United Electronic Industries

The High-Performance Alternative

DNx-PL-820

—

User Manual

Complex Programmable Logic Device Extension Board in the PowerDNA Cube or RACK series chassis

December 2017

PN Man-DNx-PL-820

CE

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please contact our support team:

Support:

Telephone: (508) 921-4600
Fax: (508) 668-2350

Also see the FAQs and online “Live Help” feature on our web site.

Internet Support:

Support: support@ueidaq.com
Web-Site: www.ueidaq.com
FTP Site: ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Specifications in this document are subject to change without notice. Check with UEI for current status.

Table of Contents

List of Figures

Chapter 1 Introduction

This document outlines the feature set and use of the DNx-PL-820 Complex Programmable Logic Device (CPLD) extension boards.

Chapter 1 contains the following sections:

- Organization of this Manual (Section 1.1)
- DNx-PL-820 Board Overview (Section 1.2)
- Features (Section 1.3)
- Indicators (Section 1.4)
- Specification (Section 1.5)
- Device Architecture (Section 1.6)
- Connectors and Wiring (pinouts) (Section 1.7)

1.1 Organization of this Manual

This DNx-PL-820 User Manual is organized as follows:

- **Introduction**
  Chapter 1 provides an overview of the DNx-PL-820 CPLD extension board features, device architecture, connectivity and logic.
- **Programming with the High-Level API**
  The PL-820 is not currently supported in the high-level API. (Chapter 2 customarily provides an overview of how to create a session, configure the session, and interpret results with the high-level framework API).
- **Programming with the Low-Level API**
  Chapter 3 describes low-level API commands called from a host application for configuring and using the DNx-PL-820 series boards.
- **Restoring & Programming a CPLD MAX10 Image File**
  Chapter 4 provides instructions for how to open, compile and run simulations in Quartus using UEI's initial image file and how to download the image file (Verilog design) to the CPLD hardware.
- **Testing the DNx-PL-820**
  Chapter 5 provides descriptions of the test cases covered in UEI sample code and provides timing diagrams / screenshots of test waveforms.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for DNx-PL-820 board(s).
- **Appendix B - Register Descriptions**
  This appendix provides tables of DNx-PL-820 registers and register bit descriptions.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

NOTE: A glossary of terms used with the PowerDNA Cube/RACK and I/O boards can be viewed or downloaded from www.ueidaq.com.

Manual Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.

CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”

Bold typeface will also represent field or button names, as in “Click **Scan Network**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbatim into the source code, initialization, or other file.

Examples of Manual Conventions

Before plugging any I/O connector into the Cube or RACKtangle, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

Usage of Terms

Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNR RACKtangle™ rack mounted system, whichever is applicable. The term DNR is a specific reference to the RACKtangle, DNA to the PowerDNA I/O Cube, and DNx to refer to both.

1.2 DNx-PL-820 Board Overview

The DNx-PL-820 is a user programmable field-programmable gate array (FPGA) board that allows an FPGA programmer to add customer FPGA functionality to the DNx family and the various platforms it supports.

The DNA-PL-820 and DNR-PL-820 boards are compatible with the UEI Cube and RACKtangle chassis respectively. All board versions are functionally identical but differ in the mounting hardware. The DNA version is designed to stack in a Cube chassis. The DNR versions are designed to plug into the backplane of a RACK chassis.

1.2.1 Board Configuration

The DNx-PL-820 is a two board product, consisting of a base-board and a daughter-board. The bottom board (left board as positioned in a DNR RACK) is the base-board, which includes a Cyclone II FPGA while the upper board (right in the RACK) is the daughter board, which provides the user-programmable Altera MAX 10 FPGA. The Cyclone board is used exclusively as the interface between the DNx chassis bus and the MAX 10 FPGA chip.

The Cyclone chip communicates with the MAX 10 using two, fully programmable serial peripheral interfaces (SPIs). A fixed synchronous serial interface (SSI) is also included but is currently reserved for system level configuration purposes.

1.2.2 Initial MAX 10 Image

The MAX 10 chip is initialized at the factory with an FPGA image that configures registers, the two serial peripheral interfaces, and the synchronous serial interface, as well as clocks and other access hardware. Our initial image sets each I/O pin as a general purpose digital input / output (DIO) pin. With the UEI API, users can set each I/O bit independently as an input or output.

NOTE: UEI provides the FPGA code of our initial MAX 10 image. To use our established API, users must wrap their FPGA design around our initial image and keep the same register map and SSI configuration.

1.2.3 Downloading Custom FPGA Images to the MAX 10 Target

The Altera Quartus II 64-bit Programmer, release 15.0, includes support for programming MAX 10 series devices. The MAX 10 chip on the CPLD daughter board can be programmed via the JTAG interface accessible on the DB62 connector using this version of the Quartus programmer.

A set of jumpers on the board allows the JTAG interface to be disconnected from the I/O connector when security concerns require it. Security is further provided by the MAX 10's implementation of a nonvolatile security key. Without the key the MAX 10's FPGA image cannot be read or written.

1.2.4 Software Support

UEI provides a high level, easy to use API that allows you to read and write registers established in our initial image. API allows the configuration and use of the two SPI ports connecting the Cyclone and MAX 10 chips. The API also allows you to program the MAX 10 chip from your host PC through the Cube/ RACK's Ethernet port and allows you to control/monitor all the GPIO included on the Cyclone II board (and the MAX 10 in its default state). UEI supplies example code for PL-820 programming in C.

The DNx-PL-820 series includes software drivers supporting all popular operating systems including Windows, Linux, QNX, VXWorks, RTX, and most other popular Real-Time Operating Systems.

1.3 Features

The PL-820 boards provide the following features:

- Interface for the user-programmable Altera MAX 10 FPGA
- 105 TTL-level general purpose I/O pins, accessible via two DB-62 connectors

Features specific to the MAX 10 FPGA include the following:

- MAX 10 FPGA type, 10M50DCF484 I7G
- Two programmable clocks, 1 kHz and 33 MHz, provided by the Cyclone II base-board
- Up to 50,000 FPGA cells available
- Up to 1683 kByte available block memory
- 144 (18x18) embedded multipliers
- 4 PLLs available
- Up to 736 kByte user flash memory
- Instant-on capability on the MAX 10, allowing MAX 10 applications to run immediately, independent to the Cube or RACK boot-up state
- User and dual configuration FLASH, DSP blocks, and Nios II embedded processor functionality

1.4 Indicators

The DNx-PL-820 indicators are described in **Table 1-1** and illustrated in **Figure 1-1**.

Table 1-1 PL-820 Indicators

LED Name	Description
RDY	Indicates board is powered up and operational
STS	Indicates which mode the board is running in: • **OFF**: Configuration mode • **ON**: Operation mode

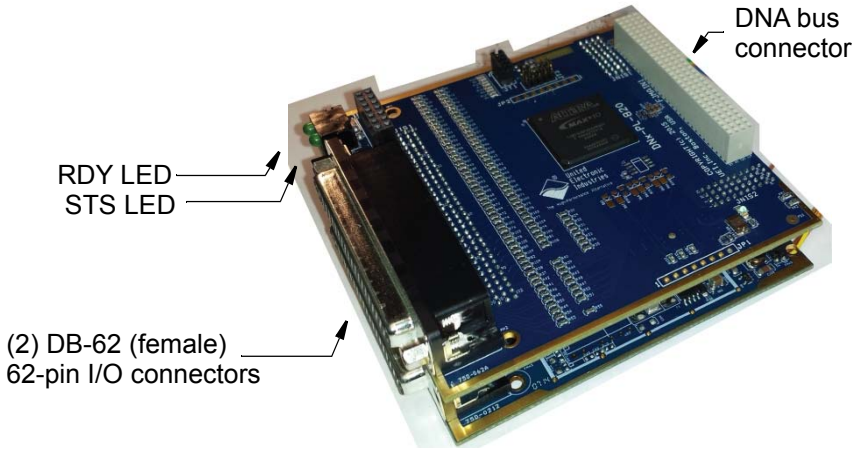


Figure 1-1 Photo of DNA-PL-820 Board

1.5 Specification

The technical specification for the DNx-PL-820 is provided in **Table 1-2**:

Table 1-2 . DNx-PL-820 Technical Specification

Formats	DNA-series for "Cube" chassis DNR-series for RACKtangle chassis
FPGA type	MAX 10, 10M50DCF484 series
Clock Frequency	two programmabe clocks,up to 33 MHz, provided by Cyclone II chip
FPGA Cells available	up to 50,000
Available block memory	up to 1,638 kiloByte
Embedded multipliers	144 (18 x 18)
PLLs available	4
User Flash memory	up to 736 kByte
I/O connectors	62-pin "D" (female DB62)
I/O configuration	Cube: Requires two consecutive slots RACK: Requires mounting in slot 12
Power consumption	5.0 W not including power sourced to external circuitry
Operating temp. (tested)	-40°C to +85°C
Operating humidity	95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500Hz, broadband random
Shock *IEC 60068-2-27*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	500,000 hours

1.6 Device Architecture

A block diagram of the architecture for the PL-820 is shown in **Figure 1-2**.

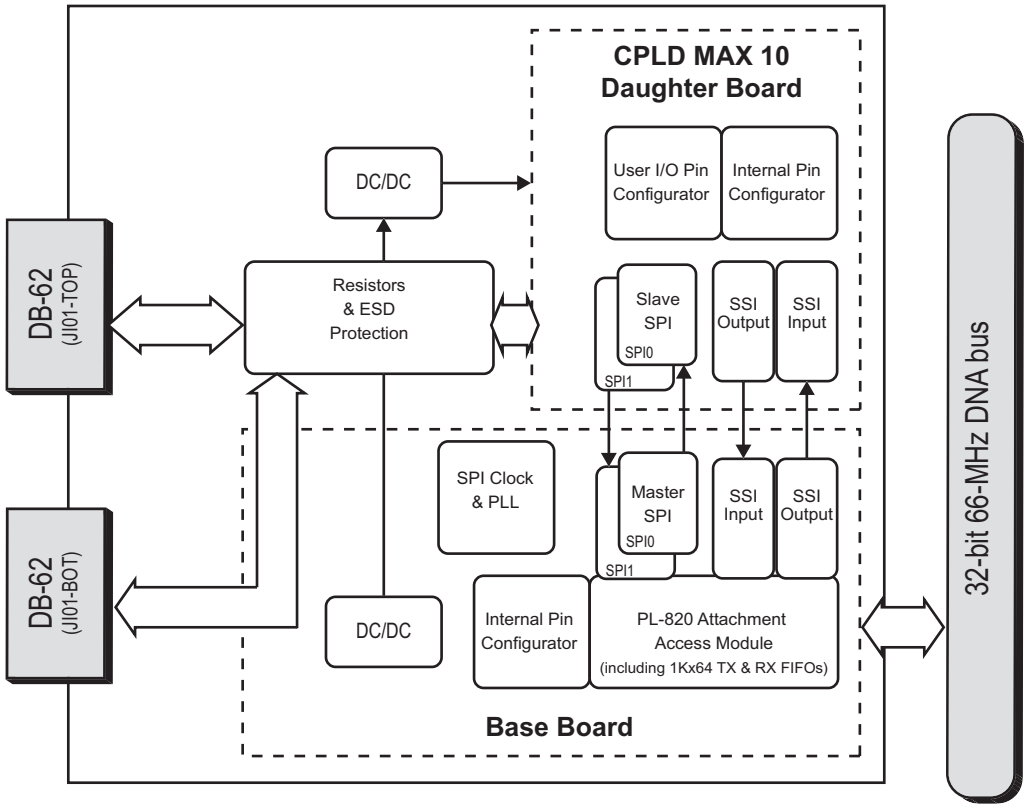


Figure 1-2 Block Diagram of DNx-PL-820 Board

The PL-820 board is a combination of a user-programmable Altera MAX 10 10M50DCF484I7G attachment (daughter) board for the Cyclone II FPGA base board.

The FPGA base board communicates with the host PC over the PowerDNA 32-bit chassis bus. The synchronous serial interface (SSI) is used for communication between the base board and the MAX 10 attachment board.

Two programmable serial periphery interfaces (SPIs) provide support for multi-word transfers. A fixed frequency (24 MHz) PLL can be used to generate the SPI clock.

Additionally, the base board provides two programmable clock signals, up to 33 MHz, to the MAX 10.

1.6.1 MAX 10 attachment board

The MAX10 CPLD uses up to 64 I/O pins to communicate with FPGA on the base layer and offers up to 105 user-accessible TTL I/O pins located on two DB-62 connectors (one DB-62 on the base board and another on the attachment). As a result PL-820 board occupies two adjacent slots in a UEI chassis.

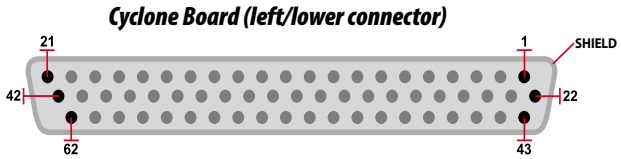
Unlike most DNx series boards, the DNx-PL-820 is not isolated from the DNx chassis. However, all I/O pins are protected with 165 Ω series resistors and low capacitance bidirectional protection diodes.

A dedicated triple channel DC/DC converter provides power to the CPLD daughter-board.

1.7 Connectors and Wiring (pinouts)

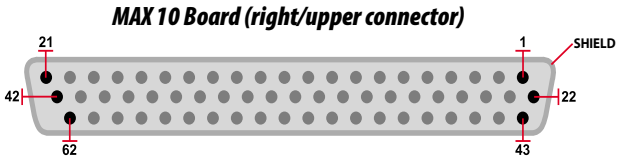
The PL-820 boards use two 62-pin female D-Sub connectors with the pinouts shown in **Figure 1-3** and **Figure 1-4**.

Signals are located across 105 pins of the two DB--62 connectors. A user can connect to each 62-pin female D-Sub connector either through a custom made cable or by connecting to a DNA/DNR-STP-62 accessory panel.



Pin	Signal	Pin	Signal	Pin	Signal
1	+3.3 V	22	GND	43	GND
2	DIO 52	23	DIO 51	44	DIO 53
3	DIO 55	24	DIO 54	45	DIO 56
4	DIO 58	25	DIO 57	46	DIO 59
5	DIO 61	26	DIO 60	47	DIO 62
6	DIO 64	27	DIO 63	48	DIO 65
7	DIO 67	28	DIO 66	49	DIO 68
8	DIO 69	29	GND	50	DIO 70
9	DIO 72	30	DIO 71	51	DIO 73
10	DIO 75	31	DIO 74	52	DIO 76
11	DIO 78	32	DIO 77	53	DIO 79
12	DIO 81	33	DIO 80	54	DIO 82
13	DIO 84	34	DIO 83	55	DIO 85
14	GND	35	DIO 86	56	DIO 87
15	DIO 89	36	DIO 88	57	DIO 90
16	DIO 92	37	DIO 91	58	DIO 93
17	DIO 95	38	DIO 94	59	DIO 96
18	DIO 98	39	DIO 97	60	DIO 99
19	DIO 101	40	DIO 100	61	DIO 102
20	DIO 104	41	DIO 103	62	GND
21	GND	42	rsvd		

Figure 1-3 Pinout of the DNA-PL-820 Base Board (Cyclone Board)



Pin	Signal	Pin	Signal	Pin	Signal
1	+3.3 V	22	GND	43	GND
2	DIO 49	23	DIO 50	44	DIO 48
3	DIO 46	24	DIO 47	45	DIO 45
4	DIO 43	25	DIO 44	46	DIO 42
5	DIO 40	26	DIO 41	47	DIO 39
6	DIO 37	27	DIO 38	48	DIO 36
7	DIO 34	28	DIO 35	49	DIO 33
8	DIO 32	29	GND	50	DIO 31
9	DIO 29	30	DIO 30	51	DIO 28
10	DIO 26	31	DIO 27	52	DIO 25
11	DIO 23	32	DIO 24	53	DIO 22
12	DIO 20	33	DIO 21	54	DIO 19
13	DIO 17	34	DIO 18	55	DIO 16
14	GND	35	DIO 15	56	DIO 14
15	DIO 12	36	DIO 13	57	DIO 11
16	DIO 9	37	DIO 10	58	DIO 8
17	DIO 6	38	DIO 7	59	DIO 5
18	DIO 3	39	DIO 4	60	DIO 2
19	DIO 0	40	DIO 1	61	JTAG TCK
20	JTAG TDO	41	JTAG TMS	62	GND
21	GND	42	JTAG TDI		

Figure 1-4 Pinout of the DNA-PL-820 MAX 10 Board

Chapter 2 Programming with the High-Level API

This chapter typically provides information about using the UeiDaq Framework high-level API; however, the DNx-PL-820 is not currently supported in the high-level API.

Please refer to Chapter 3, “Programming with the Low-Level API” for low-level programming information.

Chapter 3 Programming with the Low-Level API

This chapter provides the following information about programming the DNx-PL-820 using the low-level API:

- About the Low-level API (Section 3.1)
- Overview of Registers and Interfaces (Section 3.2)
- Low-level Functions (Section 3.3)
- Low-level Programming Techniques (Section 3.4)
  - Configuring the PL-820 (Section 3.4.1)
  - Reading Registers (Section 3.4.2)
  - Writing Registers (Section 3.4.3)
  - Writing the SPI FIFO (Section 3.4.4)
  - Reading the SPI FIFO (Section 3.4.5)
  - Constants Definitions (Section 3.4.6)

NOTE: This chapter provides information about application API available for accessing PL-820 registers and interfaces, not information about downloading an FPGA image. For information about programming the MAX 10 / CPLD hardware with a custom FPGA image, refer to **Chapter 4**.

3.1 About the Low-level API

The low-level API provides direct access to the DAQBIOS protocol structure and registers in C.

For detailed information regarding low-level programming, refer to the “PowerDNA API Reference Manual” located in:

- On Linux systems:
  <PowerDNA-x.y.z>/doc
- On Windows systems:
  Start » All Programs » UEI » PowerDNA » Documentation

3.2 Overview of Registers and Interfaces

The PL-820 API provides access to the configuration interface between the base-board FPGA and the Max 10 CPLD, as well as access to all registers on both the base-board and the CPLD sides.

3.2.1 Register Access

Registers on the base-board side are accessed immediately.

Registers on the CPLD side are accessed via the synchronous serial interface (SSI). The base-board serially transfers data over the SSI, sending 42-bit data (two start bits, a read/write bit, a 7-bit address, and 32-bit data) to the CPLD. This interface is hard-coded and not configurable. The interface works at 33 MHz, which results in data words taking 1.3 µs to transmit. Upon a read command, 32-bit data is sent back to the base-board FPGA from the CPLD (plus 2 start bits). It takes about 3 µs for a CPLD access to complete.

Network command processing for register and FIFO access is done using DaqBIOS functions (see Table 3-1). The average time for a register read or write function calls to complete is 70 µs, regardless of whether a CPLD or base-board FPGA register is accessed.This includes the time it takes to form and send the packet and receive a reply from the IOM side.

A list of registers and register bit descriptions is provided in Appendix B.

3.2.2 Serial Peripheral Interface Access

The base-board FPGA side has two master serial peripheral interfaces (SPIs). The software application communicates with the master SPI via FIFOs.

There are two FIFOs: one to write to the CPLD side and one to read from the CPLD side. When an SPI word is written into the base-board FPGA FIFO it is clocked out to the CPLD while the input word is clocked in at the same time. In multi-word mode, the transmission of words is terminated upon last the word transferred out of the FIFO and a new transmission starts with the address/data word.

CPLD side has two slave SPIs. There are two ways software can communicate with the slave SPI. In single-word mode, data from base-board FPGA appears in `SPIx_IDTx` registers on the CPLD is taken from `SPIx_ODTx` registers. Every received word overwrites the content of `SPIx_IDTx` registers, so the CPLD must store it upon receipt. To change output word, `SPIx_ODTx` needs to be written.

In multi-word mode the CPLD can use a register table, or pool of registers. The start address (the address of the top of the `SPIx_REGx` table) should be programmed into `SPIx_IADDR` register (i.e. read and write start address and enable bits). Then, when an incoming SPI word has an address equal to the programmed (as above), it is stored into this table and the index in the table is incremented. The next word is stored into the next row until up to 8 words are received.

(More information regarding single-word and multi-word testing and waveforms is provided in **Chapter 5**, and register descriptions and addresses are provided in Appendix B).

3.2.3 Internal and External Ports

There are two sets of ports.

One set is called NIS (non-isolated) and connects the lower base-board FPGA board with top CPLD board. There are 64 NIS pins and these are for internal connections between the FPGA board and CPLD boards; some of them are taken by SSI, clocks and the two SPI interfaces. These pins should not be configured and used. (Refer to the `PL820_NISx` register descriptions.)

The second set is for external user ports. There are 105 pins connected to DB-62 connectors and divided into four 32-bit blocks. These pins are controlled by the CPLD and can be used as required.

3.2.4 Initialization Mode

Upon boot-up, firmware enters an initialization mode. It is possible to set up an initial state for both NIS and user pins (tri-stated as input or set as output low or high) and store these settings into the EEPROM. Also, the EEPROM can contain information on how to set up one or both SPI interfaces and whether firmware needs to transmit defined SPI words over these interfaces. The default state of all pins and SPI interfaces is “unconfigured” or tri-stated.

When firmware is switched into shutdown mode, either by a software command or upon a watchdog timer, there is a second set of parameters in the EEPROM which can be used to set I/O into pre-defined “safe” state. If parameters are not set, firmware preserves the latest state of the pins until the chassis is reset.

Settings for initialization and shutdown modes are optional and factory programmed as “disabled” when the product is shipped.

3.3 Low-level Functions

Low-level functions are described in detail in the PowerDNA API Reference Manual. Table 3-1 provides a summary of PL-820-specific functions.

Table 3-1 Summary of Low-level API Functions for DNx-PL-820

Function	Description
`DqAdv820SetCfg`	Sets up NIS/IS pin direction, as well as SPI and PLL parameters
`DqAdv820ReadReg`	Reads a PL-820 register
`DqAdv820WriteReg`	Writes a PL-820 register
`DqAdv820ReadSPI`	Reads from the SPI0 or SPI1 FIFO
`DqAdv820WriteSPI`	Writes to the SPI0 or SPI1 FIFO

3.4 Low-level Programming Techniques

Application developers are encouraged to explore existing source code examples when first programming the PL-820. Sample code provided with the installation is self-documented and serves as a good starting point.

Code examples are located in the following directories:

- For Linux: <PowerDNA-x.y.z>/sdk/DAQLib_Samples
- For Windows: *Start » All Programs » UEI » PowerDNA » Examples*

Sample820.c provides example code for the PL-820, which incorporates several test scenarios. Refer to **Chapter 5** for a detailed description of the test scenarios and to view example waveforms. The rest of this chapter describes the API and constants used by host applications to access PL-820 registers and the interface.

3.4.1 Configuring the PL-820

The PL-820 uses the low-level API, `DqAdv820SetCfg(),` to configure the PL-820 board:

```
DqAdv820SetCfg(
      int hd,           // Handle to IOM received from DqOpenIOM()
      int devn,         // Board device # inside the IOM chassis
      uint32 cfg_mask,  // <reserved>
      pPL820CFG pCfg,   // Pointer to the configuration structure,
                        //     (see Table 3-2)
      uint32* status,   // Content of PL820_STS status register,
                        //     (see Appendix B)
);
```

Configuration is passed as the following structure, and parameters are described in Table 3-2:

```
typedef struct {
  uint32 prm_flags;

  // Control external pins (DQ_L820_PORTS: 4 32-bit ports of user I/O)
  uint32 port_dir[DQ_L820_PORTS];
  uint32 port_out[DQ_L820_PORTS];

  // PLL section
  //     DQ_L820_NIS_PORTS: 2 32-bit ports connect FPGA board to CPLD board
  uint32 pll_freq[DQ_L820_NIS_PORTS];
  uint32 pll_div[DQ_L820_NIS_PORTS];

  // SPI configuration
  uint32 spi_cfg_fpga[DQ_L820_NIS_PORTS];
  uint32 spi_wm[DQ_L820_NIS_PORTS];
  uint32 spi_acfg[DQ_L820_NIS_PORTS];
  uint32 spi_div[DQ_L820_NIS_PORTS];
  uint32 spi_cfg_cpld[DQ_L820_NIS_PORTS];

  // NIS connects FPGA and CPLD.
  // There are 64 pins total with the exception
  //    of pins 2,4,8,12,16 and 39, which are used for internal SSI & clocks
  // When the direction is set to input on FPGA a command is sent mirror
  //   FPGA configuration
  //  <nis_dir> and <nis_out> are from FPGA side.
  uint32 nis_dir[DQ_L820_NIS_PORTS];
  uint32 nis_out[DQ_L820_NIS_PORTS];
  uint32 nis_out_cpld[DQ_L820_NIS_PORTS];

 } PL820CFG, *pPL820CFG;
```

Table 3-2 PL820CFG Parameters

PL820CFG Parameter	Description
prm_flags	Flags to validate the following parameters (supported flags are listed on the next page)
port_dir[DQ_L820_PORTS]	Direction of the user pins on DB62 connectors (input or output)
port_out[DQ_L820_PORTS]	Output values
pll_freq[DQ_L820_NIS_PORTS]	Requested PLL frequency in Hz, or M/N/C/Post
pll_div[DQ_L820_NIS_PORTS]	Post-divider
spi_cfg_fpga[DQ_L820_NIS_PORTS]	SPIx configuration (SPI configuration parameters are listed in Section 3.4.4 Writing the SPI FIFO)
spi_wm[DQ_L820_NIS_PORTS]	SPIx watermark register
spi_acfg[DQ_L820_NIS_PORTS]	SPIx address configuration register

Table 3-2 PL820CFG Parameters

PL820CFG Parameter	Description
spi_div[DQ_L820_NIS_PORTS]	SPIx clock divider from PLL0 (24 MHz default) (SPI divider is described in Section 3.4.4 Writing the SPI FIFO)
spi_cfg_cpld[DQ_L820_NIS_PORTS]	SPIx configuration (SPI configuration parameters are listed in Section 3.4.4 Writing the SPI FIFO)
nis_dir[DQ_L820_NIS_PORTS]	direction of NIS pins (input or output)
nis_out[DQ_L820_NIS_PORTS]	output values for FPGA
nis_out_cpld[DQ_L820_NIS_PORTS]	output values for CPLD

<prm_flags> defines which parameters are valid, i.e. which subsystems need to be configured. It is possible to call `DqAdv820SetCfg()` for each subsystem individually or for all at the same time.

