

Full details on the Chipscope tool are provided in the *Chipscope User Guide* supplied with the product. Please refer to this User Guide for more details on the use of Chipscope.



Note that details on how to make use of Chipscope within the Xilinx System Generator tool are contained in **"Part X:Xilinx System Generator Support" on page 189.**

20.2 Connect a JTAG Download Cable

A download cable is required in order to connect to the general JTAG chain through which the User FPGAs are configured. Supported cables are the Parallel-III or Parallel-IV cables from Xilinx, which can be connected to the two headers on the board. One header supports flying lead connections from either pod and the other supports the faster ribbon. Details of these headers are included on **page 215** and **page 216.**

20.3 XtremeDSP Development Kit Pro Chipscope Example

20.3.1 Overview

This example is based around a simple 'ledsnake' design which consists of a counter connected to available LEDs, with the remaining bits connected to a header (which can be studied on a logic analyzer). The following steps should be followed to add Chipscope to a design:

- Create the ICON and ILA components using the corresponding Chipscope applications. These are effectively a controller component (ICON) and a data capture component (ILA).
- Manually instantiate the generated components in the design and connect them up appropriately. Alternatively the Chipscope Pro inserted tool can be used to manually insert components into existing designs.
- Connect to the design using a JTAG download cable, such as the Parallel-III or Parallel-IV, and analyze the design using the Chipscope Pro Analyzer.



In this example Chipscope Pro 6.3i has been used. However, similar procedures can be used for more recent versions of the tool.

20.3.2 Creating the ICON and ILA Cores

Chipscope Pro provides a tool called 'Chipscope Pro Core Generator'. This can be launched from the start menu as shown in [Figure 85 on page 164](#).

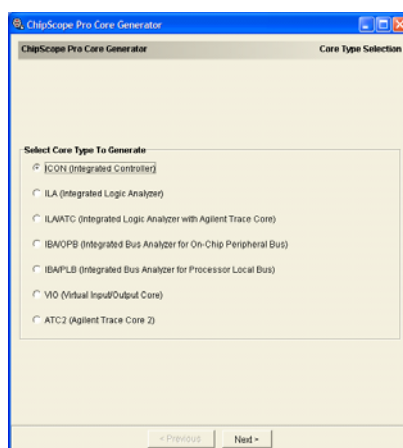


Figure 85: Chipscope Pro Core Generator

Generate the ICON Core

This first component to generate is the '**ICON (Integrated Controller)**' component. Select it and click '**Next**'.

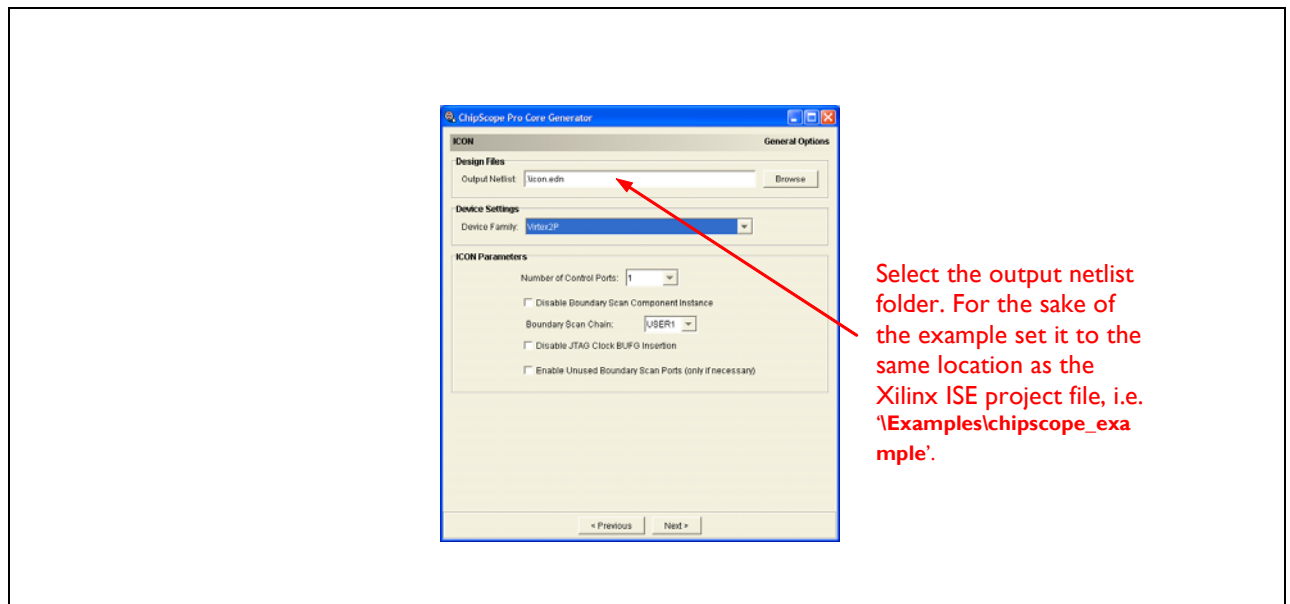


Figure 86: ICON Generation Options I

On the next dialog (Figure 87 on page 165) leave the settings as '**VHDL**', provided you are using VHDL. Then click '**Generate Core**' to produce the ICON core.

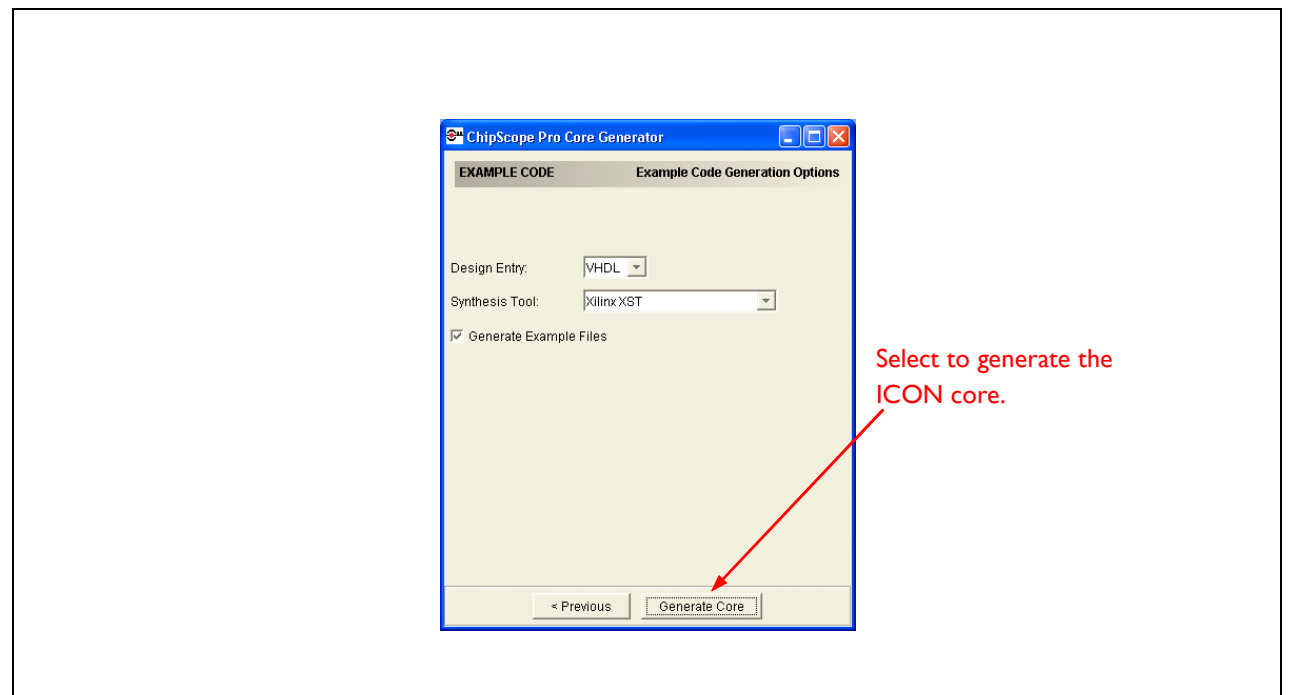
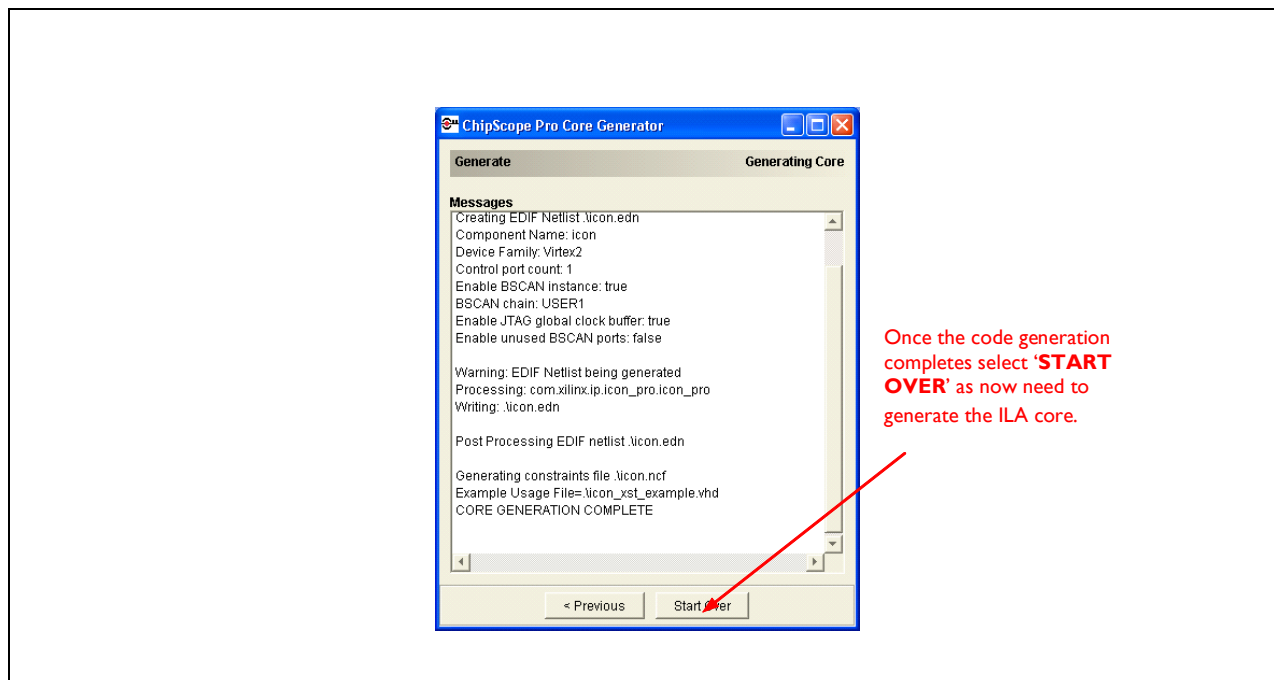


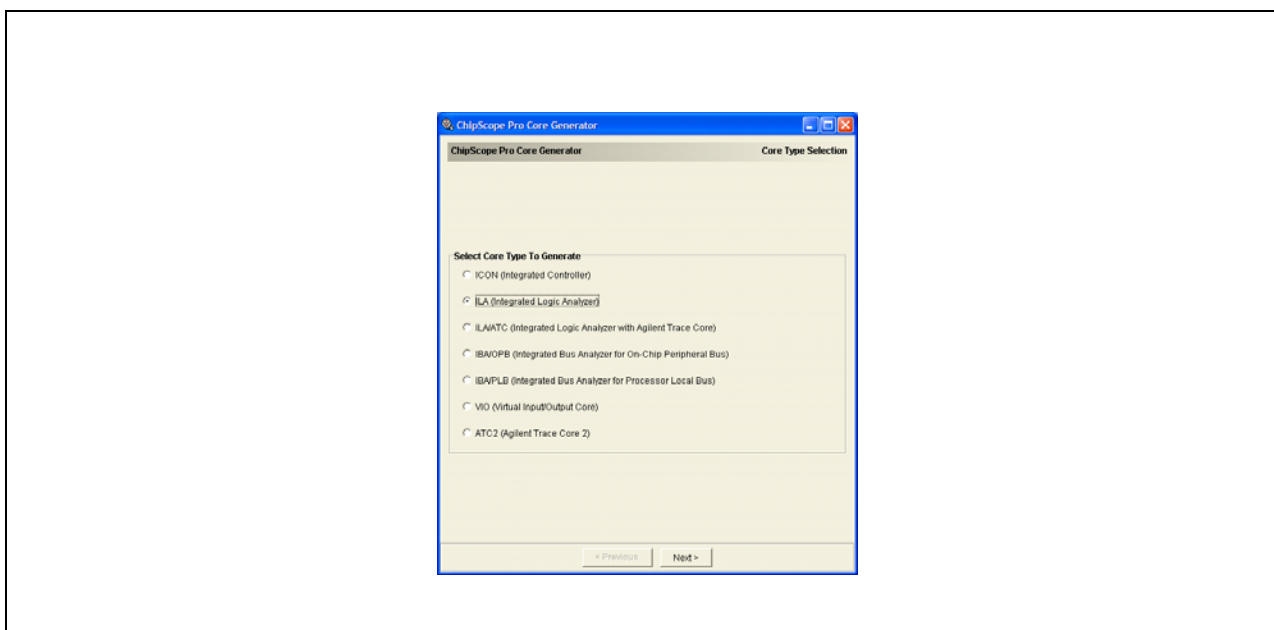
Figure 87: ICON Generation Options II

**Figure 88: Actual ICON Component Generation**

Once the ICON component has been generated (Figure 88 on page 166), click on 'Start Over' to generate the ILA component.

Generate the ILA Core

The second component to generate is the ILA (Integrated Logic Analyzer). In the screen shown in Figure 89 on page 166 select 'ILA' as the type.

**Figure 89: Actual ICON Component Generation**

Click '**Next**' to continue to the settings for the ILA component and set the output netlist folder as shown in **Figure 90** on page 167.

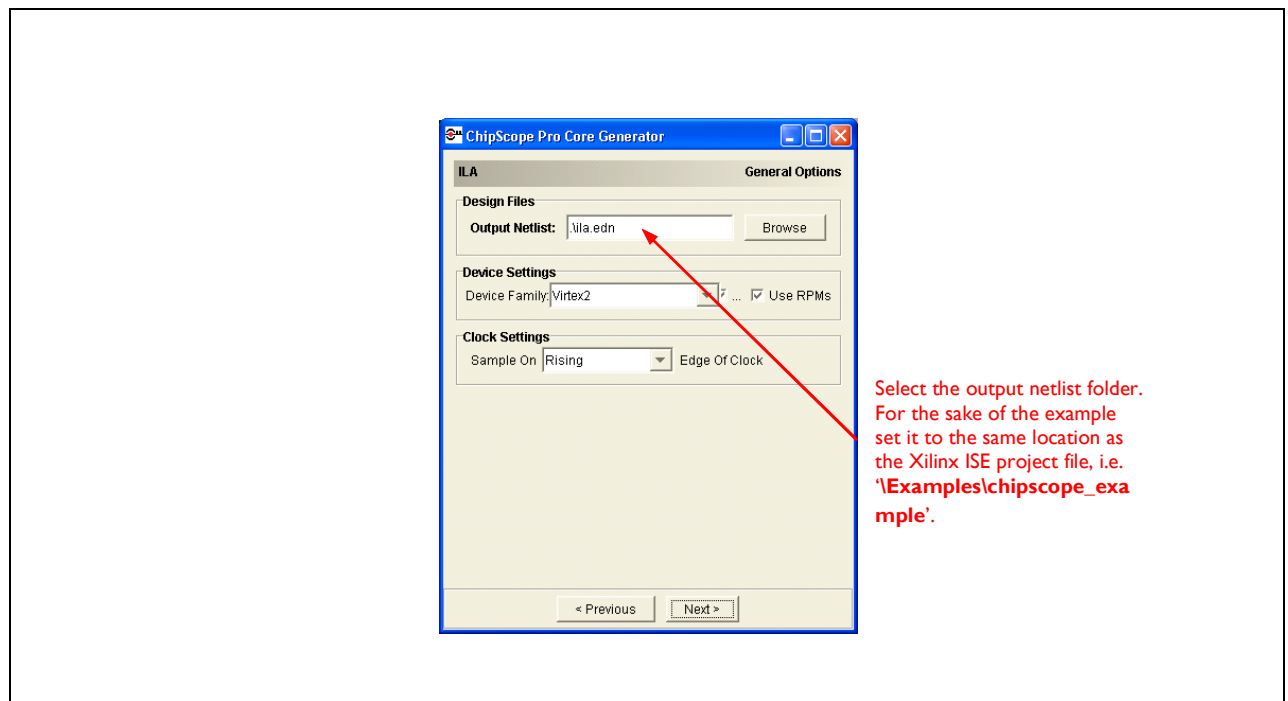


Figure 90: ICON Component Settings

Then set the trigger width and data depth settings as they are requested.

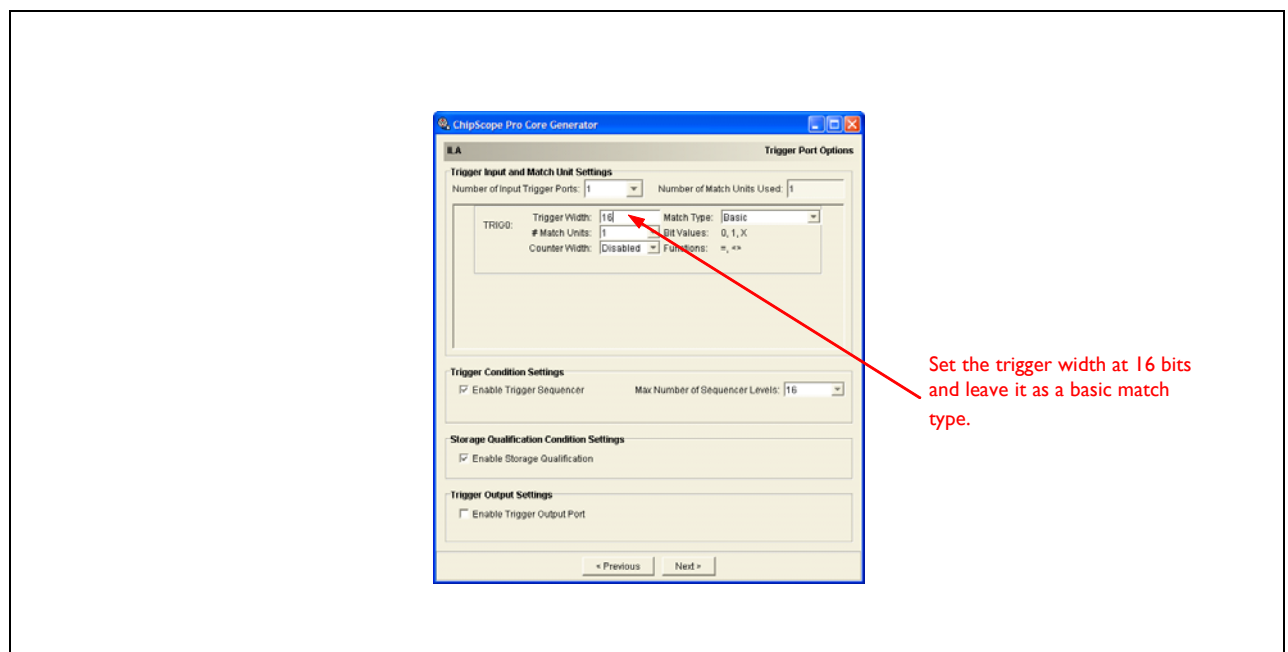
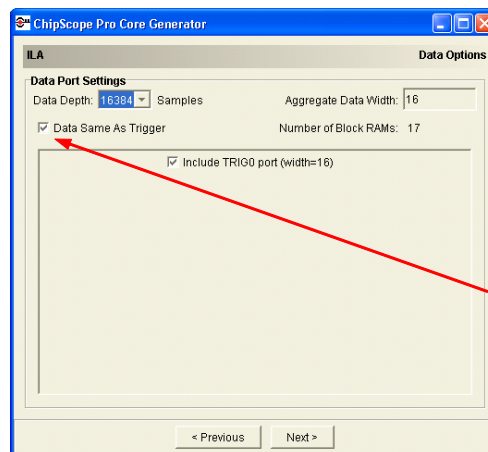
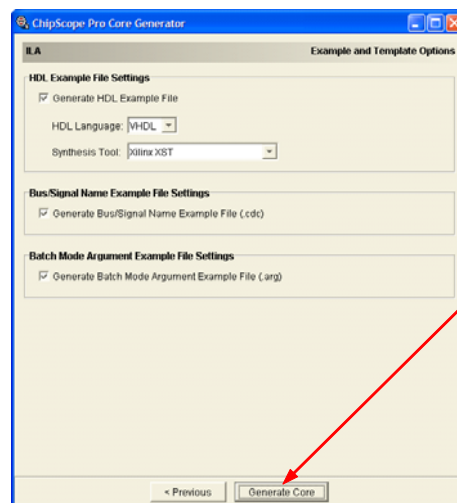


Figure 91: Setting ILA Trigger Width



Set the **Data Same as Trigger** to be checked. Also set the **Data Depth** to be 16384 samples.

Figure 92: Setting Data Capture Depth



Select to generate the ILA core.

Figure 93: Final ILA Settings

When '**Generate Core**' is selected the tool runs and produces the ICON core. Then click on '**Close Button**' to exit the Core Generator.

20.3.3 Manually Instantiating the ICON and ILA Cores in your Design

Once the cores have been created they should be inserted into the actual design you wish to monitor. In this example this is done manually in the VHDL through component declaration and instantiation of the ICON and ILA components. Details are provided in the following listing on [page 170](#).

Listing (ledsnake.vhd)

```

-- (c) Nallatech 1999
--
-- ledsnake.c
--
-- This applications snakes a running stream of lit LED
-- Intended as a confidence tester
--

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ledsnake is
  port (
    CLK : in std_logic;
    RST1 : in std_logic;
    LED : out std_logic_vector(47 downto 0)
  );
end ledsnake;

architecture ledsnake_arch of ledsnake is

  --
  -- ICON Pro core component declaration
  --

  component icon
  port
  (
    control0 : out std_logic_vector(35 downto 0)
  );
end component;

  --
  -- ILA Pro core component declaration
  --

  component ila
  port
  (
    control : in std_logic_vector(35 downto 0);
    clk : in std_logic;
    trig0 : in std_logic_vector(15 downto 0)
  );
end component;

  signal LEDi : std_logic_vector(47 downto 0);
  signal COUNT : integer;

  --
  -- ILA Pro core signal declarations
  --

  signal control : std_logic_vector(35 downto 0);
  signal trig0 : std_logic_vector(15 downto 0);

begin

  --
  -- ICON Pro core instance
  --

  i_icon : icon
  port map
  (
    control0 => control
  );

  --
  -- ILA Pro core instance
  --

  i_ila : ila
  port map
  (
    control => control,
    clk => CLK,
    trig0 => trig0
  );

  trig0 <= LEDi(15 downto 0);

  --Invert as active low LEDs
  LED <= not LEDi;

  DOSNAKE : process(RST1, CLK)
  begin
    if RST1 = '0' then -- active low
      LEDi <= (others => '0');
    elsif CLK'event and CLK = '1' then
      LEDi <= LEDi+1;
    end if;
  end process DOSNAKE;
end ledsnake_arch;

```

Declare the components for the ICON controller and the ILA logic analyzer.

Instantiate the ICON core

Instantiate the ILA core

Connect the trigger port to the internal LEDs signal.

Invert the internal LEDs signal and connect to the actual LEDs port on the top level.

Create a process that produces the actual counter that creates the ledsnake in the design.

Once this code has been updated the actual design needs to be implemented. An ISE project has already been created called '**chipscope_example.npl**', and is included on the XtremeDSP Development Kit Pro CD in the following location: 'CD ROM Drive'\Examples\chipscope_example'. A bitfile has also been created and is included in the top level of the Chipscope example called '**ledsnake.bit**'.

20.3.4 Analyzing the Design

Once the bitfile has been created the design must be analyzed using the Chipscope Pro Analyzer. Follow the steps below to analyze the 'ledsnake' design.

Setup the Design Using FUSE Probe Tool

The simplest way to setup the design in the FPGA is to use the Nallatech FUSE Probe Tool. This is opened either from the Windows start menu or by double clicking on the Nallatech icon in the Windows task bar.

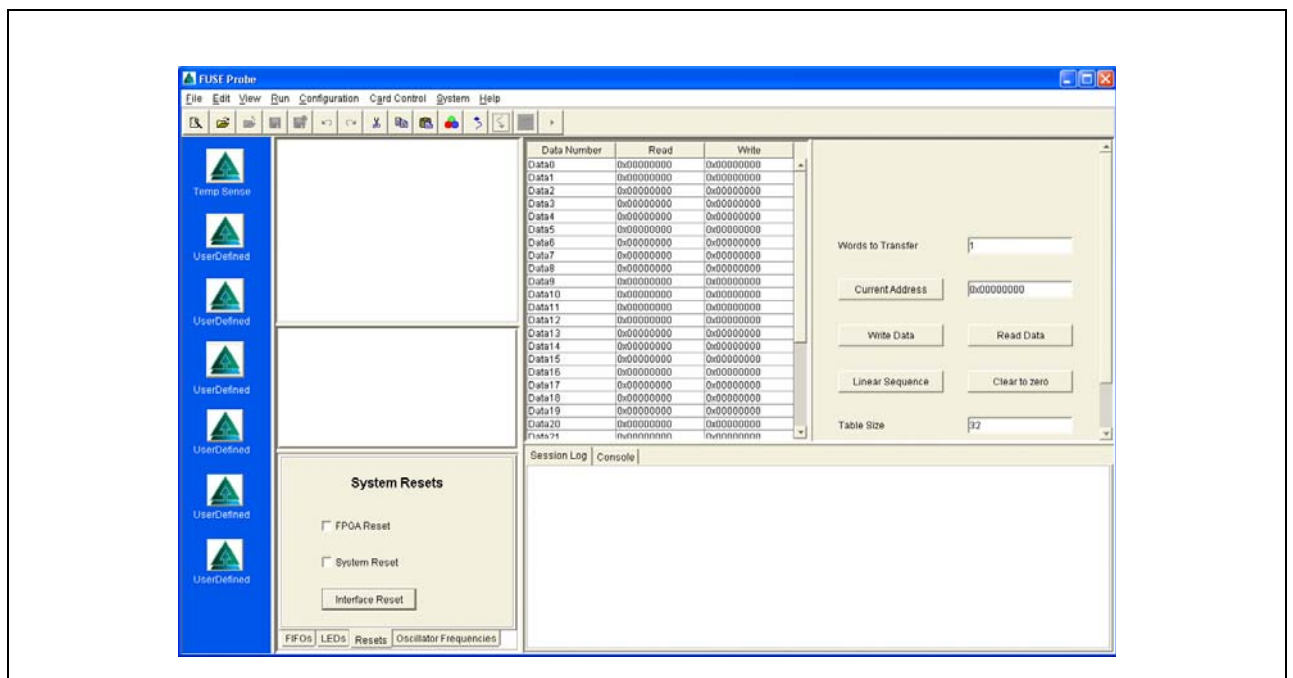


Figure 94: Initial FUSE Probe Tool Window

Once the FUSE Probe Tool has appeared, the card should be opened for access. To do this click on the **'Open Card'** button and select the appropriate interface, i.e. PCI or USB.

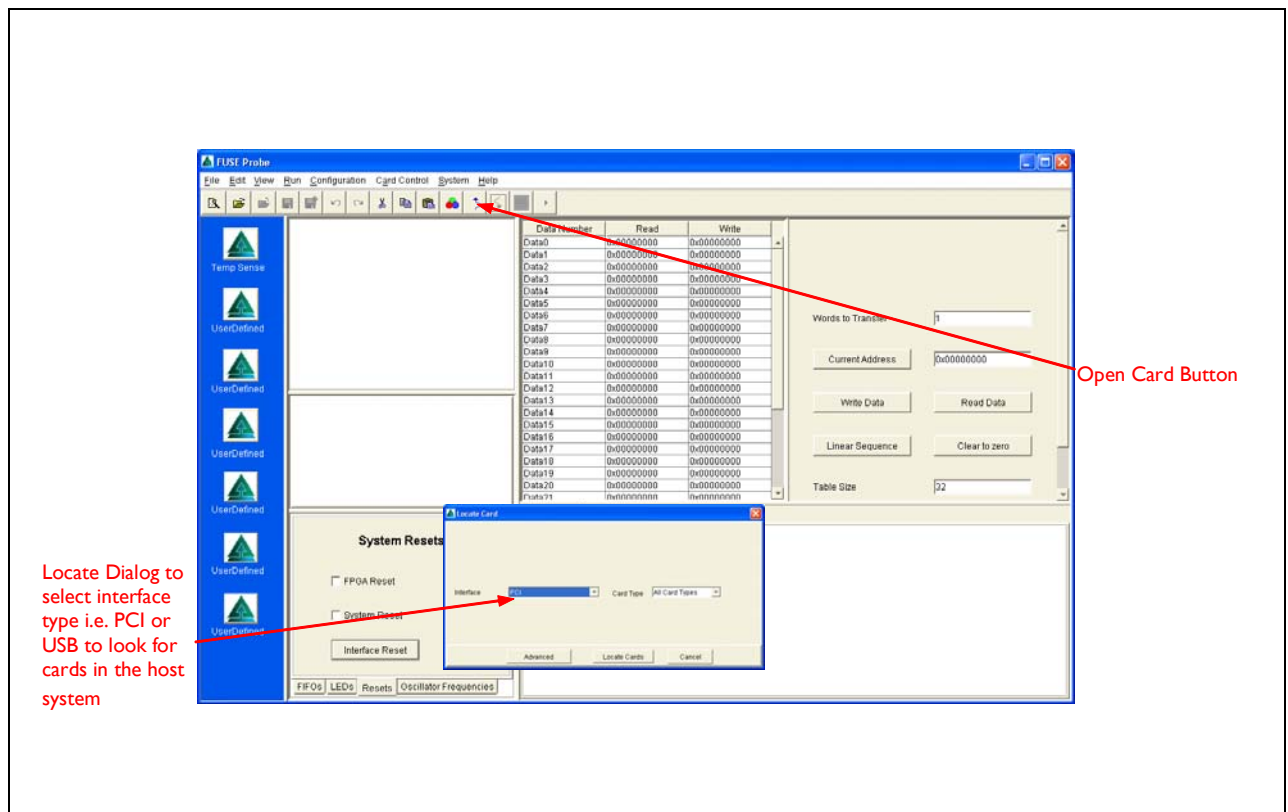


Figure 95: Open Card and Select Interface

FUSE then detects all cards in the system and returns a list of those detected on the specific interface.