The following flags are supported:

```
#define PL608_PRMFPGA_SPI1 (1L<<8) // configure SPI1
#define PL608_PRMFPGA_SPI0 (1L<<7) // configure SPI0
#define PL608_PRMCPLD_SPI1 (1L<<6) // configure SPI1
#define PL608_PRMCPLD_SPI0 (1L<<5) // configure SPI0
#define PL608_PRMFPGA_PLL1 (1L<<4) // configure PLL1
#define PL608_PRMFPGA_PLL0 (1L<<3) // configure PLL0(CycloneIII required)
#define PL608_PRMCPLD_PORT (1L<<2) // configure port side
#define PL608_PRMCPLD_NIS (1L<<1)  // configure NIS side
#define PL608_PRMFPGA_NIS (1L<<0)  // configure NIS side
```

Note:

1. For the NIS interface firmware takes care of preventing driving the same pin from both CPLD and FPGA side in case of user error
2. NIS interface has some bits reserved for CLI SPI to access registers from FPGA on CPLD side and clock lines:

```
// bits 2, 4, 8, 12, 16, 39 should be always written with 0s
    #define PL820_NIS0_MASK (~0x00011114L)
    #define PL820_NIS1_MASK (~0x00000080L)
```

<spi_cfg_fpga> Refer to the following example of setting the FPGA config parameter:

```
cfg.spi_cfg_fpga[spi] =
                    PL820_SPI_CFG_CSI |
                    PL820_SPI_CFG_ABE(ADDR_START+ADDR_SIZE-1) |
                    PL820_SPI_CFG_PSR |
                    PL820_SPI_CFG_ABS(ADDR_START) |
                    PL820_SPI_CFG_PSW |
                    PL820_SPI_CFG_DBE(DATA_START+DATA_SIZE-1) |
                    PL820_SPI_CFG_MWE |
                    PL820_SPI_CFG_DBS(DATA_START) |
                    0;
```

Table 3-3 FPGA Configuration Bits and Macros In DqAdv820SetCfg()

Configuration	Description
PL820_SPI_CFG_CSI	CS is inverted (i.e. negative)
PL820_SPI_CFG_ABE(ADDRM_START+ADDRM_SIZE-1)	Last bit of address
PL820_SPI_CFG_PSR	Receive data is valid on the rising edge
PL820_SPI_CFG_ABS(ADDRM_START)	Address starts at this bit
PL820_SPI_CFG_PSW	Transmit data is valid on the rising edge
PL820_SPI_CFG_DBE(DATAM_START+DATAM_SIZE-1)	Data ends at this bit
PL820_SPI_CFG_MWE	Multi-word w/o address in every word is enabled
PL820_SPI_CFG_DBS(DATAM_START)	Data starts at bit zero

Note that for proper operations both FPGA and CPLD have to receive the same configuration for the exception of PL820_SPI_CFG_PSR bit. PSR bit defines the clock edge when data becomes valid and needs to be set into opposite states.

PL820_SPI_CFG_DBS, _DBE, _ABS, _ABE constants define bit numbers where data word starts and ends, and where address part starts and ends. For example, for a 32-bit word with 8-bit address and 24 bit data these constants have to be set to 0, 23, 24, 31 respectively.

<spi_div> needs to be programmed with a value of the 24 MHz base clock divider minus one. This field defines frequency on the clock line of SPIx interface. Each SPI interface may have its own clock rate.

Note:

1. In multi-word mode the minimum number of uint32 to write is two for 8-32-bit word and four for 33-64 bit words. If the amount of data is less than that no words will be written.
2. Also, multi-word transaction has to be written in a single call to DqAdv820WriteSPI().

3.4.2 Reading Registers

The PL-820 uses the low-level API, `DqAdv820ReadReg()`, to read data and control registers from either the CPLD or FPGA base-board.

FPGA address range is 0x0 to 0x2ffc and CPLD address range is from 0x3000 to 0x31fc.

Refer to Appendix B for a list of available registers and bit descriptions.

```
DqAdv820ReadReg(
      int hd,           // Handle to IOM received from DqOpenIOM()
      int devn,         // Board device # inside the IOM chassis
      uint32 reg,       // Register offset
      uint32* value     // Returned value of the register
);
```

Note:

1. Registers are also described in the PL-820 Logic Interface document.

3.4.3 Writing Registers

The `DqAdv820WriteReg()` API writes a register on either the CPLD or FPGA base-board.

FPGA address range is 0x0 to 0x2FFC and CPLD address range is from 0x3000 to 0x31FC.

Refer to Appendix B for bit descriptions for each control register.

```
DqAdv820WriteReg(
      int hd,           // Handle to IOM received from DqOpenIOM()
      int devn,         // Board device # inside the IOM chassis
      uint32 reg,       // Register offset
      uint32 value      // Value to write to the register
);
```

Note:

1. Registers are described in the logic document and in powerdna.h file starting with `DQ_L820_LINES` define (see also Defined Constants section of this document)
2. The write register cannot be used to transfer multi-word (i.e. when the address comes in the first SPI word only) sequences because the logic resets transmission every time output FIFO becomes empty. Use the `DqAdv820WriteSPI()` API instead.

3.4.4 Writing the SPI FIFO

The `DqAdv820WriteSPI()` API writes words into the SPI FIFO (to be transmitted from the FPGA base-board to CPLD side).

If the word size is from 8 to 32 bits, then one write takes one uint32 to complete.

If the word size is from 33 to 64 bits, it takes two uint32s to complete one write.

In the later mode the write will not proceed if you write an odd number of words and either return zero <`written`> or ignore the extra word. <`written`> returns the actual number of uint32s written to the SPI FIFO.

```
DqAdv820WriteSPI(
      int hd,           // Handle to IOM received from DqOpenIOM()
      int devn,         // Board device # inside the IOM chassis
      uint32 spi,       // SPI interface 0 or 1
      uint32 size32,    // Number of uint32 words to write
      uint32* data,     // Array of uint32s the size of <size32>
      uint32* written,  // Actual number of words written
      uint32* available // Remained capacity of FIFO
                        //  after words were written
);
```

Note that the SPI must be configured before use. This is done by setting up the SPI configuration (e.g., `cfg.spi_cfg_fpga`) with the `DqAdv820SetCfg()` API.

3.4.5 Reading the SPI FIFO

The `DqAdv820ReadSPI()` API reads words from the SPI FIFO (to be transmitted from CPLD to FPGA side).

If the word size is from 8 to 32 bit then each word is represented in one uint32.

If the word size is from 33 to 64 it takes two uint32s to store the word.

.

```
DqAdv820ReadSPI(
      int hd,           // Handle to IOM received from DqOpenIOM()
      int devn,         // Board device # inside the IOM chassis
      uint32 spi,       // SPI interface 0 or 1
      uint32 size32,    // Number of uint32 words to write
      uint32* data,     // Array of uint32s the size of <size32>
      uint32* read,     // Actual number of words read
      uint32* avail     // Number of words available in FIFO
);
```

Note:

Only the data part of the word is stored, not the address one. Address bits are replaced with zeroes.

3.4.6 Constants Definitions

The PL-820 API defines the following constants in powerdna.h:

```
#pragma pack()

#define DQ_PL820_LINES        (104)            // number of I/O lines
#define DQ_PL820_CHAN         (2)              // working channels
#define DQ_PL820_CHANSVC      (2)              // total channels including service
#define DQ_PL820_INFOSZ       DQ_MAX_INFO_SIZE // maximum size of information
                                               //   structure
#define DQ_PL820_BASE         BUS_FREQUENCY    // 66MHz base frequency
#define DQ_PL820_PLL_DEF      24000000         // default PLL frequency

#define DQ_PL820_PORTS        (4)              // four 32-bit ports in
                                              // total available to the user outside
#define DQ_PL820_NIS_PORTS    (2)              // two 32-bit ports connect
                                               //    FPGA and CPLD layers

#define PL820_MAX_ISR_WAIT    100              // microseconds/2 to wait for IS reply

#define PL820_SPI_TX_FIFO_SZ  1024            // SPI0/1 TX FIFO size
#define PL820_SPI_RX_FIFO_SZ  1024            // SPI0/1 RX FIFO size

//
// ADC is optional and provide ability to read all voltages and temperature
//
// Bits 18-0 are encoded as listed below as delivered from LTC2486 converter
//    (sub-LSBs not included)
#define PL820IS_ADC_EOC          (1L<<18)      // "EOC" bit 0 indicates valid data
                                               // 1 - conversion result read too son
(hardware error)
#define PL820IS_ADC_DMY          (1L<<17)      // "DUMMY" bit - should be always 0
#define PL820IS_ADC_SIG          (1L<<16)     // Sign bit if this bit is same as MSB
- conversion out of the range
#define PL820IS_ADC_DATA(D)      ((D)&0xffff)  // Data 16 bit

#define PL820IS_ADC_DLY_WIDTH     8            // Number of bits in
                                               //   ADC /CS hold delay counter
#define PL820IS_ADC_CTRL_WIDTH    13           // Number of bits in ADC control word
#define PL820IS_ADC_DATA_WIDTH    19           // # of bits in ADC data word (do not
                                               //   read 12 data bits and 5 sub-lsbs)
#define PL820IS_ADCCH_NUM         5            // # of ADC channels
                                               //  (4 SE + die temperature)
#define PL820IS_ADC_CH0CFG        0x1610       // default ADC channel config
#define PL820IS_ADC_CH1CFG        0x1710       // default ADC channel config
#define PL820IS_ADC_CH2CFG        0x1630       // default ADC channel config
#define PL820IS_ADC_CH3CFG        0x1730       // default ADC channel config
#define PL820IS_ADC_CH4CFG        0x1418       // default ADC channel config
```

```
// FPGA registers - CLI side
#define PL820_CLI_STR             (0x0000)    // R layer status register
#define PL820_CLI_LCR             (0x0000)    // W layer control register,
                                              //    do readback with DQL_CLI_LCRR
#define PL820_CLI_ISDR            (0x0004)    // R isolated side RX data register
#define PL820_CLI_ISTR            (0x0004)    // W isolated side TX transmit register

#define PL820_CLI_LID             (0x0008)    // R layer identification
                                              //  {u16 BOM,u16 Model}

#define PL820_CLI_NISV            (0x000C)    // R   NIS logic revision
                                             //{u8 rsvd, u8 major, u8 minor, u8 dbg}

#define PL820_CLI_SPD0            (0x0010)   // R/W scratchpad register 0 (8 bits)
#define PL820_CLI_SPD1            (0x0014)   // R/W scratchpad register 1 (8 bits)

// PL820_CLI_LCR
#define PL820_LCR_LIOE     (1UL<<31)     // Layer I/O enable - controls iso_cmd line
#define PL820_LCR_LRE      (1UL<<30)     // Layer reset enable

#define PL820_LCR_LED      (1UL<<1)      // LED control
#define PL820_LCR_DCDC     (1UL<<0)      // DC/DC control

// PL820_CLI_STR
// Layer Status Register  PL820_CLI_LSR (0x0000+RD)
// individual bit definitions
#define PL820_LSR_BSY     (1UL<<31)  // Layer BUSY (reserved)
#define PL820_LSR_ILS     (1UL<<30)  // Interrupt line status (1=IRQ requested)
#define PL820_LSR_EDR     (1UL<<29)  // EEPROM data ready
#define PL820_LSR_EIB     (1UL<<28)  // EEPROM interface busy
#define PL820_LSR_IDR     (1UL<<27)  // IS-data ready
#define PL820_LSR_IBT     (1UL<<26)  // IS-interface TX busy
#define PL820_LSR_IBR     (1UL<<25)  // IS-interface RX busy
#define PL820_LSR_LES     (1UL<<24)  // Layer error status (reserved)
#define PL820_LSR_EXT1    (1UL<<23)  // iso_ext1 line current state
#define PL820_LSR_EXT0    (1UL<<22)  // iso_ext0 line current state
#define PL820_LSR_EXT_POS (22)       // bit position of iso_ext0
#define PL820_LSR_INT1    (1UL<<21)  // iso_int1 line current state
#define PL820_LSR_INT0    (1UL<<20)  // iso_int0 line current state
```

3.4.6.1 FPGA Register Constants

```
// FPGA registers - PL820 FPGA side
#define PL820_S            0x2000     // Base address for the FPGA

#define PL820_CFG          0x2000     // R/W  General configuration register
#define PL820_STS          0x2004     // R    General status register

// DPLL control
#define PL820_PLL0_BCFG    0x2020     // W    PLL0 configuration register
#define PL820_PLL0_BSTS    0x2020     // R    Status of the PLL0
#define PL820_PLL1_BCFG    0x2024     // W    PLL1 configuration register
#define PL820_PLL1_BSTS    0x2024     // R    Status of the PLL1
#define PL820_PLL0_DIV     0x2028     // W    Post-divider for PLL0 (0bypass divider)
#define PL820_PLL1_DIV     0x202C     // W    Post-divider for PLL1 (0bypass divider)
```

```
// NIS pins mode
#define PL820_NISMD0       0x2040     // W    NIS IO31-0 mode (0-default 1-test;
                                      //      NIS-IO 2, 4, 8, 12, 16 excluded)
#define PL820_NISMD1       0x2044     // W    NIS IO 63-32 mode (0-default 1-test;
                                        //      NIS-IO 39 excluded)
// NIS pins direction
#define PL820_NISDIR0      0x2048     // W    NIS IO 31-0 direction in test mode (1-
output)
#define PL820_NISDIR1      0x204C     // W    NIS IO 63-32 direction in test mode (1-
output)

// NIS pins values on output
#define PL820_NISOUT0      0x2050    // R/W  NIS IO 31-0 value when test mode is
                                     //      enabled and direction is output;
                                     //      read return last written value
#define PL820_NISOUT1      0x2054    // R/W  NIS IO 63-32 value when test mode is
                                     //      enabled and direction is output;
                                     //      read return last written value
// NIS pins readbask
#define PL820_NISIN0       0x2058    // R    NIS IO 31-0 current state
#define PL820_NISIN1       0x205C    // R    NIS IO 63-0 current state

// SPI0 section (user SPI)
#define PL820_SPI0_CFG     0x2100     // W  SPI0 configuration register
#define PL820_SPI0_STS     0x2104     // R  SPI0 status register
#define PL820_SPI0_FWM     0x210C     // W  SPI0 FIFO watermark register
#define PL820_SPI0_OFDT0   0x2110     // W  SPI0 output FIFO data register bits 31-0
#define PL820_SPI0_OFDT1   0x2114     // W  SPI0 output FIFO data register bits 63-32
#define PL820_SPI0_OFDT0E  0x2118     // W  SPI0 output FIFO last word
                                      //     data register bits 31-0

#define PL820_SPI0_IFDT0    0x2120    // R  SPI0 input FIFO bits 31-0 data register
#define PL820_SPI0_IFDT1    0x2124    // R  SPI0 input FIFO bits 63-32 data register

// SPI1 section (user SPI)
#define PL820_SPI1_CFG      0x2180    // W  SPI1 configuration register
#define PL820_SPI1_STS      0x2184    // R  SPI1 status register
#define PL820_SPI1_FWM      0x218C    // W  SPI1 FIFO watermark register
#define PL820_SPI1_OFDT0    0x2190    // W  SPI1 output FIFO
                                      //       data register bits 31-0

#define PL820_SPI1_OFDT1    0x2194    // W  SPI1 output FIFO
                                      //     data register bits 63-32

#define PL820_SPI1_OFDT0E   0x2198    // W  SPI1 output FIFO last word
                                      //    data register bits 31-0

#define PL820_SPI1_IFDT0    0x21A0     // R SPI1 input FIFO bits 31-0 data register
#define PL820_SPI1_IFDT1    0x21A4     // R SPI1 input FIFO bits 63-32 data register

//           PL820_CFG             0x2000     // R/W  General configuration register
#define PL820_CFG_SPI1R     (1L<<2)    // 1 - reset user SPI1
#define PL820_CFG_SPI0R     (1L<<1)    // 1 - reset user SPI0
#define PL820_CFG_CPLDR     (1L<<0)    // 1 - reset CPLD
```

```
//           PL820_STS              0x2004     // R        General status register
// Report status of the PL-820 module
// Sticky (latched) status bits auto-cleared after read
#define PL820_STS_SPI1TXFES (1L<<23)   // 1 - SPI1 TX FIFO is empty
#define PL820_STS_SPI1TXHFS (1L<<22)   // 1 - SPI1 TX FIFO is below watermark
#define PL820_STS_SPI1RXFFS (1L<<21)   // 1 - SPI1 RX FIFO is full
#define PL820_STS_SPI1RXHFS (1L<<20)   // 1 - SPI1 RX FIFO is above watermark
#define PL820_STS_SPI0TXFES (1L<<19)   // 1 - SPI0 TX FIFO is empty
#define PL820_STS_SPI0TXHFS (1L<<18)   // 1 - SPI0 TX FIFO is below watermark
#define PL820_STS_SPI0RXFFS (1L<<17)   // 1 - SPI0 RX FIFO is full
#define PL820_STS_SPI0RXHFS (1L<<16)   // 1 - SPI0 RX FIFO is above watermark

// Current (static) status bits
#define PL820_STS_SPI1TXFE  (1L<<15)   // 1 - SPI1 TX FIFO is empty
#define PL820_STS_SPI1TXHF  (1L<<14)   // 1 - SPI1 TX FIFO is below watermark
#define PL820_STS_SPI1TXFF  (1L<<13)   // 1 - SPI1 TX FIFO is full
#define PL820_STS_SPI1RXFE  (1L<<12)   // 1 - SPI1 RX FIFO is empty
#define PL820_STS_SPI1RXHF  (1L<<11)   // 1 - SPI1 RX FIFO is above watermark
#define PL820_STS_SPI1RXFF  (1L<<10)   // 1 - SPI1 RX FIFO is full
#define PL820_STS_SPI0TXFE  (1L<<9)    // 1 - SPI0 TX FIFO is empty
#define PL820_STS_SPI0TXHF  (1L<<8)    // 1 - SPI0 TX FIFO is below watermark
#define PL820_STS_SPI0TXFF  (1L<<7)    // 1 - SPI0 TX FIFO is full
#define PL820_STS_SPI0RXFE  (1L<<6)    // 1 - SPI0 RX FIFO is empty
#define PL820_STS_SPI0RXHF  (1L<<5)    // 1 - SPI0 RX FIFO is above watermark
#define PL820_STS_SPI0RXFF  (1L<<4)    // 1 - SPI0 RX FIFO is full
#define PL820_STS_SPI1BSY   (1L<<3)    // 1 - SPI1 busy
#define PL820_STS_SPI0BSY   (1L<<2)    // 1 - SPI0 busy
#define PL820_STS_ISRDBSY   (1L<<1)    // 1 - CPLD->FPGA interface busy
#define PL820_STS_ISWRBSY   (1L<<0)    // 1 - FPGA->CPLD interface busy

//           PL820_PLL0_BCFG       0x2020    // W       PLL0 configuration register
//           PL820_PLL0_BSTS       0x2020    // R       Status of the PLL0
//           PL820_PLL1_BCFG       0x2024    // W       PLL1 configuration register
//           PL820_PLL1_BSTS       0x2024    // R       Status of the PLL1
// PLL0 is currently for SPI0 clock
// PLL1 is currently for SPI1 clock

#define PL820_PLL_BCFG_RLD     (1L<<27)   // 1 - reload configuration from ROM
                                          //     check BSTS_BUSY bit
                                          //     (should be0 after data is loaded)
                                          //     reload should be followed
                                          //     by PLL update (PUPD) cycle
#define PL820_PLL_BCFG_PUPD    (1L<<26)   // 1 - update PLL with previously written
                                          //     data bit is auto-cleared
                                          //     it is applied to the PLL
                                          //     check BSTS_BUSY bit
                                          //     (should be 0 after data is written)
#define PL820_PLL_BCFG_PWR     (1L<<25)   // 1 - write PLL register bit
                                          //     is auto-cleared after it is
                                          //     applied to the PLL
                                          //     check BSTS_BUSY bit
                                          //     (should be 0 after data is written)
#define PL820_PLL_BCFG_PCLR    (1L<<24)   // 1 - reset PLL bit is auto-cleared
                                          //     after it is applied to the PLL
#define PL820_PLL_BCFG_CPRM2   (1L<<22)   // 3-bit counter parameter
                                          //    (see Cyclone
                                          //     ALTPLL_RECONFIG)
```

```
#define PL820_PLL_BCFG_CPRM(D) ((D)<<20)  // C0-C4 :  0 - high count (8-bit data);
                                          //          1 - low count (8-bit data);
                                          //          4 - bypass (1-bit data);
                                          //          5 - mode (odd/even 1-bit data)
                                          // M/N :    0145 - same as for C0-C4;
                                          //          7 - load nominal count
                                          //              (9-bit data)
#define PL820_PLL_BCFG_CTYPE3  (1L<<19)   // 4-bit counter type
#define PL820_PLL_BCFG_CTYPE(D)((D)<<16)  // 0 - N 1 - M 2 - CP/LF 3 - VCO
                                          // 4-8 - C0-C4
#define PL820_PLL_BCFG_D8      (1L<<8)    // 9-bit data that will be written
                                          //     to the PLL register
#define PL820_PLL_BCFG_D(D)   ((D)&0x1ff) //

#define PL820_PLL_BSTS_LOCKED   (1L<<1)   // 1 - PLL is locked (normal operation)
#define PL820_PLL_BSTS_BUSY     (1L<<0)   // 1 - PLL is busy
                                          //     (in reconfiguration mode)
// Master SPI can be reconfigured at any time however after each
//    re-configuration SPI reset
//    should be issued by toggling corresponding PL820_CFG_SPIxR bit
#define PL820_SPI_CFG_CSI      (1L<<31)      // Invert SCS if 1

#define PL820_SPI_CFG_ABE(N)   ((N)<<24)     // Address bit MSB set to
                                             //   "PL820_SPI_CFG_DBE"
                                             //   if address is unused
#define PL820_SPI_CFG_PSR      (1L<<23)      // If==1: receive data is valid
                                             //        on rising edge of the clock
#define PL820_SPI_CFG_ABS(N)   ((N)<<16)     // Address bit LSB set to 0
                                             //     if address is unused
#define PL820_SPI_CFG_PSW      (1L<<15)      // If==1: transmit data is valid
                                             //        on rising edge of the clock
#define PL820_SPI_CFG_DBE(N)   ((N)<<8)      // Data MSB
#define PL820_SPI_CFG_MWE      (1L<<7)       // 1 multi-word mode + auto-increment
                                             //    address once RX SPI data
                                             //    is latched

#define PL820_SPI_CFG_DBS(N)   ((N)<<0)      // Data LSB

// get parts of the config word into usable bits
#define PL820_SPI_CFGGET_CSI(W)     (((W)&PL820_SPI_CFG_CSI) >> 31)
#define PL820_SPI_CFGGET_ABE(W)     ((W & PL820_SPI_CFG_ABE(0x3F))>>24)
#define PL820_SPI_CFGGET_PSR(W)     (((W)&PL820_SPI_CFG_PSR) >>23)
#define PL820_SPI_CFGGET_ABS(W)     ((W & PL820_SPI_CFG_ABS(0x3F))>>16)
#define PL820_SPI_CFGGET_PSW(W)     (((W)&PL820_SPI_CFG_PSW) >> 15)
#define PL820_SPI_CFGGET_DBE(W)     (((W) & PL820_SPI_CFG_DBE(0x3F))>>8)
#define PL820_SPI_CFGGET_MWE(W)     (((W)&PL820_SPI_CFG_MWE) >> 7)
#define PL820_SPI_CFGGET_DBS(W)     ((W & PL820_SPI_CFG_DBS(0x3F))>>0)

//            PL820_SPI0_STS           0x2104   // R        SPI0 status register
//            PL820_SPI1_STS           0x2184   // R        SPI1 status register

// Report current FIFO levels and status of the SPI interface
#define PL820_SPI_STS_TXFE   (1L<<31)   // 1 if TX FIFO was ever empty
                                        //   since the last read
#define PL820_SPI_STS_RXFF   (1L<<30)   // 1 if RX FIFO was ever full
                                        //   since the last read
```

```
// Bits 26-0 are static i.e. current values are reported
#define PL820_SPI_STS_TXFUW_N(W)  (((W)>>16)&(PL820_SPI_TX_FIFO_SZ-1))
                                          //  # of occupied words in the TX FIFO

#define PL820_SPI_STS_SPIB        (1L<<15)   //  1 if SPI transaction is in progress

#define PL820_SPI_STS_RXFUW_N(W) ((W)&(PL820_SPI_RX_FIFO_SZ-1))
                                          // # of occupied words in the RX FIFO

//            PL820_SPI0_FWM       0x210C   // W       SPI0 FIFO watermark register
//            PL820_SPI1_FWM       0x218C   // W       SPI1 FIFO watermark register
// Watermark registers are used to set FIFO half-full interrupt levels
#define PL820_SPI_FWM_TXWM(N)   (((N)&(PL820_SPI_TX_FIFO_SZ-1))<<16)
                                          //    TX FIFO watermark level

#define PL820_SPI_FWM_RXWM(N)   ((N)&(PL820_SPI_RX_FIFO_SZ-1))
                                          //    RX FIFO watermark level

#define PL820_SPI_FWMGET_TXWM(N)
                           (((N)&PL820_SPI_FWM_TXWM((PL820_SPI_RX_FIFO_SZ-1)))>>16)
                                          //    TX FIFO watermark level

#define PL820_SPI_FWMGET_RXWM(N)
                          ((N)&PL820_SPI_FWM_RXWM((PL820_SPI_RX_FIFO_SZ-1)))
                                          //    RX FIFO watermark level

// PL820_SPI0_OFDT0    0x2110   // W       SPI0 output FIFO data register bits 31-0
// PL820_SPI0_OFDT1    0x2114   // W       SPI0 output FIFO data register bits 63-32
// PL820_SPI0_OFDT0E   0x2118   // W       SPI0 output FIFO last word data register
//                              //               bits 31-0
// PL820_SPI1_OFDT0    0x2190   // W       SPI1 output FIFO data register bits 31-0
// PL820_SPI1_OFDT1    0x2194   // W       SPI1 output FIFO data register bits 63-32
// PL820_SPI1_OFDT0E   0x2198   // W       SPI1 output FIFO last word data register
//                                                bits 31-0

// Up to 64-bits of the output data are split into the two registers.
// For the >32 bits write to
// PL820_SPIx_OFDT1 should proceed write to PL820_SPIx_OFDT0 or PL820_SPIx_OFDT0E.
// Write to PL820_SPI0_OFDT0E indicates end of the frame for
// the multi-word transaction

// PL820_SPI0_IFDT0   0x2120   // R      SPI0 input FIFO bits 31-0 data register
// PL820_SPI0_IFDT1   0x2124   // R      SPI0 input FIFO bits 63-32 data register
// PL820_SPI1_IFDT0   0x21A0   // R      SPI1 input FIFO bits 31-0 data register
// PL820_SPI1_IFDT1   0x21A4   // R      SPI1 input FIFO bits 63-32 data register

// Up to 64-bits of the input data are split into the two registers.
// For the >32 bits read from
// PL820_SPIx_IFDT1 should preceed read from PL820_SPIx_IFDT0.
// Read from PL820_SPIx_IFDT0 advances RX FIFO.

#define PL820_E               0x2FFC   // Last address in FPGA address space
```