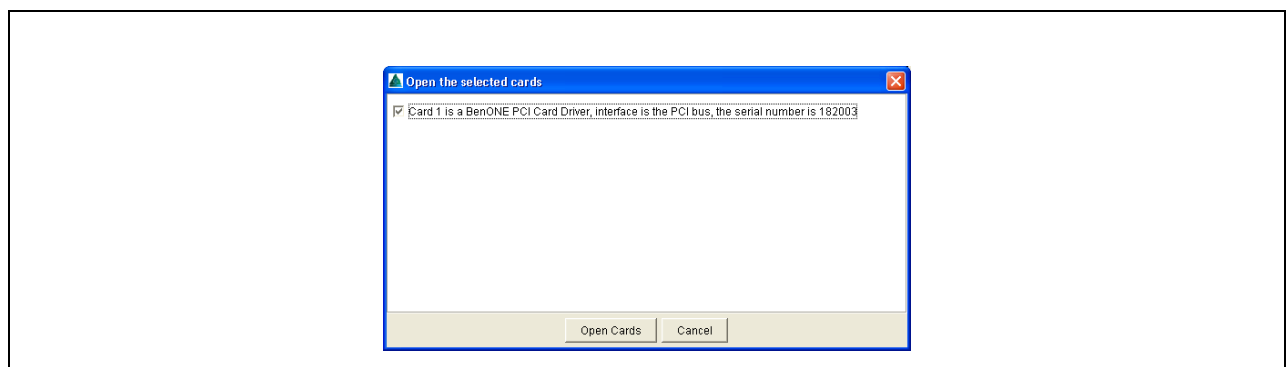


Figure 96: Open Card Dialog

Select a specific card and then click the **'Open Cards'** button to open the card in FUSE as shown in **Figure 97** on page 173.

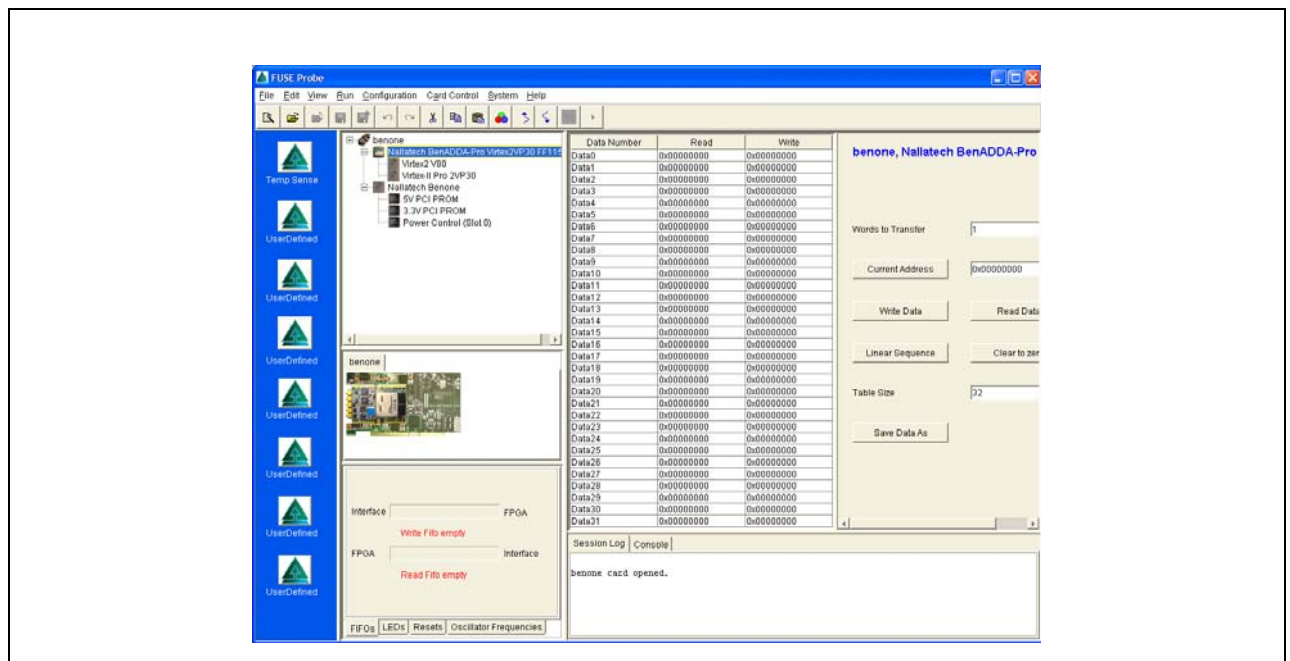


Figure 97: XtremeDSP Development Kit Pro Hardware Opened in FUSE

Now assign the **'ledsnake.bit'** bitfile to the main User FPGA (XC2VP30-4FF1152). To do this, right click on the Virtex-II Pro XC2VP30-4FF1152 device listed in the device tree. Then select **'Assign Bitfile'** to bring up a dialog to browse to the bitfile. Select the ledsnake bitfile from 'Examples\chipscope_example' folder. Now click on the Resets Tab and enable all the resets, as shown in **Figure 98** on page 173.

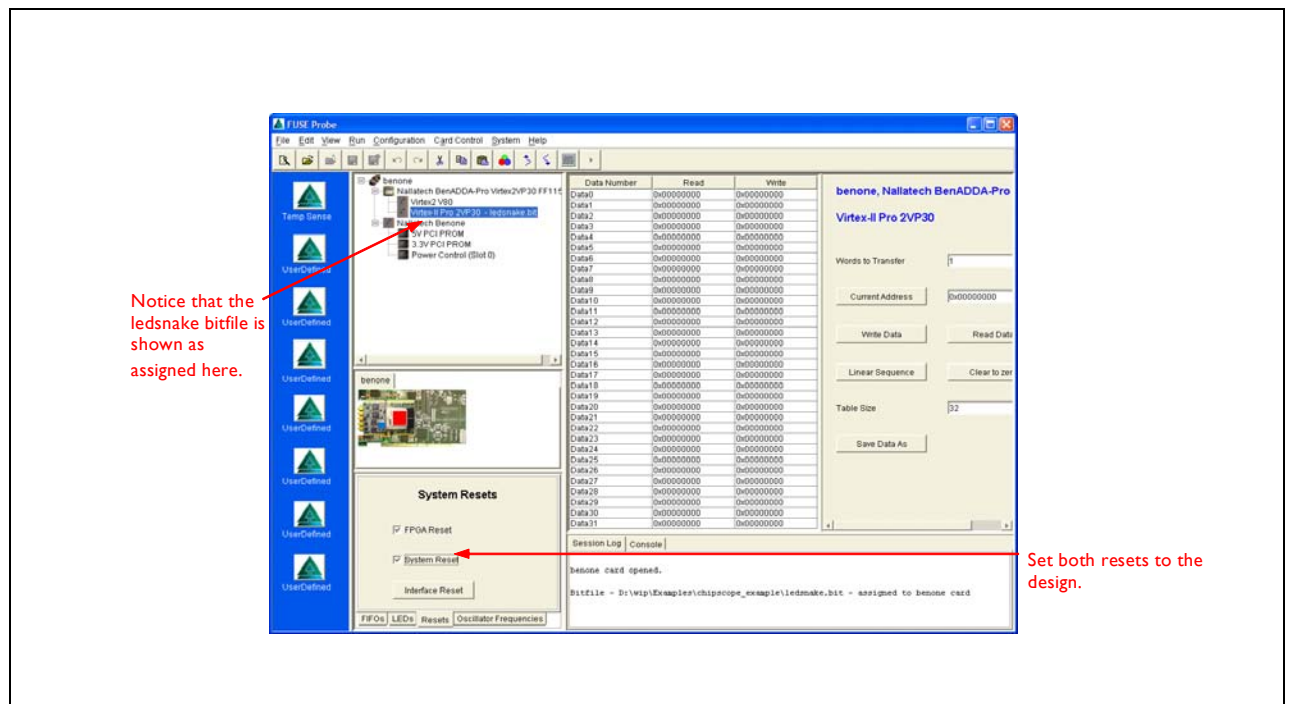


Figure 98: Enable Resets

Once the resets are enabled, the clocks should be set. In this example only one of the clocks is used and so only one clock is set.

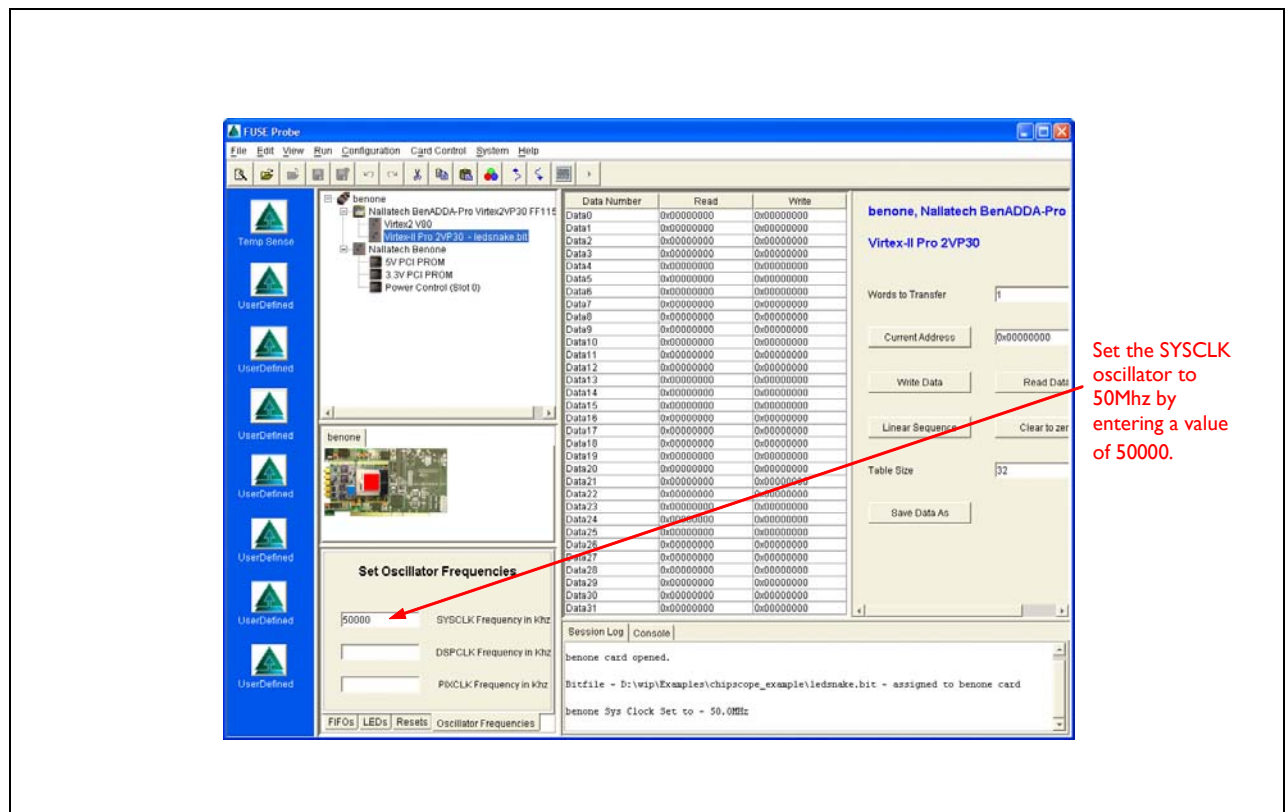


Figure 99: Setting the Oscillator

Now configure the main User FPGA by right clicking on the Virtex-II Pro XC2VP30-4FFI I52 and configure the device. After a delay, which is significantly longer on USB, a message is returned reporting the configuration of the device.

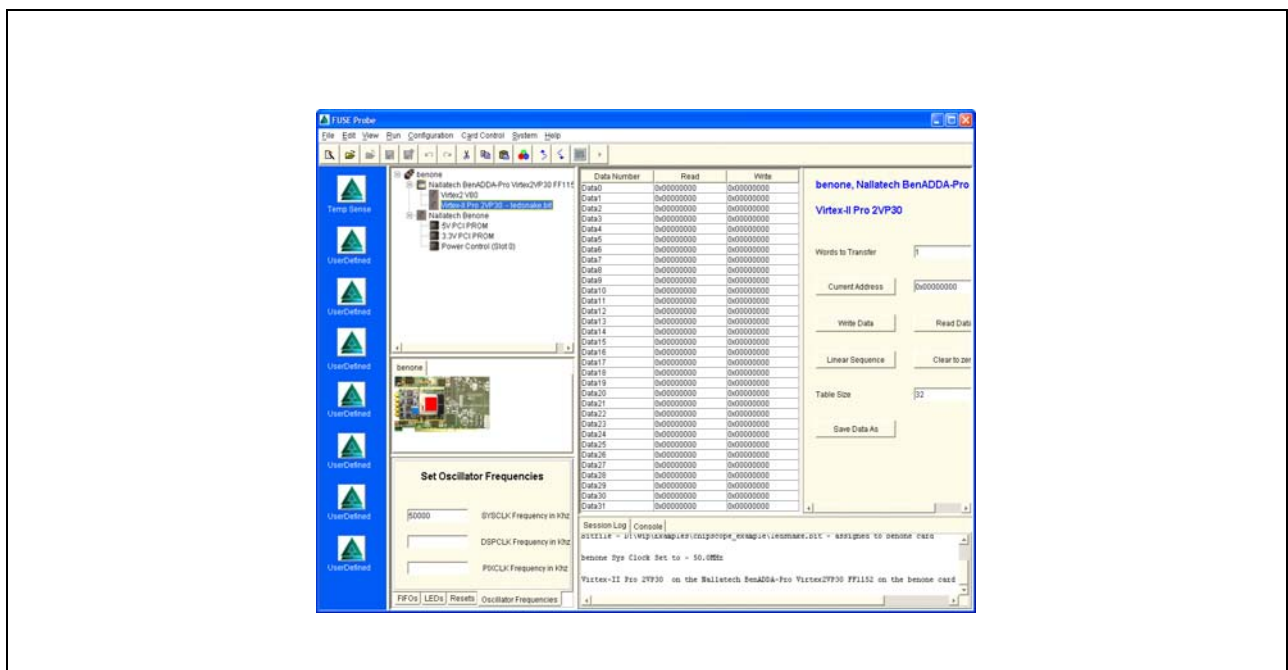


Figure 100: Log Configuration Result

Finally, go back to the **Resets** tab and do the following:

- Uncheck FPGA Reset
- Uncheck System Reset
- Click the '**Interface Reset**' button
- Check the FPGA Reset again
- Check the System Reset again
- Uncheck the System Reset again
- Uncheck the FPGA Reset again

Chipscope Pro Analyzer

The design is now ready to be examined in the Chipscope Pro Analyser tool. This can be opened from the Windows Start menu.

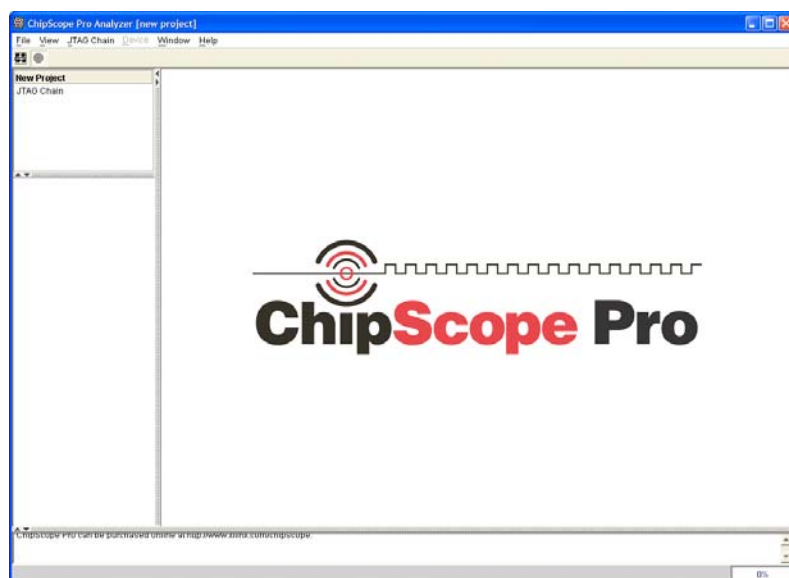


Figure 101: Initial Chipscope Pro Analyzer Start-up screen

Once the analyzer has started it must now establish a connection through the JTAG cable. Go to the menu and select **'Cable->Xilinx Parallel Cable'**. The tool scans the JTAG chain and provides a list of devices detected.

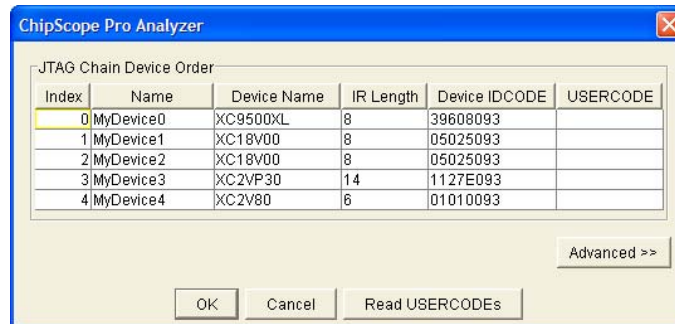


Figure 102: Detected Devices in Chain (should be 5)

Click **'OK'** to continue. The communication should now be established and the ILA units and signals are displayed. Further details on driving the Analyzer are provided in the Chipscope documentation.

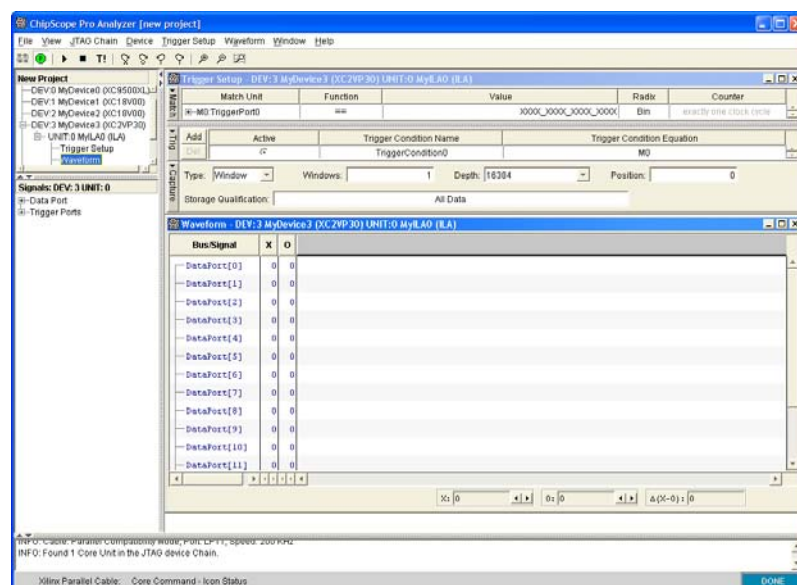


Figure 103: Connected to ILA Units

The design should now run on the chip - to test it request a capture of samples by clicking on the black arrow button in the top left toolbar in [Figure 104 on page 177](#). After a short delay the samples are shown in the waveform and you can see a counting sequence as expected.

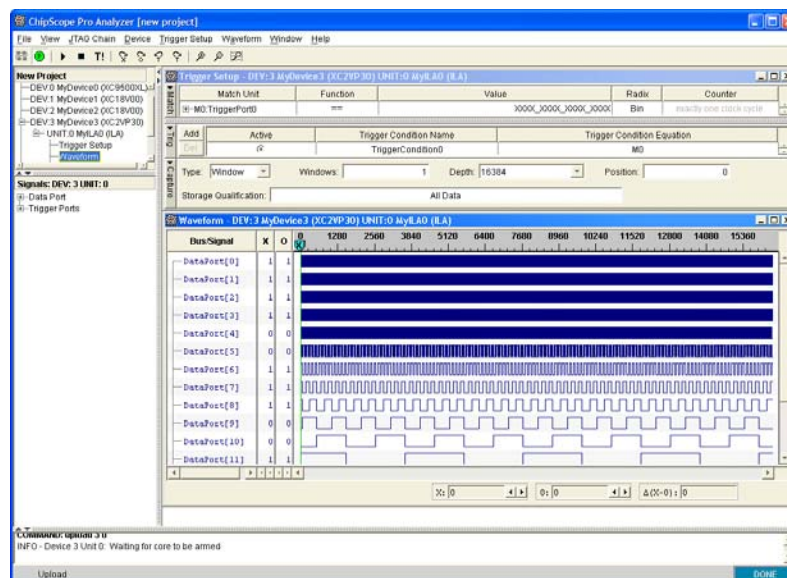


Figure 104: Captured Data



NALLATECH
The High Performance FPGA Solutions Company

Part IX: Xilinx Embedded Developer's Support

This part of the User Guide provides details on how to use the Xilinx Embedded Developer's Support Tool in conjunction with the XtremeDSP Development Kit Pro.

Section 2 I

Xilinx Embedded Developer's Kit (EDK) Support

In this section:

- Introduction
 - Design Notes
 - EDK Example
-

21.1 Introduction

The Xilinx Embedded Developers Kit is a set of tools which allows designers to create designs for Xilinx FPGAs that make use of either the Microblaze soft processor core or the embedded PowerPC core in the Virtex-II Pro silicon. The EDK tools are not included in the Kit and the details provided here are provided to support users who have access to the EDK tools and wish to make use of them with the Kit.

21.2 Design Notes

21.2.1 Resets

The resets that can be controlled from the host interface are all active-LOW which is important to consider when designing a system. You can create a microprocessor design as a submodule in a top level design, where the clock management and reset management are carried out.

21.2.2 Using the ZBT as a Peripheral

The ZBT can be connected to the embedded systems using the OPB EMC (External Memory Controller) peripheral provided in the EDK distribution. There are a number of parameters that need set for this controller as follows:

PARAMETER C_MEM0_WIDTH = 32

PARAMETER C_SYNCH_MEM_0 = 1

PARAMETER C_MAX_MEM_WIDTH = 32

The following pins are used to connect to the ZBT chips:

PORT Mem_DQ = fpga_0_Generic_External_Memory_Mem_DQ, which is a STD_LOGIC_VECTOR(31 downto 0) port.

PORT Mem_A = fpga_0_Generic_External_Memory_Mem_A, which is a STD_LOGIC_VECTOR(31 downto 0) port.

PORT Mem_WEN = fpga_0_Generic_External_Memory_Mem_WEN, which is a STD_LOGIC port.

PORT Mem_CKEN = fpga_0_Generic_External_Memory_Mem_CKEN, which is a STD_LOGIC port.

PORT Mem_OEN = fpga_0_Generic_External_Memory_Mem_OEN, which is a STD_LOGIC port.

PORT Mem_CEN = fpga_0_Generic_External_Memory_Mem_CEN, which is a STD_LOGIC port.

PORT Mem_ADV_LDN = fpga_0_Generic_External_Memory_Mem_ADV_LDN, which is a STD_LOGIC port.

When physically connecting the address bus from the EMC peripheral to the address lines of the ZBT note that some translation of the address pins is required. In the case of the XtremeDSP Development Kit Pro note the following mapping:

ZBT_A(0 to 18) <= fpga_0_Generic_External_Memory_Mem_A(11 to 29)



Note that this can simply be replicated for the second independent bank of ZBT memory on the board.

21.2.3 XMD Support

XMD is used to debug code running on embedded microprocessors. It is possible to connect to XMD via a JTAG or UART connection. The JTAG headers on the board can be used to provide XMD access via a JTAG connection, i.e. a JTAG_OPB_UART peripheral. Note that the main User FPGA is device 4 in the JTAG chain for XMD

Alternatively a serial port connection can be used. There is a Nallatech specific test header (J9 and J12) that also provides RS232 level shifters. No cable is provided in the Kit to connect to this header; however the pinouts are given in **"Part XII:Reference Information"** on page 229. The header allows a straight wiring connection to be made with the serial ports on a PC. The RS232 level shifters translate between the voltage specification of the FPGA and that of the serial ports on the PC.

21.3 EDK Example

A simple example is provided on the XtremeDSP Development Kit Pro CD in the following location: 'CD ROM Drive\Examples\edk_example' folder. This example platform contains the following:

- PPC core
- jtagppc_cntlr
- processor reset block
- Two EMC peripherals to connect to the two independent ZBT banks
- OPB_GPIO peripheral
- OPB_UartLite UART peripheral for STDIO
- DCM block for handling the system clock
- Two DCM blocks for handling the clock to the ZBT banks

The intention of this design example is to show how certain aspects of the Kit can be used in the EDK tool, rather than providing insight into using EDK. The EDK tools are supplied with an extensive documentation set to explain the flow, tools and components used. The main EDK Platform Studio project is provided in the example folder and is called 'system.xmp'. Open this file to show the main project which is displayed in **Figure 105 on page 183**.

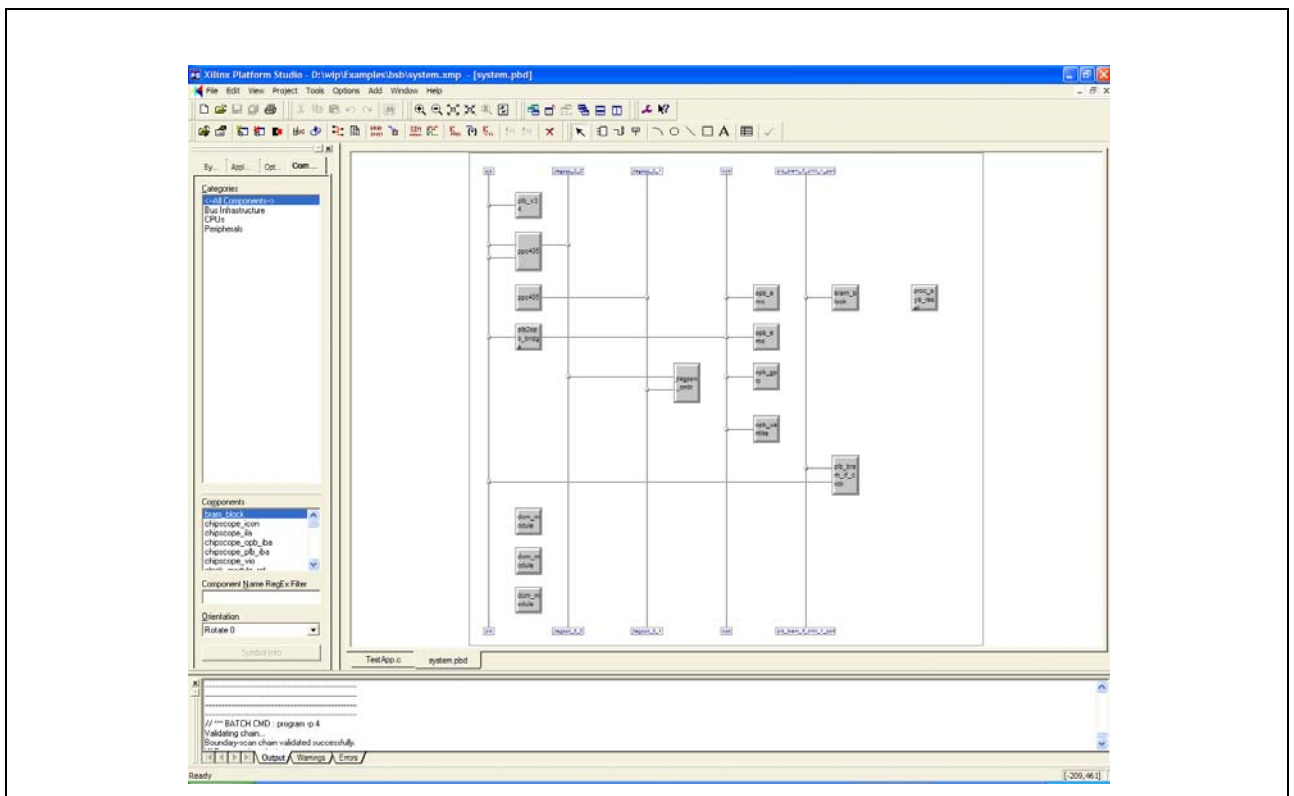


Figure 105: Main XPS Project

The design has already been created and the ISE project is included in the proj_nav subfolder. A User Constraint File (UCF) is also included called system.ucf.

21.3.1 Load the FPGA Design with FUSE

Prior to connecting the debugger to the design, the design should be loaded, the oscillators setup and the resets to the design toggled. Launch the FUSE Probe Tool and assign the '.\implementation\download.bit' file to the main User FPGA (XC2VP30-4FFI152).

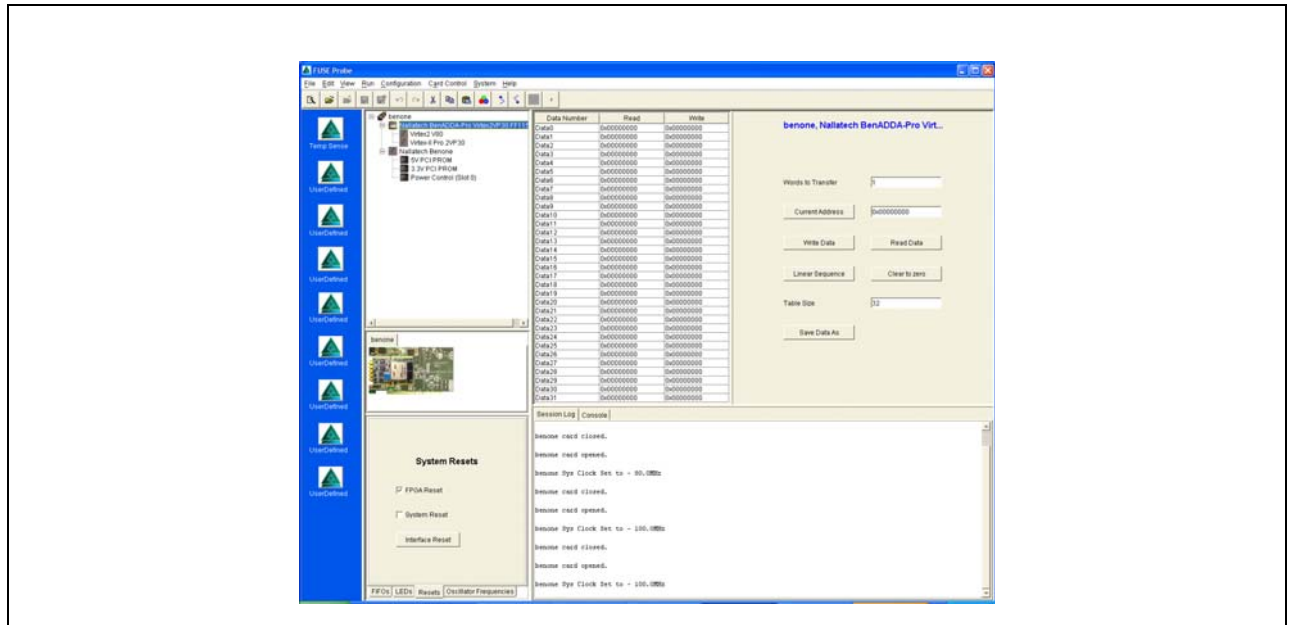


Figure 106: EDK Example Assigning 'download.bit' File

Right click on the XC2VP30 and select to configure it. You will see the result in the main session log as shown in [Figure 106 on page 184](#). Set the Oscillators to 100Mhz i.e. 100000 in the FUSE Probe Tool. Finally toggle the resets to the design. Check and then uncheck both the System Reset signal to ensure proper startup of the design. At this point you will see the LEDs (numbers 1 and 2) on the module toggling through a short sequence showing that a test program is running on the Kit.

21.3.2 Connect via XMD

XMD is then used to connect to the design for the purposes of debugging. From within XPS, go to tools and the click on 'XMD'.

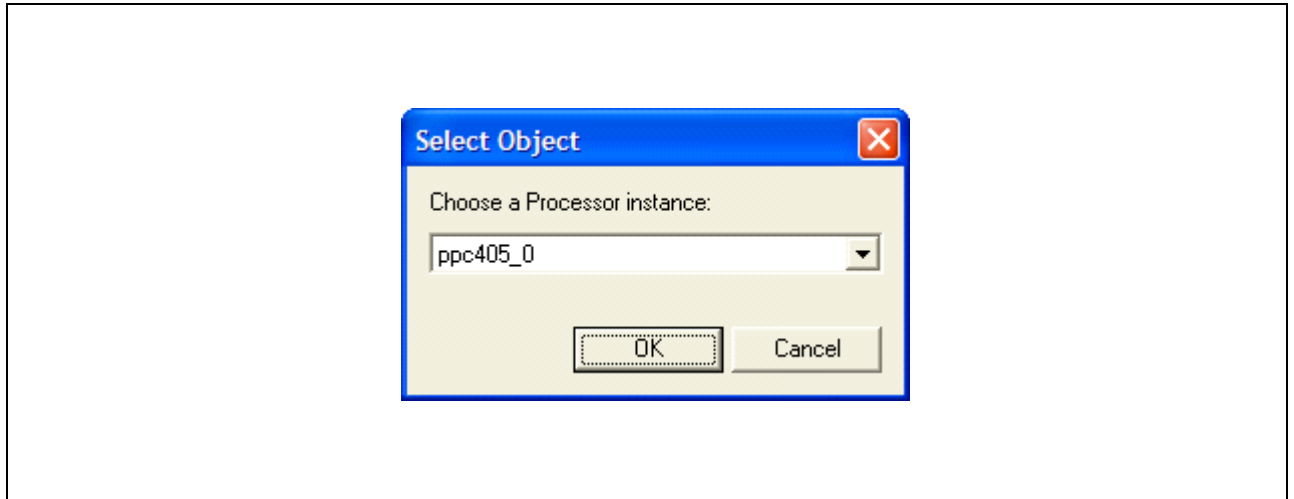


Figure 107: Choosing the processor for XMD connection

In the main User FPGA there are two PowerPC processors. Therefore when using XMD you should connect to the one you wish to debug. In this example the program running is in ppc405_0. Select this in the dialog box shown in [Figure 107 on page 185](#). XMD will then connect to the chain and then the ppc405_0 in the main User FPGA design. The successful connection is shown in [Figure 108 on page 185](#).

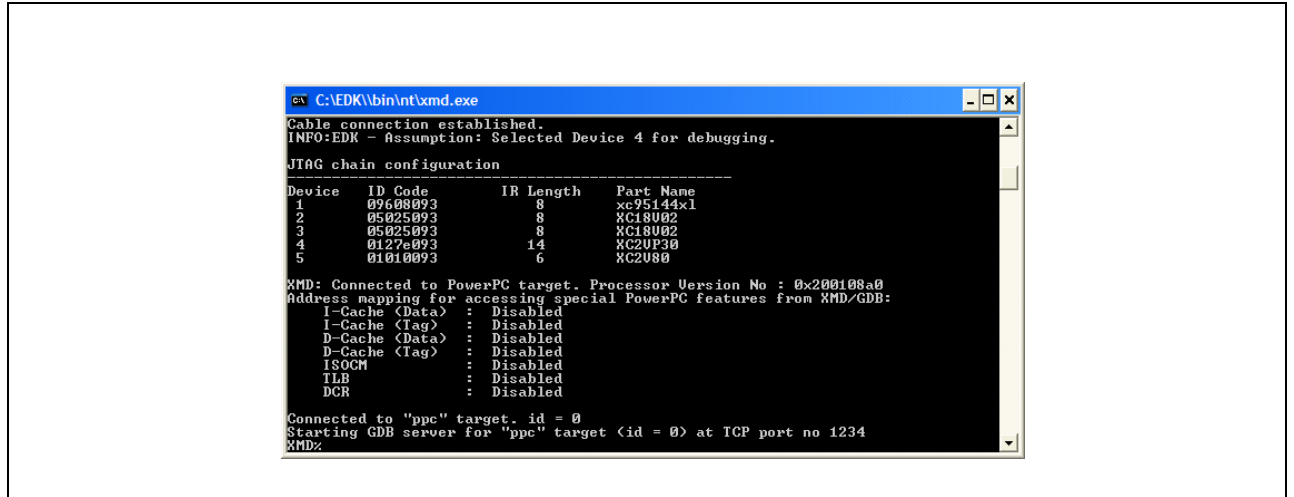


Figure 108: Successful XMD Connection

21.3.3 Start the Software Debugger

Once XMD has successfully connected, the software debugger can be started. In XPS go to 'tools' and then 'Software Debugger'.

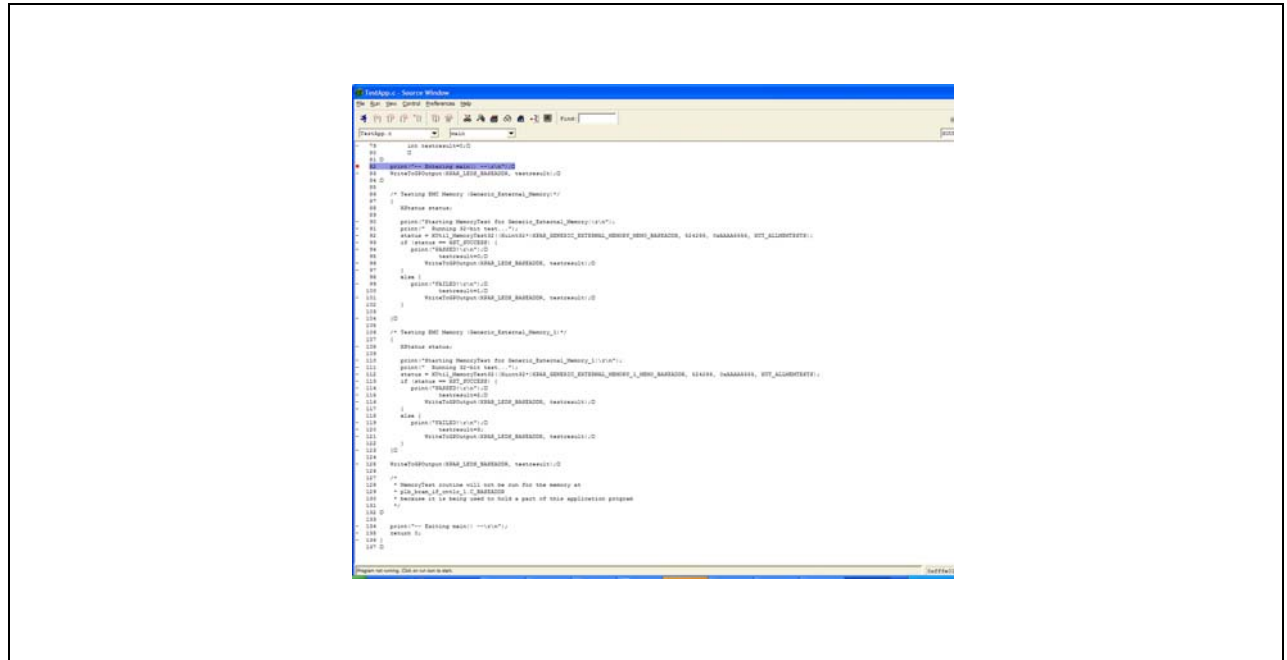


Figure I 09: Software Debugger

The software debugger starts showing the main code. It is now necessary to connect the software debugger to the TCP port opened by XMD. In the software debugger go to **Run->Connect to target**. In the dialog that appears select the target type and hostname as shown in Figure I 10 on page 186. Click 'OK' to bring up a prompt which states that a connection was established.

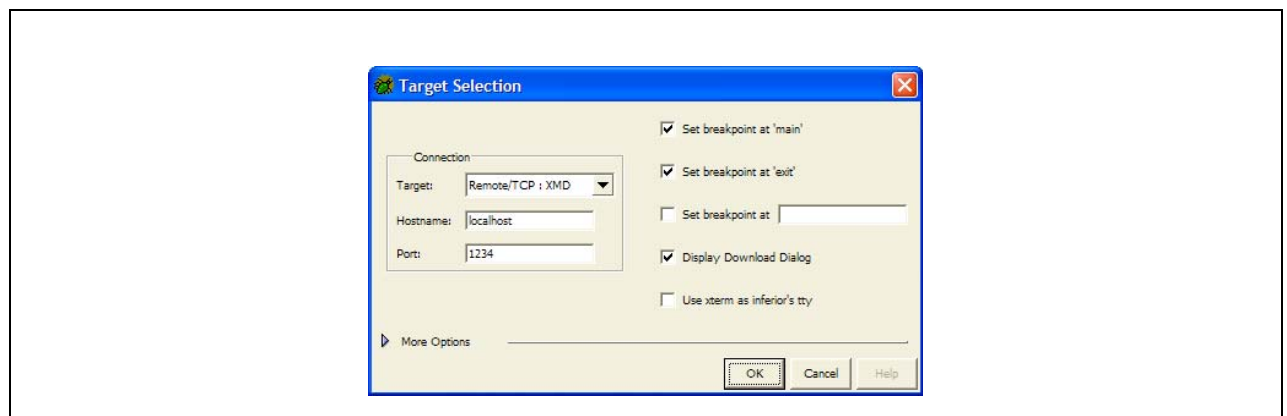


Figure I 10: Set the Target Selection

Then go to 'Run->Run'. You will see the program ELF file being downloaded to the BRAM in the design. Once it is complete it will breakpoint at the first valid line as shown in **Figure III on page 187**.

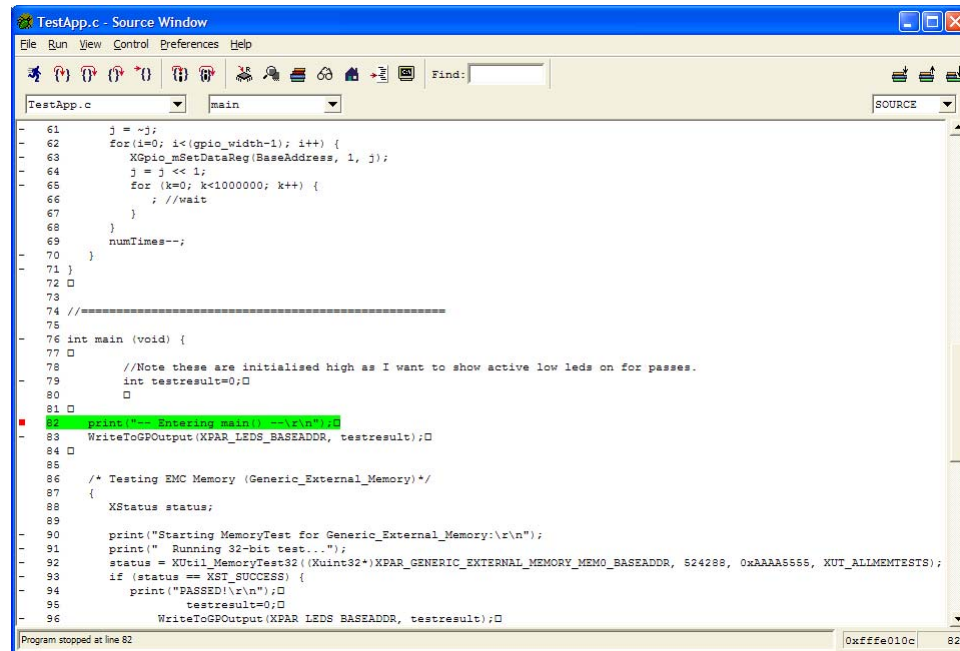


Figure III: First Line Breakpoint

From this point the debugging controls can be used to step through the program and look at local variable and memory.



NALLATECH
The High Performance FPGA Solutions Company

Part X:Xilinx System Generator Support

This part of the User Guide provides details on how to use the Xilinx System Generator Tool in conjunction with the XtremeDSP Development Kit Pro.

Section 22

System Generator Support

In this section:

- Introduction
 - Overview of Xilinx System Generator
 - Using the HDL Netlist Flow
 - Using the Cosimulation Flow
 - Chipscope Use in System Generator
-

22.1 Introduction

Xilinx have developed a tool called System Generator which may be included as an evaluation version in the XtremeDSP Development Kit Pro pack. Alternatively contact Xilinx for details on the availability of evaluation versions of System Generator. This section provides information on using System Generator through worked examples that are also included on the Kit CD in the folder 'CD ROM Drive\Examples\system_generator_examples'. If you have selected to install the examples locally then this folder will also be present on the PC.

Full details on the System Generator are provided through online help accessible within MATLAB. Also please refer to the *System Generator documentation* for further details on the use of System Generator. Details of System Generator are included here as some users may wish to use it on the XtremeDSP Development Kit Pro. Three main examples are included here to show the standard HDL netlist flow when using the Kit (QAM example), cosimulation flow (MAC FIR example), and finally an example of using Chipscope ILA with System Generator.

22.2 Overview of System Generator

System Generator is a system level modelling tool that facilitates FPGA hardware design and extends Simulink® in various ways in order to provide a powerful modelling environment that is well suited to hardware design. The tool provides high-level abstractions that are automatically compiled into an FPGA at the push of a button. The tool also provides access to underlying FPGA resources through lower level abstractions, allowing you to implement highly efficient FPGA designs.

Programming an FPGA using System Generator means describing a computation as a Simulink model, generating a hardware description from this model, and then compiling this hardware description into an FPGA configuration file, called a bitstream. The final step of compiling a hardware description into a bitstream is not unique to System Generator.

System Generator blocksets allow you to construct bit-accurate and cycle-accurate models of an FPGA circuit in Simulink. Nevertheless, it is universally true that a hardware engineer wants to see the design running in hardware. System Generator provides hardware cosimulation interfaces that make it possible to incorporate an FPGA directly into a Simulink simulation. The code generator has "Hardware Cosimulation" compilation targets (analogous to the HDL Netlist target) that automatically create a bitstream. After creating the bitstream, System Generator

automatically incorporates an FPGA hardware platform configured with this bitstream back into Simulink as a run-time block. When the design is simulated in Simulink, results for the compiled portion are calculated in hardware. This allows the compiled portion to be tested in actual hardware, and can speed up simulation dramatically.

22.3 Using the HDL Netlist Flow (QAM Example)

22.3.1 Example Overview

This example shows a QAM example that is worked through the HDL Netlist Compilation flow. The output of System Generator is taken through ISE to produce a bitfile that can then be programmed onto the hardware.

Tool Requirements: ISE 6.3i and System Generator 6.3. The example will also require the use of an oscilloscope with a minimum of 2 channels.



ISE 6.3i must be used in conjunction with System Generator 6.3. Earlier versions of ISE will not function with System Generator 6.3.

22.3.2 Simulating the QAM Example



To simulate the QAM Example use the following procedures:

1. From the MATLAB console, change the directory to the folder '\Examples\system_generator_examples\QAM'. If you have chosen to install the examples in the XtremeDSP Development Kit Pro then this folder will be present where you chose to install the Kit software. If you have chosen not to install these files, they can be copied from the examples folder on the supplied Kit CD. The following file is located in this directory:

Sysgenqam16_dplr.mdl - Your working model.

2. Open sysgenqam16_dplr.mdl model from the MATLAB console. This design implements an equalized 16-QAM demodulator for use in a software defined radio. The receiver architecture provides sub-systems that demonstrate adaptive channel equalization and carrier tracking on a random QAM data source.

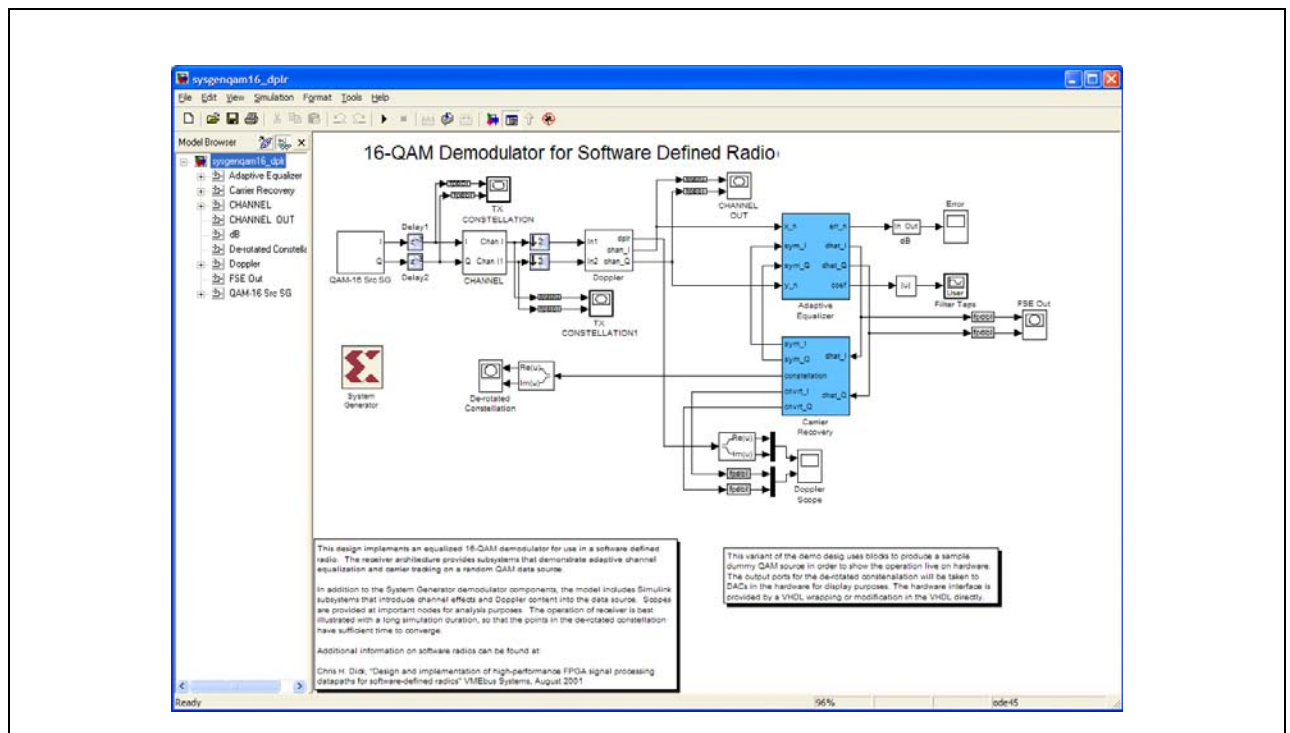


Figure 112: 16-QAM Demodulator Example

In addition to the System Generator demodulator components, the model includes Simulink subsystems that introduce channel effects and Doppler content into the data source. Scopes are provided at important nodes for analysis purposes. The operation of receiver is best illustrated with a long simulation duration, so that the points in the de-rotated constellation have sufficient time to converge.

3. Simulate the model by clicking on the 'Start simulation' icon shown right. At this point, without modifying the model, you should be able to see the plots shown in [Figure 113 on page 194](#).



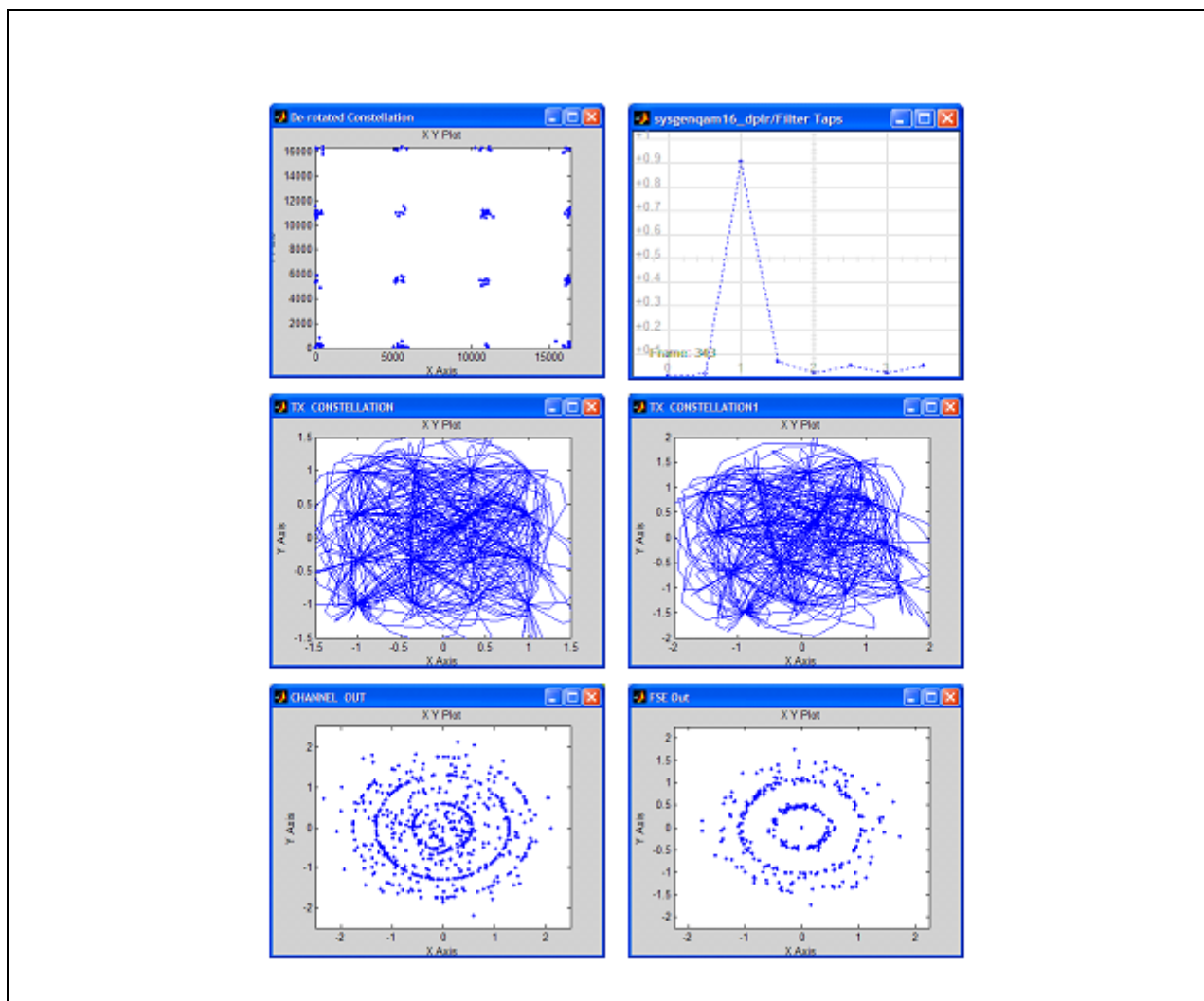


Figure 113: Simulink Simulation Output

You will see the filter values settling and the constellation output changing. The next step is to create the VHDL for the model and take the output to the Xilinx ISE tools to create a bitstream to program the FPGA.

4. Double click on the System Generator block in the model to bring up the parameters to set the compilation target.

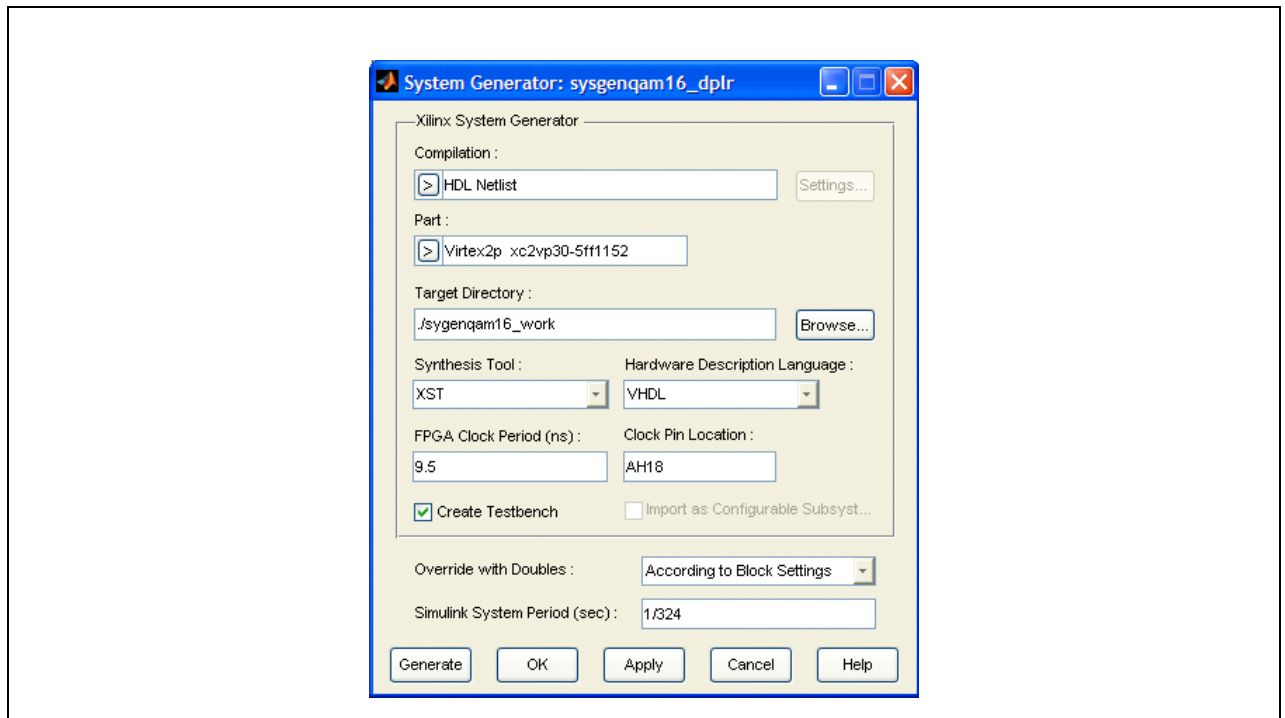


Figure 114: QAM System Generator Options

- Set the compilation target at HDL Netlist.
 - Set the part as Virtex-II Pro XC2VP30-4FFI152 if using the XtremeDSP Development Kit Pro.
 - Note the FPGA Clock Period setting (9.5ns in this case) as this determines how fast the design is capable of running if using a free running clock. The dialog box should then be similar to that shown in **Figure 114 on page 195**.
5. Once these parameters have been set click on 'generate' to start the System Generator flow. This will create the VHDL for the design and will also create a Xilinx ISE Project Navigator File for the next stage of creating the implemented design on the FPGA itself.
 6. Now open the ISE Project Navigator file that has been created. The file will be located in the folder you selected as the target directory. The default is '\Examples\system_generator_examples\QAM\sysgenqam16_work'. Open the file 'sysgenqam16_dplr_clk_wrapper.npl' within Xilinx ISE Project Navigator as shown in **Figure 115 on page 196**.

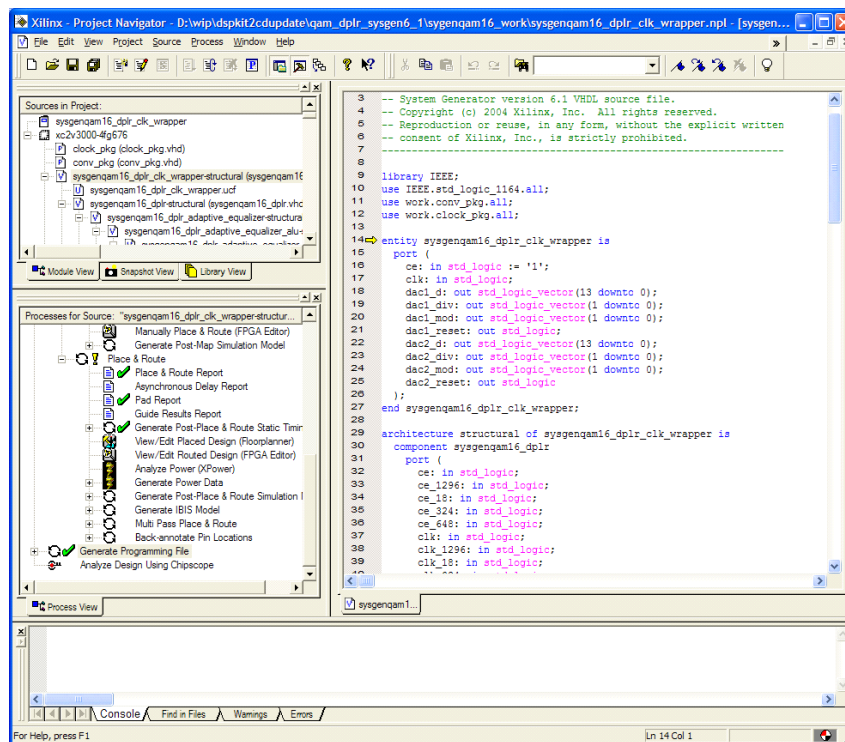


Figure I I5: QAM Design in ISE Project Navigator

7. Right click on the process for 'Generate Programming File' and select 'Properties'. This brings up the options for creating the bitstream for the FPGA itself. Click on the Startup tab and set the startup clock to 'JTAG' as shown in Figure I I6 on page 196.