3.4.6.2 CPLD Register Constants

```
// All CPLD PL820IS_xx registers are accessed via the serial interface using
// NIS-IO pins 2,4,8,12,16,39 in serial mode and thus require delay
// of ~(42) 33MHz clocks
// for the write and ~(42+34) 33MHz clocks for the read operation
// Also access can be interrupt-driven or polled - based on status bits in LSR
// register

#define PL820IS_S              0x3000    // Base address for the CPLD

#define PL820IS_CFG            0x3000    // R/W  General configuration register
#define PL820IS_STS            0x3004    // R    General status register

// Format for the UEI serial interface -  40-bit FPGA-->CPLD
// MSB is a write bit (1 - write to the register) following 7 bits - address of the
// register in the CPLD shifted 2 bits to the right
//   and ANDed with 16'h7F and 32 bits data
// CPLD-->FPGA returns back 32 bits of data.
// In both directions transmission starts with two start bits
//
#define PL820IS_WREN_BIT     (1L<<39)           // 1 - write to the register
#define PL820IS_ADDR_A8                         // (Register address >> 2)
#define PL820IS_ADDR_A2_8(A) (((A)&0x7f)<<32)   // - mapped to 0x3000-0x31FC
                                                //     address space
#define PL820IS_ADDRDNA_A8                      // DNA address bits mapped to CPLD
#define PL820IS_ADDRDNA_A2_8(A)
                             (((A)&0x7f)<<2)    // - address space
#define PL820IS_DATA_D31                        // 32-bit data
#define PL820IS_DATA_D(D)    ((D)&0xffffffff)

// Interrupt on the CPLD side
#define PL820IS_IER          0x3010             // W   Interrupt enable register
#define PL820IS_ISR          0x3014             // R   Interrupt status register
#define PL820IS_ICR          0x3018             // R   Interrupt clear register

// NIS pins mode
#define PL820IS_NISMD0       0x3030             // W   NIS IO 31-0 mode
                                                //     (0-default 1-test;
                                                // NIS-IO 2, 4, 8, 12, 16 excluded)
#define PL820IS_NISMD1       0x3034             // W   NIS IO 63-32 mode
                                                //     (0-default 1-test;
                                                //     NIS-IO 39 excluded)

// NIS pins direction
#define PL820IS_NISDIR0      0x3038             // W   NIS IO 31-0 direction
                                                //     in test mode (1-output)
#define PL820IS_NISDIR1      0x303C             // W   NIS IO 63-32 direction
                                                //     in test mode (1-output)

// NIS pins value
#define PL820IS_NISOUT0      0x3040             // R/W NIS IO 31-0 value
                                                //     when test mode is enabled
                                                //     and direction is output;
                                                //     read return last
                                                //     written value
```

```
#define PL820IS_NISOUT1      0x3044              // R/W NIS IO 63-32 value
                                                 //     when test mode is enabled
                                                 //     and direction is output;
                                                 //     read return last
                                                 //     written value
// NIS pins readback
#define PL820IS_NISIN0       0x3048              // R    NIS IO 31-0 current state
#define PL820IS_NISIN1       0x304C              // R    NIS IO 63-0 current state

// ADC readback registers
#define PL820IS_ADC0         0x3050    // R    ADC Channel 0 data (3+16LSB valid)
#define PL820IS_ADC1         0x3054    // R    ADC Channel 1 data (3+16LSB valid)
#define PL820IS_ADC2         0x3058    // R    ADC Channel 2 data (3+16LSB valid)
#define PL820IS_ADC3         0x305C    // R    ADC Channel 3 data (3+16LSB valid)
#define PL820IS_ADCTC         0x3060   // R    ADC Channel Temperature data
                                       //         (3+16LSB valid)
// external pins direction
#define PL820IS_USRDIR0      0x3080     // W    USER IO 31-0 direction (1-output)
#define PL820IS_USRDIR1      0x3084     // W    USER IO 63-32 direction (1-output)
#define PL820IS_USRDIR2      0x3088     // W    USER IO 95-64 direction (1-output)
#define PL820IS_USRDIR3      0x308C     // W    USER IO 104-96 direction (1-output)

// external pins value (if output is selected)
#define PL820IS_USROUT0      0x3090     // R/W  USER IO 31-0 value
#define PL820IS_USROUT1      0x3094     // R/W  USER IO 63-32 value
#define PL820IS_USROUT2      0x3098     // R/W  USER IO 95-64 value
#define PL820IS_USROUT3      0x309C     // R/W  USER IO 104-96 value

// external pins readback
#define PL820IS_USRIN0       0x30A0     // R    USER IO 31-0 current state
#define PL820IS_USRIN1       0x30A4     // R    USER IO 63-32 current state
#define PL820IS_USRIN2       0x30A8     // R    USER IO 95-64 current state
#define PL820IS_USRIN3       0x30AC     // R    USER IO 104-96 current state

// SPI0 section (user)
#define PL820IS_SPI0_CFG     0x3100     // W    SPI0 configuration register
#define PL820IS_SPI0_STS     0x3104     // R    SPI0 status register
#define PL820IS_SPI0_ODT0    0x3108     // W    SPI0 output data register bits 31-0
#define PL820IS_SPI0_ODT1    0x310C     // W    SPI0 output data register bits 63-32
#define PL820IS_SPI0_IDT0    0x3110     // R    SPI0 input bits 31-0 data register
bits 31-0
#define PL820IS_SPI0_IDT1    0x3114     // R    SPI0 input bits 63-32 data register
bits 63-32
#define PL820IS_SPI0_IADDR   0x3118     // R    SPI0 input address bits register
#define PL820IS_SPI0_ACFG    0x311C     // W    SPI0 address configuration register
#define PL820IS_SPI0_REG0    0x3120     // W    SPI0 register pool,
                                        //       8 32-bit registers
#define PL820IS_SPI0_REG1    0x3124     // R/W
#define PL820IS_SPI0_REG2    0x3128     // R/W
#define PL820IS_SPI0_REG3    0x312C     // R/W
#define PL820IS_SPI0_REG4    0x3130     // R/W
#define PL820IS_SPI0_REG5    0x3134     // R/W
#define PL820IS_SPI0_REG6    0x3138     // R/W
#define PL820IS_SPI0_REG7    0x313C     // R/W

// SPI1 section (user)
#define PL820IS_SPI1_CFG     0x3180     // W    SPI1 configuration register
#define PL820IS_SPI1_STS     0x3184     // R    SPI1 status register
#define PL820IS_SPI1_ODT0    0x3188     // W    SPI1 output data register bits 31-0
```

```
#define PL820IS_SPI1_ODT1    0x318C     // W    SPI1 output data register bits 63-32
#define PL820IS_SPI1_IDT0    0x3190     // R    SPI1 input bits 31-0 data register
                                        //      bits 31-0
#define PL820IS_SPI1_IDT1    0x3194     // R    SPI1 input bits 63-32 data register
                                        //      bits 63-32
#define PL820IS_SPI1_IADDR   0x3198     // R    SPI1 input address bits register
#define PL820IS_SPI1_ACFG    0x319C     // W    SPI1 address configuration register
#define PL820IS_SPI1_REG0    0x31A0     // R/W  SPI1 register pool,
                                        //      8 32-bit registers
#define PL820IS_SPI1_REG1    0x31A4     // R/W
#define PL820IS_SPI1_REG2    0x31A8     // R/W
#define PL820IS_SPI1_REG3    0x31AC     // R/W
#define PL820IS_SPI1_REG4    0x31B0     // R/W
#define PL820IS_SPI1_REG5    0x31B4     // R/W
#define PL820IS_SPI1_REG6    0x31B8     // R/W
#define PL820IS_SPI1_REG7    0x31BC     // R/W
// PL820IS_CFG                0x3000    // R/W  General configuration register
#define PL820IS_CFG_ADCR     (1L<<3)    // 1 - reset ADC
#define PL820IS_CFG_SPI1R    (1L<<2)    // 1 - reset user SPI1
#define PL820IS_CFG_SPI0R    (1L<<1)    // 1 - reset user SPI0
#define PL820IS_CFG_LED      (1L<<0)    // 1 - turn ON user LED

// PL820IS_STS                0x3004    // R        General status register
#define PL820IS_STS_ADCDR    (1L<<18)   // 1 - data ready from ADC (cleared after
                                        //     read)
#define PL820IS_STS_SPI1DR   (1L<<17)   // 1 - data ready from SPI1 (cleared after
                                        //     read)
#define PL820IS_STS_SPI0DR   (1L<<16)   // 1 - data ready from SPI0 (cleared after
                                        //     read)

// Current (static) status bits
#define PL820IS_STS_SPI1BSY  (1L<<9)    // 1 - SPI1 busy
#define PL820IS_STS_SPI0BSY  (1L<<8)    // 1 - SPI0 busy
#define PL820IS_STS_LVER7    (1L<<7)    // Report logic version
#define PL820IS_STS_LVER0    (1L<<0)    //

// PL820IS_IER         0x3010   // W       Interrupt enable register
// PL820IS_ISR         0x3014   // R        Interrupt status register
// PL820IS_ICR         0x3018   // R        Interrupt clear register
#define PL820IS_IR_ADCDR     (1L<<5)    // 1 - data ready from ADC
#define PL820IS_IR_SPI1WR    (1L<<3)    // 1 - SPI1 write request (RX data received)
#define PL820IS_IR_SPI1RDY   (1L<<2)    // 1 - SPI1 ready (detects end of "busy")
#define PL820IS_IR_SPI0WR    (1L<<1)    // 1 - SPI0 write request (RX data received)
#define PL820IS_IR_SPI0RDY   (1L<<0)    // 1 - SPI0 ready (detects end of "busy")
```

```
// PL820IS_NISMD0      0x3040            // W    NIS IO31-0 mode
//                                       //      (0-default 1-test;
//                                       //         NIS-IO 2, 4, 8, 12, 16 excluded)
// PL820IS_NISMD1      0x3044            // W    NIS IO 63-32 mode
//                                       //      (0-default 1-test;
//                                       //         NIS-IO 39 excluded)
// PL820IS_NISDIR0     0x3048            // W    NIS IO 31-0 direction
//                                       //         in test mode (1-output)
// PL820IS_NISDIR1     0x304C            // W    NIS IO 63-32 direction
//                                       //         in test mode (1-output)
// PL820IS_NISOUT0     0x3040            // R/W  NIS IO 31-0 value
//                                       //         when test mode is enabled
//                                       //         and direction is output;
//                                       //         read return last
//                                       //         written value
// PL820IS_NISOUT1     0x3044            // R/W  NIS IO 63-32 value
//                                       //         when test mode is enabled
//                                       //         and direction is output;
//                                       //         read return last
//                                       //         written value
// PL820IS_NISIN0      0x3048            // R        NIS IO 31-0 current state
// PL820IS_NISIN1      0x304C            // R        NIS IO 63-0 current state
// All registers above are 32-bit wide; bits 2, 4, 8, 12, 16, 39 are always written
with 0s
#define PL820_NIS0_MASK (~0x00011114L)
#define PL820_NIS1_MASK (~0x00000080L)
```

```
// PL820IS_ADC0         0x3050   // R        ADC Channel 0 data (3+16LSB valid)
// PL820IS_ADC1         0x3054   // R        ADC Channel 1 data (3+16LSB valid)
// PL820IS_ADC2         0x3058   // R        ADC Channel 2 data (3+16LSB valid)
// PL820IS_ADC3         0x305C   // R        ADC Channel 3 data (3+16LSB valid)
// PL820IS_ADCTC        0x3060   // R        ADC Channel Temperature data
//                               //         (3+16LSB valid)

#define PL820IS_ADCCH_V3        0            // ADC channel 3
#define PL820IS_ADCCH_V2        1            // ADC channel 2
#define PL820IS_ADCCH_V1        2            // ADC channel 1
#define PL820IS_ADCCH_V0        3            // ADC channel 0
#define PL820IS_ADCCH_TEMP      4            // Internal ADC channel number
                                             //   for temperature

// PL820IS_USRDIR0      0x3080   // W    USER IO 31-0 direction (1-output)
// PL820IS_USRDIR1      0x3084   // W    USER IO 63-32 direction (1-output)
// PL820IS_USRDIR2      0x3088   // W    USER IO 95-64 direction (1-output)
// PL820IS_USRDIR3      0x308C   // W    USER IO 104-96 direction (1-output)
// PL820IS_USROUT0      0x3090   // R/W  USER IO 31-0 value
// PL820IS_USROUT1      0x3094   // R/W  USER IO 63-32 value
// PL820IS_USROUT2      0x3098   // R/W  USER IO 95-64 value
// PL820IS_USROUT3      0x309C   // R/W  USER IO 104-96 value
// PL820IS_USRIN0       0x30A0   // R    USER IO 31-0 current state
// PL820IS_USRIN1       0x30A4   // R    USER IO 63-32 current state
// PL820IS_USRIN2       0x30A8   // R    USER IO 95-64 current state
// PL820IS_USRIN3       0x30AC   // R     USER IO 104-96 current state
// All registers above are 32-bit wide however only bits 8:0 are used for
//   xx_USRxx3 registers

// PL820IS_SPI0_CFG     0x3100   // W     SPI0 configuration register
// PL820IS_SPI0_STS     0x3104   // R     SPI0 status register
// PL820IS_SPI0_ODT0    0x3108   // W     SPI0 output data register bits 31-0
// PL820IS_SPI0_ODT1    0x310C   // W     SPI0 output data register bits 63-32
// PL820IS_SPI0_IDT0    0x3110   // R     SPI0 input bits 31-0 data
//                               //         register bits 31-0
// PL820IS_SPI0_IDT1    0x3114   // R     SPI0 input bits 63-32 data
//                               //         register bits 63-32
// PL820IS_SPI0_IADDR   0x3118   // R     SPI0 input address bits register
//
// PL820IS_SPI1_CFG     0x3180   // W     SPI1 configuration register
// PL820IS_SPI1_STS     0x3184   // R     SPI1 status register
// PL820IS_SPI1_ODT0    0x3188   // W     SPI1 output data register bits 31-0
// PL820IS_SPI1_ODT1    0x318C   // W     SPI1 output data register bits 63-32
// PL820IS_SPI1_IDT0    0x3190   // R     SPI1 input bits 31-0 data
//                               //         register bits 31-0

// PL820IS_SPI1_IDT1    0x3194   // R     SPI1 input bits 63-32
//                               //         data register bits 63-32
// PL820IS_SPI0_IADDR   0x3198   // R     SPI1 input
//                               //         address bits register

#define PL820IS_SPI_CFG_CSI        (1L<<31)         // Invert SCS if 1
#define PL820IS_SPI_CFG_ABE_E      29               // Address bit MSB set to
                                                    //    "PL820IS_SPI_CFG_DBE"
#define PL820IS_SPI_CFG_ABE(D)     ((D)<<24)        // if address is unused
#define PL820IS_SPI_CFG_PSR        (1L<<23)         // If==1: receive data is valid
                                                    //   on rising edge of the clock
#define PL820IS_SPI_CFG_ABS_E      21               // Address bit LSB set
                                                    //   to 0 if address is unused
```

```
#define PL820IS_SPI_CFG_ABS(D)     ((D)<<16)        //
#define PL820IS_SPI_CFG_PSW        (1L<<15)         // If==1: transmit data is valid
                                                    //   on rising edge of the clock
#define PL820IS_SPI_CFG_DBE_E      13               // Data bit MSB
#define PL820IS_SPI_CFG_DBE(D)     ((D)<<8)         //
#define PL820IS_SPI_CFG_MWE        (1L<<7)          // 1 auto-increment address
                                                    //   once RX SPI data is latched
#define PL820IS_SPI_CFG_DBS_E      5                // Data bit LSB
#define PL820IS_SPI_CFG_DBS(D)     ((D)&0x3f)

#define PL820IS_SPI_ACFG_REN       (1L<<31)         // register pool read enabled
#define PL820IS_SPI_ACFG_WEN       (1L<<23)         // register pool write enabled
#define PL820IS_SPI_ACFG_RAD(N)    (((N)&0xff)<<8)  // read address
#define PL820IS_SPI_ACFG_WAD(N)    ((N)&0xff)       // write address



#define PL820IS_E                 0x33FC     // Last address in CPLD address space


// setparam_820 commands
#define DQL_IOCTL820_SETCHNL_CFG 1

#define PL608_PRMFPGA_SPI1       (1L<<8)     // configure SPI1 (FPGA)
#define PL608_PRMFPGA_SPI0       (1L<<7)     // configure SPI0 (FPGA)

#define PL608_PRMCPLD_SPI1       (1L<<6)     // configure SPI1 (CPLD)
#define PL608_PRMCPLD_SPI0       (1L<<5)     // configure SPI0 (CPLD)

#define PL608_PRMFPGA_PLL1       (1L<<4)     // configure PLL1 (FPGA)
#define PL608_PRMFPGA_PLL0       (1L<<3)     // configure PLL0 (CyclonIII required)

#define PL608_PRMCPLD_PORT       (1L<<2)     // configure port side
#define PL608_PRMCPLD_NIS        (1L<<1)     // configure NIS side
#define PL608_PRMFPGA_NIS        (1L<<0)     // configure NIS side
```

Chapter 4 Restoring & Programming the CPLD Image File

This chapter provides instructions for restoring the initial MAX10 archived image provided by UEI, instructions for running a simulation, and instructions for programming the Verilog design onto the DNx-PL-820 CPLD MAX10 hardware.

- Restoring an Archived Quartus Image (Section 4.1)
- Running a Simulation (Section 4.2)
- Programming the Hardware (Section 4.3)

4.1 Restoring an Archived Quartus Image

The following section provides instruction for restoring UEI's initial image from the Quartus archive file delivered with hardware.

Altera Quartus II 64-bit Programmer from Quartus release 15.0 includes support for the MAX10 series and should be used.

STEP 1: Create a C:/QWork directory on your PC.

STEP 2: Copy the .qar archive file into the QWork directory. The archive name is in the format 820_top_0x06_<year><month><day>.qar

STEP 3: Open Quartus II 15.0 (64-bit) Web Edition:

- In the Windows start menu, click
  *Start >> All Programs >> Altera*** >> Quartus II Web** >> Quartus II Web (executable)*

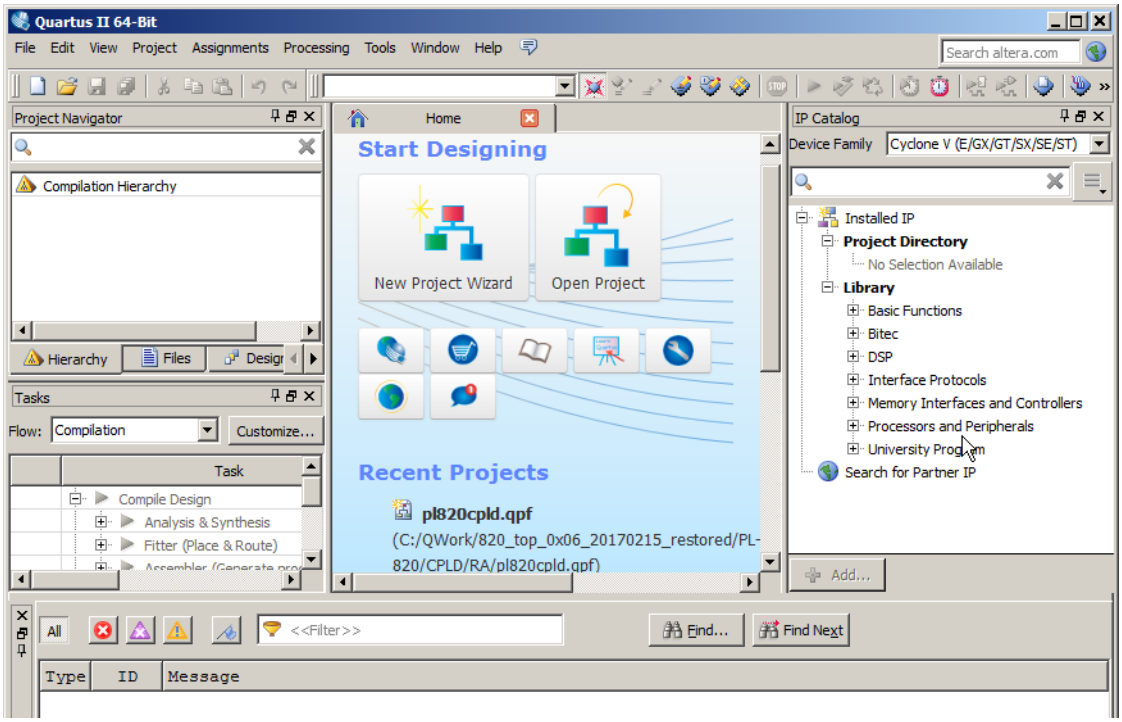


Figure 4-1 Quartus II 15.0

STEP 4: In the Quartus II menu bar, click **Project >> Restore Archived Project...**

STEP 5: In the Restore pop up window, click **...** and navigate to C:/QWork, and click xxx.qar filename. Click **Open**.