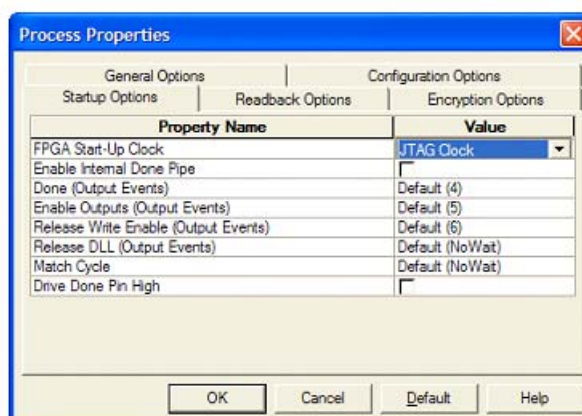


Figure I I6: Setting the JTAG Startup Option

8. Now click on 'Generate Programming File' and select 'Run'. You will see ISE run through synthesis, place and route and finally generation of the programming file. Once this is complete ISE should be closed.
9. Open FUSE to power up the board.
 - Select Start->Programs-> FUSE -> Software -> FUSE Probe
 - From FUSE, select Card Control -> Open Card. Then select USB for the interface and click on '**Locate Card**'. At this point FUSE locates the BenONE-Kit Motherboard and the top left corner of the FUSE Probe Tool displays the FPGA information as shown in **Figure I 17 on page 197**. If you have the card plugged into a PCI slot then select to open the card via PCI.

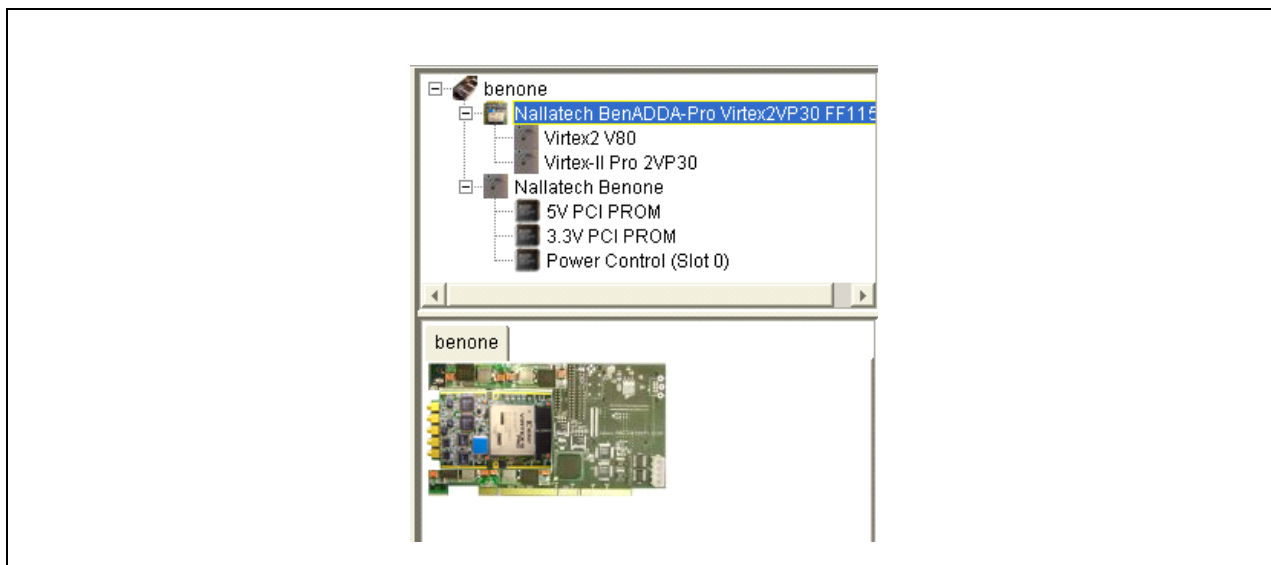


Figure I 17: Opening the Card in FUSE

10. Assign two bitfiles to the devices as follows.
 - Right click on the Virtex2 2v80 in the card browser and select 'Assign Bitfile'. In the file window that appears browse to '\Examples\system_generator_examples\QAM\bitfiles' and select the 'osc_clock_2v80.bit' file.
 - Right click on the Virtex-II Pro XC2VP30-4FF1152 in the card browser and select 'Assign Bitfile'. In the file window that appears browse to '\Examples\system_generator_examples\QAM\syngenqam16_work' and select the 'syngenqam16_dplr_clk_wrapper.bit' file. Note that if you have not generated this bitfile yourself though ISE in the previous steps a pre-generated bitfile is provided in '\Examples\system_generator_examples\QAM\bitfiles' called 'syngenqam16_dplr_XC2VP30-4FF1152.bit'.
 - Right click on the Virtex2 2v80 in the card browser and select 'Configure Device' to configure the FPGA.
 - Right click on the Virtex-II Pro XC2VP30-4FF1152 in the card browser and select 'Configure Device' to configure the FPGA.

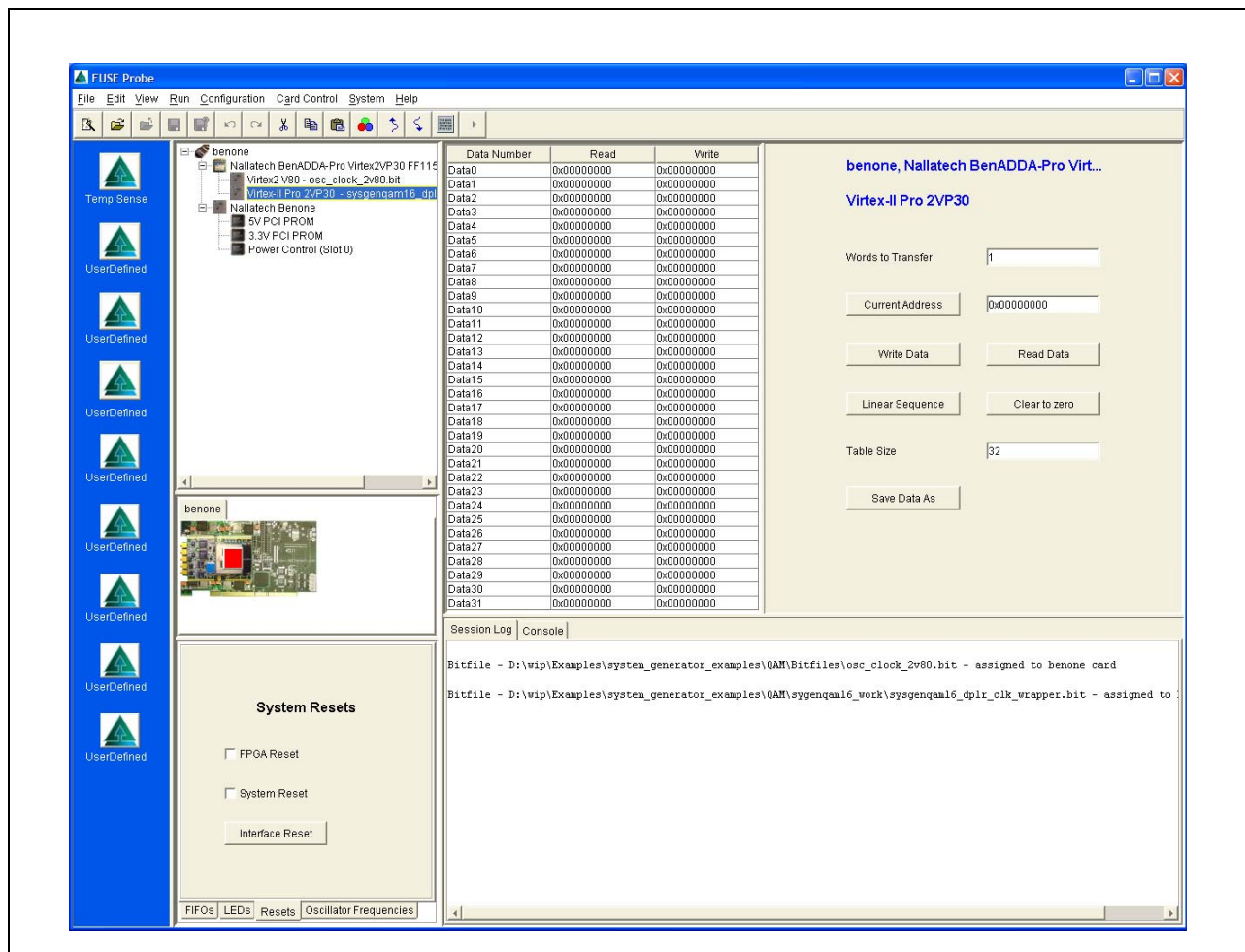


Figure 118: Card Opened in FUSE - Assigning Bitfiles

11. View the output on the oscilloscope. Connect two of the MCX-BNC cables from the DAC outputs on the board to channels on the oscilloscope. Set the display mode of the scope to X-Y if it supports it. You will see either two streams of data (Figure 120 on page 199) or the constellation if using X-Y (Figure 119 on page 199). Some modification of time and voltage scales may be necessary dependent upon the scope used. Remember to set the input coupling for the channels to 50Ω on the oscilloscope.



22.4 Using the Cosimulation Flow (MAC FIR Example)

22.4.1 Example Overview

This example shows how the cosimulation flow can be used with the Kit. The example shows a 32 tap MAC-based FIR design. The design of coefficients to produce a low pass filter is carried out using the Filter Design and Analysis Tool (FDATool). The example shows how the MAC FIR example can be targeted to be run on the actual Kit hardware using the cosimulation flow.

Tool Requirements: ISE 6.3i or later, System Generator 6.3i or later.



Please note that you will probably need to copy the MACFIR example folder to a local folder that contains no spaces in the complete folder path, i.e. C:\temp\MACFIR. By default the installer places the XtremeDSP Development Kit Pro software and examples to C:\Program Files\FUSE\XtremeDSP Development Kit Pro. The space in 'Program Files' causes an error in the generation of the cosimulation files - you will get a popup stating that an error has occurred. The examples should be run from a folder that has no spaces in the path to remove this problem.

22.4.2 Simulating the MAC FIR Cosimulation Example



To simulate the MAC FIR Cosimulation example use the following procedures:

1. From the MATLAB console, change the directory to the folder '\Examples\system_generator_examples\MACFIR'. If you have chosen to install the examples in the XtremeDSP Development Kit Pro then this folder will be present where you chose to install the Kit software. If you have chosen not to install these files, they can be copied from the examples folder on the supplied Kit CD. The following file is located in this directory:

mac_fir_xtremedsokit_demo.mdl - Your working model.

2. Open mac_fir_xtremedsokit_demo.mdl model from the MATLAB console. This model represents a simple 32 tap MAC FIR with stimulus noise and sinewave sources as shown in **Figure 121 on page 200**.

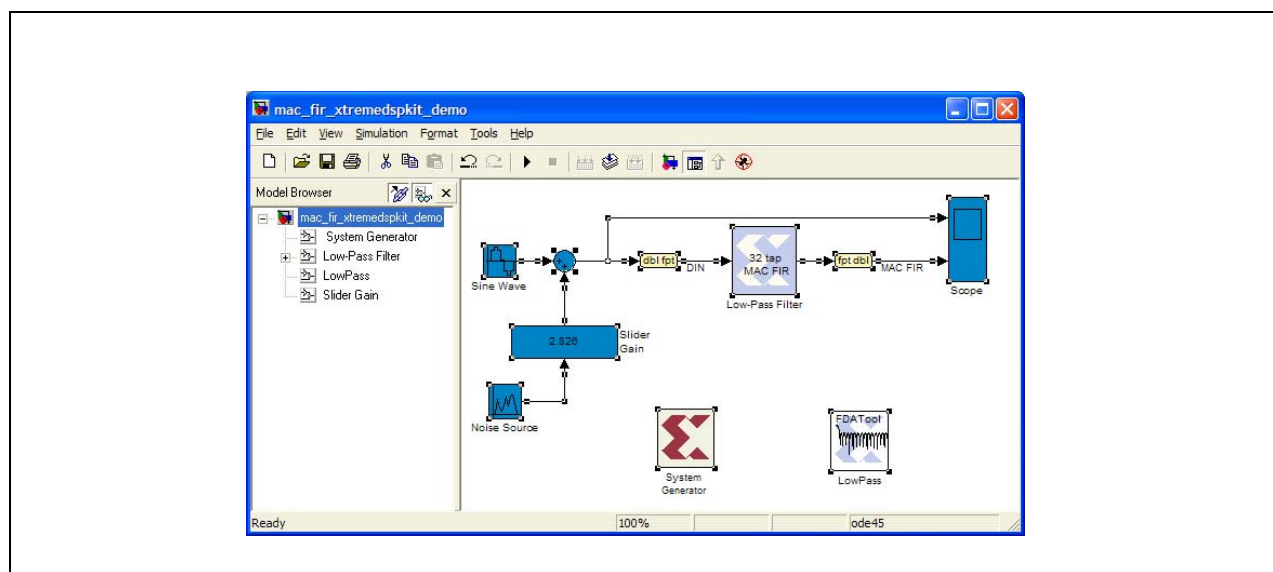


Figure 121: Simulink Model: 'mac_fir_xtremedsokit_demo.mdl'

The model shows two input sources, a sinewave simulink source and a slider gain controlled noise source. These sources are combined and fed through the MAC FIR to be displayed on the scope block along with the source signal. The gateway blocks (yellow) allow for translation of the double to fix point numbers. A FDATool block is also included. Double click on the FDATool block to launch the tool. The filter design chosen here is for illustrative purposes only.

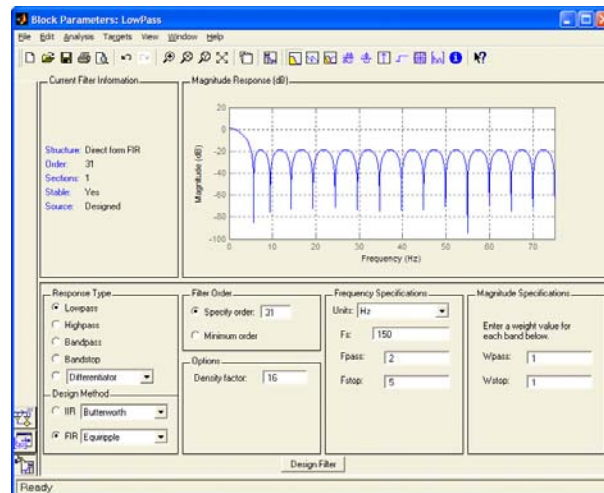


Figure 122: Simple Low Pass Filter Design in FDATool

The filter coefficients for the designed filter can be used in the actual MAC FIR block by using the `xlfsd_numerator` function. This function returns the numerator of the filter object stored in the Xilinx FDATool block shown in [Figure 122 on page 201](#).

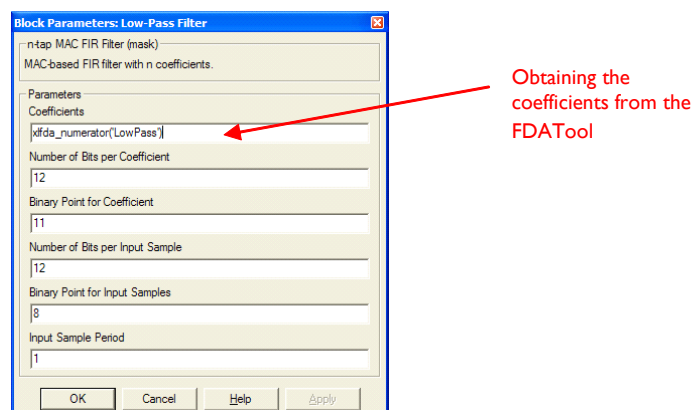


Figure 123: Obtaining the Coefficients from the FDATool

3. Simulate the model by clicking on the 'Start simulation' Icon shown right. At this point, without modifying the model, you should be able to see the plot shown in [Figure 124 on page 202](#).



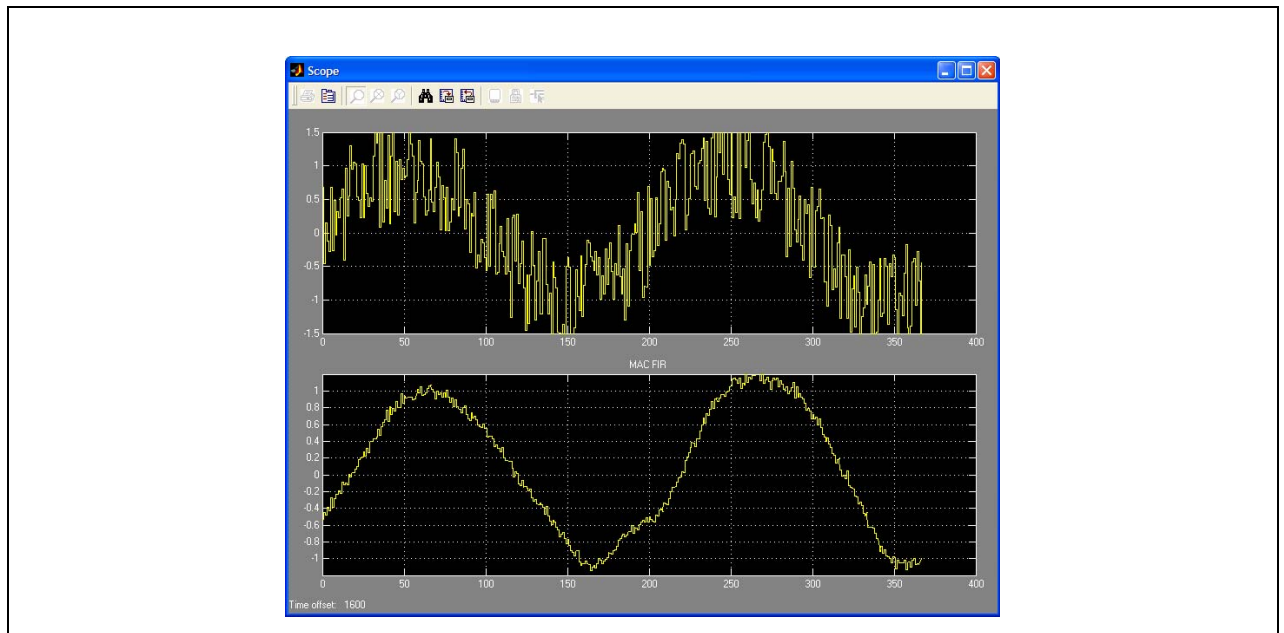


Figure 124: Scope Waveform

4. Double click on the Slider Gain Block and move the slider (Figure 125 on page 202) to change the noise level. Then view the changes on the scope.

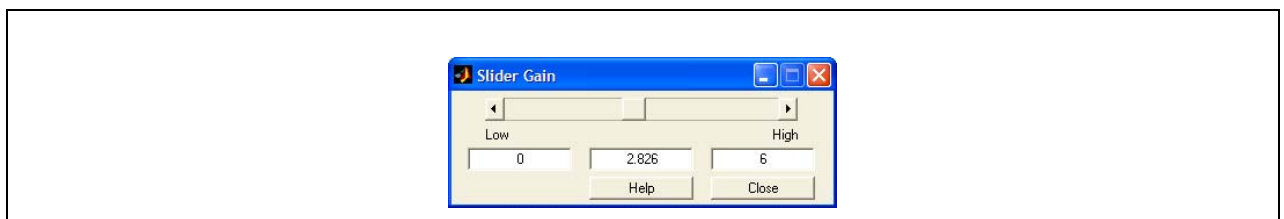


Figure 125: Slider Gain Control

22.4.3 Cosimulation Flow

After simulating the model within Simulink, the next step is to make use of the cosimulation flow. If the Kit is used with the USB connection and the external power supply, use the JTAG cosimulation flow with a Parallel-IV JTAG download cable. If the Kit is plugged into a PCI slot then the cosimulation can be carried out through the PCI interface itself.

Setting up for JTAG Cosimulation Flow

If the Kit is used via the standalone (non-PCI) then the cosimulation flow can be carried out using a Parallel-IV download cable. Connect the Parallel Cable to the available JTAG headers on the Kit. If using the Parallel-IV pod with the ribbon cable, a keyed socket (J24) is available on the board. If using a flying lead connection a 0.1" pitch General JTAG header (J14) is provided.



To enable the power supplies to the User FPGA (XC2VP30-4FF1152) without opening the card via USB ensure J17 has its jumper fitted. This enables the power supplies shortly after power is applied to the board by the external power unit.

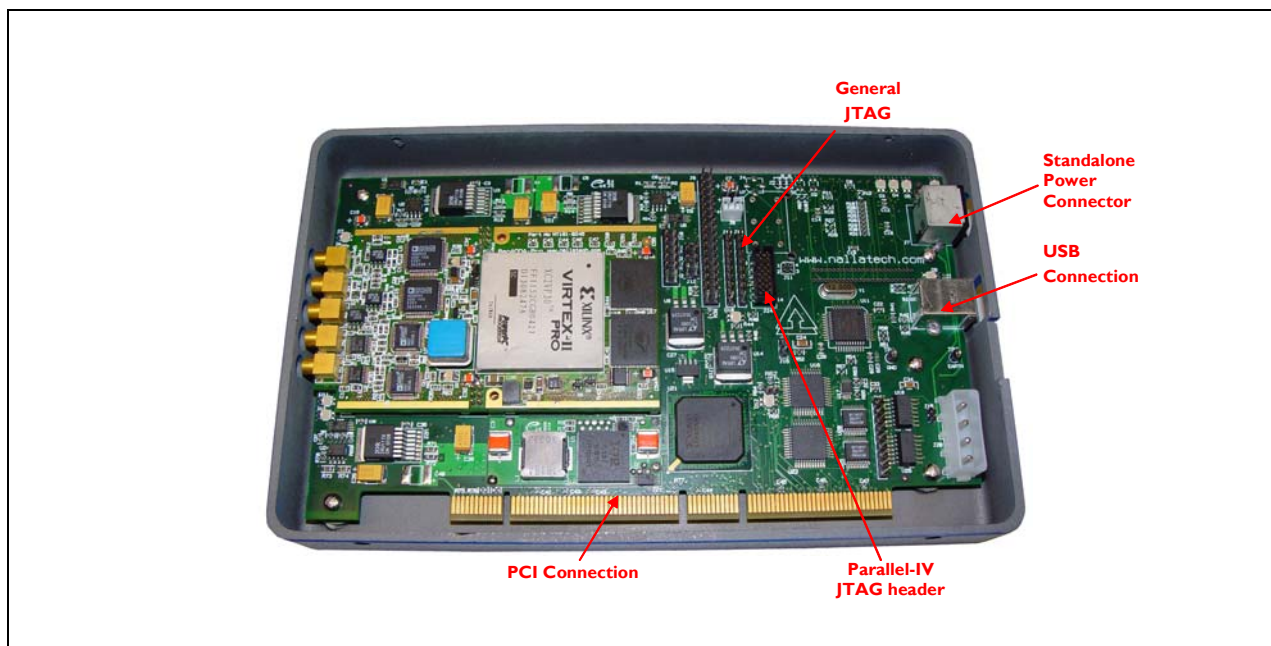


Figure 126: JTAG Header Positions on Card

General JTAG Header (Flying Lead Connection)

This is a JTAG header that connects to the chain which runs through the standard JTAG pins on the User FPGA devices in the Kit and also some of the PROMs and CPLDs. It is a standard 0.1" pitch header and can therefore support flying lead connections for the Parallel-IV pods. The header is shown in [Figure 127 on page 203](#).

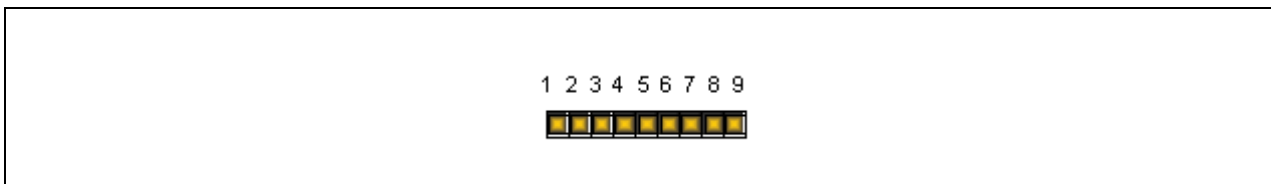


Figure 127: General JTAG Connector J14

Pin #	Name	Description
1	3.3V	3.3 Volts Supply
2	GND	Signal Ground
3	N/C	Not connected - do not use
4	TCK	ALT JTAG TCK Signal
5	N/C	Not connected - do not use
6	TDO	ALT JTAG TDO Signal
7	TDI	ALT JTAG TDI Signal
8	TRST#	ALT JTAG TRST# Signal

Table 51: General JTAG Header J14 Pinouts

Pin #	Name	Description
9	TMS	ALT JTAG TMS Signal

Table 51: General JTAG Header J14 Pinouts

Fitting a Parallel-IV Ribbon Cable

The header (J24 on the motherboard) allows a Xilinx Parallel-IV ribbon cable to be plugged in. Note that this header is in parallel with the General JTAG header J14. The connector is keyed and follows the pinout on the *datasheet* supplied with the Parallel-IV cable from Xilinx.



Please note that the Kit is not supplied with a Parallel-IV ribbon cable. This section provides information on how to connect the cable only if one is available.



This is the recommended header if the Xilinx JTAG Cosimulation option in the Xilinx System Generator tool is chosen to be used with a Parallel-IV ribbon cable.

When the board is used inside the blue board case, the cable is brought out through the small gap marked P-IV in the side of the case. This case should be opened, the cable fitted and then the case closed again.

To fit the cable remove the four screws round the sides of the case and remove the lid. Then plug one end of the cable into the keyed header and bring the cable out to the small opening in the base of the case as shown in [Figure I28 on page 204](#).

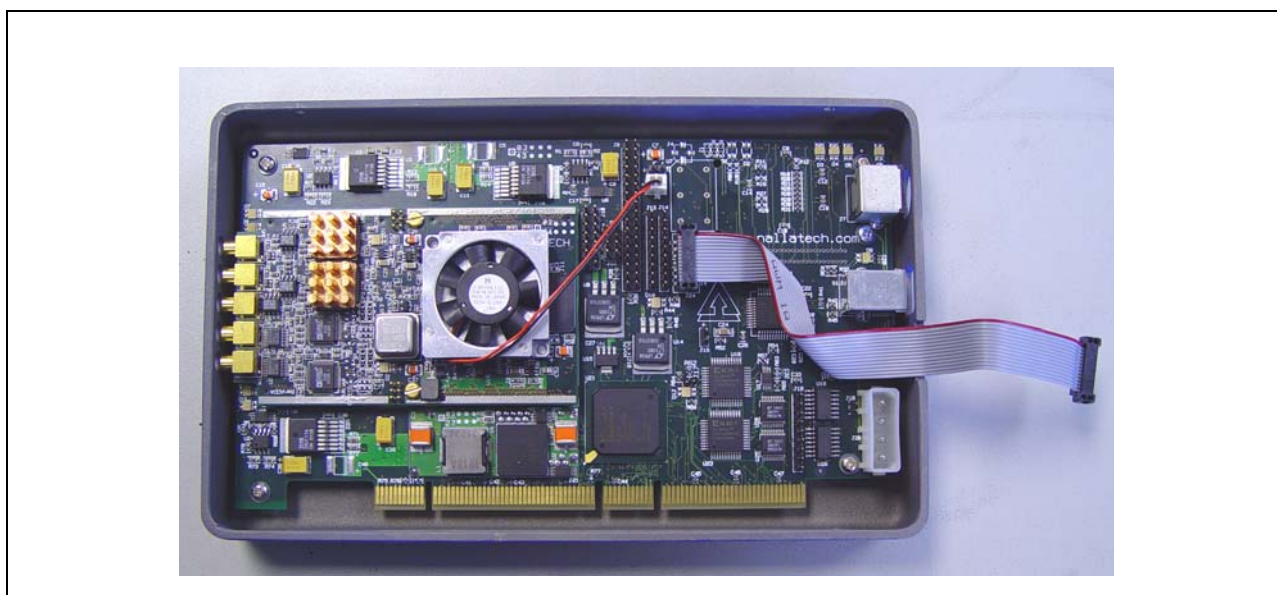


Figure I28: Routing the Parallel-IV Cable Out from Header

Then fit the lid back onto the case, refitting the screws if necessary, and ensure that the ribbon cable is not caught between the top and bottom of the blue case. Finally, connect the Parallel-IV pod to the exposed end of the ribbon

cable as shown in [Figure 129 on page 205](#). Both the connector on the cable and the socket on the pod are keyed for ease of use.

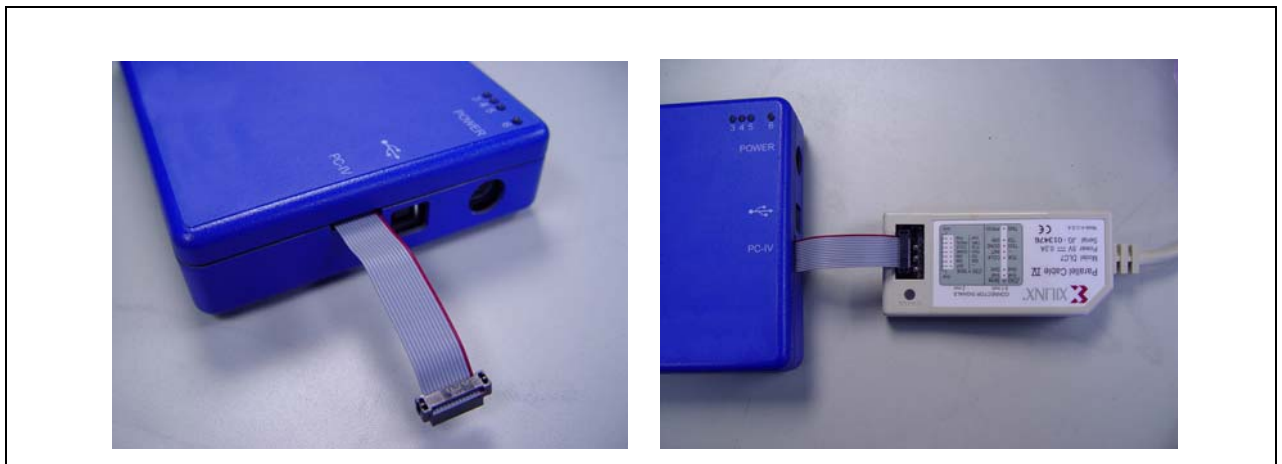


Figure 129: Re-fitting the Case Lid (first), Connecting the Parallel-IV Pod (second)

Setting up for PCI Cosimulation Flow

Using the cosimulation flow via the PCI interface assumes that you have the board properly installed into an available PCI slot. Please follow the instructions in the *Getting Started Guide* supplied with your Kit for details on how to install the hardware into a PCI slot. The *Getting Started Guide* is also available on the Kit CD in the following location: '<CD ROM Drive>\Documentation\Product'.