- The Restore pop up window will automatically populate the **Destination folder** as the .qar file name with a "_restored" added to the end. If you keep this name, you will not need to update any library paths. If your restore directory is not C:/QWork/"name_of_qar"_restore, then you will need to update library paths and update simulation paths.

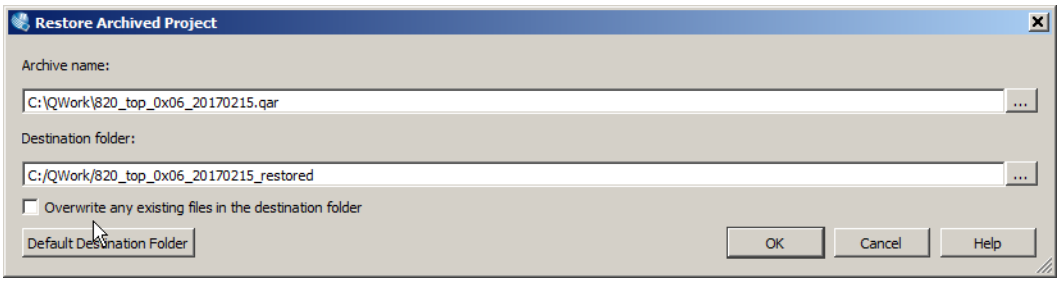


Figure 4-2 Restore Archived Quartus Project

STEP 6: Click **OK** to restore archive and load into Quartus II.

STEP 7: To compile the project, click the purple triangle in the menu bar or click **Processing > Start Compilation**.

As you compile, you will see warning messages in the Messages window, and you will see the status of the design compilation in the Task window (green check marks next to a section indicate the section is compiled).
Refer to **Figure 4-3**.
A successful compile will verify paths and project state.

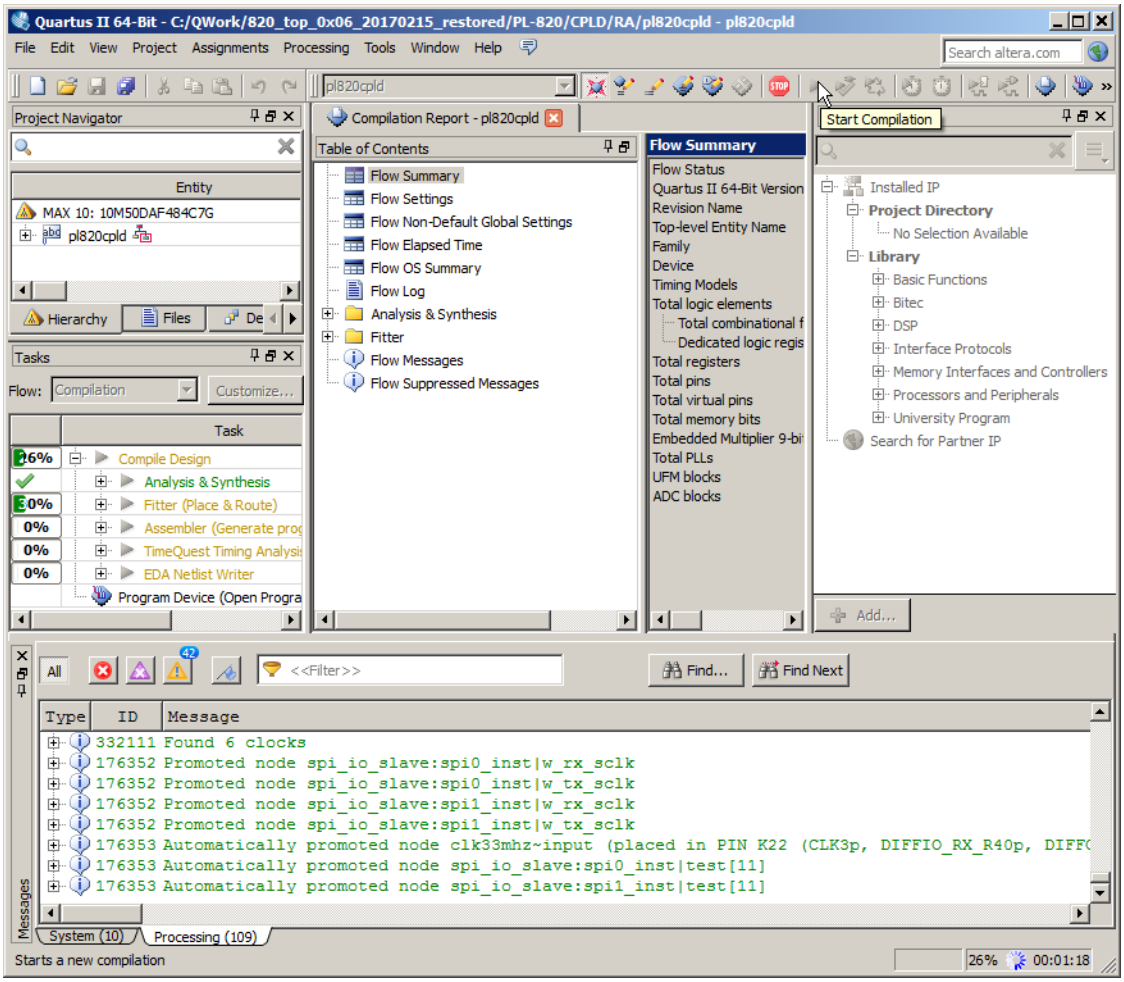


Figure 4-3 Compile Quartus Image

NOTE: You are now ready to run a simulation, or program the hardware (download the image to the MAX10 CPLD).

4.2 Running a Simulation

The following steps open and run a simulation script in Altera's ModelSim software simulator.

STEP 1: Open Altera's ModelSim software simulator:

- In the Windows start menu, click
  *Start >> All Programs >> Altera*** >> ModelSim Altera Starter Edition** >> ModelSim Altera-10.3d (executable)*

STEP 2: Open the testbed file:

- In the ModelSim menu bar, click **File > Open**, and navigate to C:\QWork\820_top_0x06_xxxxxx_restored\PL-820\CPLD\RA\simulation\modelsim
- Select pl820cpld.vt and click **Open.**

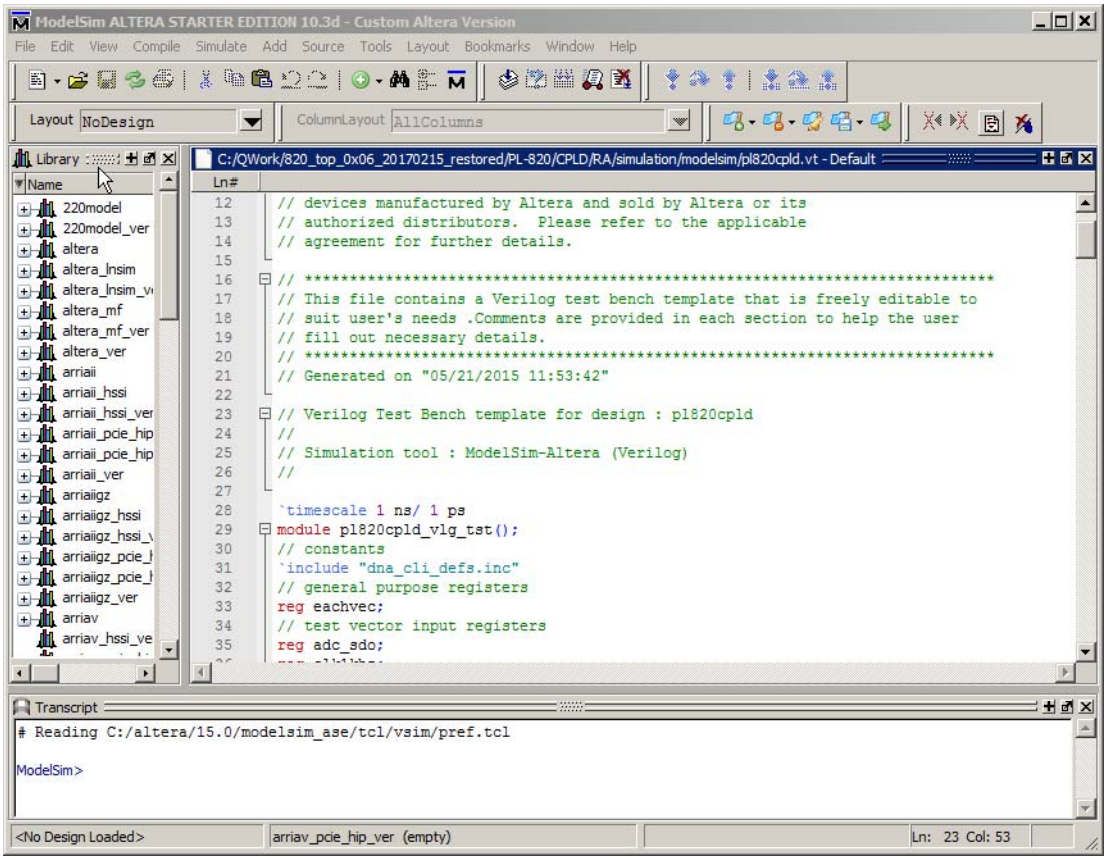


Figure 4-4 ModelSim Window

STEP 3: In the Transcript window at the bottom of the ModelSim GUI, change directory to the simulation directory:

```
ModelSim> cd C:/QWork/820_top_0x06_xxxxxxxx_restored/PL-820/CPLD/RA/simulation/modelsim
```

NOTE: A "`# reading modelsim.ini`" message will display.

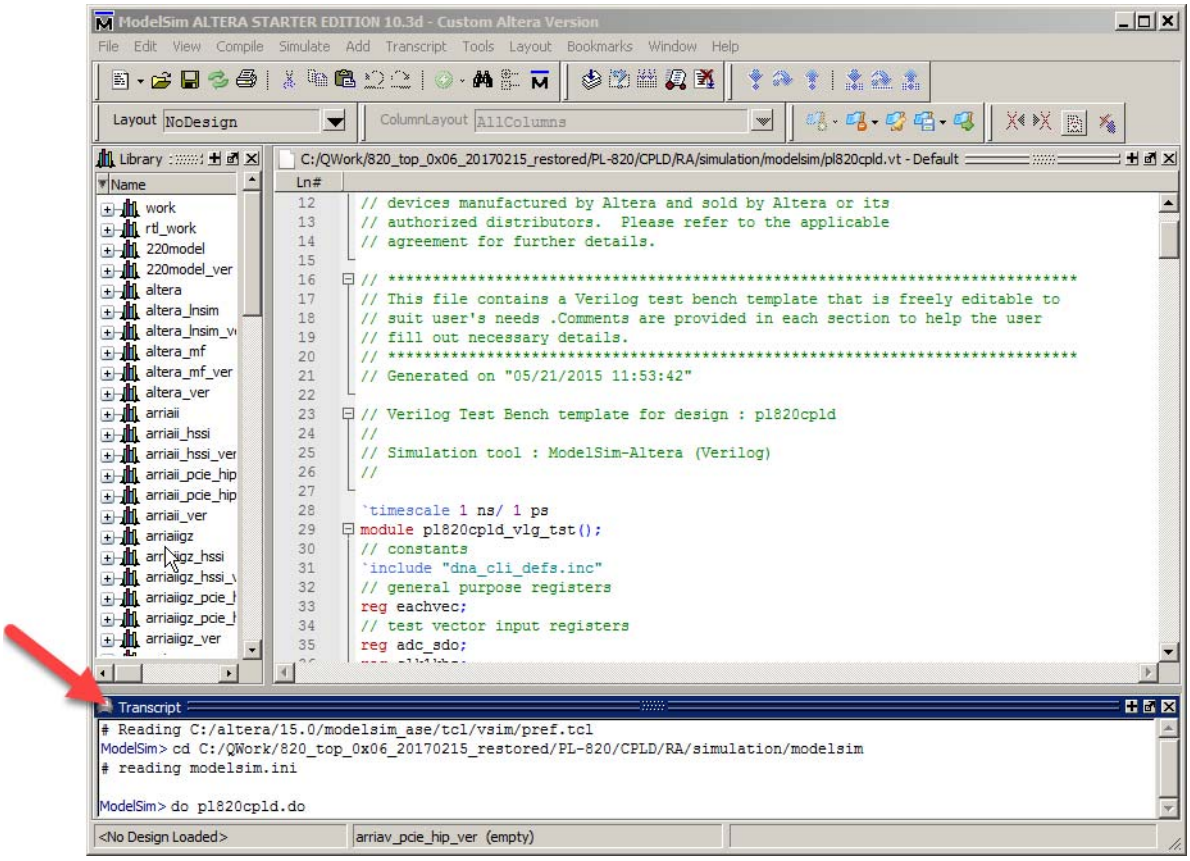


Figure 4-5 Start Simulation

STEP 4: Start the simulation script by typing the "do" command at the ModelSim prompt:

```
ModelSim> do pl820cpld.do
```

NOTE: 1 us of a simulation will run and a waveform display will open.

STEP 5: Run for an additional 100 µs to see serial ports toggle and register values update:

```
VSIM 3> run 100us
```

At this point, you can debug, view internal states, and interact with the design as you wish (see **Figure 4-6**).

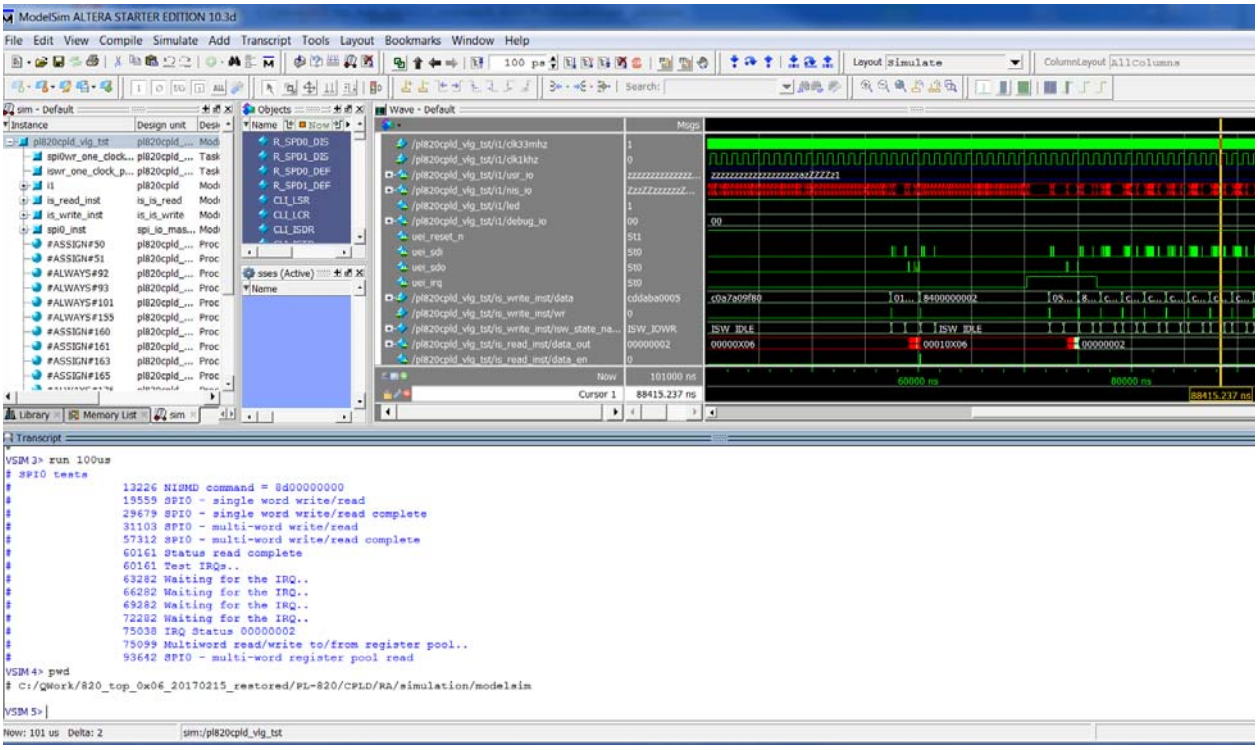


Figure 4-6 View Simulation Waveforms and Internal States

4.3 Programming the Hardware

Use the following steps to download an image onto the Altera MAX10 (program your PL-820 with your custom image or restore an existing image).

STEP 1: The Altera MAX10 10M50DCF484 device is connected to the front connector of the CPLD-820 board (the upper board in the stack of FPGA-820 and CPLD-820 subassemblies).

STEP 2: The following connections are made from DB-62 connector to the terminal board (refer to **Figure 4-7**):

JTAG	DB-62	Signal	Color
1	61	TCK	Green
2	22,21,62	GND	Black/Violet
3	20	TDO	Yellow
4	1	+3.3V	Red
5	41	TMS	Orange
8	1	JTAGEN	via 4.7k resistor
9	42	TDI	Brown
10	22,21,62	GND	Violet

Table 4-1 JTAG to DB-62 Connections

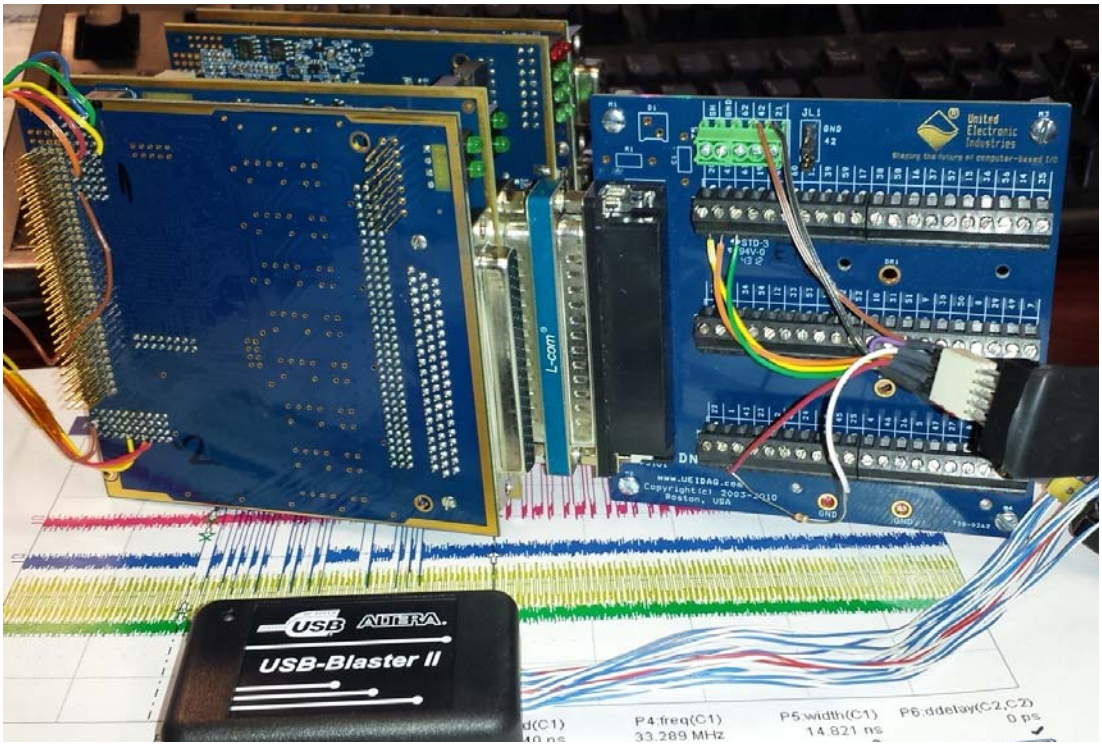


Figure 4-7 JTAG to DB-62 Connections for Quartus II Bit Programming

NOTE: Note that UEI does not recommend using the DNA-CBL-62 cable between STP-62 terminal and CPLD-820 board, but instead we recommend connecting the STP-62 directly to the CLPD-820 boards. A gender-changer (UEI part number 503-6262) allows a direct connection between the STP-62 and CPLD-820. Lines need to be as short as possible.

STEP 3: Open Altera Quartus II 64-bit Programmer from Quartus release 15.0. This version includes support for MAX10 series and should be used.

STEP 4: Set Altera Programmer TCK frequency to 6 MHz to improve reliability. You can do this by issuing the following command:

```
C:\Altera\15.0\programmer\bin64\jtagconfig.exe --setparam USB-BlasterII
JtagClock 6M
```

STEP 5: Open compiled byte code, and load the xxx.pof Programmer Object File, (e.g., pl820cpld.pof).

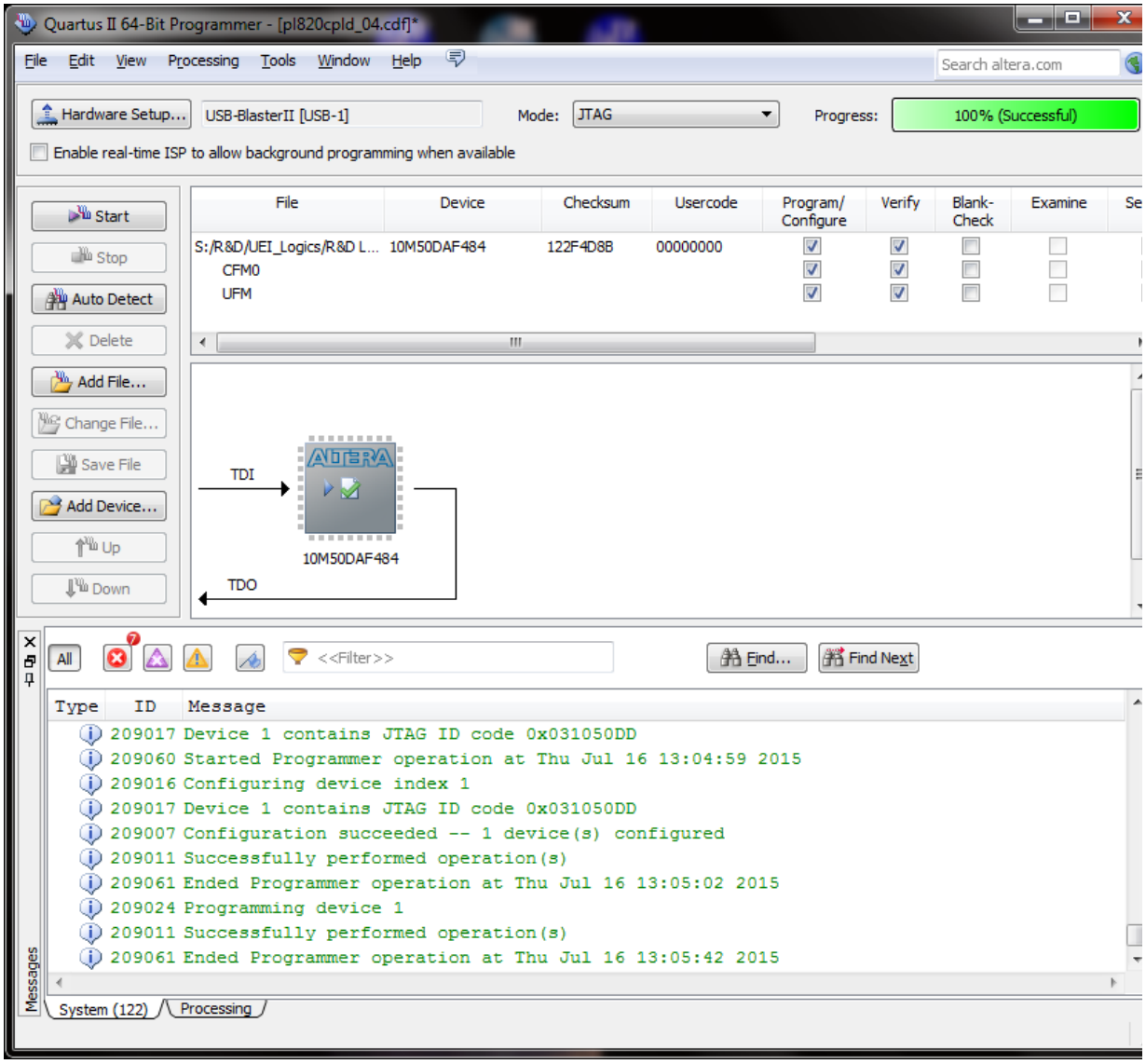


Figure 4-8 Quartus II Bit-Programmer

STEP 6: Set **Hardware Setup** to USB-Blaster II, and **Mode** to JTAG mode (refer to **Figure 4-8**).The Altera Programmer will automatically select the proper device (10M50DAF484 in this case).

STEP 7: Power down the cube or rack chassis, and connect the Altera USB-Blaster II to the terminal (see **Figure 4-7**).