Performing the Cosimulation

The following steps describe how to run the MAC FIR example using the cosimulation flow.

▼ To perform the cosimulation use the following procedures:

1. If the `mac_fir_xtremedspkit_demo.mdl` model is not open, open the model.
2. Double click on the System Generator block.

Set the compilation target. If the PCI connection is used, select 'Hardware cosimulation -> XtremeDSP Development Kit -> PCI'. If the JTAG Parallel cable connection is used, select 'Hardware cosimulation -> XtremeDSP Development Kit -> JTAG'.

Set the part as XC2VP30-4FF1152 if using the XtremeDSP Development Kit Pro.

Note the FPGA Clock Period setting (25ns in this case) as this determines how fast the design is capable of running if using a free running clock. The dialog box should be similar to that shown in [Figure 130 on page 206](#).

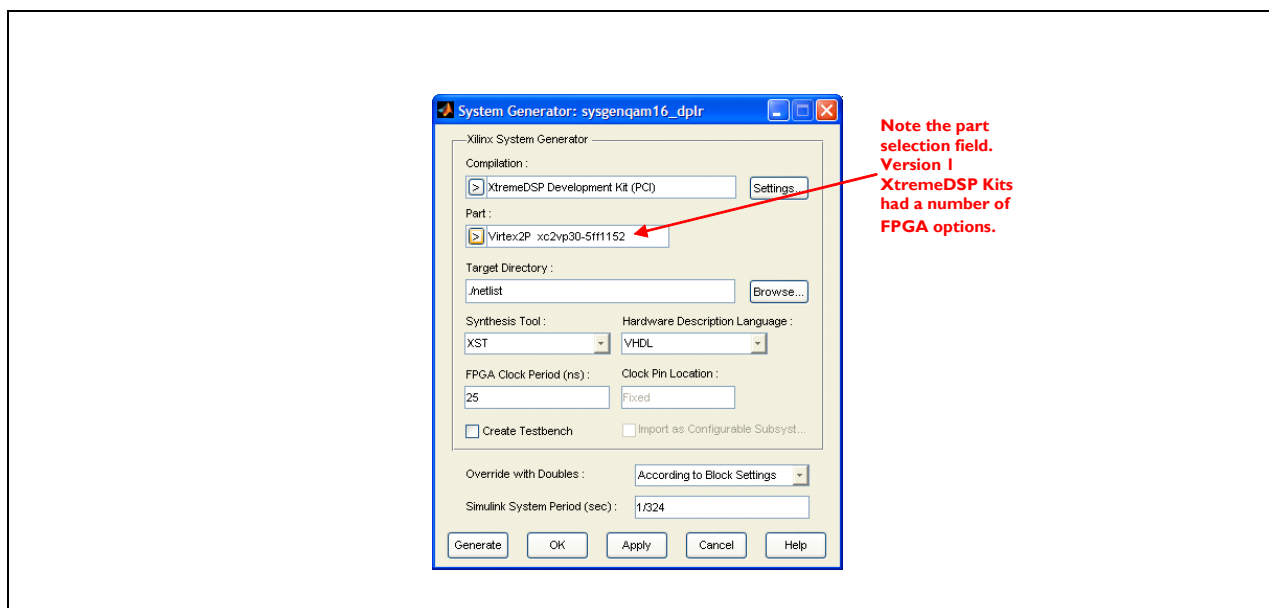


Figure 130: System Generator Settings for PCI Cosimulation Flow

- Once the parameters have been set, click on the '**Generate**' button to start the System Generator flow. The flow will create the netlist of the design but will also run the design through the synthesis and implementation tools to create a bitfile automatically. The process execution is carried out in a command window (shown in Figure 131 on page 206). Once complete, a new library is created which contains a single block for the cosimulation of the design. This is the runtime block.

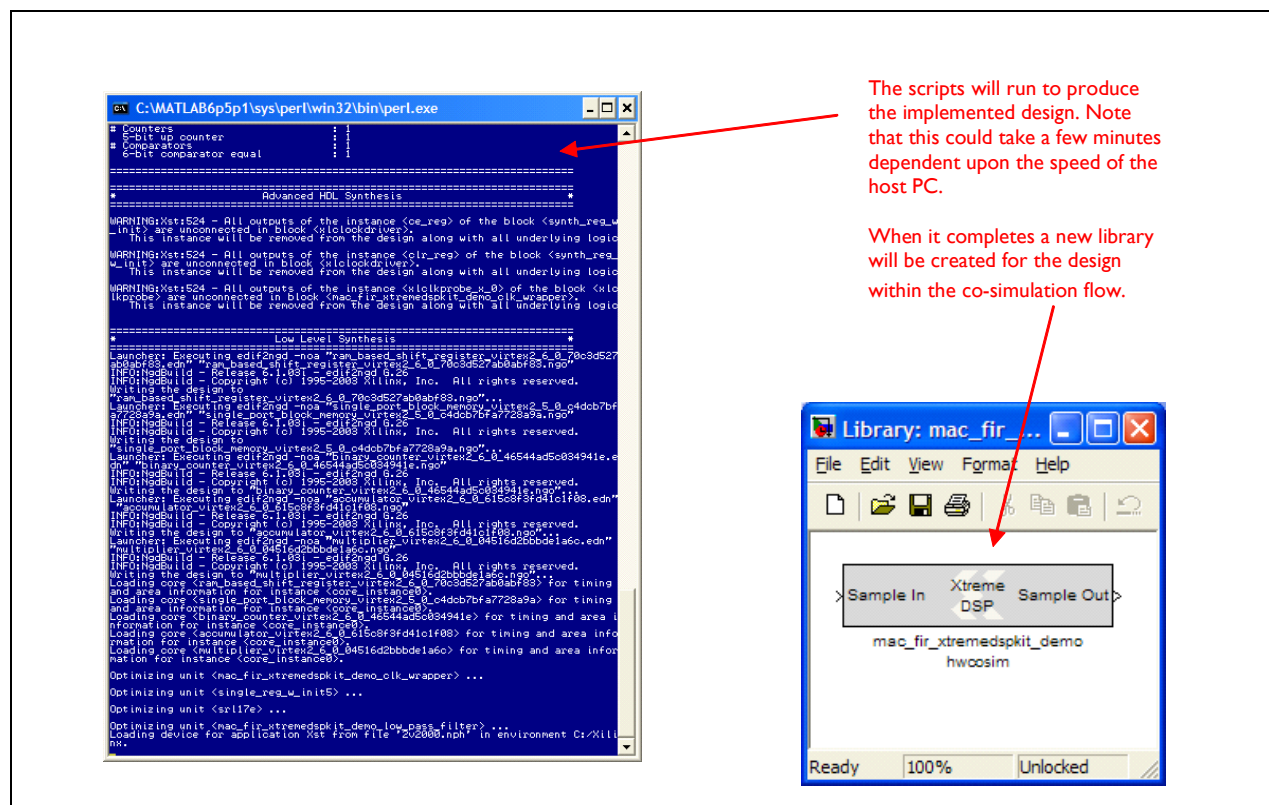


Figure 131: Command Window and Generated Runtime Library

4. Drag the generated block into the 'mac_fir_xtremedsplit_demo' model. Connect an output gateway block and another scope as shown in **Figure 132 on page 207**.

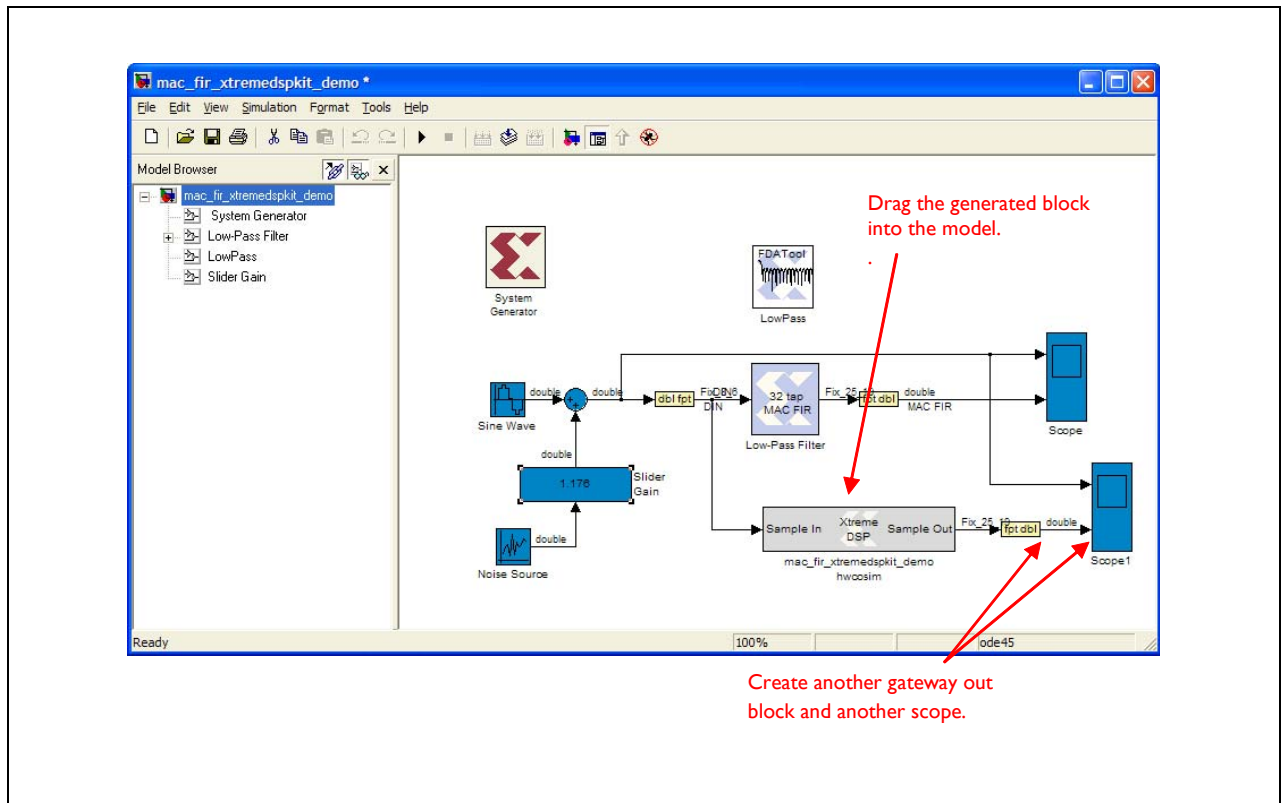


Figure 132: Adding the Runtime Cosimulation Block to the Model

5. Double click on the 'mac_fir_xtremedsplit_demo_hwcossim' block to bring up the control dialog shown in **Figure 133 on page 207**.

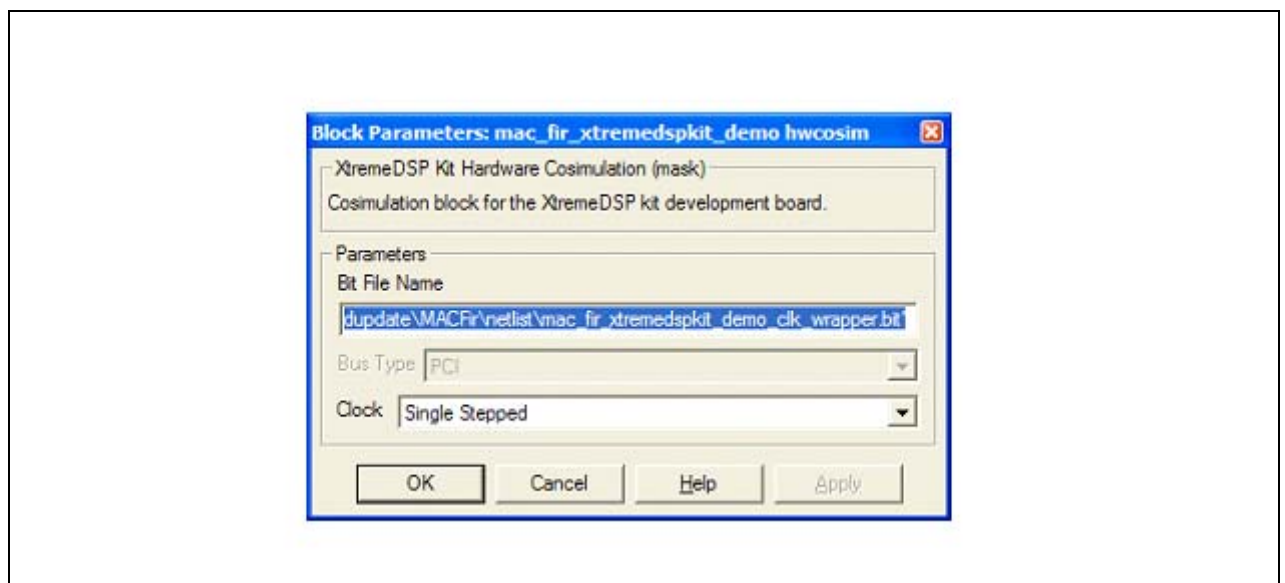


Figure 133: Hwcossim Parameters

These block parameters control the selection of the bitfile (which is set automatically during the generation) and also the selection of the clock. In this case make sure that the single stepped clock is selected.

6. Simulate the model by clicking on the 'Start simulation' icon shown right. Open both scopes and also the gain slider control as shown in [Figure 134 on page 208](#). After a few seconds, during which the FPGA is configured, the simulation begins and waveforms appear in the scopes.

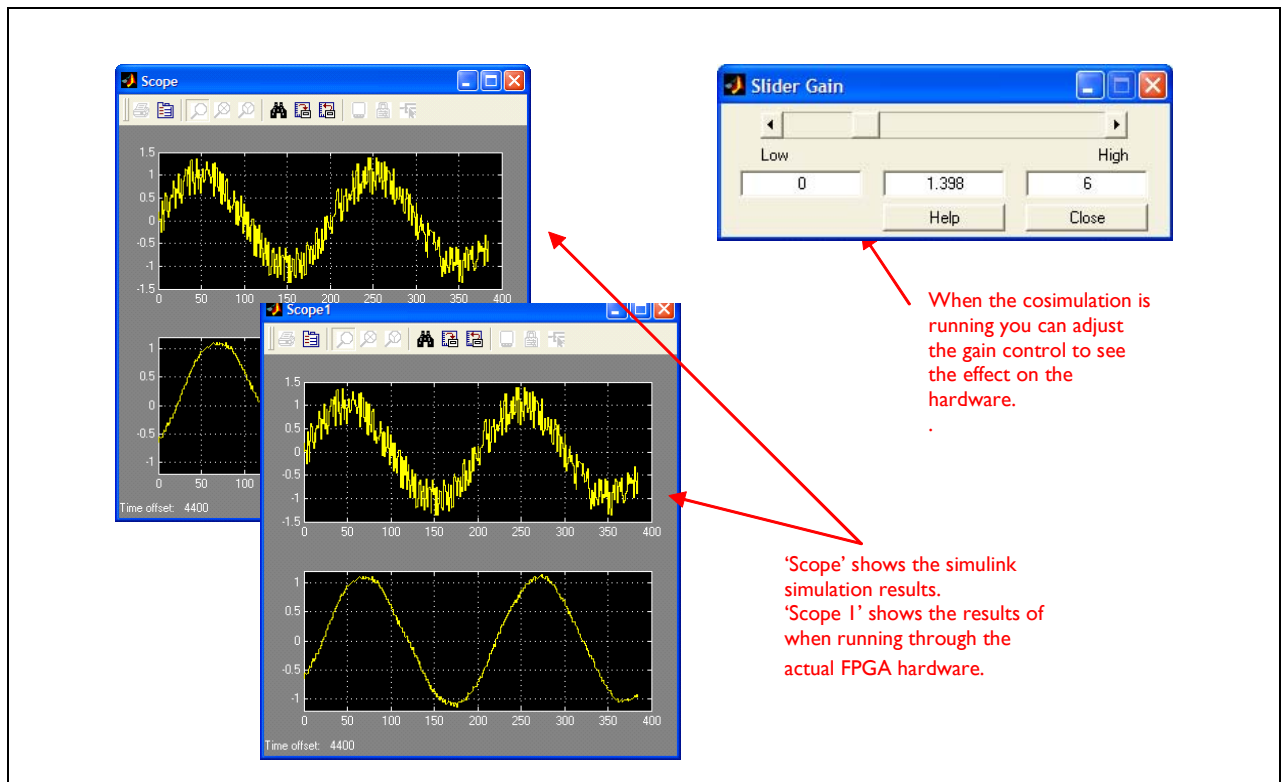


Figure 134: Cosimulation Results

22.5 Chipscope Use in System Generator

22.5.1 Example Overview

This example demonstrates how to connect and use Chipscope Pro, the Xilinx Debug Tool, within Xilinx System Generator. The integration of Chipscope Pro in the System Generator flow allows real-time debugging at system speed. By inserting a Chipscope block into your System Generator design, you can debug and verify all the internal signals and nodes within the FPGA. After reviewing some characteristics of the Chipscope Pro Debug tool, this example then describes the process of running the Chipscope block to a simple Simulink® model, deploying it on a hardware platform and probing internal signals. The following topics are discussed:

- Chipscope Pro Overview
- Chipscope within System Generator
- Real-Time Debug with the Chipscope Pro Analyzer
- Importing data back into MATLAB® workspace from Chipscope

Tool Requirements: ISE 6.3i, System Generator 6.3i or later, Chipscope PRO 6.3i or later.

22.5.2 Chipscope Pro Overview

As the density of FPGA devices increases, so does the impracticality of attaching test equipment probes to these devices under test. The Chipscope Pro tools integrate key logic analyzer hardware components with the target design inside Xilinx Virtex, Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE and Spartan-3 devices. The Chipscope Pro

tools communicate with these components during system operation and in effect provide the designer with a logic analyzer for nodes inside the Xilinx FPGA. Chipscope provides a deep trace memory, fast clock speeds and multiple trigger options, which can vary in complexity. It is possible to capture and view signal activity inside an FPGA without having to dedicate critical logic space, come up with complex capture schemes, or allocate additional I/O pins. Data samples are captured based on user-defined trigger conditions and stored in internal block memory. All control and data transfer is done via the JTAG port eliminating the need to drive data off-chip using I/O pins. Please refer to the following Web page for further details on Chipscope Pro: <http://www.xilinx.com/chipscope>

22.5.3 Chipscope in System Generator

This example shows how to modify a Simulink model to integrate the Chipscope block and to select the data to be captured and viewed for debugging. The steps are as follows:

▼ **To modify a Simulink model for Chipscope integration use the following procedures:**

1. From the MATLAB console, change the directory to \$SYSGEN/examples/chipscope. Here, \$SYSGEN represents the location in which you have installed System Generator (the default location is \$MATLAB/toolbox/xilinx/sysgen). The following files are located in this directory:
 - **chip.mdl** - Your working model.
 - **chip_soln.mdl** - Solution model, including the Chipscope blockset.
 - **osc_clock_2v80.bit** - Bitstream to program the XC2V80 device on the Kit. This device is used for clock management and is required for proper operation.
2. Open chip.mdl model from the MATLAB console. This model represents a simple sine/cosine table driven by an 8-bit counter. Both sine and cosine functions have been selected, enabling you to probe and later plot both waveforms.
3. The 8-bit counter counts modulo 255. This is used to trigger Chipscope. The most significant bit is extracted with a slice block and drives a LED on the XtremeDSP Development Kit Pro.

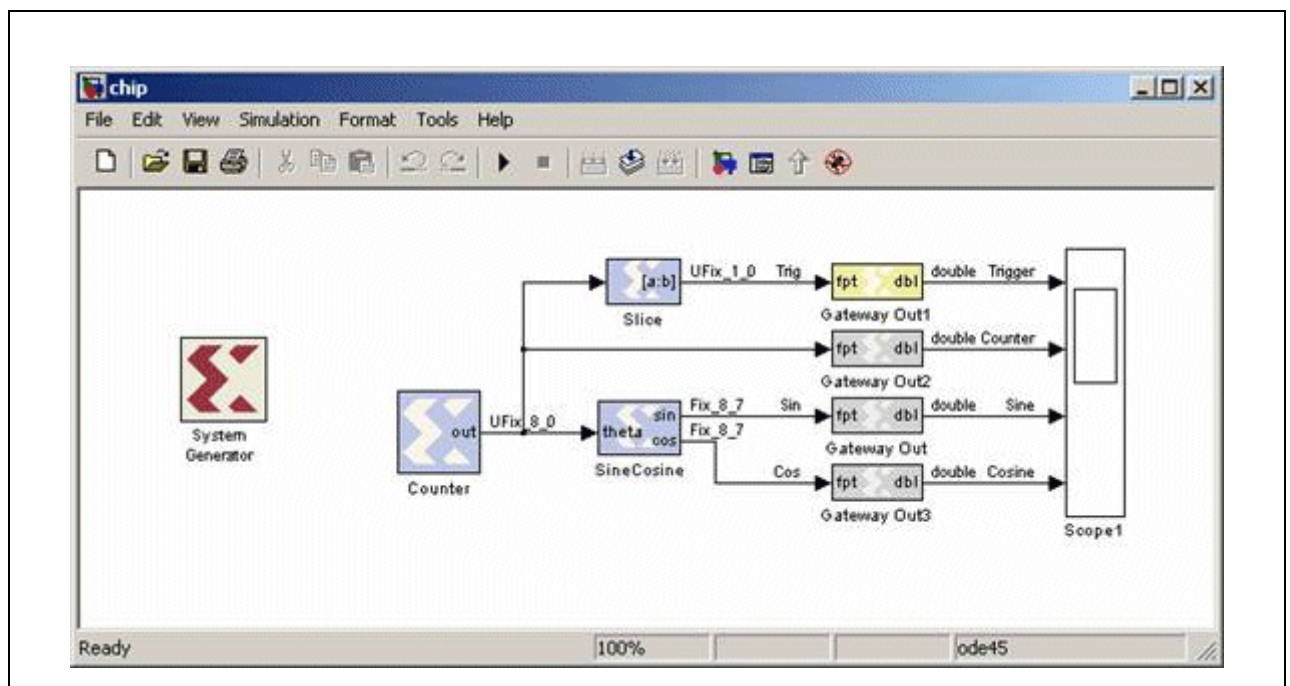


Figure 135: chip.mdl

4. Simulate the model by clicking on the 'Start simulation' icon shown right. At this point, without modifying the model, the plot shown in [Figure 136 on page 210](#) should appear.

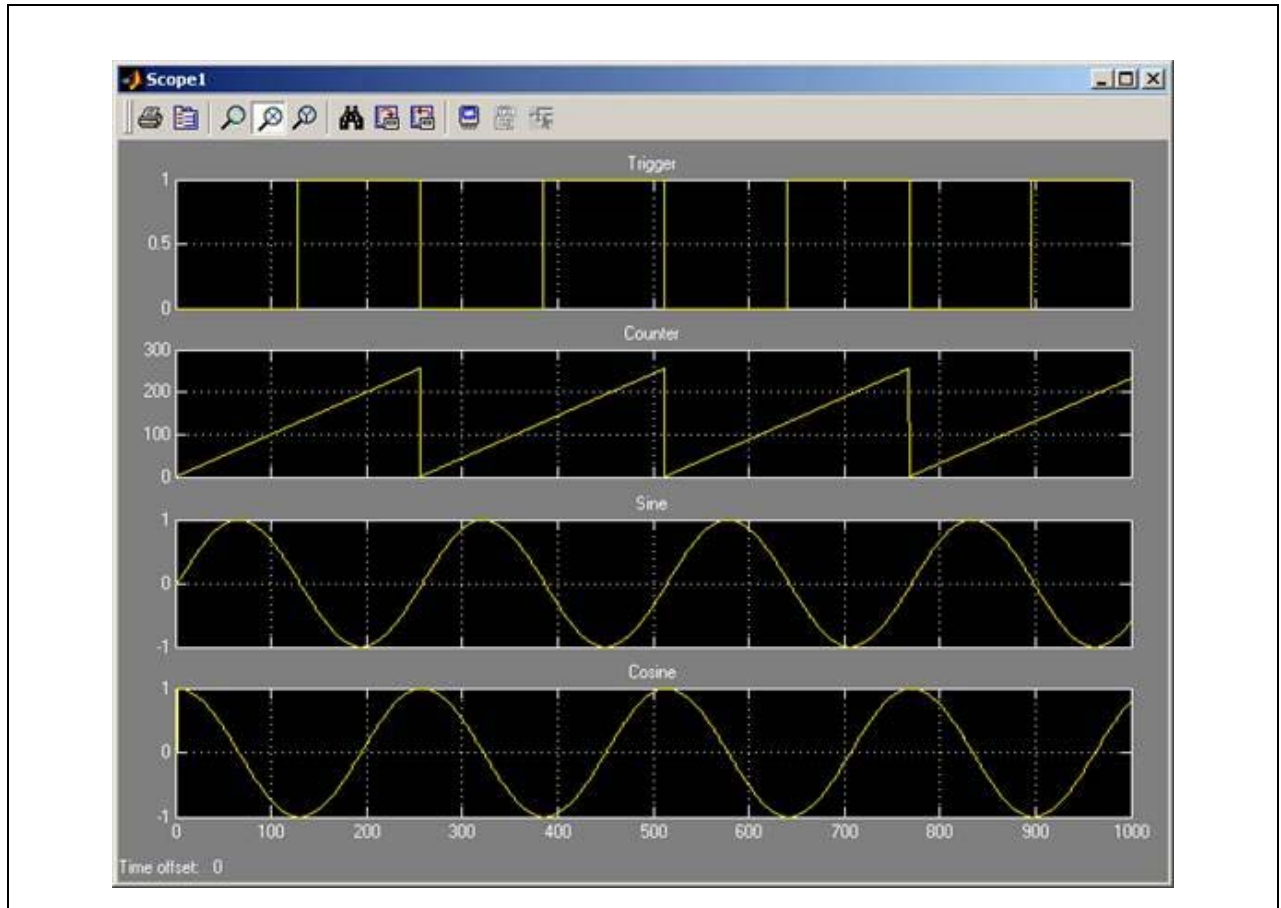


Figure 136: Scope1 Displayed Waveforms

- The first plot represents the most significant bit of the 8-bit counter. The MSB becomes 1 when the counter output is within the range of 128 through 255.
 - The second plot represents the full output of the counter.
 - The third plot shows the sine. Zoom in at the beginning of time to see a 2 clock-cycle delay due to the pipelining implemented in the SineCosine table.
 - The fourth output represents the cosine. This output has the same 2 clock-cycle delay as the Sine Wave.
5. Integrate Chipscope in the Simulink model. The Chipscope block can be found in the Simulink Library Browser in the Xilinx Blockset, under the Tools library. While holding down the left mouse button, select the Chipscope block and drag it into the open chip Simulink model.

6. Double click on the Chipscope block in order to set the following parameters:

- **Number of Trigger Ports:** Multiple trigger ports allow a larger range of events to be detected and can reduce the number of values that must be stored. Up to 16 trigger ports can be selected. In this example, only one is used.
- **Display Options for Trigger Port:** For each trigger port, the number of match units and the match type need to be set. The pull-down menu displays options for a particular trigger port. For N ports, the display options for trigger port 0 to N-1 can be shown. In this example, one Trigger port named Trig0 is used. This option should therefore be set to 0.
- **Number of Match Units for Trigger Port:** Using multiple match units per trigger port increases the flexibility of event detection. One to four match units can be used in conjunction to test for a trigger event. In this example, this option should be set to 1 since only one condition is checked for (i.e. the 8-bit counter value). Set the trigger value at run-time in the Chipscope Pro Analyzer.
- **Match Type for Trigger Port:** This option can be set to one of the following six types:
 - Basic: performs = or <> comparisons
 - Basic With Edges: in addition to the basic operations high/low, low/high transitions can also be detected
 - Extended: performs =, <>, >, <, <=, >= comparisons
 - Extended With Edges: in addition to the extended operations, high/low, low/high transitions can also be detected.
 - Range: performs =, <>, >, >=, <, <=, in range, not in range comparisons
 - Range With Edges: in addition to the range operations, high/low, low/high transitions can also be detected.

In this example, set the Match Type to **Basic With Edges**.

- **Number of Data Ports:** Up to 256 bits can be captured per sample, meaning that the sum over all ports of the bits used per port must be less than or equal to 256. System Generator propagates the data width automatically; therefore, only the number of data ports needs to be specified. In this example the sine and cosine should be viewed, therefore enter 2.
- **Depth of Capture Buffer:** The depth of the capture buffer is a power of 2, up to 16384 samples for Virtex-II, Virtex-II Pro, and Spartan-3 device families, and 4096 for Virtex, Virtex-E, Spartan-II and Spartan-IIE device families. In this example, set the depth to 512.

After parameterization the Chipscope GUI should look similar to that shown in [Figure I37 on page 212](#).

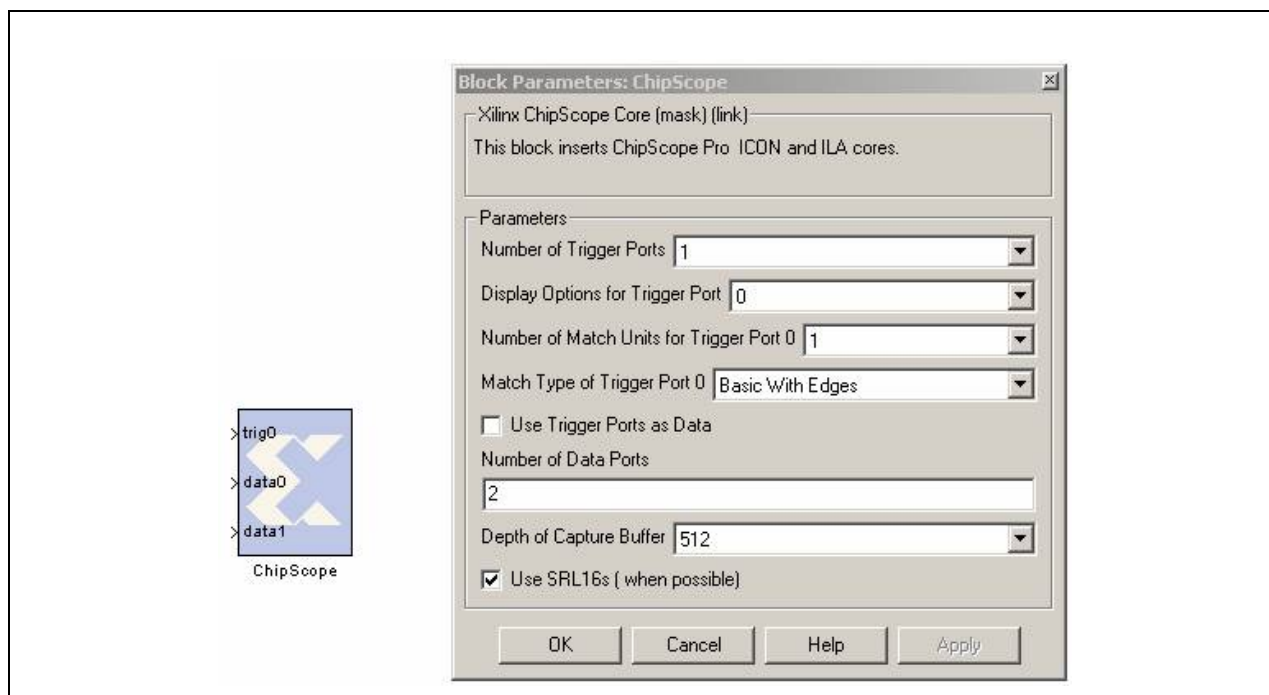


Figure I37: Parameterized Chipscope GUI

7. Connect the Chipscope Block. The signal used to trigger Chipscope is the counter output. The two buses to probe are the sine and cosine from the Sine/Cosine table. Connect the signals appropriately as shown in [Figure I38 on page 212](#).

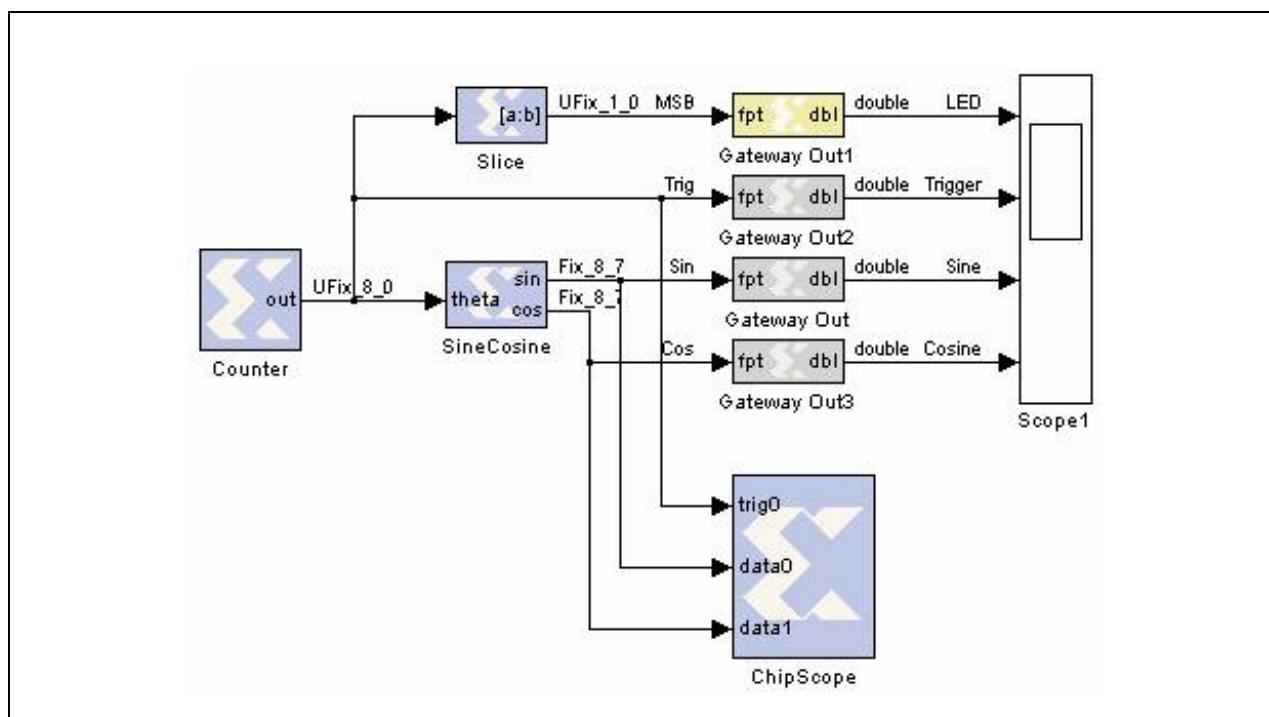


Figure I38: Connecting the Chipscope Block

8. Prepare for JTAG download. Now that the design is fully implemented and simulates correctly, the next step is to prepare it for connection to the hardware target. Although it can work on any hardware platform, the process is described for the XtremeDSP Development Kit Pro. Two pins should be locked down in this design - the trigger pin (to connect it to an LED) and the clock pin.
- Trigger Pin: Double click on the Trigger Gateway Out, select Specify IOB Location constraint and type in {'Y11'} (note the need for single quotes) as shown in **Figure 139 on page 213**.

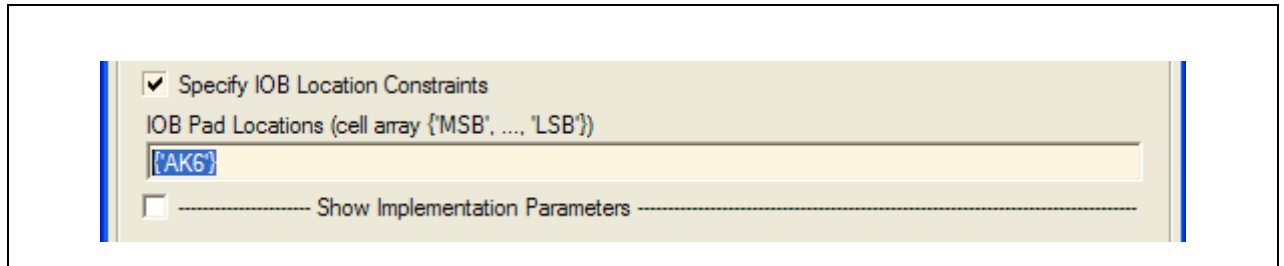


Figure 139: Trigger Pin Location

- Clock Pin: Double click on the System Generator block, set the clock period to 25ns and the clock pin location to Y13 as shown in **Figure 140 on page 213**.

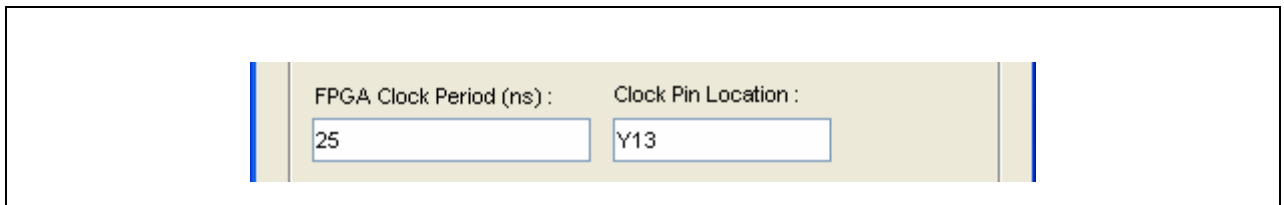


Figure 140: Clock Pin Location

If a customized board is used, pin location should be modified appropriately.

9. Generate the netlist. The last parameter to be updated before generating the netlist is the target device.
- Double click on the System Generator token and select the following part: Virtex-II Pro XC2VP30-4FF1152 depending on which part is on the board.

- Check that the System Generator parameters match the ones shown in **Figure 141** on page 214 and press Generate.

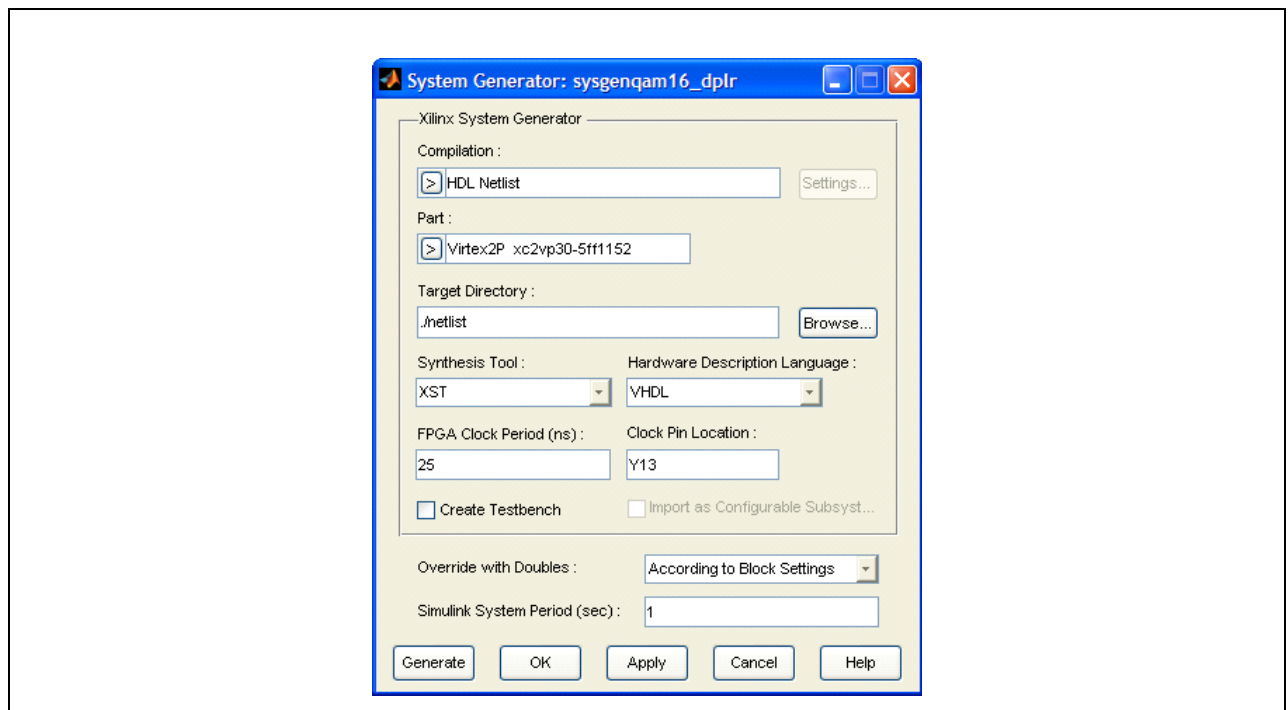


Figure 141: System Generator Parameters for Chiptest Example

10. HDL Netlist Generation. At this point, the Xilinx System Generator software calls both the Core Generator and Chipscope generator to create the netlist and cores.
 - The CORE Generatortm is used to generate the Sine/Cosine table and Counter netlists.
 - Chipscope Generator is called to create an Internal Logic Analyzer (ILA) core and an ICON core to communicate with the Chipscope Pro software via the JTAG port.
11. Generate Programming file. Once the System Generator generates the file successfully, navigate to your \$SYSGEN/examples/chipscope/netlist directory and double click on chip_clk_wrapper.npl file in order to open Xilinx Project Navigator.
 - In Project Navigator, highlight chip_clk_wrapper-structural under the Sources in Project window.
 - Under the Processes for Source window, right click on Generate Programming files and select Properties. For the XtremeDSP Development Kit Pro, the following parameter needs to be setup: Properties -> Startup Options -> FPGA Start-Up Clock -> JTAG Clock. Set up the appropriate parameters if a different board is used.
 - Generate the bitstream by double clicking on the Generate Programming File process.

22.5.4 Real-time debug

The next step is to run the design on the Kit and view the probed outputs with the Chipscope Pro Analyzer.

- I. Connect the Parallel Cable to the available JTAG headers on the Kit. If a Parallel-IV pod is used with the ribbon cable, a keyed socket (J24) is available on the board. If a flying lead connection is used a 0.1" pitch General JTAG header (J14) is provided. It is also necessary to connect the board to the USB port for power-up if using the standalone power supply.

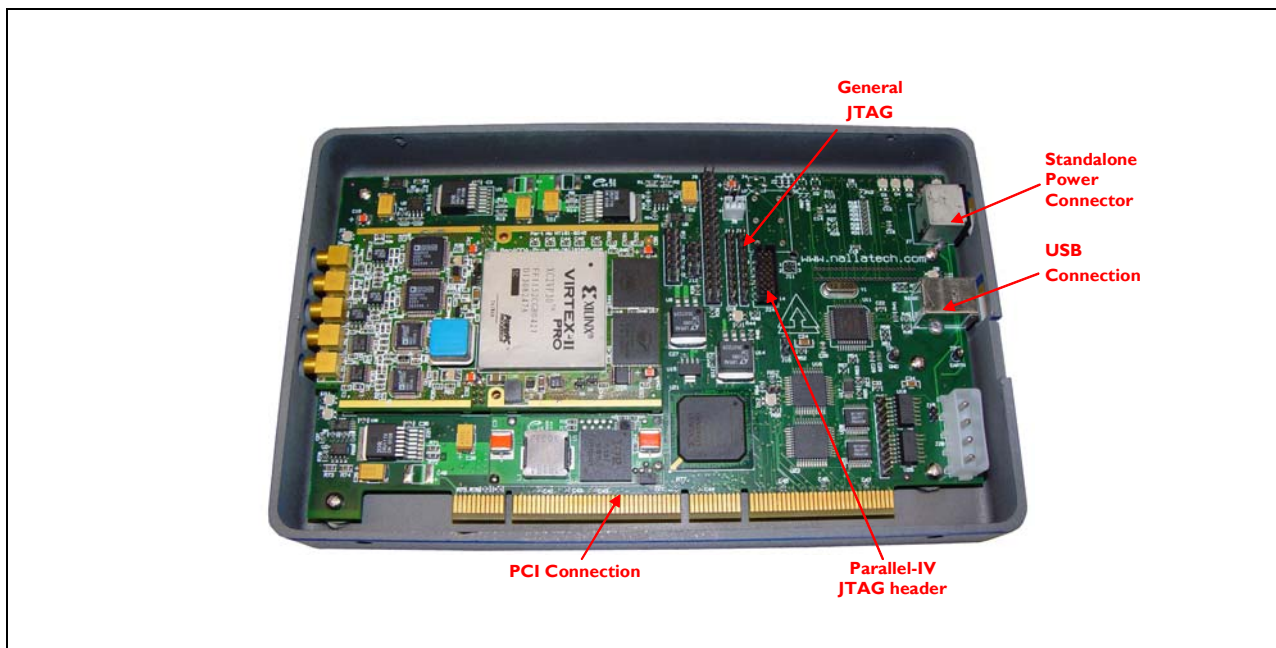


Figure 142: JTAG Header Positions on Card

General JTAG Header (Flying Lead Connection)

This is a JTAG header that connects to the chain which runs through the standard JTAG pins on the user FPGA devices in the Kit and also some of the PROMs and CPLDs. It is a standard 0.1" pitch header and can therefore support flying lead connections for the Parallel-IV pod. The header is shown in Figure 143 on page 215.

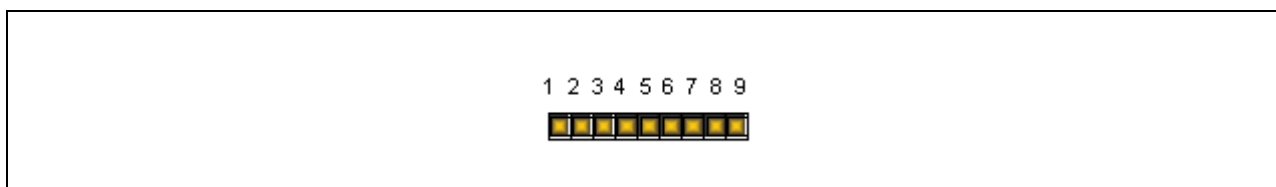


Figure 143: General JTAG Connector J14

Pin #	Name	Description
1	3.3V	3.3 Volts Supply
2	GND	Signal Ground
3	N/C	Not connected - do not use
4	TCK	ALT JTAG TCK Signal

Table 52: General JTAG Header J14 Pinouts

Pin #	Name	Description
5	N/C	Not connected - do not use
6	TDO	ALT JTAG TDO Signal
7	TDI	ALT JTAG TDI Signal
8	TRST#	ALT JTAG TRST# Signal
9	TMS	ALT JTAG TMS Signal

Table 52: General JTAG Header J14 Pinouts

Fitting a Parallel-IV Ribbon Cable

The header (J24 on the motherboard) allows a Xilinx Parallel-IV cable to be plugged in. Note that this header is simply in parallel with the General JTAG header J14. The connector is keyed and follows the pinout on the *datasheet* supplied with the Parallel-IV cable from Xilinx.



Please note that the Kit is not supplied with a Parallel-IV download cable. This section provides information on how to connect the cable only if one is available.



This is the recommended header if the Xilinx JTAG Cosimulation option in the Xilinx System Generator tool is chosen.

When the board is used inside the blue board case, the Parallel-IV ribbon cable is brought out through the small gap marked P-IV in the side of the case. This case should be opened, the Parallel-IV ribbon cable fitted and then the case closed again. To fit the cable remove the four screws round the sides of the case and remove the lid. Then plug one end of the Parallel-IV ribbon cable into the keyed header and bring the cable out to the small opening in the base of the case as shown in [Figure 144 on page 217](#).

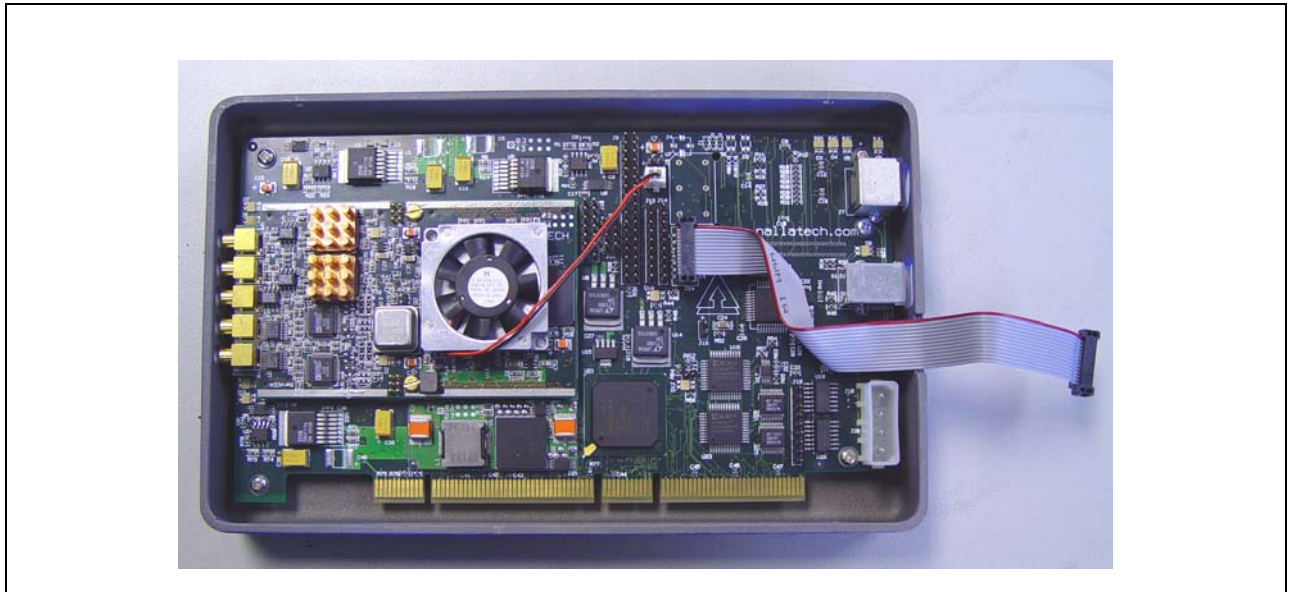


Figure 144: Routing the Parallel-IV Cable Out from Header

Then fit the lid back onto the case, refitting the screws if necessary, and ensure that the ribbon cable is not caught between the top and bottom of the blue case. Finally, connect the Parallel-IV pod to the exposed end of the ribbon cable as shown in [Figure 145 on page 217](#). Both the connector on the cable and the socket on the pod are keyed for ease of use.

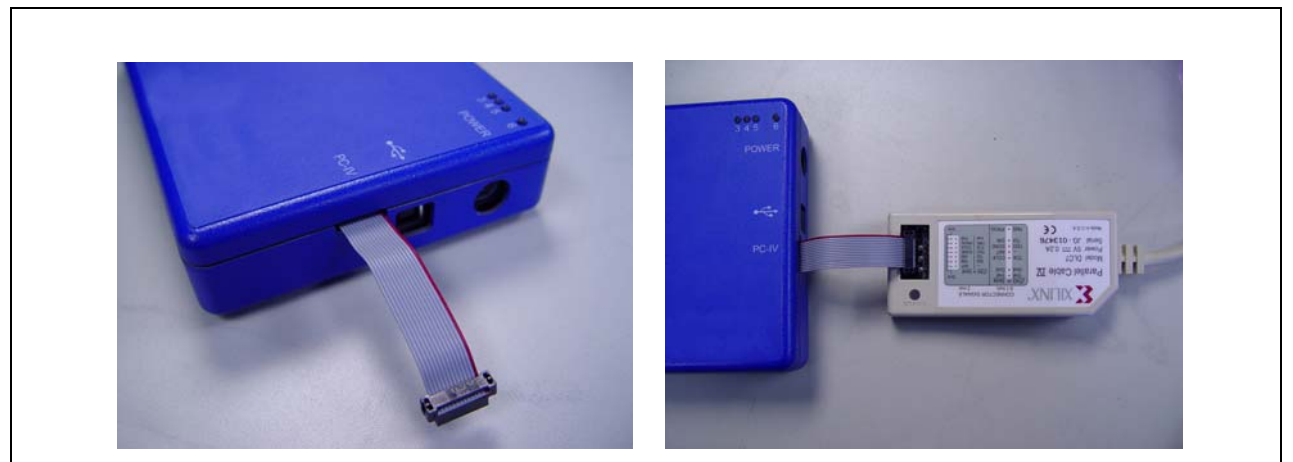


Figure 145: Re-fitting the Case Lid (first), Connecting the Parallel-IV Pod (second)

2. Open FUSE to power up the board.
 - Select Start->Programs-> FUSE -> Software -> FUSE Probe
 - From FUSE, select Card Control -> Open Card. Then select USB for the interface and click on Locate Card. At this point FUSE locates the BenONE-Kit Motherboard and the top left corner of the FUSE Probe Tool displays the FPGA information as shown in [Figure 117 on page 197](#). If you have the card plugged into a PCI slot then select to open the card via PCI.

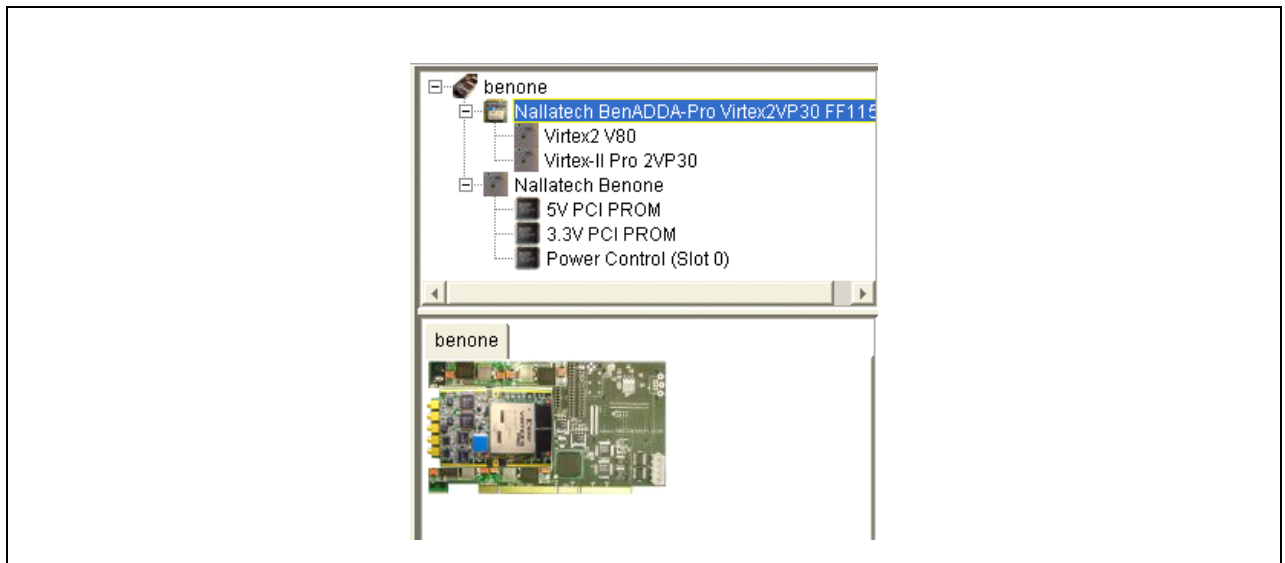


Figure I46: Opening the Card in FUSE

3. Open Chipscope Pro Analyzer:
 - Select Start -> Programs -> Chipscope Pro -> Chipscope Pro Analyzer.
 - Open JTAG Chain by clicking on JTAG Chain -> Xilinx Parallel Cable and selecting Xilinx Parallel IV Cable.
 - Note the index for the Virtex-II Pro devices available on the XtremeDSP Kit: XC2VP30-4FF1152 is index 3, and xc2v80 is index 4.
4. Configure the FPGAs using the following procedures:
 - Under the New Project Window, right click on Device 3 and select Configure-> Device 3. At this point, you need to look for the bistream which was generated in Step 11 on [page 214](#). Select New File and scroll to your project directory: \$SYSGEN/examples/chipscope/netlist/chip_clk_wrapper.bit. After configuration, the status window at the bottom of the Chipscope Analyzer should reflect that one Chipscope core was found in the JTAG chain.
 - Similarly, select Device 4 and Configure it with the osc_clock_2v80.bitfile provided in the Chipscope project directory. This second bitstream is used to program the clock driver FPGA on the board.
5. Import Chipscope Project File. System generator creates a project file for Chipscope in order to group data signals into busses. A bus is created for each data port so that it can be viewed in the same manner (sign and precision) in which it was viewed in the Simulink environment. Load this project file by going under File-> Import -> Select New File, and select chip_chipscope.cdc.
6. Plot the Sine Waves. In the New Project window, under Device 3 -> Unit 0 ILA, double click on Bus Plot. A Bus Plot window appears as shown in [Figure I47 on page 219](#). Select data0 and data1 in the Bus Selection section and then arm the trigger.

Since no trigger conditions have been set, values are captured immediately. Both the sine and cosine appear as shown in [Figure I47 on page 219](#). The display option can be changed to represent the waveforms with points, lines, or both.

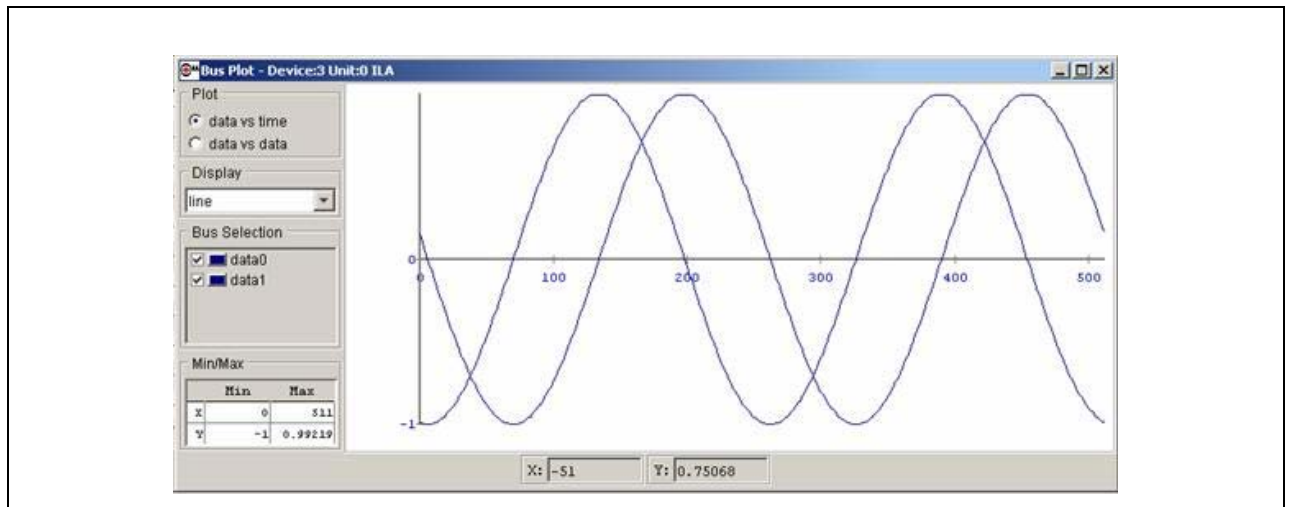


Figure 147: Chipscope Bus Plot

7. Setup Trigger. In the Trigger Setup window, shown in Figure 148 on page 219, change the current XXXX-XXXX value with 0000-0000. Once the counter hits 0, Chipscope starts capturing values. Earlier, the buffer was setup to 512, so 512 data points can be visualized in Chipscope. Re-capture the data.

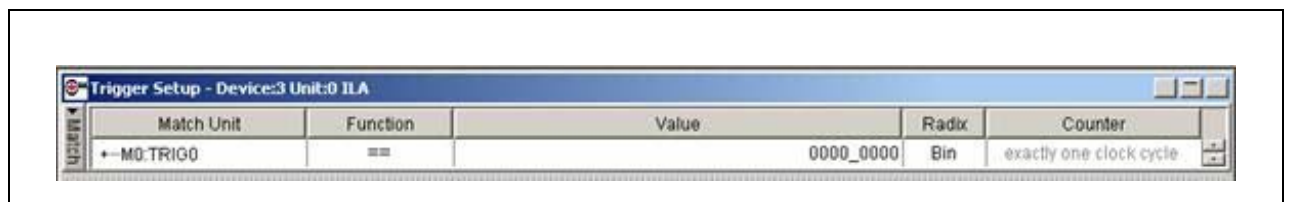


Figure 148: Trigger Setup

Again, a 2 clock delay is seen at the beginning of time on the sine due to the 2 clock latency through the SineCos Look Up Table. Modify the trigger value to 0000-0010 (decimal 2) and re-capture the data. Now the sine and cosine start at 0 and 1 respectively as shown in Figure 149 on page 219.