STEP 8: Power up cube or rack and wait until all POST operations are done and the CPLD board is powered by the driver enabling DC/DC (all green lights on the power layer with COM blinking).

STEP 9: Check **Program/Configure** and **Verify,** and press the **Start** button to program CPLD.

STEP 10: Power down the cube or rack, and then power it up again. Read the `PL820IS_STS` register (on CPLD side) to verify the version of logic in CPLD chip. (See Sample820.c, and test descriptions in **Chapter 5**.)

Chapter 5 Testing the DNx-PL-820

This chapter provides instructions for testing the PL-820 using PowerDNA API and UEI provided sample code (Sample802.c).

- Test Overview and Terminology (Section 5.1)
- Installation Overview (Section 5.2)
- PL-820 Testing (Section 5.3)
- Timing Diagrams (Section 5.4)

5.1 Test Overview and Terminology

DNx-PL-820 is a COTS production module consisting of two PCB boards. The bottom board is called the FPGA-820 or simply the FPGA in this chapter. The top one is called the CPLD-820 or simply the CPLD. The FPGA board has an Altera Cyclone II FPGA on it. The CPLD board has an Altera MAX10 device on it.

These boards are interconnected using three headers – JNIS1 and JNIS2 connect the CPLD to the FPGA via the NIS port. This port includes an SSI interface, two SPI interfaces, clocks and multiple general-purpose I/O lines. JIS1 is used to route the second half of user I/O pins from the CPLD device to the DB-62 header of the bottom board (FPGA board).

UEI provides sample code for the following test cases:

Test 1: Test communication port between CPLD and FPGA

Test 2: SPI test of address-multiple data word transactions

Test 3: SPI test of address-data single word (8 to 64 bit word is valid)

Test 4: Test user I/O port (using cable)

Test 5: Test user I/O port writes/reads

Test 6: Test network register read/write time

Test 7: LED test

Test 8: Bit flip on CPLD or FPGA port

5.2 Installation Overview

Before the CPLD board can be tested, the PL-820 should be installed properly in the IOM cube or RACKtangle. Base addresses are dependent on what position a board is installed in the chassis. For the PL-820, the base address is selected on the FPGA layer (bottom). See **PowerDNA Field Installation Manual** for a reference of board insertion positions in a cube chassis. Once the board is installed, the IOM chassis can be powered up.

Note that only the FPGA board is recognizable on the PowerDNA bus. All communication to CPLD board is done via the SSI port on the FPGA.

Register `PL820IS_ID` (0x3008) on the CPLD board is used to report the CPLD model number. The model number is returned as 0x000vA820, where “v” is PCB revision of the installed CPLD board.

5.3 PL-820 Testing

UEI Sample820.c provides test code for each of the test cases described in this chapter.

UEI code examples are located in the following directories:

- For Linux: <PowerDNA-x.y.z>/sdk/DAQLib_Samples
- For Windows: *Start » All Programs » UEI » PowerDNA » Examples*

5.3.1 Checking Version/ Getting Started

Initial testing is to verify the version of the logic installed. To do that, Sample820.c reads two registers: `PL820_CLI_NISV` (FPGA logic version 0x01021172) and `PL820IS_STS` (CPLD logic version 6). The logic versions shown here are for example only.

Once you open Sample820.c, set up the Custom Configuration fields (#defines in the header) based on your system:

1. Set `IOM_IPADDR0` to the IP address of your chassis. For example:
   `#define IOM_IPADDR0 "192.168.100.3"`
2. Verify the board position of the PL-820 as installed in your chassis:
   `#define DEVN -1`
   The `DEVN` constant should be set to -1 and shouldn't be touched unless the IOM has more than one PL-820 layers. The sample will auto detect and default to the last PL-820 in the cube or rack.

NOTE: Test cases can be selected by setting the `TEST_CASE` constant in the header at the top of Sample820.c file to one of the values from 1 to 8 (refer to the list of test cases in Section 5.1 on the previous page). Each test will repeat itself *ad infinitum* until Ctrl+C is pressed.

5.3.2 Test 1: Test Communication between FPGA and CPLD

The first interface on the board to test is the NIS interface. It connects the FPGA on the bottom the board with the CPLD on the upper board.

To run this test, set the test case to 1:

```
#define TEST_CASE   1
```

The test works as follows:

1. Sets all NIS lines into the test mode
2. Writes a pattern to the FPGA and reads back from the CPLD
3. Then reverses to write the pattern to the CPLD and read from FPGA

With the exception of SSI pins, all pins that are required to communicate to CPLD are tested. (Note that the SSI interface is the control communication interface between the boards: without the SSI interface nothing can be written or read from CPLD side; therefore, this test tests it by default.)

Refer to the following page for an example of the output of this test (see **Figure 5-1**).

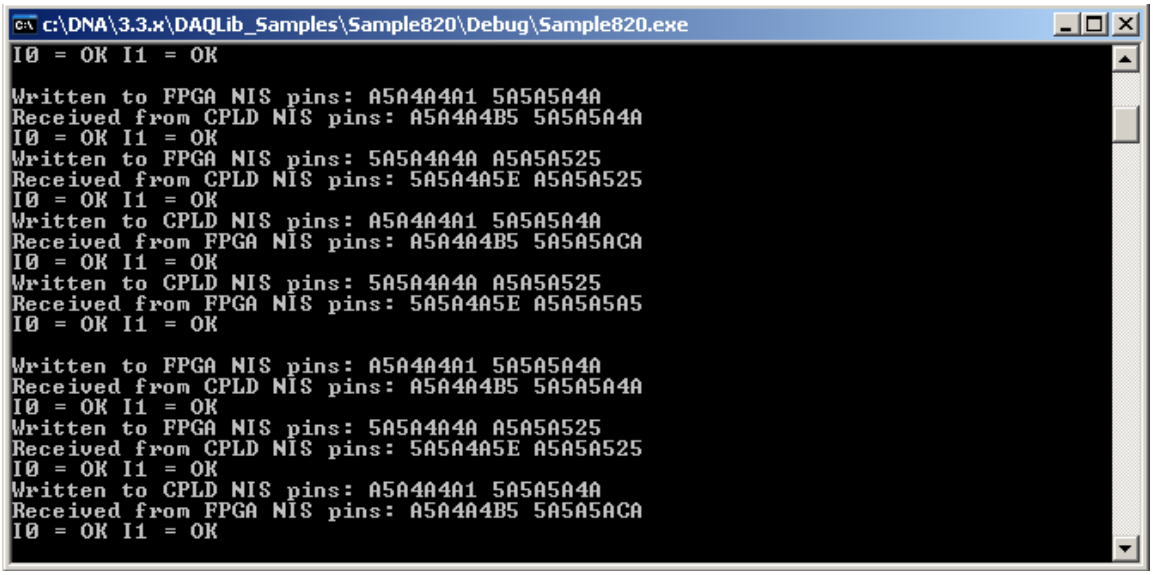


Figure 5-1 Test 1 NIS Interface Test Output Display

NOTE: Note that IOs 2, 4, 8, 12, 16 on NIS0 and IO39 on NIS1 are dedicated to SSI and clocks, thus if they don't work there is no communication between FPGA and CPLD.

5.3.3 Test 2: Test SPI Ports

Test 2 is the SPI test of address-multiple data word transactions.

To run this test, set the test case to 2:

```
#define TEST_CASE    2
```

and select which SPI to test:

```
#define TEST_SPI     0   // SPI0 (0) or SPI1 (1)
```

The following constants control testing different sizes and numbers of words:

```
#define NUM_SPI_REGx  8
//
#define DATAM_START   0
#define DATAM_SIZE    20          // size of data word
#define ADDRM_START   DATAM_SIZE  // address position
#define ADDRM_SIZE    8           // address size
```

Please note that the current single-address multiple-data mode (MWE) implementation in CPLD supports word sizes (i.e. address+data) up to 32 bits. There are up to eight SPI slave addresses which are implemented in the CPLD

```
#define NUM_SPI_REGx    8        // up to 8
```

First test SPI0 and then SPI1. Refer to the following page for an example output display (see **Figure 5-2**).

Note that in the current CPLD slave SPI implementation, the address is set to a constant 8 bits.

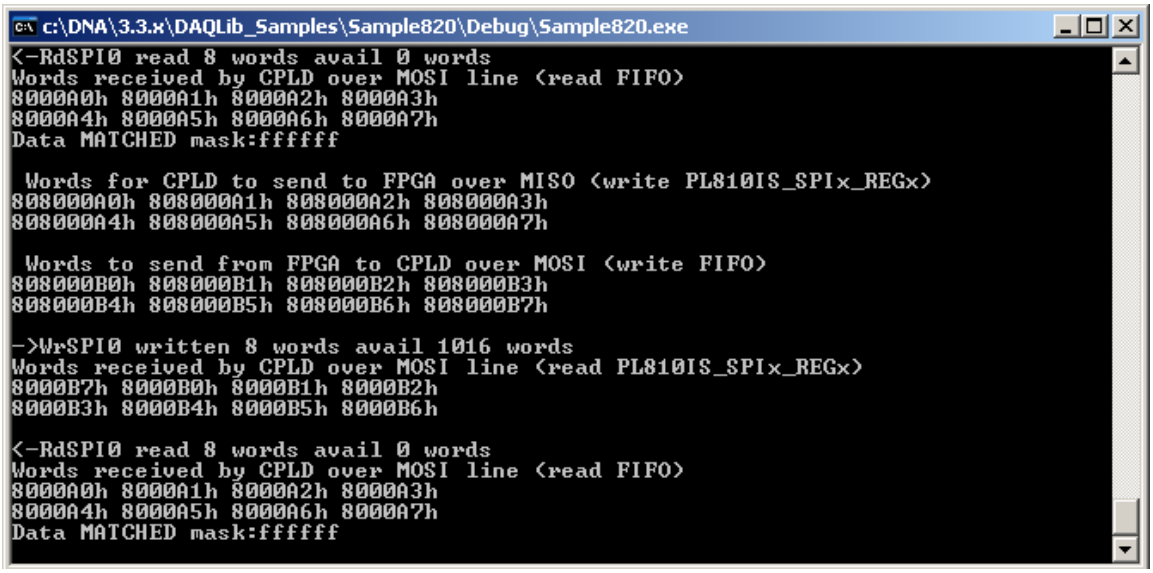


Figure 5-2 Test 2 SPI Single-address Multiple-data Output Display

5.3.4 Test 3: Test SPI in Address/Data Mode

Test 3 is the SPI test of address-single data word transactions.

To run this test, set the test case to 3:

```
#define TEST_CASE    3
```

and select which SPI to test:

```
#define TEST_SPI     0   // SPI0 (0) or SPI1 (1)
```

Default parameters test 48-bit words with 40-bit data and 8-bit addresses. The test automatically defines whether one or two words need to be written to the SPI FIFO (and also read and compared):

```
#define DATA_START   0
#define DATA_SIZE    40          // size of data word
#define ADDR_START   DATA_SIZE   // address position
#define ADDR_SIZE    8           // address size
```

Refer to the following page for an example output display (see **Figure 5-3**).

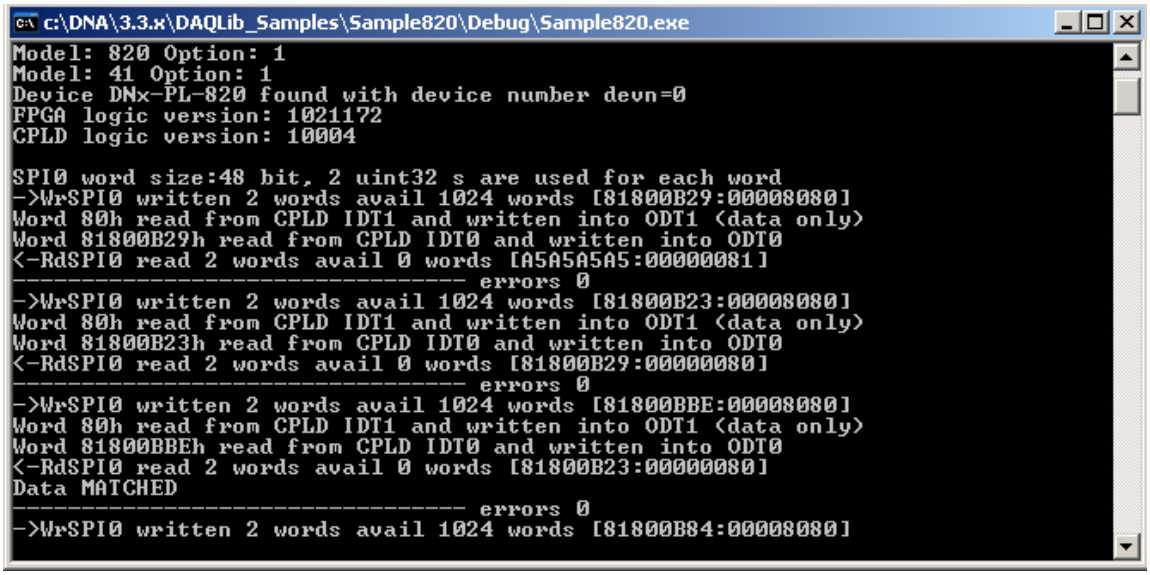


Figure 5-3 Test 3 SPI Single-address Single-data Output Display

To test timing with the scope, uncomment the following line to send fixed value:

```
data_out[1] = 0x1; data_out[0] = 0x808001
```

Refer to “Timing Diagrams ” on page 49 for screenshots of example timing diagrams.

5.3.5 Test 4: Test User I/O Pins (using cable)

Test 4 tests the user I/O port, using a cable.

To run this test, set the test case to 4:

```
#define TEST_CASE    4
```

This test relies on a CBL-96 to be plugged in on one side into the bottom of the FPGA board and on the other side into the top of the CPLD board.

The test selects paired lines and tests them in both directions.

Refer to the following page for an example output display (see **Figure 5-4**).

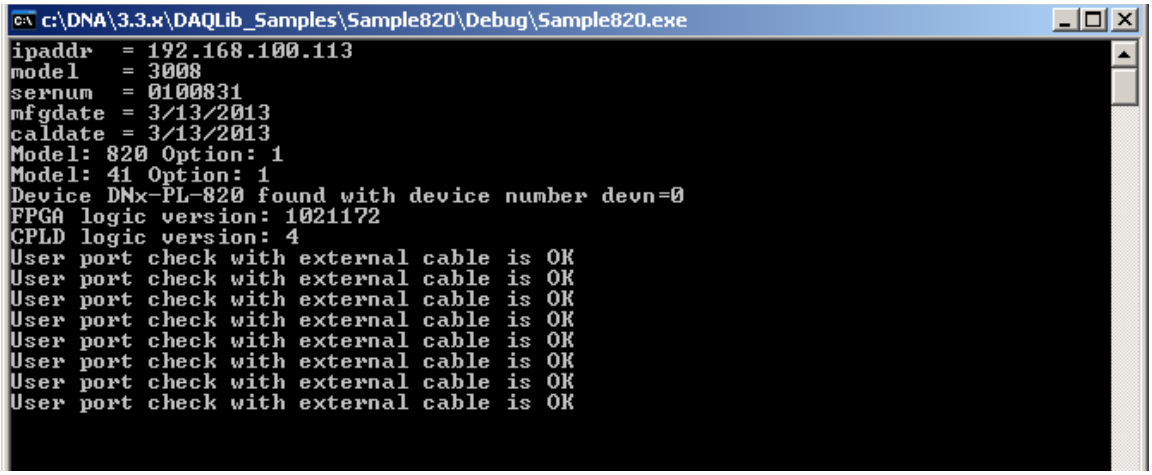


Figure 5-4 Test 4 User IO Test Output Display

The function tests the user port interface through the straight CBL-96 cable, which can be plugged into both CPLD and FPGA boards connecting I/O as in the following table. These are pairs of pins connected to each other when both FPGA and CPLD layers are connected using CBL-62 cable. For example, pin 19 on both connectors is line 0 of port 0 on the FPGA board and bit 101 of port 3 on the CPLD board.

```
#define CONN_PIN    0
#define CPLD_IO     1
#define FPGA_IO     2
//                   { pin, I/O CPLD, I/O FPGA }

uint32 port_pairs[51][3] = {{19,  0, 101},
```

The following functions convert IO pin numbers into port number and bit number:

```
int get_user_port_num(int io_num); and
int get_user_bit_num(int io_num);
```

The table of connected pairs mapping pin number (either on top or bottom connector) and associated CPLD and FPGA digital I/O pins:

Pin	I/O CPLD	I/O FPGA	Pin	I/O CPLD	I/O FPGA	Pin	I/O CPLD	I/O FPGA
19	0	101	40	1	100	60	2	99
18	3	98	39	4	97	59	5	96
17	6	95	38	7	94	58	8	93
16	9	92	37	10	91	57	11	90
15	12	89	36	13	88	56	14	87

Table 5-1 Pin Number to CPLD to FPGA Pin Mapping

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

Table 5-1 Pin Number to CPLD to FPGA Pin Mapping (Cont.)

5.3.6 Test 5: Test User I/O Port Reads/Writes

Test 5 tests reading and writing over the user I/O port.

To run this test, set the test case to 5:

```
#define TEST_CASE    5
```

This test uses the function `test_user_io_by_port()` to write an alternating bit pattern on top and lower board user I/O. I/O pins need to be connected to a screw terminal panel to measure voltage levels (preferably with the scope).

5.3.7 Test 6: Test Network Register Read/ Write Time

Test 6 provides a command timing test.

To run this test, set the test case to 6:

```
#define TEST_CASE    6
```

This test uses the function `test_network_interface()` to perform three tests. First, it tests register I/O for the FPGA board, then register I/O for the CPLD board and then executes configuration IOCTL command. Each test is performed 10^4 times and the average value is returned. The test uses a built-in timestamp counter on the FPGA board to measure the time to perform 10^4 iterations.

Refer to the following page for an example output display (see **Figure 5-5**).

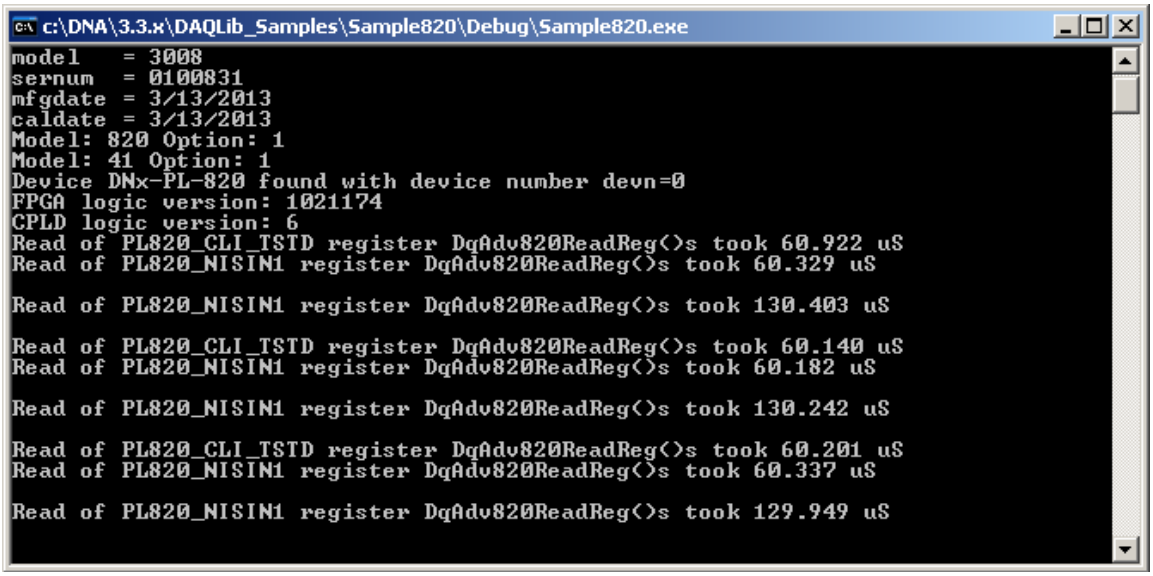


Figure 5-5 Test 6 Timing Test Display

5.3.8 Test 7: LED Test

Test 7 tests PL-820 LEDs.

To run this test, set the test case to 7:

```
#define TEST_CASE    7
```

Run this test to continuously turn LEDs on the base and attachment board on and off.

5.3.9 Test 8: Test Bit States on CPLD or FPGA Port

Test 8 allows verification (with the oscilloscope) of the shape of the edges on the user I/O pins.

To run this test, set the test case to 8:

```
#define TEST_CASE    8
```

The function `test_rapid_io_change()` can be set using

```
#define TEST_TOP     1
```

to turn on/off bit IO0 bit 0 port 0 on CPLD board (0) or IO101 bit 18 port 3 on FPGA board. Both bits are connected to pin 19 on the related DB62 connector (refer to Table 5-1 for pin mappings).

Refer to the following page for an example oscilloscope waveform (see **Figure 5-6**).

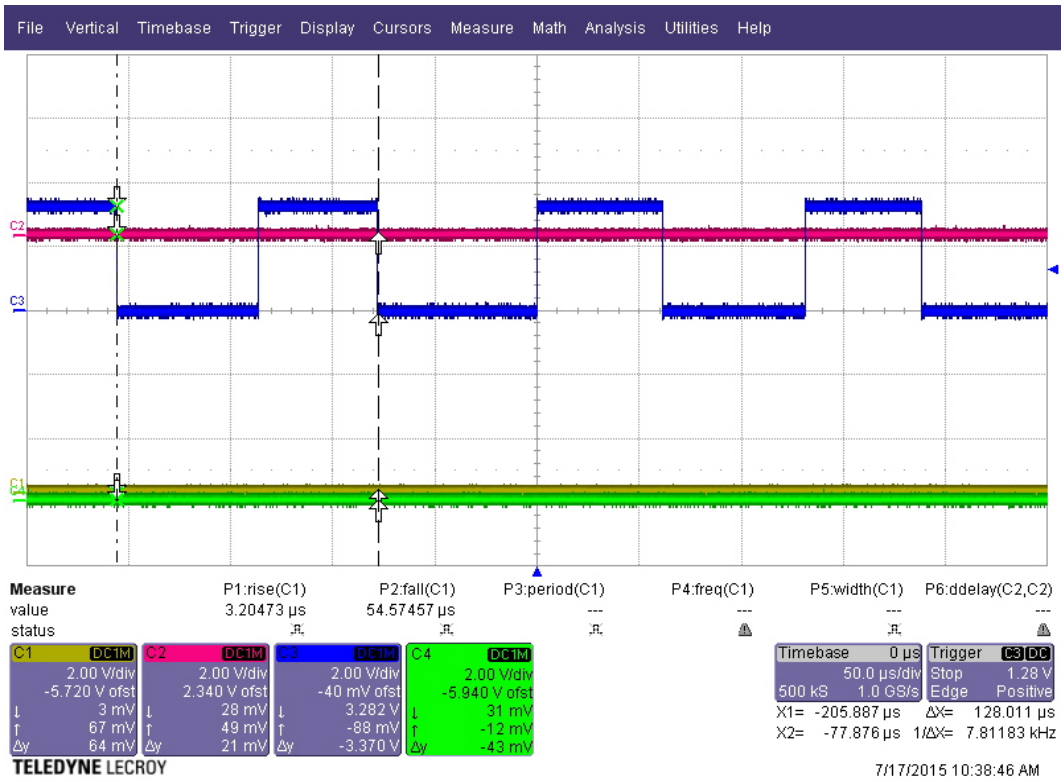


Figure 5-6 Test 8 I/O Pin State Changes Oscilloscope Display

5.4 Timing Diagrams

The following section provides example waveforms for PL-820 SSI, SPI, and external I/O interface timing.

5.4.1 SSI Timing

The following four timing diagrams demonstrate timing on the synchronous serial interface (SSI) between FPGA and CPLD PCB boards. The SSI is used to write and read to/from the CPLD logic registers. Clock (yellow) is a continuous 33 MHz clock (IO39). UEI-SDO (data from FPGA) is red (IO8) and UEI_SDI (data from CPLD) is blue (IO12).