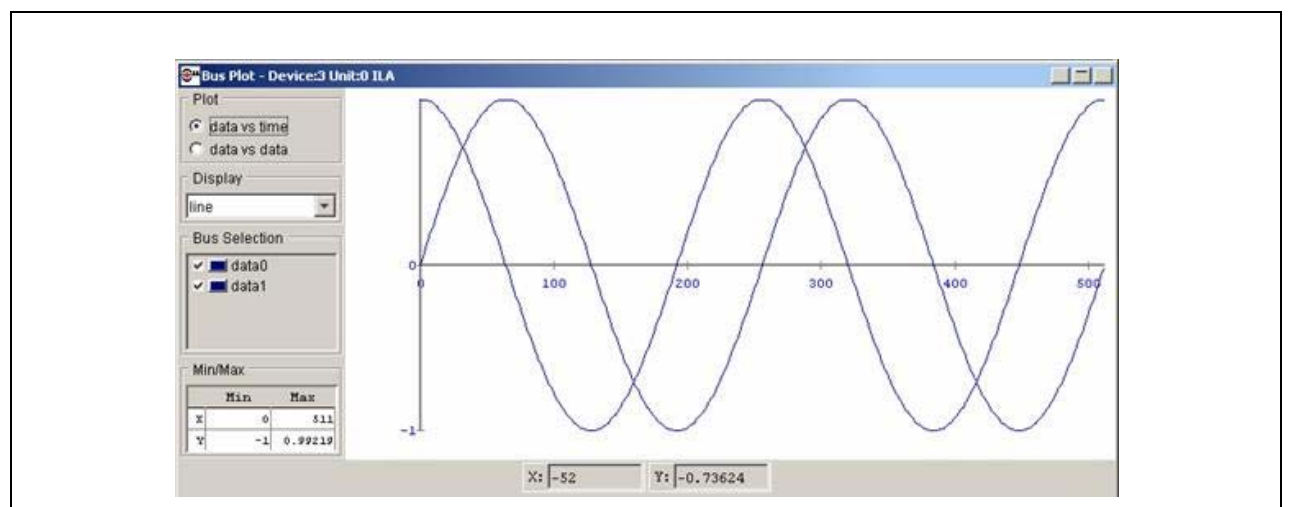


Figure 149: Chipscope Bus Plot

22.5.5 Importing Data Into MATLAB Workspace From Chipscope

The data captured by Chipscope can now be exported back into the MATLAB workspace.

1. Export data from Chipscope Pro Analyzer. Select File -> Export option from within Chipscope Pro Analyzer. Select ASCII format and choose All Signals/Buses to export. Press the Export button and save the file as `sinecos.prn`.
2. Start MATLAB and change the current working directory to the location where you saved `sinecos.prn`. Type `xlLoadChipScopeData('sinecos.prn');` This loads the data from the .prn file into the MATLAB workspace. In the workspace there are two new arrays named `data0` and `data1`.
3. The values can be plotted using the MATLAB plot function. Type: `plot(1:512, data0, 1:512, data1)` which gives the plot shown in **Figure 150 on page 220**.

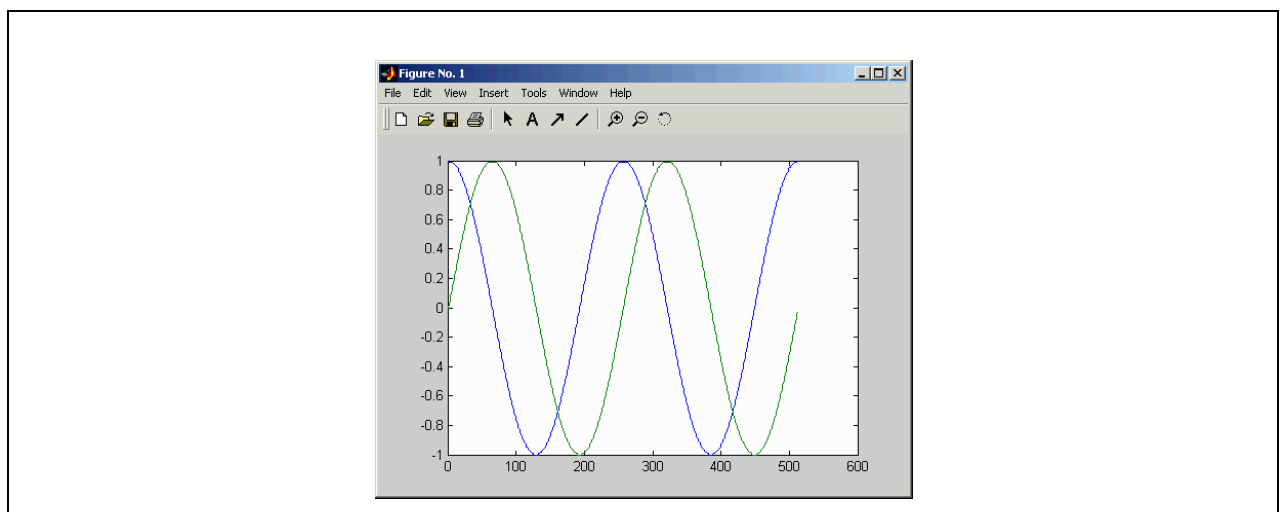


Figure 150: Plotted Data in MATLAB

22.6 System Generator and UltraController Reference Design

This reference design shows how the UltraController can be brought into System Generator using the Black Box interface, and then co-simulated along with existing System Generator models using the XtremeDSP Development Kit Pro. Also presented is a step-by-step example showing how to co-simulate the UltraController alongside an LMS-based adaptive filter data path. A tutorial documenting this reference design may be found at the following URL:

http://www.xilinx.com/products/software/sysgen/app_docs/user_guide_Chapter_7_Section_17_subsection_3.htm



The demonstration models included in this zip file should be simulated using System Generator v6.3 with Xilinx ISE v6.3 Service Pack 3.

22.6.1 Reference Design Contents

The example is installed to the local machine with the other examples. The files are located in:

<install location>\XtremeDSP Development Kit Pro\Examples\system_generator_examples\uc_xdspkit_demo

In this route directory the files listed in [Table 53 on page 221](#) are provided.

File	Description
balanced_w_uc_bmm.opt	XFLOW options file that specifies the Xilinx tool options necessary to incorporate the appropriate Block RAM Memory Map (BMM) file for the UltraController.
run_data2mem.m	Example script that runs the Xilinx Data2Mem utility to configure the UltraController co-simulation bitstream with the example ELF file program image.
sysgen_ultracontroller.mdl	Provides the minimum blocks in which the UltraController black box block can be added and compiled for hardware co-simulation.
ultracontroller_bb.ngc	Xilinx NGC netlist file that comprises the UltraController reference design and provides a port interface suitable for use in System Generator. Note: Uses the 4i_4d_2ppc UltraController design.
ultracontroller_bb.vhd	VHDL black box wrapper that instantiates the UltraController 4i_4d_2ppc as a component.
ultracontroller_bb_config.m	Black box configuration m-function for the UltraController reference design.
ultracontroller_bb_lib.mdl	Library that includes a Black Box block pre-configured with the settings necessary to import the UltraController reference design in System Generator.
ultracontroller_db_calc.c	C source code that computes the LMS filter error driving the UltraController 32-bit GPIO input in dB. Input values are assumed to have 18 fractional bits. Error value in dB is written to the GPIO port (also with 18 fractional bits).
ultracontroller_db_calc.elf	Example PowerPC 405 ELF object file.
ultracontroller_lms_filter_ex.mdl	System Generator model that contains an LMS adaptive filter implementation. The UltraController hardware co-simulation block is used to report the filter error in dB.
xlCreateXDSPKitUCBmm.p	Generates an appropriate BMM file for the UltraController when the design is compiled into hardware.

Table 53: Description of Files in Route Directory

In the sub folder ".\uc_xdspkit_demo\lms_demo_jtag" the files listed in [Table 54 on page 221](#) are contained.

File	Description
ultracontroller_lms_filter_ex_jtag.mdl	LMS filter example model with pre-compiled hardware co-simulation block for the XtremeDSP Development Kit Pro using the JTAG interface.
download.bit	Pre-compiled JTAG hardware co-simulation bitstream for the XtremeDSP Development Kit Pro.

Table 54: Description of Files in Sub Folder

In the sub folder ".\uc_xdspkit_demo\lms_demo_pci" the files listed in [Table 55 on page 222](#) are contained:

File	Description
ultracontroller_lms_filter_ex_pci.mdl	LMS filter example model with pre-compiled hardware co-simulation block for the XtremeDSP Development Kit Prousing the PCI interface.
download.bit	Pre-compiled PCI hardware co-simulation bitstream for the XtremeDSP Development Kit Pro.

Table 55: Description of Files in Sub Folder

22.6.2 Running the Examples

Please follow the weblink at the start of this section to the Xilinx website for full details of the example. To quickly run the examples there are pre-compiled versions for both PCI and JTAG in the folders 'lms_demo_pci' and 'lms_demo_jtag' respectively. Simply open the Simulink model (MDL) files and click on the 'play' button to start the simulation.

Note that when using JTAG you will need the Parallel-IV cable connected to the board as described previously in the System Generator section.



NALLATECH
The High Performance FPGA Solutions Company

Part XI: Board Firmware

This part of the User Guide provides details on how to update the firmware supplied with the XtremeDSP Development Kit Pro.

Section 23

Changing the Firmware

In this section:

- Introduction
 - The Firmware Utilities
 - How to Perform a Firmware Update
-

23.1 Introduction

The Kit comes preconfigured with a 32bit/33MHz PCI core for 5V PCI signal environment and USB1.1 core for connecting the board via USB cable. It may be necessary to carry out a firmware update or to change the firmware so the card can be used in a 3.3V PCI signalling PCI slot as highlighted in the *Getting Started Guide*.

Please note that the interface firmware is provided in two PROMs. One prom can be used for 5V PCI signalling whilst the other can be used for either 3.3V PCI signalling or USB. Therefore if you configure the second PROM to allow for use in a 3.3V PCI signalling slot the USB interface on the Kit is disabled. However you can change the firmware back if necessary. This section describes two utility programs for loading in the specific firmware.



Please note that if power is removed from the card during reconfiguration, or there is a power outage, then the card may need to be returned for default configuration re-programming.

23.2 The Firmware Utilities

Getting the Firmware Update Utilities

Please note that these utilities are not provided on the CD and can be downloaded from the support lounge if required.

Benone_32PCI_USB.exe

This file allows the BenONE-Kit Motherboard to be configured with a 32-bit/33MHz PCI core for the 5V PCI signal environment and a USB core for a USB1.1 connection to the host PC.

Benone_32PCI_64PCI.exe

This file allows the BenONE-Kit Motherboard to be configured with a 32-bit/33MHz PCI core for the 5V PCI signal environment and a 64bit/33MHz PCI core for the 3.3V PCI signal environment.

23.3 Performing a Firmware Update



It is strongly advised to update the firmware via a PCI slot if available - generally a 32-bit PCI slot. Updating over USB takes significantly longer and therefore raises the risk of problems due to power or host computer problems. USB should only be used for changing the board firmware if there is no way of using a 32-bit PCI slot.

23.3.1 Converting from 5VIO PCI and USB firmware to 5VIO and 3.3VIO PCI firmware

The firmware can be updated using the PCI connection or the USB connection - however using a 32-bit PCI slot is highly recommended.

Updating via PCI

▼ **To update via PCI use the following procedures:**

1. With the PC powered OFF, insert your Kit in a PCI slot.
2. After the PC starts up, click on the Windows start menu and launch '**Programs->XtremeDSP Development Kit Pro->Software->Update to PCI Firmware Only**'. Alternatively open Windows Explorer and navigate to the firmware folder in the XtremeDSP Development Kit Pro installation and run the Benone_32PCI_64PCI.exe program (see [Figure 151 on page 226](#)).
3. Click on the Card Menu and select the PCI interface to detect cards on. A list of detected cards is displayed. Click on the selected card.
4. Click on '**Update Firmware**'.
5. Programming should take approximately 10 to 20 seconds to configure via PCI.
6. The firmware change is implemented the next time the power is cycled.

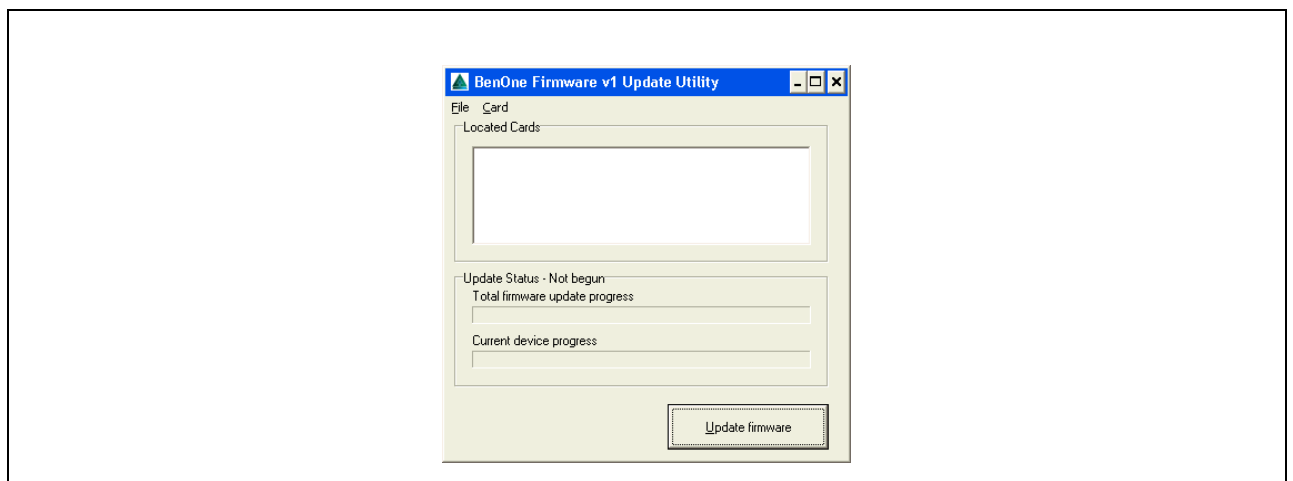


Figure 151: Firmware Update Utility

Updating via USB

▼ **To update via USB use the following procedures:**

1. Apply power to the Kit via the external power connector.
2. Attach the Kit to your host PC using the USB cable, once the PC has finished starting up.
3. Double click on file Benone_32PCI_32PCI.exe to launch the application (see **Figure 151 on page 226**).
4. Click on the Card Menu and select the USB interface to detect cards on. A list of detected cards is displayed. Click on the selected card.
5. Click on 'Update Firmware'.
6. Programming should take approximately between 18 and 22 minutes to configure via USB, although this time depends upon the USB configuration.
7. The firmware change is implemented the next time the power is cycled.

23.3.2 Converting from PCI Firmware only to 5VIO PCI and USB firmware



This procedure can only be performed with the card plugged into a PCI slot as the USB firmware is not currently present in the board.

▼ **To convert from PCI firmware only to 5VIO PCI and USB firmware use the following procedures:**

1. With the PC powered OFF, insert your Kit in a PCI slot.
2. After the PC starts up open Windows Explorer and navigate to the firmware folder containing the Benone_32PCI_USB.exe program.
3. Double click on file Benone_32PCI_USB.exe to launch the application.
4. Click on the Card Menu and select the PCI interface to detect cards on. A list of detected cards is displayed. Click on the selected card.
5. Click on 'Update Firmware'.
6. Programming should take approximately 10 to 20 seconds to configure via PCI.
7. The firmware change is implemented the next time the power is cycled.



NALLATECH
The High Performance FPGA Solutions Company

Part XII:Reference Information

This part of the User Guide provides reference information on the XtremeDSP Development Kit Pro, and includes pinout information.

Section 24

Pinout Information

In this section:

- Local Bus Pinouts - Interface Communications
- Adjacent Out Bus - Interface Communications
- Adjacent Header
- PLinks 0 - PLink Header (J10)
- PLinks 7 - Nallatech RS232 Connections
- DIME-II Control and Monitoring Signals
- ZBT SRAM
- Inter-FPGA Clock Infrastructure Signals
- DAC Signal Pinouts
- ADC Signal Pinouts
- User I/O Header

24.1 Local Bus Pinouts - Interface Communications

Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
LBUS<0>	PB1	F4
LBUS<1>	PB2	H2
LBUS<2>	PB3	H1
LBUS<3>	PB4	K5
LBUS<4>	PB6	K4
LBUS<5>	PB7	J2
LBUS<6>	PB8	K2
LBUS<7>	PB9	K1
LBUS<8>	PB10	L8
LBUS<9>	PB11	L7

Table 56: Local Bus Pinouts (XC2VP30-4FF1152)

Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
LBUS<10>	PB12	L6
LBUS<11>	PB13	M10
LBUS<12>	PB15	N10
LBUS<13>	PB16	M9
LBUS<14>	PB17	N9
LBUS<15>	PB18	N8
LBUS<16>	PB19	M7
LBUS<17>	PB20	N7
LBUS<18>	PB21	M6
LBUS<19>	PB22	L3
LBUS<20>	PB24	L2
LBUS<21>	PB25	L1
LBUS<22>	PB26	L5
LBUS<23>	PB27	N4
LBUS<24>	PB28	L4
LBUS<25>	PB29	M2
LBUS<26>	PB30	P10
LBUS<27>	PB31	P9
LBUS<28>	PB33	M4
LBUS<29>	PB34	N6
LBUS<30>	PB35	N5
LBUS<31>	PB36	P2

Table 56: Local Bus Pinouts (XC2VP30-4FF1152)

24.2 Adjacent Out Bus - Interface Communications

Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
ADJOUT<0>	PD29	U28
ADJOUT<1>	PD30	U27
ADJOUT<2>	PD31	V33
ADJOUT<3>	PD32	U33
ADJOUT<4>	PD33	U32
ADJOUT<5>	PD34	U31
ADJOUT<6>	PD35	V29

Table 57: Adjacent OUT BUS Pinouts - User FPGA (XC2VP30-4FF1152)

24.3 Adjacent Header (J8)

Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
ADJIN<0>	PA29	AL17
ADJIN<1>	PA30	AK17
ADJIN<2>	PA31	W11
ADJIN<3>	PA32	V11
ADJIN<4>	PA33	AA1
ADJIN<5>	PA34	Y1
ADJIN<6>	PA35	V10
ADJIN<7>	PA36	V9
ADJIN<8>	PA38	Y2
ADJIN<9>	PA39	W2
ADJIN<10>	PA40	W8
ADJIN<11>	PA41	W7
ADJIN<12>	PA42	W6
ADJIN<13>	PA43	W5
ADJIN<14>	PA44	W4
ADJIN<15>	PA45	W3
ADJIN<16>	PA47	Y4
ADJIN<17>	PA48	Y3
ADJIN<18>	PA49	W10
ADJIN<19>	PA50	W9
ADJIN<20>	PA51	Y10
ADJIN<21>	PA52	Y9
ADJIN<22>	PA53	Y7
ADJIN<23>	PA54	Y6
ADJIN<24>	PA56	AA6
ADJIN<25>	PA57	AA5
ADJIN<26>	PA58	AA4
ADJIN<27>	PA59	AA3

Table 58: Adjacent IN BUS Pinouts - User FPGA (XC2VP30-4FF1152)

24.4 PLINKS 0: PLINK Header (J10)

Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
PP0LK<0>	PA2	U7
PP0LK<1>	PA3	U8
PP0LK<2>	PA4	U5
PP0LK<3>	PA5	U6
PP0LK<4>	PA6	U3
PP0LK<5>	PA7	U4
PP0LK<6>	PA8	V4
PP0LK<7>	PA9	V3
PP0LK<8>	PA11	V2
PP0LK<9>	PA12	U2
PP0LK<10>	PA13	V8
PP0LK<11>	PA14	V7

Table 59: PLINK Pinouts - User FPGA (XC2VP30-4FF1152)

24.5 PLINKS 7: Nallatech RS232 Connections in Part

Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
PP7LK<0>	SC47	AF31
PP7LK<1>	SC48	AF32
PP7LK<2>	SC49	AK31
PP7LK<3>	SC51	AK32
PP7LK<4>	SC52	AH29
PP7LK<5>	SC53	AH30
PP7LK<6>	SC54	AH27
PP7LK<7>	SC55	AG28
PP7LK<8>	SC56	AJ27
PP7LK<9>	SC57	AJ28
PP7LK<10>	SC58	AL33
PP7LK<11>	SC60	AL34

Table 60: PLINK Pinouts - User FPGA (XC2VP30-4FF1152)

24.5.1 Nallatech RS232 Test Header (J9 and J12)

Note that this header is intended for Nallatech debug purposes. No cable is supplied for connection to this header. Details are provided here for fullness of information about the headers on the card.

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No	Header Pin
R2	AF31	J9 pin 2
T2	AF32	J9 pin 1
R1	AH29	J12 pin 2
T1	AH30	J12 pin 1

Table 61: Nallatech RS232 Test Header Pinouts

- Note that pins 3 and 4 on both J9 and J12 are ground connections.
- Pin 1 on J9 or J12 is marked on the silk-screen as the top left pin on each header.

24.6 DIME-II Control and Monitoring Signals

24.6.1 DIME-II Specific Pins

DIME-II Connector PIN No	Signal Name	DIME-II Connector PIN No	User FPGA (XC2VP30-4FF1152) PIN No
CONFIG_DONE	FPGA_DONE	PC40	L32
N/C	CONFIG_DONE ^a	N/a	L31
CLK0	CLKA	PC24	H17
CLK1	CLKB	PC31	AJ17
CLK2	CLKC	PC42	J18
RESETI	RESETI	PC15	P28
SLOT_ID0	SLOT_ID0	PC51	N32
SLOT_ID1	SLOT_ID1	PC52	N31

Table 62: User FPGA Specific Pinouts (XC2VP30-4FF1152)

- a. CONFIG_DONE is driven from the I/O of the Virtex-II into the base of the transistor to signal that the on board User FPGA has been configured successfully. User to drive this pin LOW once the FPGA is configured.

24.6.2 User LEDs

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
LED_Green1	AK6
LED_Red1	AK7

Table 63: User LED Pinouts - User FPGA (XC2VP30-4FF1152)

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
LED_Green2	AL29
LED_Red2	AL30

Table 63: User LED Pinouts - User FPGA (XC2VP30-4FF1152)

24.6.3 On Board Temperature Sensor

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ALERTI	F31

Table 64: Temperature Sensor Pinouts (XC2VP30-4FF1152)

24.7 ZBT SRAM

24.7.1 Clock and Control Signals for ZBT

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTA_CLK	G18
ZBTA_CLK_FB_OUT	F18
ZBTA_CLK_FB_IN	H18
ZBTA_ADV	J27
ZBTA_CKEI	F30
ZBTA_CSI<0>	J28
ZBTA_CSI<1>	E34
ZBTA_OEI	E32
ZBTA_WEI	E33
ZBTB_CLK	F17
ZBTB_CLK_FB_OUT	G17
ZBTB_CLK_FB_IN	D17
ZBTB_ADV	J7
ZBTB_CKEI	E2
ZBTB_CSI<0>	E1
ZBTB_CSI<1>	J8
ZBTB_OEI	F5
ZBTB_WEI	E3

Table 65: ZBT Clock and Control Signals

24.7.2 ZBT Address Signals for ZBT Bank A

Signal Name	User FPGA (XC2VP30-4FFI152) PIN No	Signal Name	User FPGA (XC2VP30-4FFI152) PIN No
ZBTA_A<0>	D29	ZBTA_A<11>	J25
ZBTA_A<1>	G25	ZBTA_A<12>	E26
ZBTA_A<2>	K23	ZBTA_A<13>	F28
ZBTA_A<3>	E28	ZBTA_A<14>	E29
ZBTA_A<4>	G22	ZBTA_A<15>	D30
ZBTA_A<5>	J23	ZBTA_A<16>	E31
ZBTA_A<6>	F26	ZBTA_A<17>	K24
ZBTA_A<7>	H22	ZBTA_A<18>	J24
ZBTA_A<8>	G26	ZBTA_A<19>	F27
ZBTA_A<9>	H26		
ZBTA_A<10>	H25		

Table 66: ZBT Address Signals Pinouts Bank A

24.7.3 ZBT Data Signals for ZBT Bank A

Signal Name	User FPGA (XC2VP30-4FFI152) PIN No
ZBT_D<0>	E21
ZBT_D<1>	C22
ZBT_D<2>	E22
ZBT_D<3>	D23
ZBT_D<4>	C24
ZBT_D<5>	D24
ZBT_D<6>	D25
ZBT_D<7>	E25
ZBT_D<8>	D19
ZBT_D<9>	D20
ZBT_D<10>	C21
ZBT_D<11>	D22
ZBT_D<12>	D18
ZBT_D<13>	E18
ZBT_D<14>	E19
ZBT_D<15>	D21

Table 67: ZBT Data Signals Pinouts Bank A

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBT_D<16>	G19
ZBT_D<17>	J19
ZBT_D<18>	F20
ZBT_D<19>	F21
ZBT_D<20>	H19
ZBT_D<21>	K18
ZBT_D<22>	K19
ZBT_D<23>	J20
ZBT_D<24>	G20
ZBT_D<25>	H21
ZBT_D<26>	F22
ZBT_D<27>	L18
ZBT_D<28>	F19
ZBT_D<29>	G21
ZBT_D<30>	K20
ZBT_D<31>	L19

Table 67: ZBT Data Signals Pinouts Bank A

24.7.4 ZBT Address Signals for Bank B

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No	Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTB_A<0>	E9	ZBTB_A<11>	E7
ZBTB_A<1>	F9	ZBTB_A<12>	F8
ZBTB_A<2>	K11	ZBTB_A<13>	H10
ZBTB_A<3>	E6	ZBTB_A<14>	H9
ZBTB_A<4>	H13	ZBTB_A<15>	E4
ZBTB_A<5>	J11	ZBTB_A<16>	F7
ZBTB_A<6>	D6	ZBTB_A<17>	G13
ZBTB_A<7>	K12	ZBTB_A<18>	J12
ZBTB_A<8>	J10	ZBTB_A<19>	D5
ZBTB_A<9>	G10		
ZBTB_A<10>	G9		

Table 68: ZBT Address Signals Pinouts Bank B

24.7.5 ZBT Data Signals for Bank B

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
ZBTB_D<0>	E14
ZBTB_D<1>	F15
ZBTB_D<2>	F14
ZBTB_D<3>	G15
ZBTB_D<4>	G14
ZBTB_D<5>	H14
ZBTB_D<6>	J15
ZBTB_D<7>	L16
ZBTB_D<8>	J17
ZBTB_D<9>	L17
ZBTB_D<10>	G16
ZBTB_D<11>	K15
ZBTB_D<12>	K17
ZBTB_D<13>	J16
ZBTB_D<14>	K16
ZBTB_D<15>	H16
ZBTB_D<16>	D16
ZBTB_D<17>	D15
ZBTB_D<18>	D14
ZBTB_D<19>	D13
ZBTB_D<20>	E17
ZBTB_D<21>	E16
ZBTB_D<22>	F16
ZBTB_D<23>	C14
ZBTB_D<24>	D12
ZBTB_D<25>	C11
ZBTB_D<26>	D11
ZBTB_D<27>	D10
ZBTB_D<28>	C13
ZBTB_D<29>	F13
ZBTB_D<30>	E13
ZBTB_D<31>	E10

Table 69: ZBT Data Signals Pinouts Bank B

24.8 Inter-FPGA Clock Infrastructure Signals

24.8.1 Clock sources arriving at CLOCK FPGA

Signal Name	Clock FPGA (XC2V80-4CS144) Pin No	User FPGA (XC2VP30-4FF1152) PIN No	Signal Description
CLK_Op_Amp	B6 (GCLK6S)	N/a	External Clock source via Op_Amp
CLK_Op_Ampl	C6 (GCLK7P)	N/a	Complement of External Clock source via Op_Amp
Osc_CLK	M6 (GCLK4P)	N/a	LVTTTL Clock Oscillator
GEN_CLKA	K7 (GCLK0P)	AF18	Generated Clock A
GEN_CLKC	N8 (GCLK1S)	AG18	Generated Clock C
GEN_CLKB	M7 (GCLK6P)	AK19	Generated Clock B
GEN_CLKD	N7 (GCLK7S)	AL19	Generated Clock D

Table 70: Clock Signals at Clock FPGA

24.8.2 Clock Feedback Signals

Signal Name	Clock FPGA (XC2V80-4CS144) Pin No	User FPGA (XC2VP30-4FF1152) PIN No	Signal Description
CLK1_FB	J2	AH18	Feedback to User FPGA
CLK3_FB	H4	AJ18	Feedback to User FPGA
CLK2_FB	H12	AH19	Feedback to User FPGA

Table 71: Clock Feedback Signals

24.8.3 Clocking Pinouts for DACs and ADCs

Signal Name	Clock FPGA (XC2V80-4CS144) Pin No
ADC_CLKA	E4
ADC_CLKAI	D1
ADC_CLKB	G1
ADC_CLKBI	F1
DAC_CLKA	D13

Table 72: Clocking Pinouts for DACs and ADCs

Signal Name	Clock FPGA (XC2V80-4CS144) Pin No
DAC_CLKAI	D12
DAC_CLKB	G10
DAC_CLKBI	F12

Table 72: Clocking Pinouts for DACs and ADCs

24.9 DAC Signal Pinouts

Signal Name (DAC 1)	User FPGA (XC2VP30-4FF1152) PIN No	Signal Name (DAC 2)	User FPGA (XC2VP30-4FF1152) PIN No
DAC1_D<0>	AE22	DAC2_D<0>	AK21
DAC1_D<1>	AE21	DAC2_D<1>	AJ20
DAC1_D<2>	AF21	DAC2_D<2>	AM21
DAC1_D<3>	AK27	DAC2_D<3>	AH20
DAC1_D<4>	AL27	DAC2_D<4>	AL23
DAC1_D<5>	AH25	DAC2_D<5>	AL21
DAC1_D<6>	AG25	DAC2_D<6>	AJ21
DAC1_D<7>	AF24	DAC2_D<7>	AH21
DAC1_D<8>	AH22	DAC2_D<8>	AM22
DAC1_D<9>	AG22	DAC2_D<9>	AL22
DAC1_D<10>	AJ22	DAC2_D<10>	AG19
DAC1_D<11>	AK24	DAC2_D<11>	AF19
DAC1_D<12>	AJ24	DAC2_D<12>	AL20
DAC1_D<13>	AF22	DAC2_D<13>	AD18
DAC1_DIV0	AG26	DAC2_DIV0	AH19
DAC1_DIV1	AH26	DAC2_DIV1	AJ19
DAC1_MOD0	AK29	DAC2_MOD0	AF20
DAC1_MOD1	AK28	DAC2_MOD1	AG21
DAC1_PLLLOCK	AE24	DAC2_PLLLOCK	AL24
DAC1_RESET	AF25	DAC2_RESET	AM24

Table 73: DACs Signal Pinouts

24.10 ADC Signal Pinouts

Signal Name (DAC 1)	User FPGA (XC2VP30-4FF1152) PIN No	Signal Name (DAC 2)	User FPGA (XC2VP30-4FF1152) PIN No
ADCI_D<0>	AE16	ADC2_D<0>	AH13
ADCI_D<1>	AK16	ADC2_D<1>	AG13
ADCI_D<2>	AJ16	ADC2_D<2>	AJ11
ADCI_D<3>	AH16	ADC2_D<3>	AK11
ADCI_D<4>	AJ15	ADC2_D<4>	AJ13
ADCI_D<5>	AL15	ADC2_D<5>	AE15
ADCI_D<6>	AL14	ADC2_D<6>	AL11
ADCI_D<7>	AL13	ADC2_D<7>	AE14
ADCI_D<8>	AL12	ADC2_D<8>	AK13
ADCI_D<9>	AG16	ADC2_D<9>	AF15
ADCI_D<10>	AH15	ADC2_D<10>	AF14
ADCI_D<11>	AF16	ADC2_D<11>	AG14
ADCI_D<12>	AK14	ADC2_D<12>	AH14
ADCI_D<13>	AJ14	ADC2_D<13>	AM11
ADCI_DRY	AF17	ADC2_DRY	AF13
ADCI_OVR	AG17	ADC2_OVR	AE13

Table 74: ADC Signal Pinouts

24.11 User I/O Header (J16 on module)

Signal Name	User FPGA (XC2VP30-4FF1152) PIN No
User_IO_1	N29
User_IO_2	N30

Table 75: User I/O Header Pinouts



NALLATECH
The High Performance FPGA Solutions Company

Part XIII: Troubleshooting

This part of the User Guide provides troubleshooting information on the XtremeDSP Development Kit Pro. It includes Kit information in the form of a Frequently Asked Questions list.

Product Registration



Updated troubleshooting information or product updates may be available in the Nallatech XtremeDSP Development Kit Pro support lounge. The XtremeDSP Development Kit Pro is provided with an initial 90-day access to the support lounge upon registration. This lounge provides access to Nallatech software updates and relevant application notes as they become available. Continued access to this lounge, beyond 90 days, is available on establishment of a maintenance agreement with Nallatech. The support lounge is available on the internet at:

www.nallatech.com/solutions/products/kits

In the actions section of this web page you will see an option to register your product. Registration requires the serial number of the XtremeDSP Development Kit Pro. The serial number is located on the bottom of the small blue board case. You can also refer to the Xilinx Answers Database that can be searched by going to:

<http://support.xilinx.com/>

Section 25

FAQ

In this section:

- Frequently Asked Questions



Please note that this FAQ contains a list of questions common at the time of print. The current FAQ can be obtained after registration from within the Nallatech Support lounge during the support period.

25.1 XtremeDSP Development Kit Pro FAQs

How do I Identify the Type of PCI Connection?

PCI defines two types of signalling environment, which operate at either 3.3v or 5v. The BenONE-Kit Motherboard is a universal card and can therefore be used in either signalling environment as shown in [Figure 152 on page 245](#).

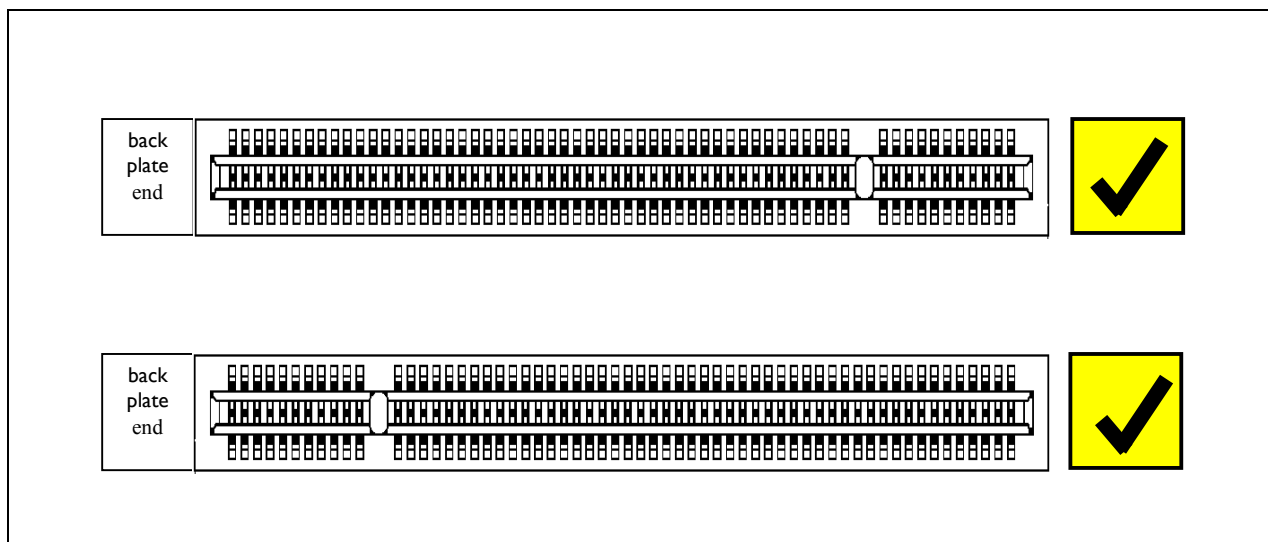


Figure 152: 5V (top) and 3.3V (bottom) signalling PCI Connectors



If you wish to install the BenONE-Kit Motherboard in a PC using a PCI slot please note that in the default configuration provided with the Kit User Guide, the BenONE-Kit Motherboard will only function correctly in a 5V PCI Signalling environment. This is because one of the XC1800 proms is programmed with a 5VIO PCI bitstream and the other PROM that can contain the 3.3VIO PCI bitstream has been used for the USB bitstream. This means that if you wish to use the hardware in a 3.3VIO PCI

slot you will need to update the firmware. Please refer to “Changing the Firmware” on page 225 for details on how this is performed.

What State Should the Module Power Supply LEDs be Showing?

Once the BenONE-Kit Motherboard and BenADDA DIME-II module have been opened, the power LEDs for the supplies used, change from red to green as shown below in Figure 153 on page 246.

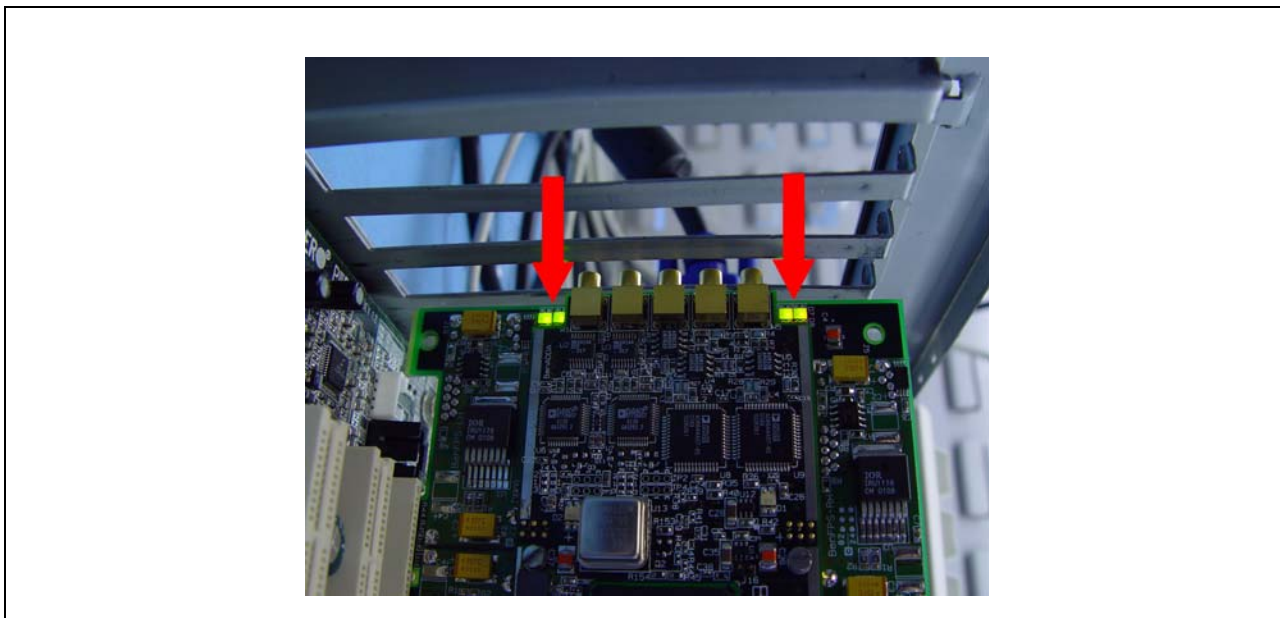


Figure 153: Power LEDs



Note that if a power supply is not used then the power indicator for that supply will remain RED indicating it is not switched on. For example, on the BenADDA-Pro DIME-II module all four supplies are required therefore the four power LEDs will change from green to red.

I Have Problems Opening Multiple Cards over USB, i.e. a StrathNUEY and a BenONE (XtremeDSP Kit). How do I Resolve This?

There is a known issue when opening a StrathNUEY and then trying to open a BenONE (XtremeDSP Kit) card. There is a current work-around: if you open the cards by type there doesn't seem to be a problem i.e. Select **Open Card**, specify USB and for card type select **BenONE** ... then ... Select **Open Card**, specify USB and for card type select **StrathNUEY**. This currently gets round the problem.

Does the FUSE API Support Borland C++?

Yes, the FUSE API supports Borland C++. Libraries are provided in both COFF and OMF formats to allow for compatibility. The OMF format is required by Borland. The libraries can be found in the following locations:

1. In the 'include' folder in the location on the host machine where FUSE was installed, commonly 'C:\Program Files\FUSE\include'

OR

2. The FUSE CD in, <CDROM>:\Software\Include

Where Can I find the FUSE API Header and Library Files?

The files are available in the following locations:

1. In the 'include' folder in the location on the host machine where FUSE was installed, commonly 'C:\Program Files\FUSE\include'

OR

2. The FUSE CD in, <CDROM>:\Software\Include

How many Programmable Clocks are there on the BenONE?

There are two programmable clocks, CLKA and CLKB. Historically these clocks have sometimes been referred to as SYSCLK and DSPCLK.

Is CLKC Programmable in the XtremeDSP Kit?

CLKC is not a programmable oscillator in the Kit. This is connected to a fixed oscillator socket for a Fixed Oscillator on the motherboard. By default there is no crystal fitted as this is socket allows users to fit a specific crystal if needed.

When Configuring the FPGA via the FUSE software, is it loading via JTAG or via a Serial or Select Map Mode?

JTAG is the configuration method. Hence JTAG clock must be selected in bitfile options as the startup clock. If CCLK is selected the programming fails and errors out.

Standard Terms and Conditions

GENERAL

These Terms and Conditions shall apply to all contracts for goods sold or work done by Nallatech Limited. (hereinafter referred to as the "company" or Nallatech) and purchased by any customer (hereinafter referred to as the customer).

Nallatech Limited trading in the style Nallatech (the company), submits all quotations and price lists and accepts all orders subject to the following conditions of contract which apply to all contracts for goods supplied or work done by them or their employees to the exclusion of all other representations, conditions or warranties, express or implied.

The buyer agrees to execute and return any license agreements as may be required by the company in order to authorize the use of those licensable items. If the licensable item is to be resold this condition shall be enforced by the re-seller on the end customer.

Each order received by the company will be deemed to form a separate contract to which these conditions apply and any waiver or any act of non-enforcement or variation of these terms or part thereof shall not bind or prejudice the company in relation to any other contract.

The company reserves the right to re-issue its price list at any time and to refuse to accept orders at a price other than at the price stated on the price list in force at the time of order.

The company reserves the right to vary the specification or withdraw from the offer any of its products without prior warning.

The company reserves the right to refuse to accept any contract that is deemed to be contrary to the companies policies in force at the time.

PRICING

All prices shown on the company's price list, or on quotations offered by them, are based upon the acceptance of these conditions. Any variation of these conditions requested by the buyer could result in changes in the offered pricing or refusal to supply.

All quoted pricing is in Pounds Sterling and is exclusive of Value Added Tax (VAT) and delivery. In addition to the invoiced value the buyer is liable for all import duty as may be applicable in the buyer's location. If there is any documentation required for import formalities, whether or not for the purposes of duty assessment, the buyer shall make this clear at the time of order.

Quotations are made by Nallatech upon the customer's request but there is no obligation for either party until Nallatech accepts the customer's order.

Nallatech reserves the right to increase the price of goods agreed to be sold in proportion to any increase of costs to Nallatech between the date of acceptance of the order and the date of delivery or where the increase is due to any act or default of the customer, including the cancellation or rescheduling by the customer of part of any order.

Nallatech reserves the right (without prejudice to any other remedy) to cancel any uncompleted order or to suspend delivery in the event of any of the customer's commitment with Nallatech not being met.

DELIVERY

All delivery times offered by the company are to be treated as best estimates and no penalty can be accepted for non compliance with them.

Delivery shall be made by the company using a courier service of its choice. The cost of the delivery plus a nominal fee for administration will be added to the invoice issued. Payment of all inward customs duties and fees are the sole responsibility of the buyer. If multiple shipments are requested by the buyer, multiple delivery charges will be made. In the case of multiple deliveries separate invoices will be raised.

If requested at the time of ordering an alternative delivery service can be used, but only if account details are supplied to the company so that the delivery can be invoiced directly to the buyer by the delivery service.

The buyer accepts that any 'to be advised' scheduled orders not completed within twelve months from the date of acceptance of the original order, or orders held up by the buyers lack of action regarding delivery, can be shipped and invoiced by the company and paid in full by the buyer, immediately after completion of that twelve month period.

INSURANCE

All shipments from the company are insured by them. If any goods received by the buyer are in an unsatisfactory condition, the following courses of action shall be taken.

If the outer packaging is visibly damaged, then the goods should not be accepted from the courier, or they should be signed for only after noting that the packaging has sustained damage.

If the goods are found to be damaged after unpacking, the company must be informed immediately.

Under no circumstances should the damaged goods be returned, unless expressly authorized by the company.

If the damage is not reported within 48 hours of receipt, the insurers of the company shall bear no liability.

Any returns made to the company for any reason, at any time shall be packaged in the original packaging, or its direct equivalent and must be adequately insured by the buyer.

Any equipment sent to the company for any purpose, including but not limited to equipment originally supplied by the company must be adequately insured by the buyer while on the premises of the company.

PAYMENT

Nallatech Ltd. terms of payment are 30 days net.

Any charges incurred in making the payment, either currency conversion or otherwise shall be paid by the buyer.

The company reserves the right to charge interest at a rate of 2% above the base rate of the Bank of Scotland PLC on any overdue accounts. The interest will be charged on any outstanding amount from said due date of payment, until payment is made in full, such interest will accrue on a daily basis.

TECHNICAL SUPPORT

The company offers a dedicated technical support via telephone and an E-mail address. It will also accept faxed support queries.

Technical support will be given free of charge for 90 days from the date of invoice, for queries regarding the use of the products in the system configuration for which they were sold. Features not documented in the user manual or a written offer of the company will not be supported. Interfacing with other products other than those that are pre-approved by the company as compatible will not be supported. If the development tools and system hardware is demonstrably working, no support can be given with application level problems.

WARRANTY

The company offers as part of a purchase contract 12 months warranty against parts and defective workmanship of hardware elements of a system. The basis of this warranty is that the fault be discussed with the companies technical support staff before any return is made. If it is agreed that a return for repair is necessary then the faulty item and any other component of the system as requested by those staff shall be returned carriage paid to the company. Insurance terms as discussed in the INSURANCE Section will apply.

Returned goods will not be accepted by the company unless this has been expressly authorized.

After warranty repair, goods will be returned to the buyer carriage paid by the company using their preferred method.

Faults incurred by abuse of the product (as defined by the company) are not covered by the warranty.

Attempted repair or alteration of the goods as supplied by the company, by another party immediately invalidates the warranty offered.

The said warranty is contingent upon the proper use of the goods by the customer and does not cover any part of the goods which has been modified without Nallatech's prior written consent or which has been subjected to unusual physical or electrical stress or on which the original identification marks have been removed or altered. Nor will such warranty apply if repair or parts required as a result of causes other than ordinary authorized use including without limitation accident, air conditioning, humidity control or other environmental conditions.

Under no circumstances will the company be liable for any incidental or consequential damage or expense of any kind, including, but not limited to, personal injuries and loss of profits arising in connection with any contract or with the use, abuse, unsafe use or inability to use the companies goods. The

company's maximum liability shall not exceed and the customers remedy is limited to, either:

- i. repair or replacement of the defective part or product or at the companies option.
- ii. return of the product and refund of the purchase price and such remedy shall be the customer's entire and exclusive remedy.

Warranty of the software written by the company shall be limited to 90 days warranty that the media is free from defects and no warranty express or implied is given that the computer software will be free from error or will meet the specification requirements of the buyer.

The terms of any warranty offered by a third party whose software is supplied by the company will be honoured by the company exactly. No other warranty is offered by the company on these products.

Return of faulty equipment after the warranty period has expired, the company may at its discretion make a quotation for repair of the equipment or declare that the equipment is beyond repair.

PASSING OF RISK AND TITLE

The passing of risk for any supply made by the company shall occur at the time of delivery. The title however shall not pass to the buyer until payment has been received in full by the company. And no other sums whatever shall be due from the customer to Nallatech.

If the customer (who shall in such case act on his own account and not as agent for Nallatech) shall sell the goods prior to making payment in full for them, the beneficial entitlement of Nallatech therein shall attach to the proceeds of such sale or to the claim for such proceeds.

The customer shall store any goods owned by Nallatech in such a way that they are clearly identifiable as Nallatech's property and shall maintain records of them identifying them as Nallatech's property. The customer will allow Nallatech to inspect these records and the goods themselves upon request.

In the event of failure by the customer to pay any part of the price of the goods, in addition to any other remedies available to Nallatech under these terms and conditions or otherwise, Nallatech shall be entitled to repossess the goods. The customer will assist and allow Nallatech to repossess the goods as aforesaid and for this purpose admit or procure the admission of Nallatech or its employees and agents to the premises in which the goods are situated.

INTELLECTUAL PROPERTY

The buyer agrees to preserve the Intellectual Property Rights (IPR) of the company at all times and that no contract for supply of goods involves loss of IPR by the company unless expressly offered as part of the contract by the company.

GOVERNING LAW

This agreement and performance of both parties shall be governed by Scottish law.

Any disputes under any contract entered into by the company shall be settled in a court if the company's choice operating under Scottish law and the buyer agrees to attend any such proceedings. No action can be brought arising out of any contract more than 12 months after the completion of the contract.

INDEMNITY

The buyer shall indemnify the company against all claims made against the company by a third party in respect of the goods supplied by the company.

SEVERABILITY

If any part of these terms and conditions is found to be illegal, void or unenforceable for any reason, then such clause or Section shall be severable from the remaining clauses and Sections of these terms and conditions which shall remain in force.

NOTICES

Any notice to be given hereunder shall be in writing and shall be deemed to have been duly given if sent or delivered to the party concerned at its address specified on the invoice or such other addresses as that party may from time to time notify in writing and shall be deemed to have been served, if sent by post, 48 hours after posting.

SOFTWARE LICENSING AGREEMENT

Nallatech Ltd. software is licensed for use by end users under the following conditions. By installing the software you agree to be bound by the terms of this license. If you do not agree with the terms of this license, do not install the Software and promptly return it to the place where you obtained it.:

1. **License:** Nallatech Ltd. grants you a licence to use the software programs and documentation in this package ("Licensed materials"). If you have a single license, on only one computer at a time or by only one user at a time;

if you have acquired multiple licenses, the Software may be used on either stand alone computers or on computer networks, by a number of simultaneous users equal to or less than the number of licenses that you have acquired; and, if you maintain the confidentiality of the Software and documentation at all times.

2. **Restrictions:** This software contains trade secrets in its human perceivable form and, to protect them, except as permitted by applicable law, you may not reverse engineer, disassemble or otherwise reduce the software to any human perceivable form. You may not modify, translate, rent, lease, loan or create derivative works based upon the software or part thereof without a specific run-time licence from Nallatech Ltd.
3. **Copyright:** The Licensed Materials are Copyrighted. Accordingly, you may either make one copy of the Licensed Materials for backup and/or archival purposes or copy the Licensed Materials to another medium and keep the original Licensed Materials for backup and/or

archival purposes. Additionally, if the package contains multiple versions of the Licensed Materials, then you may only use the Licensed Materials in one version on a single computer. In no event may you use two copies of the Licensed Materials at the same time.

4. **Warranty:** Nallatech Ltd. warrants the media to be free from defects in material and workmanship and that the software will substantially conform to the related documentation for a period of ninety (90) days after the date of your purchase. Nallatech Ltd. does not warrant that the Licensed Materials will be free from error or will meet your specific requirements.
5. **Limitations:** Nallatech Ltd. makes no warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the Licensed Materials.

Neither Nallatech Ltd. nor any applicable Licensor will be liable for any incidental or consequential damages, including but not limited to lost profits.

6. **Export Control:** The Software is subject to the export control laws of the United States and of the United Kingdom. The Software may not be shipped, transferred, or re-exported directly or indirectly into any country prohibited by the United States Export Administration Act 1969 as amended, and the regulations there under, or be used for any purpose prohibited by the Act.

USER GUIDE CONDITIONS

Information in this User Guide is subject to change without notice. Any changes will be included in future versions of this document. Information within this manual may include technical, typing or printing inaccuracies or errors and no liability will arise therefrom.

This User Guide is supplied without warranty or condition, either expressed or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding the information provided herein.

Under no circumstances will Nallatech Limited be liable for any incidental or consequential damage or expense of any kind, including, but not limited to, loss of profits, arising in connection with the use of the information provided herein.

Index

A

abbreviations	xiv
ADCs	19
analog inputs.....	21
analog performance	22
architecture	20
assembly drawing	27
clocking.....	22
coupling specifications.....	26
default configuration settings	26
filter details	24
channel 1	24
channel 2.....	25
firmware	27
timing constraints	28
front end configuration.....	23
hardware description	21
key features	20
maximum clock input to ADC devices	22
operation.....	20
overview.....	19
ADIO signal	115
Adjacent In bus	81
Adjacent Out bus	81
AS/DS# signal	115

B

BenADDA.....	4
BenONE	4
board case layout	16
front	16
left side	16
right side.....	17
board monitoring capabilities	85
overview.....	85
temperature monitoring.....	86
temperature sensor.....	86
using FUSE API	88
using FUSE Probe Tool.....	87
bus structure	81
bus naming.....	81
DIME-II communication busses.....	82
functional diagram	83
overview.....	81
BUSY signal	115

C

Chipscope ILA support.....	161
Chipscope example	164

analyzing the design	171
creating the ICON and ILA cores	164
manually instantiating ICON and ILA cores	169
overview	164
connecting a JTAG download cable.....	163
overview.....	163
clock FPGA	18
clock sources	22
external clock input via MCX connector.....	22
on board 105MHz crystal	22
programmable oscillators	22
clocks	65
ADC and DAC clocks	71
clock configuration	67
clock selection and jitter	76
clock skew	65
clocking the ADCs and DACs	75
DCM wizard.....	73
general notes	74
DIME-II system clocks.....	65
available clocks	65
external clock input via module MCX connection ...	68
firmware.....	72
fixed/external oscillator.....	68
inter-FPGA clock management.....	70
clock feedbacks for de-skewing and clock routing	70
generated clock signals from User FPGA	70
jitter.....	77
names.....	xvi
on board 65MHz oscillator.....	69
overview.....	65
physical clock resources.....	66
programmable oscillators.....	67
set clocks in FUSE API	78
set clocks using FUSE Probe Tool	77
source descriptions	67
using module 65MHz crystal or external MXC clock	75
using the DCM locked signal for reset	74
using the DIME clocks.....	76
using the Virtex-II DCMs	72
ZBT clocks.....	71
ZBT SRAM clocking example.....	72
comms parallel link	81
communications bus protocol.....	115
Config DONE signal	97
contact details	iii

D

DACs	31
architecture	32
clocking.....	38

firmware	39
timing constraints	41
two's complement vs offset binary format	40
hardware	33
analog DAC outputs	35
modes of operation	34
PLL clock multiplier	33
key features	32
output configurations	36
differential outputs using termination resistors	38
single-ended DC coupling using op-amp	37
overview	31
digital I/O	51
firmware	54
hardware	52
adjacent bus header (J8)	52
PLink bus header (J10)	52
user I/O header interfacing	52
introduction	51
DIME-II communication busses	82
DIME-II standard	10
DIME-II system clocks	65
DIMEscript	11
features	11
overview	11
DIMEtalk	11
example network	12
overview	11

E

EMPTY signal	115
environmental specifications	107
operating temperatures	107
relative humidity	107
storage temperatures	107
evaluation software	5

F

FAQ	245
firmware	223
firmware utilities	225
performing a firmware update	226
update the firmware	225
fixed power supply	102
FPGA configuration	121, 123
key steps	124
access through JTAG headers	125
using FUSE API or DIMEscript	124
module and device numbering within FUSE	123
using FUSE Probe Tool	125
using general JTAG chain	126
using the FUSE APIs	125
FPGA power requirements	100

FPGAs on Kit	6
functional diagram	6
FUSE naming conventions	xvi
FUSE overview	10
FUSE interfaces	10
key features	10

H

hardware examples	129
ADC hookup example	131
ADC to DAC example	135
host interface example	138
overview	131
relevant application notes	146
hardware features	4

I

installation	3
installation pack contents	5
introduction	1

J

JTAG chain access	94
configuration status monitoring	97
general JTAG header	95
parallel-IV JTAG header	95
fitting the cable	96
physical order of devices in JTAG chain	94

K

key features	4
Kit features	13

L

LEDs	55
firmware	58
interface LEDs	57
DIME-II module power supply LEDs	57
interface FPGA configuration LED	57
motherboard main power LEDs	58
USB physical interface LED	57
overview	55
physical location	56
set LEDs in FUSE API	60
set LEDs in FUSE Probe Tool	59
user LEDs	58
module user LEDs	58
motherboard user status LEDs	58
Local Bus	82
LVTTL clock oscillator	69

M

MATLAB.....	11
FUSE Toolbox for MATLAB.....	11
MCX connectors.....	21
cable assembly.....	22
design	22
interfacing to connectors	22

N

Nallatech systems and solutions	9
---------------------------------------	---

P

physical layout.....	15
back view	18
front view.....	17
overview.....	15
pinout information	231
ADC signals	242
adjacent header (J8).....	233
adjacent out bus	232
DAC signals	241
DIME-II control and monitoring signals	235
inter-FPGA clock infrastructure signals	240
local bus.....	231
Nallatech RS232 connections.....	234
PLink Header (J10).....	234
User I/O header	242
ZBT SRAM.....	236
power specifications	99
ADC heatsinks.....	105
cooling considerations	104
main FPGA.....	104
external connectors	106
additional power connector (J20).....	106
standalone power connector	106
external standalone power supply	99
Fixed Power Supply	101, 102
overview.....	99
PCI power specification.....	100
power supply LEDs.....	103
DIME-II module power supply LEDs.....	103
motherboard main power LEDs	103
Programmable Power Supply	101
recommended power max for Kit.....	100
recommended specification.....	100
using the Kit in the board case	104
using the Kit outside the board case.....	104
product registration.....	243
programmable clocks	247
programmable power supply	102

R

reading from Interface FPGA to User FPGA.....	115
reference information	229
related documentation.....	xiv
REN#/WEN# signal	115
resets.....	61
firmware.....	62
overview.....	61
reset control in FUSE API.....	64
signal names.....	62
structure	61
toggle using FUSE probe tool.....	63
revision	ii

S

scope of manual	xvi
Spartan-II device	6
system level design.....	109
building a host interface.....	111
design partitioning	111
interface communications bus	113
designing with the FUSE API	118
interface performance	119
interface FPGA to User FPGA interface core	114
further information on core.....	118
implementing the comms mechanism	114

T

temperature monitoring	
using FUSE Probe Tool.....	87
using the FUSE API	88
DIME_ModuleControl.....	88
DIME_ModuleStatus	90
DIME_PPSSStatus.....	92
temperature sensor	86
ALERT1 signal	86
datasheet.....	86
pinouts.....	86
timing information.....	117
troubleshooting	243
typographical considerations	xv

U

user guide format.....	xiii
user guide symbols.....	xiii

V

Virtex-II device.....	6
voltage controlled oscillator	34

W

writing to Interface FPGA from User FPGA 116

X

Xilinx EDK support	179
design notes.....	181
resets	181
using the ZBT as a peripheral	181
XMD support	182
EDK example	183
connect via XMD	185
load the FPGA design with FUSE	184
start the software debugger	186
overview.....	181
Xilinx Impact Support	153
overview.....	155
set up a connection	155
connecting JTAG download cable.....	155
launch impact.....	157
open card to enable power supplies	155
Xilinx ISE support.....	147
common ISE settings	149
synthesis and implementation settings	149
Xilinx System Generator support.....	189
Chipscope use in System Generator	208
Chipscope Pro overview.....	208
importing data into MATLAB workspace from Chip-	
scope	220
overview	208
real time debug.....	215
overview.....	191
System Generator and UltraController reference de-	
sign	220
using the co-simulation flow	
overview	200
using the cosimulation flow	200
using the HDL Netlist Flow.....	192
overview	192
simulating the example	192
XtremeDSP Development Kit-III overview	3

Z

ZBT SRAM memory	43
bus utilization	43
clocking.....	49, 71
cycle times	43
firmware.....	49
hardware	44
clock and control signals for ZBT bank.....	47
ZBT address signals.....	47, 238
ZBT data signals	47, 239
memory size.....	43

overview.....	43
ZBT SRAM to User FPGA interconnect.....	44

If you want Nallatech to reply to your comments, please include your name, address and telephone number.