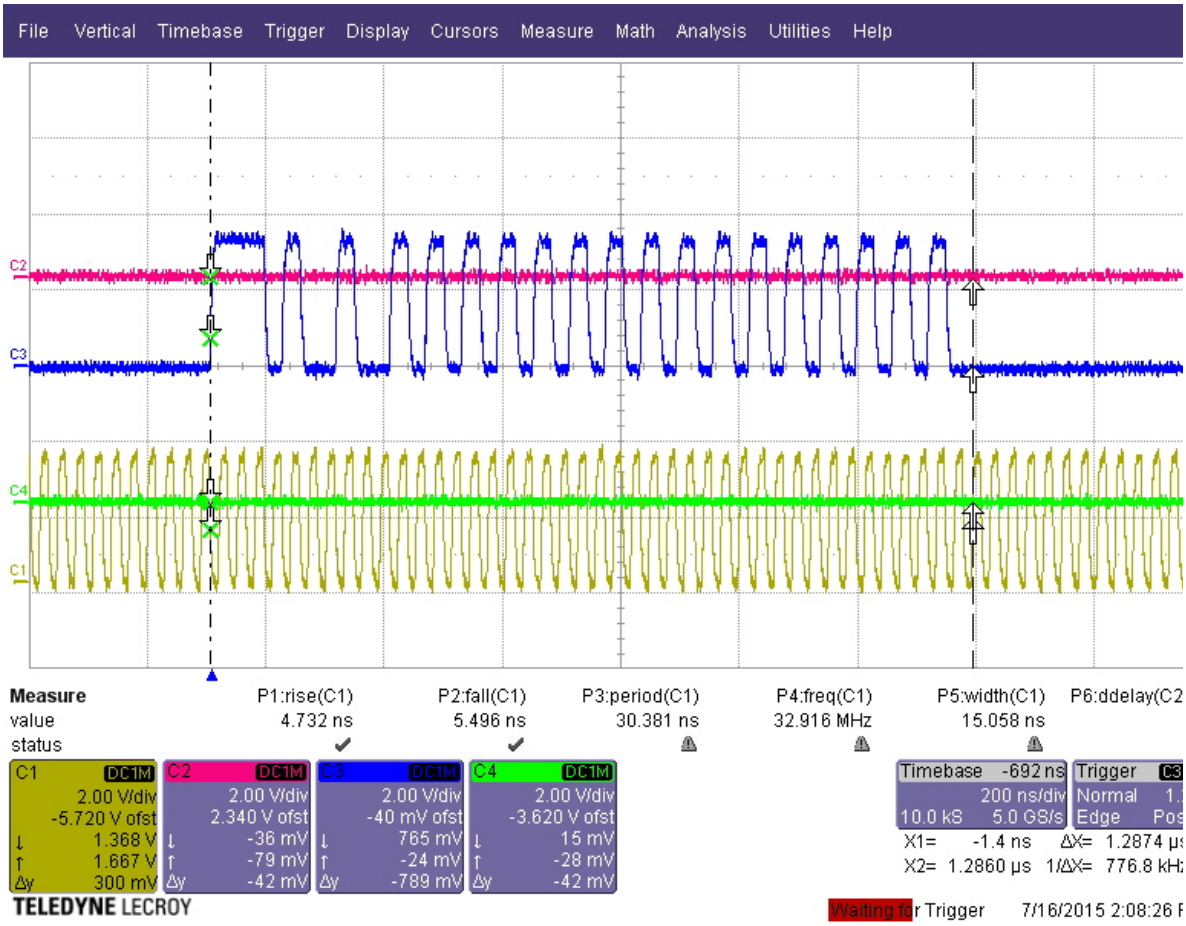


Figure 5-7 Write 0xAAAAAAAA to CPLD Register 0x90 Oscilloscope Display

The write sequence starts from two start bits, write (1) or read (0) bit, seven address bits (only bits 8 through 2 are used to form 32-bit address from 0 to 0x1fc (accessible via the FPGA at 0x3000 range), then follows the 32-bit data. The start sequence and edges relative to the clocks are clearly visible in **Figure 5-8**:

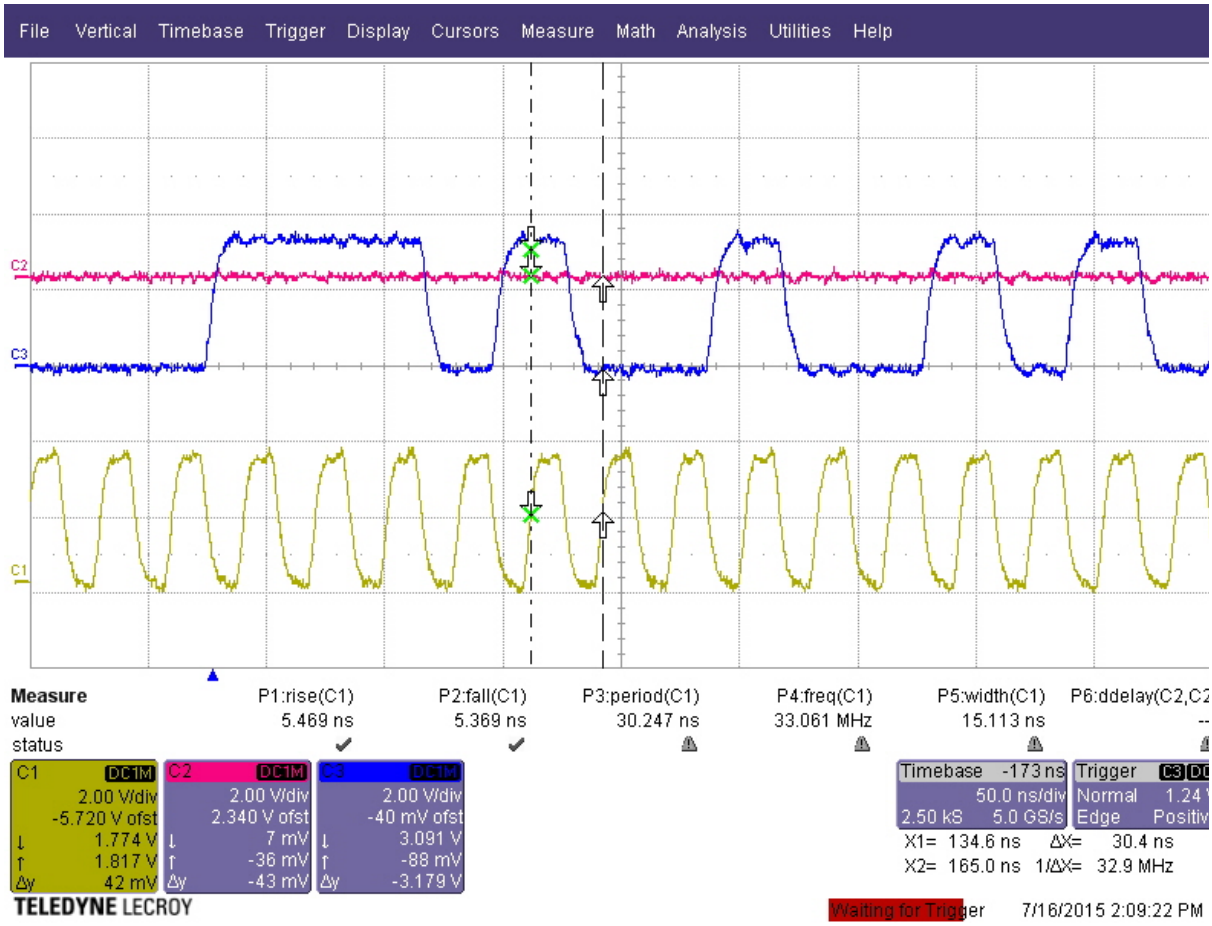


Figure 5-8 Close-up of the SSI Start and Address Bits (Write Command 0x90) Oscilloscope Display

The full read sequence (reads 0xAAAAAAAA for register 0x90) is shown in **Figure 5-9** and in the close-up waveform, **Figure 5-10**.

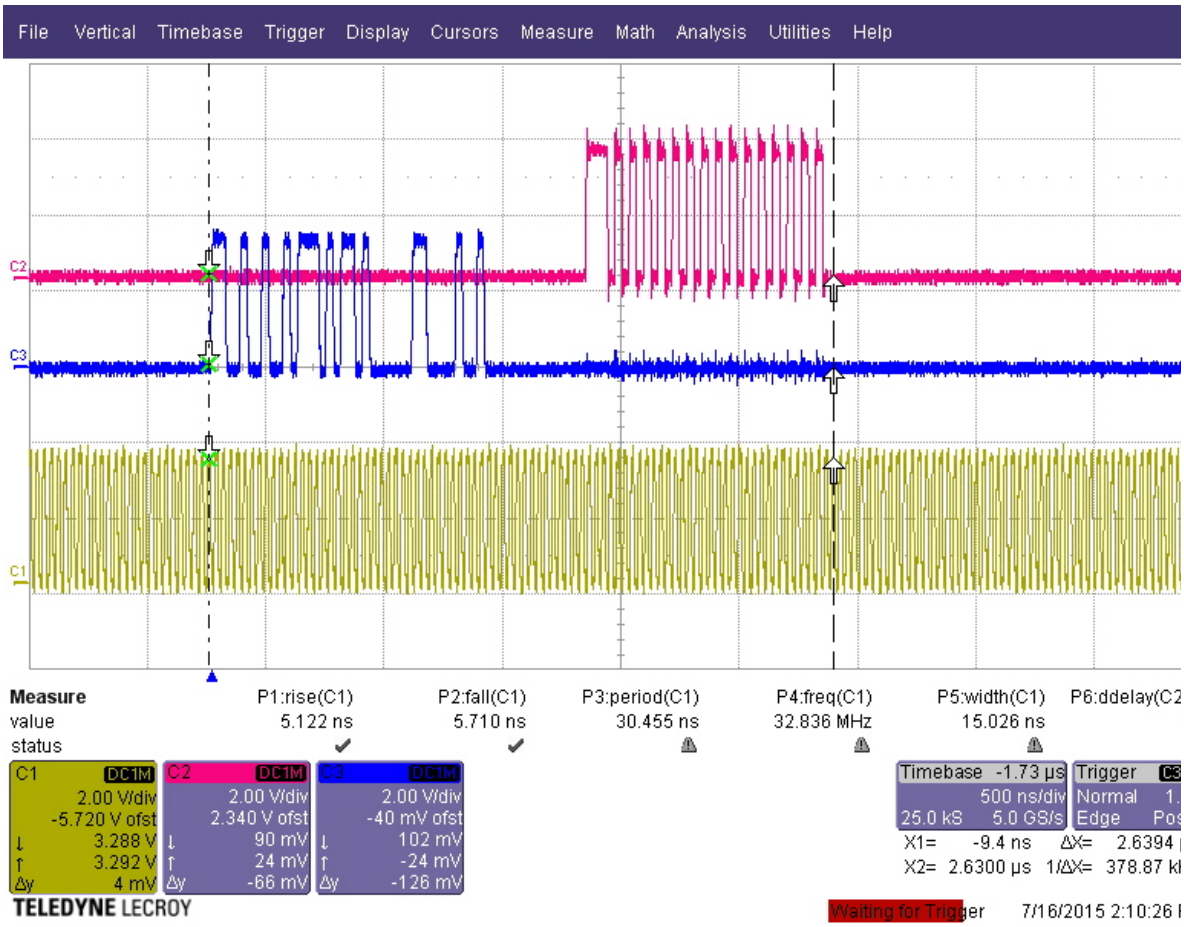


Figure 5-9 Read 0xAAAAAAAA from CPLD Register 0x90 Oscilloscope Display

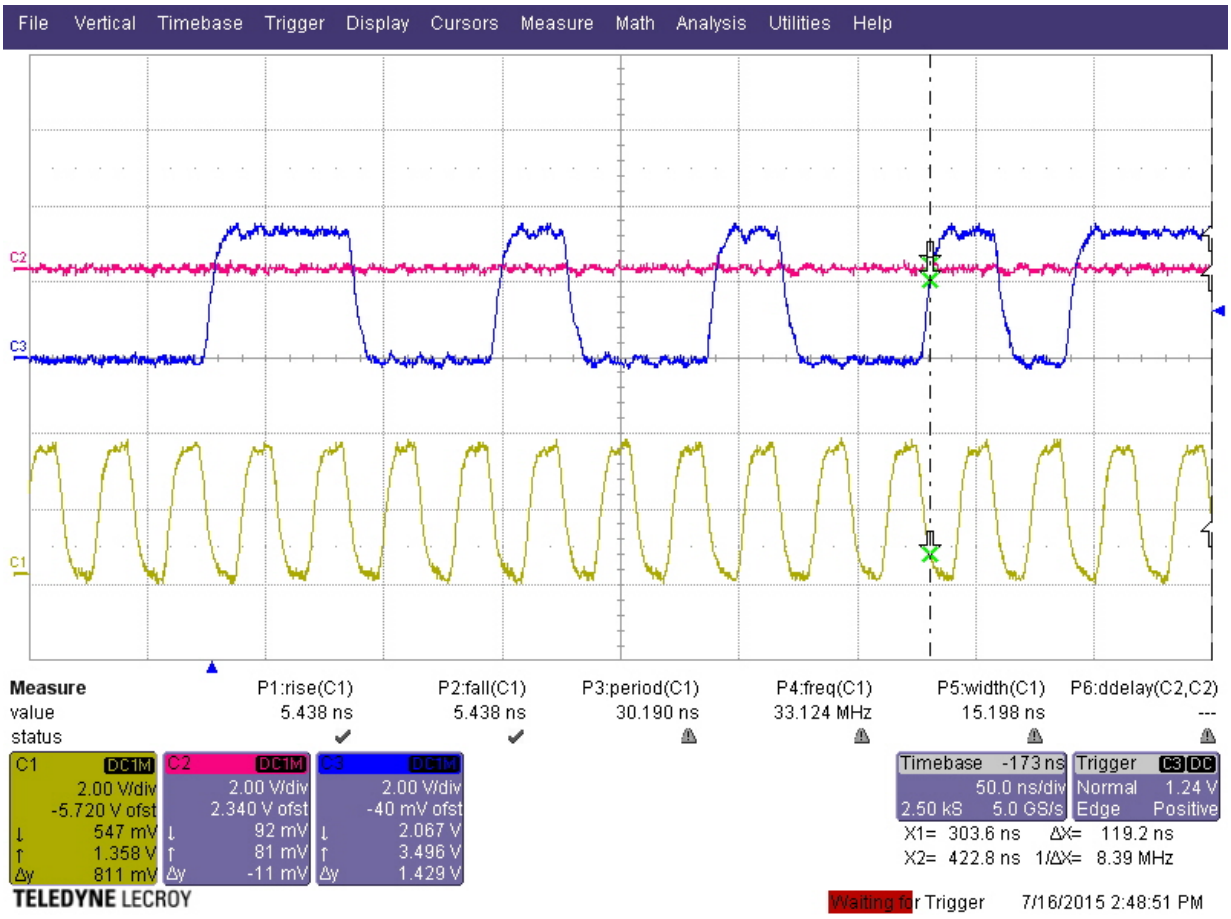


Figure 5-10 Close-up of the SSI Start and Address Bits (Read Command 0x90) Oscilloscope Display

5.4.2 SPI Timing

The DNA-PL-820 board has two independent serial peripheral interfaces (SPIs) that support from 3- to 64-bit data words, including for up to 8 bit addresses. FPGA code incorporates the master SPI implementation and the CPLD the slave one. This interface allows two modes of operation:

1. Every word consists of address and data
2. Only the first word consists of address and data, followed by the words with data only. CS signals the end of the transaction

The Clock can be derived from a 24 MHz base clock via a divider. The SPI interface works at the 8 MHz and below rates (with a 31-bit divider value of 3 and up – divider value 2=3-1).

The SPI interface consists of the following signals:

- SPI_CSx – (Green) – chip select (FPGA output, CPLD input, pin IO50 for SPI0 and IO15 for SPI1)
- SPI_SCLKx – (Yellow) – clock (FPGA output, CPLD input, pin IO47 for SPI0 and IO11 for SPI1)

- SPI_MOSI – (Blue) – master output slave input (FPGA output, CPLD input, pin IO62 for SPI0 and IO20 for SPI1)

SPI_MISO – (Red) – master input slave output (CPLD output, FPGA input, pin IO51 for SPI0 and IO9 for SPI1)

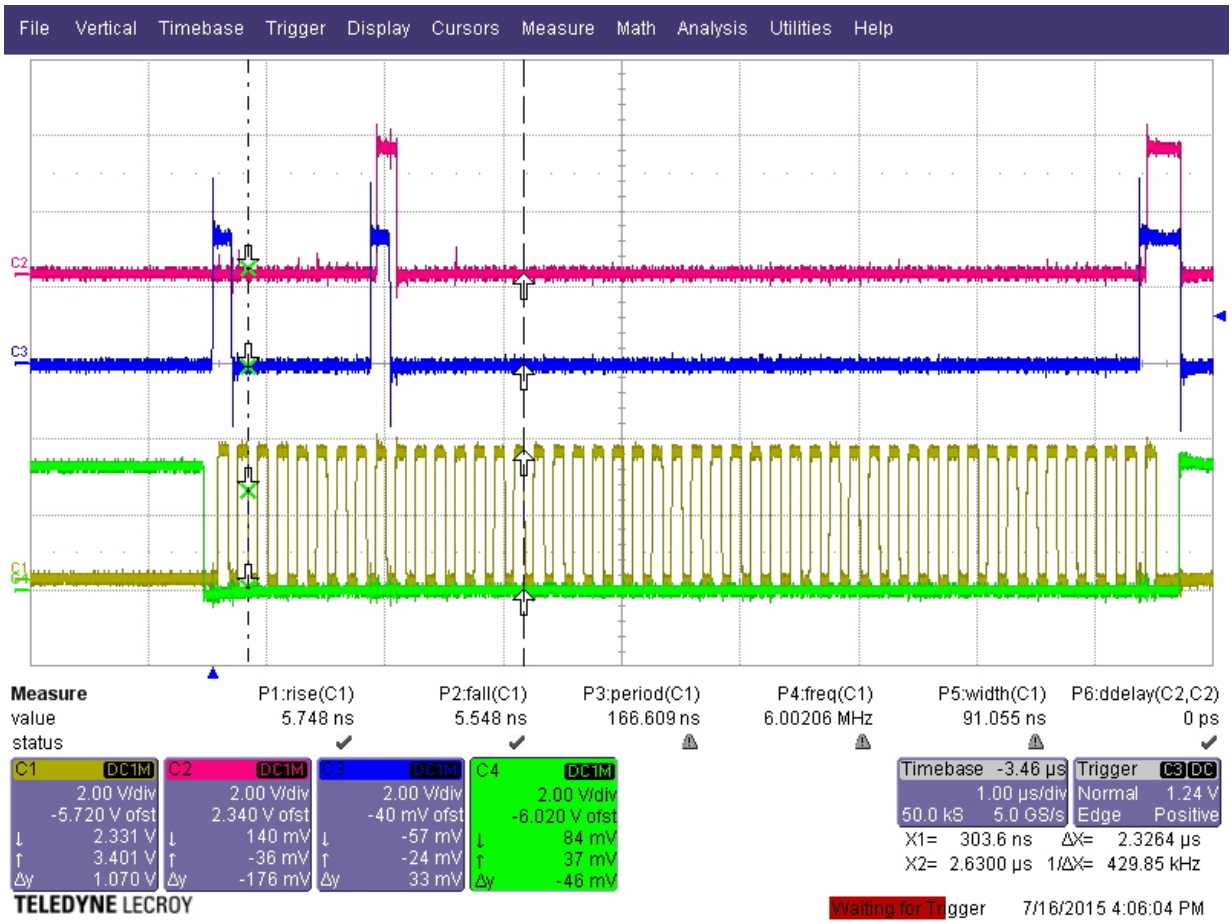


Figure 5-11 SPI Command 0x80 & Data Word 0x80 0000 0001 Oscilloscope Display

Figure 5-11 demonstrates a 48-bit SPI command. The MOSI line transmits the 8-bit command 0x80 and 40-bit word 0x80 0000 0001, and MISO mirrors back 0x80 0000 0001 received on the previous cycle. Note the shift of the rising edge relative to the clock on MOSI and MISO lines: the configuration is set for the CPLD to receive data on the rising edge of the clock and for the FPGA on the falling edge of the same clock (**Figure 5-12**).

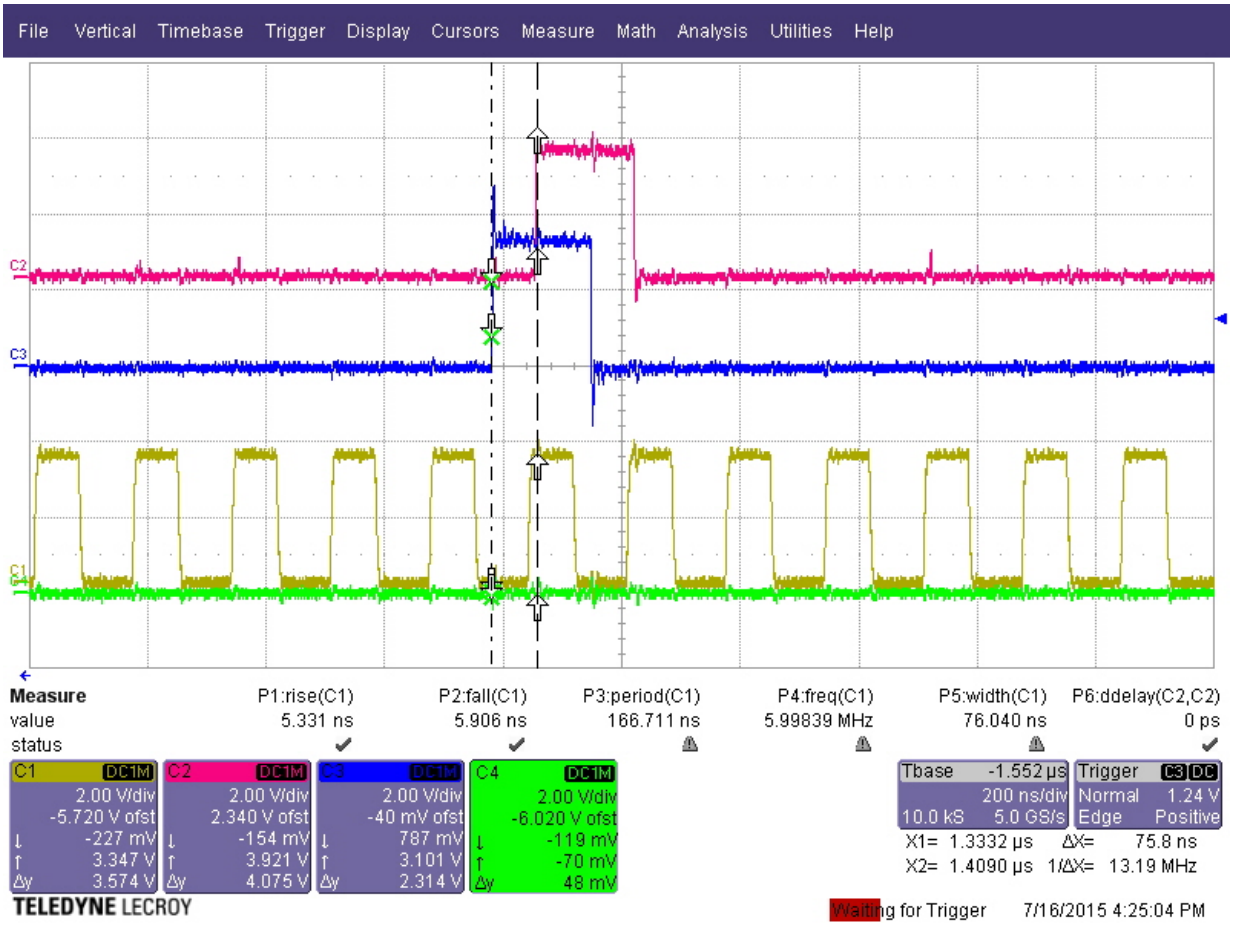


Figure 5-12 MOSI Clock Valid on Rising Edge - MISO on the Falling Edge

The following image (**Figure 5-13**) is an example of 24-bit transaction with 8 bit of address and 16 bit of data.

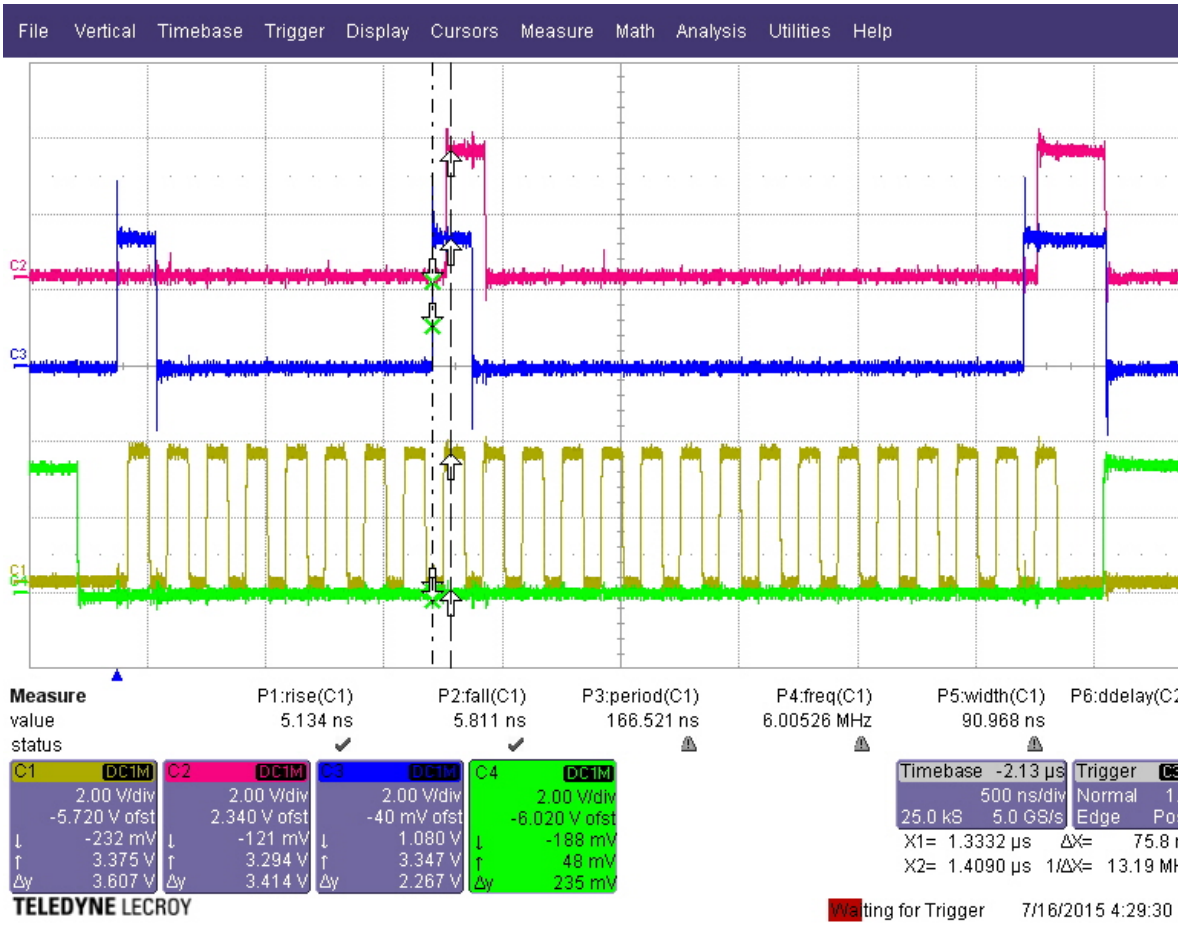


Figure 5-13 24-bit Word/8-bit Address (0x80)/16-bit Data (0x8001)

In multi-word mode, the first word contains the address and data and the following words contain data only. It is assumed that addresses are automatically incremented for each following word.

Figure 5-14 shows an oscilloscope image of a transaction with the first word with the address 0x80 following by the seven words with data only for addresses 0x81, 0x82, etc.

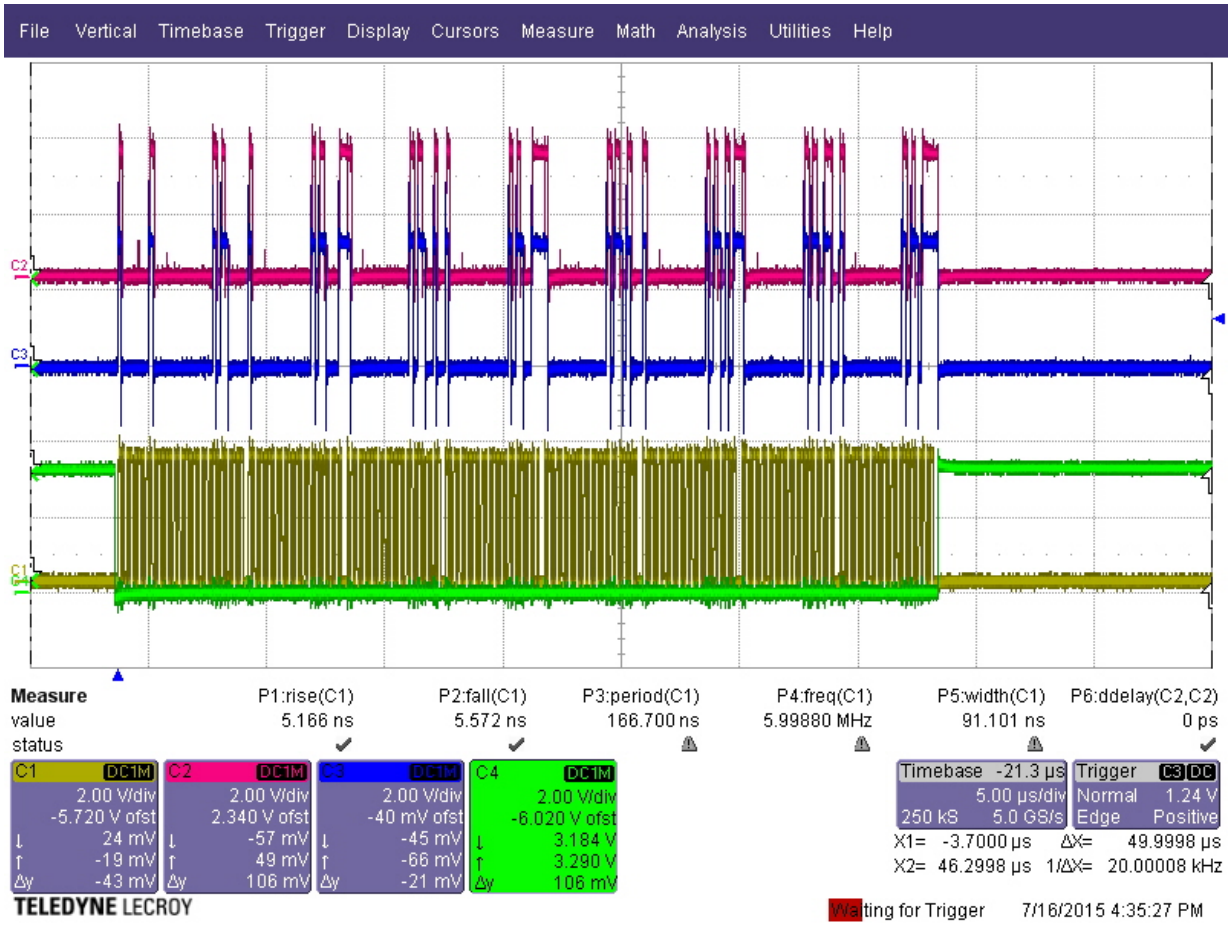


Figure 5-14 Multi-Word Transmission

The first word takes 32 clock cycles to transmit and each following word takes 24 cycles for the transmission (**Figure 5-15**):

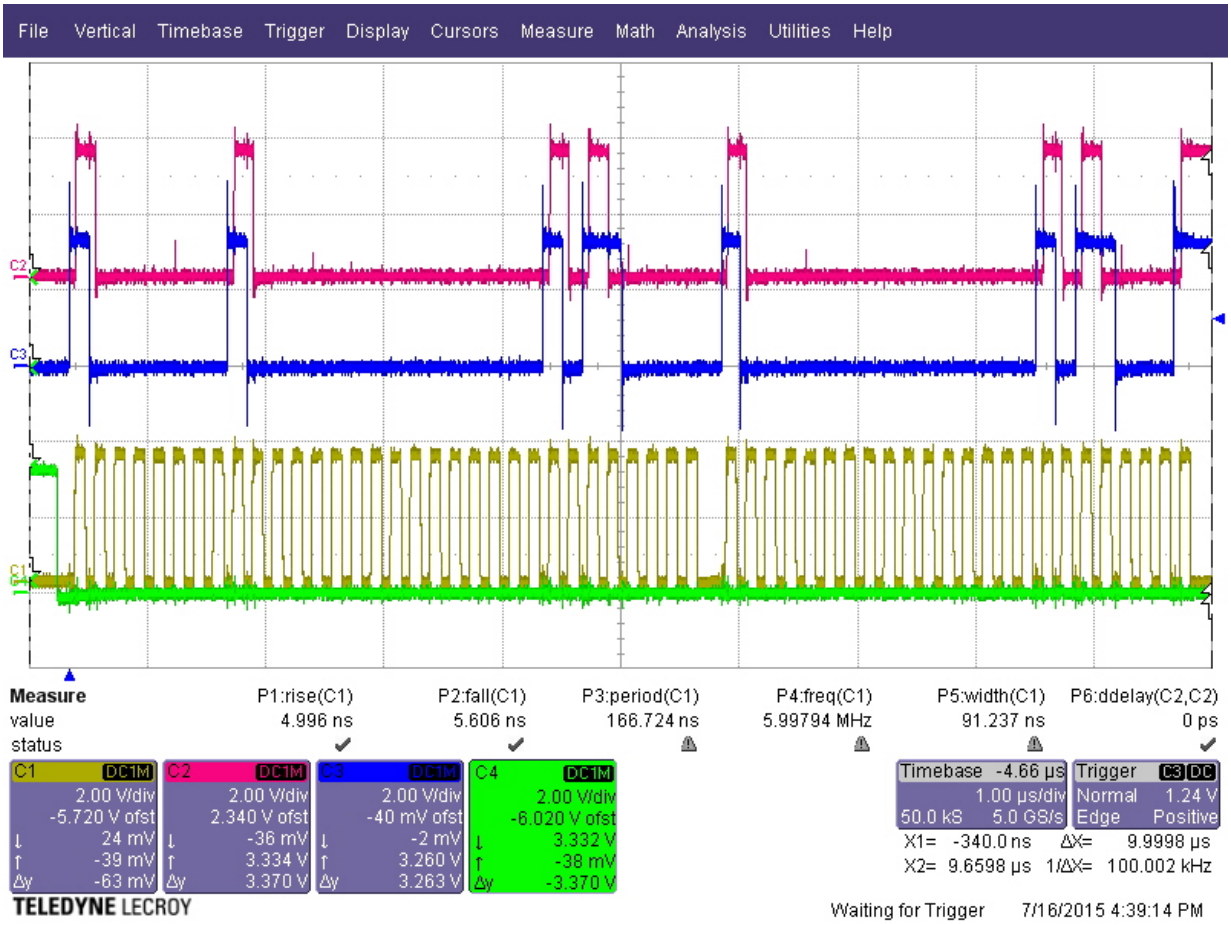


Figure 5-15 The First Two Words of a Multi-Word Transmission

As a demonstration of the interface capabilities, **Figure 5-16** shows a 4-word transaction with 3-bit address and 5-bit data:

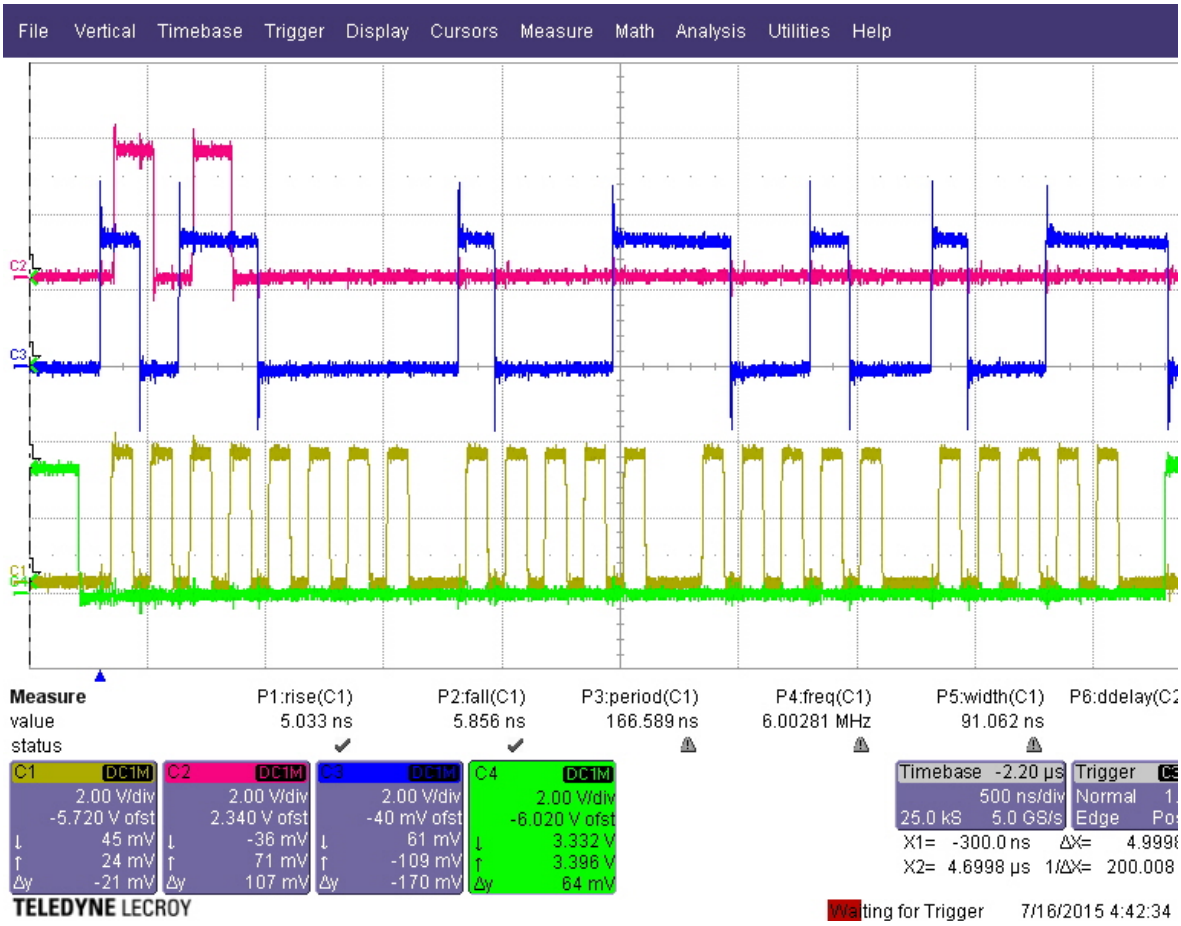


Figure 5-16 Four Word Transmission with 3-bit Address & 5-bit Data

5.4.3 User Port Timing

User port timing was tested for both the top (CPLD) and bottom (FPGA) boards' DB-62 connectors. The sample program sent commands to drive port bits to logical 0 and 1 as fast as possible in a tight loop using API function calls via the network interface. The update time was around 65 µs with the maximum update time encountered around 120 µs.

This was measured using a Windows XP SP3 system tuned for the shorter response time.

Since Windows is not a real-time system, it is feasible to wait for 1-2 ms on the IP stack upon receiving a reply message. Note that the cube and rack use real-time µC/OS and response time is fixed.

Figure 5-17 Output Register Writes on CPLD Board

User port 0 pin 19 on CPLD board is represented in **Figure 5-17**. Each edge is caused by calling the API function call to write an output register on the CPLD side.

The edges themselves look sharp and without overshoot/undershoot. The transition between states takes less then 40 ns to complete (**Figure 5-18**).

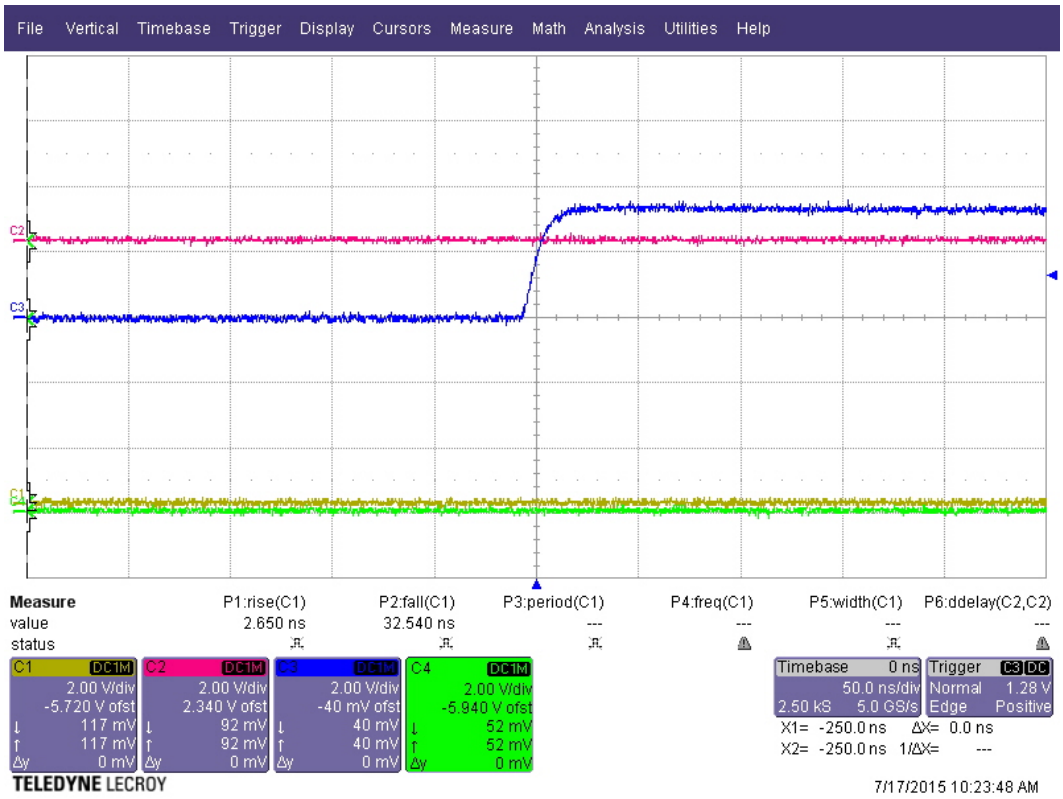


Figure 5-18 Rising Edge on User Port

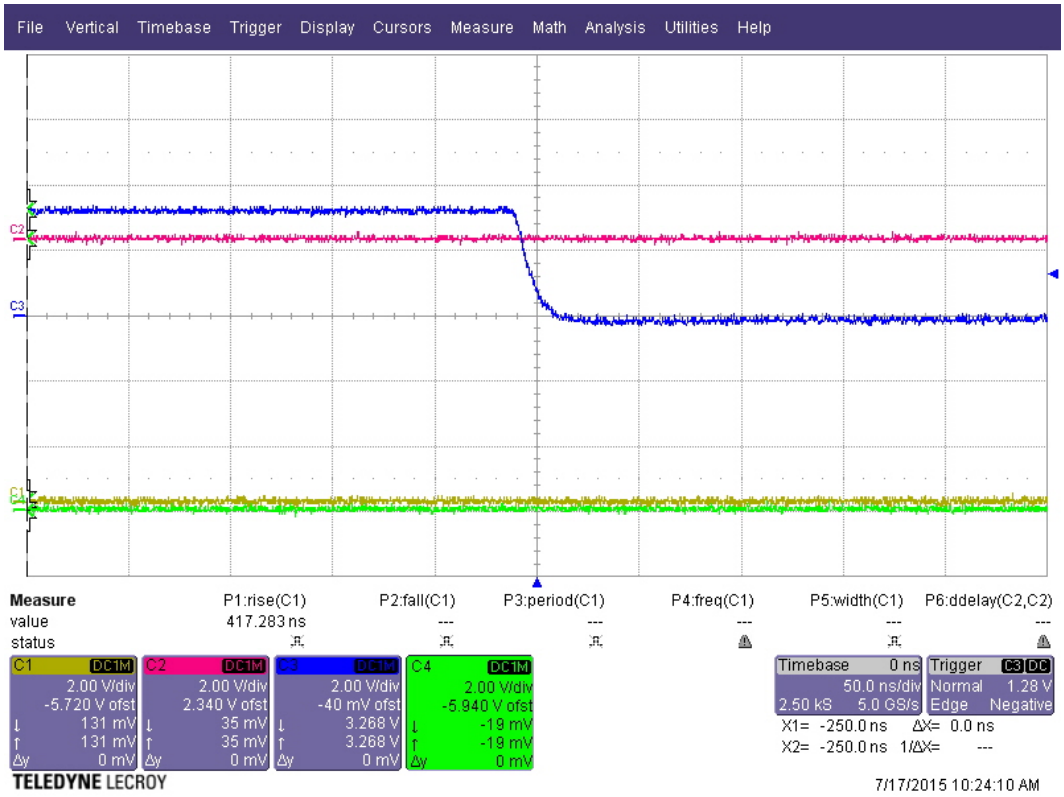


Figure 5-19 Falling Edge on User Port

Appendix A

A.1 Accessories

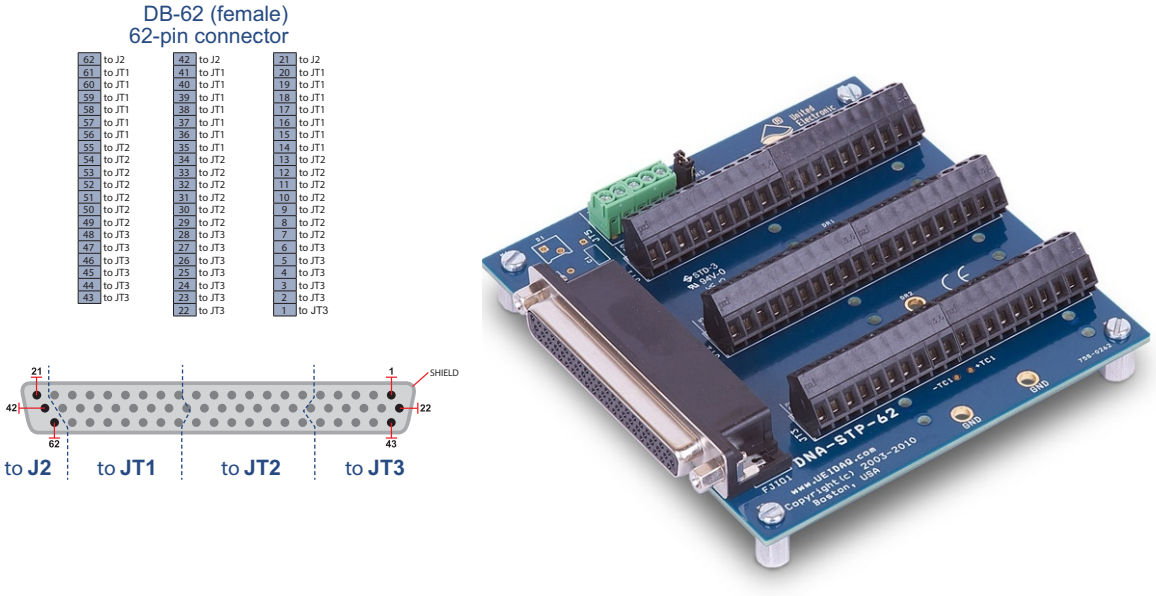
The following cables and STP boards are available for the PL-820 boards.

DNA-CBL-62

This is a 62-conductor round shielded cable with 62-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 2.5 ft (75 cm) long, weight of 9.49 ounces or 269 grams; up to 10ft (305cm) and 20ft (610cm).

DNA-STP-62

The STP-62 is a Screw Terminal Panel with three 20-position terminal blocks (JT1, JT2, and JT3) plus one 3-position terminal block (J2). The dimensions of the STP-62 board are 4w x 3.8d x1.2h inch or 10.2 x 9.7 x 3 cm (with standoffs). The weight of the STP-62 board is 3.89 ounces or 110 grams.



Pinout and Photo of DNA-STP-62 Screw Terminal Panel

Appendix B

PL-820 Memory Map & Register Descriptions

B.1 Memory Map Overview

The following sections provide tables of PL-820 memory mapped registers and their bit descriptions. For additional information, refer to register and constant definitions in powerdna.h or the PL-820 Logic Interface document.

All I/O boards in a UEI chassis are defined as a set of read (R), write (W), and R/W registers in a system memory space. R/W access to pre-defined registers is the only way to communicate with I/O boards. Note that the register address is relative to the base address of the I/O board as positioned in the chassis.

All I/O boards have a set of Common Logic Interface (CLI) registers that are standard to each board.

The PL-820 is also accessed via board-specific address space (0x2000-0x3FFC).

The following register descriptions are provided in this appendix:

- Section B.2: Standard CLI Registers (0x0 - 0xC)
- Section B.3: FPGA Base-Board Registers (0x2000-0x2FFC)
- Section B.4: CPLD/MAX10 Daughter-Board Registers(0x3000-0x3FFC)

NOTE: Note that a read/write to the CPLD address space (0x3000-0x31FC) is translated into 40-bit serial command/data stream that is transferred via UEI's synchronous serial interface to the CPLD. Read accesses also include 32-bit reply available in the board's ISDR (0x4); data delivered from the DNA bus during read from 0x3000-0x3FFC is undefined and should be discarded.

B.2 Standard CLI Registers

Standard Common Logic Interface registers are listed below. They provide configuration and status data and IS-logic access. Bit descriptions for each register are provided in the subsections that follow (Section B.2.1 thru Section B.2.6).

Register Address	Read Access	Write Access	Note
0x0	LSR Layer Status Register (Layers are I/O boards)	LCR Layer Control Register	Layer status/control registers contains pre-defined and layer-specific status and control bits For normal I/O operations, the register should be properly initialized.
0x4	ISDR/DINR Isolated side ISDR data RX register/data input register (DINR)	ISTR/DOUR Isolated side ISTR data TX register/data output register (DOUR)	ISDR/ISTR represents a window to the IS logic. Any write to ISTR will initialize a write cycle over the isolation interface. If any data is returned back, it will be available in the ISDR register. The data transfer format is layer-specific and defined in the accompanying section.
0x8	LID Layer Identification Register	NA	Layer identification, which defines type of layer, and build version, based on internal build of materials (BOM) #
0xC	NISV Non-isolated-side (NIS) logic version	NA	Returns CLI logic version. Logics should have the 0x0102xxxx revision to be compatible with this document

Table B-1 Standard CLI Register Descriptions

B.2.1 LSR Bit Description

0x0 RD – LSR – Layer status register: The Layer status register reports the current hardware status of the I/O board. This register should be used to define layer status and is in a state which allows execution of other commands.

Bit Number	Name	Description	Reset State
31	BSY	Layer busy – if set, the layer is busy executing previously issued command and will not respond to the current command. When the layer is busy, all other status bits are not valid. The only command which the layer will respond when it is busy is write to the **LRR** (layer reset register) which will initiate a reset for this layer only; however, the **LRE** (layer reset enable) bit should be previously set in the **LCR** (layer control register)	0
30	ILS	Interrupt line status. 0 represents no interrupt and 1 – interrupt has been requested	0
29	EDR	EEPROM data ready. 1 – Data is valid. Cleared after any write to the **ECR** EEPROM Control Register	0
28	EIB	EEPROM interface busy. When set to 1 indicates, that EEPROM state machine process EEPROM access operation.	0
27	IDR	Isolated side data ready. 1 – IS data is ready. This flag is set after data was successfully received from the IS-side logic and stored in the internal register. Bit is cleared by the any write to the ISTR.	0
26	IBT	Isolated interface busy transmitting data. When set to one indicates, that NIS → IS interface is busy processing IS access command previously issued.	0
25	IBR	Isolated interface busy receiving data. When set to one indicates, that NIS ← IS interface is busy processing IS access command previously issued.	0
24	LES	Layer error status. **Reserved**	0
23	EXT1	Current value of the ISO_EXT1 line	0
22	EXT0	Current value of the ISO_EXT0 line	0
21	INT1	Current value of the ISO_INT1 line	0
20	INT0	Current value of the ISO_INT0 line	0

Table B-2 LSR Register Bit Descriptions

Bit Number	Name	Description	Reset State
19-16	RSV	Reserved for the future use and should not be used by the individual layer's logic	0
15-0	AVL	Available for the layer-specific status bits	0

Table B-2 LSR Register Bit Descriptions (Cont.)

B.2.2 LCR Bit Description

0x0 WR – LCR – Layer control register: The Layer status/control register contains pre-defined (global) and layer-specific status and control bits.

Bit Number	Name	Description	Reset State
31	LIOE	Layer I/O enable – when one is written to this bit, I/O functionality of the layer become available. Registers available with LIOE set to 0 are: • Layer status/control LSR/LCR • Layer ID LID • Layer NIS logic version NISV • Scratchpad registers SPDx • SYNC registers • Interrupt registers • EEPROM status/control ESR/ECR • EEPROM read/write ERD/EWR	0
30	LRE	Layer software reset enable – if set to one write to LRR (0x7C) (Layer reset register will initiate reset sequence for the layer)	0
29	ICLE	Input channel list enable. When set – enables data transfers to the input channel list area from the **ISDR** (IS-side RX register, on some layers from other sources, see individual layer's descriptions for the details) .	0
28	OCLE	Output channel list enable. When set – enables data transfers from the output channel list area to the **ISTR** (IS-side TX register, or other local destination registers, layer-dependent, see individual layer's descriptions for the details).	0
27	ICCE	Input channel list continuous operation enable. Valid only when ICLE bit is set. If ICCE = 0 CLI logics wait for the channel list clock before restarting channel list.	0

Table B-3 LCR Register Bit Descriptions

Bit Number	Name	Description	Reset State
26	OCCE	Output channel list continuous operation enable. Valid only when OCLE bit is set. If OCCE = 0 CLO logics wait for the channel list clock before restarting channel list.	0
25	CLOM	If=1 – Mask writes to bits 31-16 of the output channel list data area (CDO) and output FIFO data register (OFDR). This feature may be used to program fixed part of the output channel list data area once and then constantly update output 16-bit data only upon request from the layer	0
24	ICLD	AVAILABLE for use only if STAT1(0x34) bit ICDE (1<<22) is set to 1 : Use iso_sdo1 line to disable iso_clk line. This bit work as follows if set to 1, then presence of the output iso_clk signal is defined by the iso_sdo1 line and by value, loaded into the **ICDR (0x54)** register. Falling edge on iso_sdo1 line will disable iso_clk line for at least the number of system clocks defined in bits [31:0] of ICLD. Clock is disabled/ enabled without any glitch during disable/enable transition. Clock will be re-enabled on next Input Channel List CL clock. If this bit is 0 iso_clk is always enabled and produces clock for the isolation interface	0
23	CLIF	Enable FIFO mode of the Input Channel list. Only one type of access to the channel list should be used : FIFO (CLIF=1) or Memory mapped (CLIF=0)	0
22	CLOF	Enable FIFO mode of the Output Channel list. Only one type of access to the channel list should be used : FIFO (CLOF=1) or Memory mapped (CLOF=0)	0
21	CLOR	Enable Output CL regenerate mode (only if CLOF=1)	0
20	RST	Reset input and output channel list engines and corresponding FIFOs	0

Table B-3 LCR Register Bit Descriptions (Cont.)

Bit Number	Name	Description	Reset State
19	TAE	Enable /TA line output. If enabled, TA line will be asserted every time when transaction on the DNA data bus is completed from the layer's standpoint. /TA line may be used in conjunction with DNA-PPC-CPU layers when different access delays are required for the different layers on the same chip select line	0
18	DMARD_EN	Enable read DMA operation (if available, check STAT1 register): Slave layers : B_A0 functions as DMAREQ# and B_A1 as DMARD#, DMA read FIFO is enabled Master layers: in addition to B_A0/B_A1 functionality all accesses from PPC via CS1 translated into DMARD# regardless of the actual read/write operation; DMAREQ# assertion translated to PPCC IRQ3 and DREQ0 (MPC8347 only)	0
17	BLK_TSTD	Only from logics 0x010210F6 =1 - block all writes (including broadcast) to TSTD, can be used if timestamp should continue to count on the selected layer even when all other timestamps are reset; this option is used for example by the DNR-MIL backplane logic to support hour counter	0
16-8	RSV	Reserved for the future use and should not be used by the individual layer's logic	0
7-2	AVL	Available for the layer-specific control bits	0
1	LCR61x _LED	Enable (turn-on)/ Disable (turn-off) status LED. 1 in this bit turn on status LED.	0
0	LCR61x_DCDIS	DC/DC control on 61x (1=disable)	0

Table B-3 LCR Register Bit Descriptions (Cont.)

B.2.3 ISDR/DINR Bit Description

0x4 RD – ISDR isolated side data RX register/DINR data input register:

NOTE: Access to the IS- logic is done using ISDR/ISTR registers. Note, that LIOE bit (1<<31) in the LCR register (0x0) should be set to 1 prior to communicating with IS- logic.

ISDR/ISTR represents a window to the IS logic. Access to this logic is initiated by the write to the **ISTR** register and may require a substantial amount of time (37 isolation clocks/write). This access should be either interrupt-driven or based on the status bits (IBT and IDR bits) in the LSR (0x0). The data format for the ISTR and ISDR is layer-specific and data word should be created via the combination of the provided with drvXX.h layer-specific firmware constants.

When I/O operation, initialized via write to ISTR register requires some data to be delivered back to the NIS logic, this data is accessible via ISDR register.

B.2.4 ISTR/DOUR Description

0x4 WR – ISTR isolated side data TX register/DOUR data output register:

Every time that an operation requires access to the IS logic, this access is performed via a write to ISTR. Because each layer has unique functionality for its IS logic, data which is written to this register should include a combination of the layer-specific constants.

There are two types of the I/O operations that may be initialized by writing data to ISTR:

- write-only, when data is written to IS logic and there are no return value expected
- write-and-read, when data is written to IS-logic and IS-logic responds with result, result will be placed into ISTR register

Data is transmitted over the NIS <> IS isolation barrier is always in 32-bit DWORD; however, the format of the data transmitted is layer-dependent.

B.2.5 LID Bit Description

0x8 RD – LID – Layer identification register: Layer identification register identifies the type of the layer and the build version. The build version reflects internal BOM revision # used to build this particular layer. This information is hardwired and cannot be updated from the firmware.

Bit Number	Name	Description	Reset State
31-16	LIDV	BOM revision #	Hardwired
0-15	LIDM	Layer identification #	Hardwired

Table B-4 LSR Register Bit Descriptions

B.2.6 NISV Bit Descriptions

0xC RD – NISV – NIS Logic version:

The NIS logic version is available via this register and may be used for the verification purposes.

The NISV register should be equal 0x010201xx to comply with this document.

.

Bit Number	Name	Description	Reset State
31-24	NISL	NIS Logic level	1
23-16	NISV	NIS Logic version, major	Hardwired
15-8	NISM	NIS Logic version, minor	Hardwired
7-0	NISD	NIS Logic build	Hardwired

Table B-5 LSR Register Bit Descriptions

B.3 FPGA Base-Board Registers

The FPGA base-board interfaces between the CPLD daughter-board and the host application. The following table lists data and control registers for the FPGA base-board. Bit descriptions for each register are provided in the subsections that follow (Section B.3.1 thru Section B.3.10).

Register Address	Read Access	Write Access	Note	Reset State
0x2000	PL820_CFG General configuration register	PL820_CFG General configuration register	General “global” configuration register	0x0
0x2004	PL820_STS General status register	NA	Current status of the board, “sticky” bits are auto-cleared after each read	0x0
NOTE: PLL configuration registers are reserved for the future version of the FPGA base layers with user re-programmable PLLs. Current PLL implementation generates fixed 32 MHz output frequency				
0x2020 0x2024	PL820_PLLx_BSTS PLL0/1 status register	PL820_PLLx_BCFG PLL0/1 configuration register	PLL configuration/ status register, allows custom precise clock rate creation by re-programming FPGA’s PLL. PLL0/1 outputs are internally connected to iso_ext0/1 which can be routed to the SYNC bus or used as a channel list clock	0
0x2028 0x202C	NA	PL820_PLLx_DIV PLL Post-Divider (SPIx 2x reference clock)	Post-divider for PLL0 Post-divider for PLL1 Output of post-divider drives corresponding SPI clock input. Actual SPI clock = (PLL_clock /PLL_DIV+1))/2	0x0

Table B-6 FPGA Register Descriptions

Register Address	Read Access	Write Access	Note	Reset State
0x2040 0x2044	NA	PL820_NISMD0 PL820_NISMD1 NIS pins mode register	NIS IO31-0 mode NIS IO63-32 mode (0-default, 1-test; NIS-IO 2,4,8,12,16,39 are excluded and should be written with 0)	0x0
0x2048 0x204C	NA	PL820_NISDIR0 PL820_NISDIR1 NIS pins direction register	NIS IO31-0 direction NIS IO63-32 direction Set direction for the corresponding pin in test mode (1-output)	0x0
0x2050 0x2054	NA	PL820_NISOUT0 PL820_NISOUT1 NIS pins output register	NIS IO31-0 output value NIS IO63-32 output value Set output value of the corresponding pin when test mode is enabled and direction is output; read returns last written value (1-output)	0x0
0x2058 0x205C	PL820_NISIN0 PL820_NISIN1 NIS pins input register	NA	NIS IO31-0 current state NIS IO63-32 current state	0x0
0x2100 0x2180	NA	PL820_SPI0_CFG PL820_SPI1_CFG Master SPI configuration register	SPI0/1 configuration register. Set number of bits and other SPI parameters, SPI module should be reset every time when parameters are changed	0x0

Table B-6 FPGA Register Descriptions (Cont.)

Register Address	Read Access	Write Access	Note	Reset State
0x2104 0x2180	PL820_SPI0_STS PL820_SPI1_STS Master SPI Status register	NA	SPI0/1 status register. Report current FIFO levels and status of the SPI interface	0x0
0x210C 0x218C	NA	PL820_SPI0_FWM PL820_SPI1_ FWM SPI FIFO watermark register	SPI0/1 watermark register. Watermark registers are used to set FIFO half-full interrupt levels	0x0
0x2110 0x2190	NA	PL820_SPI0_OFDT0 PL820_SPI1_ OFDT0 SPI Output FIFO LSBs register	SPI0/1 Output FIFO LSB (31-0) data register. Write LSBs to the output FIFO, advances FIFO counter	0x0
0x2114 0x2194	NA	PL820_SPI0_OFDT1 PL820_SPI1_ OFDT1 SPI Output FIFO MSBs register	SPI0/1 Output FIFO MSB (63-32) data register. Write MSBs to the output FIFO, only needed for the transfers over 32-bits	0x0
0x2118 0x2198	NA	PL820_SPI0_OFDT0E PL820_SPI1_ OFDT1E SPI Output FIFO LSBs + EOF register	SPI0/1 Output FIFO LSB (31-0) last data word register. Write LSBs to the output FIFO, advances FIFO counter, sets EOF flag for multi-word transmission	0x0
0x2120 0x21A0	PL820_SPI0_IFDT0 PL820_SPI1_ IFDT0 SPI Input FIFO LSBs register	NA	SPI0/1 Input FIFO MSBs (31-0) data register. Read LSBs to the input FIFO, advances FIFO counter	0x0

Table B-6 FPGA Register Descriptions (Cont.)

Register Address	Read Access	Write Access	Note	Reset State
0x2124 0x21A4	PL820_SPI0_IFDT1 PL820_SPI1_ IFDT1 SPI Input FIFO MSBs register		SPI0/1 Input FIFO MSB (63-32) data register. Read MSBs to the input FIFO, only needed for the transfers over 32-bits	0x0

Table B-6 FPGA Register Descriptions (Cont.)

B.3.1 PL820_CFG Bit Descriptions

0x2000 + RD/WR – PL820_CFG – General configuration register

This general configuration register is used to enable/disable major subsystems on PI-820.

Bit Number	Name	Description	Reset State
31-3	RSV	Reserved	0
2	SPI1R	=1 - reset master SPI1. SPI should be reset every time the SPI configuration is changed	0
1	SPI0R	=1 - reset master SPI0. SPI should be reset every time the SPI configuration is changed	0
0	CPLDR	=1 - reset CPLD (assert UEI_RESET_N)	0

Table B-7 PL820_CFG Register Bit Descriptions

B.3.2 PL820_STS Bit Descriptions

0x2004 + RD/WR – PL820_STS – General status register

This register reports the status of the PL-820. Sticky bits reflect accumulated status and cleared after each read..

Bit Number	Name	Description	Reset State
31-24	RSV	Reserved	0
Sticky (latched) status bits, autocleared after a read:			
23	SPI1TXFES	=1 - SPI1 TX FIFO was/is empty	0
22	SPI1TXHFS	=1 - SPI1 TX FIFO was/is below watermark	0
21	SPI1RXFFS	=1 - SPI1 RX FIFO was/is full	0
20	SPI1RXHFS	=1 - SPI1 RX FIFO was/is above watermark	0
19	SPI0TXFES	=1 - SPI0 TX FIFO was/is empty	0
18	SPI0TXHFS	=1 - SPI0 TX FIFO was/is below watermark	0
17	SPI0RXFFS	=1 - SPI0 RX FIFO was/is full	0
16	SPI0RXHFS	=1 - SPI0 RX FIFO was/is above watermark	0
Current (static) status bits:			
15	SPI1TXFE	=1 - SPI1 TX FIFO is empty	0
14	SPI1TXHF	=1 - SPI1 TX FIFO is below watermark	0
13	SPI1TXFF	=1 - SPI1 TX FIFO is full	0
12	SPI1RXFE	=1 - SPI1 RX FIFO is empty	0
11	SPI1RXHF	=1 - SPI1 RX FIFO is above watermark	0
10	SPI1RXFF	=1 - SPI1 RX FIFO is full	0
9	SPI0TXFE	=1 - SPI0 TX FIFO is empty	0
8	SPI0TXHF	=1 - SPI0 TX FIFO is below watermark	0
7	SPI0TXFF	=1 - SPI0 TX FIFO is full	0
6	SPI0RXFE	=1 - SPI0 RX FIFO is empty	0
5	SPI0RXHF	=1 - SPI0 RX FIFO is above watermark	0
4	SPI0RXFF	=1 - SPI0 RX FIFO is full	0
3	SPI1BSY	=1 - SPI1 busy	0
2	SPI0BSY	=1 - SPI0 busy	0
1	ISRDBSY	=1 - CPLD->FPGA interface busy	0
0	ISWRBSY	=1 - FPGA->CPLD interface busy	0

Table B-8 PL820_STS Register Bit Descriptions

B.3.3 PL820_PLL0/1_BCFG & _BSTS Bit Descriptions

0x2020/2024 + RD/WR – PL820_PLL0/1_BCFG/BSTS PLL Access/Status registers

NOTE: Programmable PLL functionality is reserved for the future revisions of the 61xR FPGA base layer. Current implementation uses a 24 MHz standard non-reprogrammable PLL.

The _BSTS register is used report the status of the PLL baud. Note that the BSTS_LOCKED bit should be set to 1 during the normal PLL operation and reading it as zero for more than 100 mS after the PLL reset indicates a hardware error.

Bit Number	Name	Description	Reset State
31-2	RSV	Reserved	0
1	LOCKED	=1 - PLL is locked (normal operation)	0
0	RSV	Reserved	0

Table B-9 PL820_PLLx_BCFG/PL820_PLLx_BSTS Register Bit Descriptions

B.3.4 PL820_NISx NIS Register Descriptions and Configurable IO pins

The FPGA on the base layer interfaces to the CPLD on the attachment daughter-board via 64 I/O lines distributed over two 40-pin connectors, NIS1 and NIS2. The additional 16 connections over the NIS connectors are powers and grounds.

Most of the lines are fully programmable and can be assigned to be either an input or output, thus allowing a full connectivity test for almost every line.

Following I/Os have fixed assignments that can't be reprogrammed:

- IO2: UEI-RESET-N
- IO4: CLK-1KHZ (used as the SSI clock)
- IO8: UEI-SDO (SSI serial data out from FPGA to CPLD)
- IO12: UEI-SDI (SSI serial data in from CPLD to FPGA)
- IO16: UEI-IRQ
- IO39: CLK-33MHZ

User-configurable NIS I/O lines can be in one of three states:

1. Default state, pre-defined for the following I/O pins, all pins not listed above or below are inputs by default:
   - IO62 SPI_MOSI0 (FPGA input/CPLD output)
   - IO51 SPI_MISO0 (FPGA output/CPLD input)
   - IO47 SPI_SCLK0 (FPGA output/CPLD input)
   - IO50 SPI_CS0 (FPGA output/CPLD input)

   - IO20 SPI_MOSI1 (FPGA input/CPLD output)
   - IO9 SPI_MISO1 (FPGA output/CPLD input)
   - IO11 SPI_SCLK1 (FPGA output/CPLD input)
   - IO15 SPI_CS1 (FPGA output/CPLD input)

In default state pin functionality is pre-defined as specified above.

2. Test state, input
3. Test state, output

Register	Description
PL820xx_NISMDx	used to set a mode of the operation of the internal NIS I/O pins and allowing switching between default and test modes (setting the corresponding bit to 1 switches pin to test mode).
PL820xx_NISDIRx	In test mode, PL820xx_NISDIRx is used to set a direction of each internal pin, and when the pin is configured as an output in this register, PL820xx_NISOUTx sets the value.
PL820xx_NISOUTx	sets the value of the internal output pin (for pins configured as outputs in PL820xx_NISDIRx register).
PL820xx_NISINx	report status (current value) of each internal pin regardless of the mode.

Table B-10 NIS Register Descriptions

B.3.5 Master/Slave SPI Interface Register Descriptions

The FPGA provides two master SPI interfaces that can be configured for the 4-64 bit command/data transfers over SPI bus. The CPLD has two matching slave SPI modules.

The master SPI on the FPGA side uses 24 MHz PLL-generated clock as a reference clock source for the SPI data transfers with a post-divider that reduces SPI clock frequency; 4 MHz is a nominal maximum value for the SPI clock. The master SPI module internally runs from the main clock (66 MHz) and all SPI I/O signals are synchronized with the main clock.

The SPI has configurable polarity of the chip select and edges when data should be read and written.

Two main modes of the operation are supported – single word and multi-word transfers.

- In single word mode each SPI transaction starts from the address/command portion followed by the data bits within the same chip select cycle.
- Multi-word transmission transmits address/command once with multiple data words to follow.

On the master side SPI, TX/RX data is accessed via FIFOs: if the full SPI word is over 32 bits, MSBs should be written to the `PL820_SPIx_OFDT1` register followed by LSBs written to `PL820_SPIx_OFDT0` register. For multi-word transfers, the last LSB data word should be written to the `PL820_SPIx_OFDT0E` register indicating an EOF condition.

Each TX data word from the master SPI results in an RX data word that is stored into the RX FIFO. The FIFO should be read MSB word first (`PL820_SPIx_IFDT1`) followed by LSB word (`PL820_SPIx_IFDT0`).

The slave SPI on the CPLD side is implemented similarly to the master SPI; however, due to the fact that the slave SPI has the address/command part that should be used to select RX/TX data, registers are used for the incoming and outgoing data along with dedicated address/command register. Also sample implementation of eight 32-bit registers for the multi-word transfer is provided for future expansion by the customer.

Sample timing diagrams are provided in **Chapter 5** and in the PL-820 Logic Interface document.

B.3.6 PL820_SPI0/1_CFG Configuration Register Descriptions

0x2100/2180 + WR – PL820_SPI0/1_CFG Configuration registers

The FPGA provides two master SPI interfaces that can be configured for the 4-64 bit command/data transfers over SPI bus. The CPLD has two matching slave SPI modules.

SPI configuration registers on a base layer are used to set the desired Master SPI mode. Master SPI can be reconfigured at any time; however, after reconfiguration SPI reset should be issued by toggling corresponding `PL820_CFG_SPIxR` bit.

Bit Number	Name	Description	Reset State
31	CSI	=1 - Inverted SCS	0
30	RSV30	Reserved	0
29-24	ABE	Address bit MSB#, set to "`PL820_SPI_CFG_DBE`" if address is unused	0
23	PSR	=1 - receive data is valid on rising edge of the clock	0
22	RSV22	Reserved	0
21-16	ABS	Address bit LSB#, set to 0 if address is unused	0

Table B-11 PL820_SPIx_CFG Register Bit Descriptions

Bit Number	Name	Description	Reset State
15	PSW	=1 - transmit data is valid on rising edge of the clock	0
14	RSV14	Reserved	0
13-8	DBE	MSB data bit# in the SPI word	0
7	MWE	=1 multi-word mode. In multi-word mode address/command part of the SPI word transmitted once followed by multiple data words	0
6	RSV6	Reserved	0
5-0	DBS	LSB data bit# in the SPI word	0

Table B-11 PL820_SPIx_CFG Register Bit Descriptions (Cont.)

B.3.7 PL820_SPI0/1_STS Status Register Descriptions

0x2104/2184 + RD – PL820_SPI0/1_STS Status registers

These registers report the current FIFO levels and status of the SPI interface.

Bit Number	Name	Description	Reset State
Sticky (latched) status bits, auto-cleared after read			
31	TXFE	=1 if TX FIFO was ever empty since the last read	0
30	RXFF	=1 if RX FIFO was ever full since the last read	0
29-27	RSV29_27	Reserved	0
Current (static) status bits			
26-16	TXFUW	# of occupied words in the TX FIFO	0
15	SPIB	=1 if SPI transaction is in progress	0
14-11	RSV14_11	Reserved	0
10-0	RXFUW	# of occupied words in the RX FIFO	0

Table B-12 PL820_SPIx_STS Register Bit Descriptions

B.3.8 PL820_SPI0/1_FWM Watermark Register Descriptions

0x210C/218C + WR – PL820_SPI0/1_FWM FIFO Watermark registers

The watermark registers are used to set FIFO half-full interrupt levels.

Bit Number	Name	Description	Reset State
31-27	RSV31_27	Reserved	0
26-16	TXFW	TX FIFO watermark level	0
15-11	RSV15_11	Reserved	0
10-0	RXFW	RX FIFO watermark level	0

Table B-13 PL820_SPIx_FWM Register Bit Descriptions

B.3.9 PL820_SPI0/1_OFDTx[E] Output Registers Descriptions

0x2110/2190, 0x2114/2194, 0x2118/2198 + WR – PL820_SPI0/1_OFDTx[E] Output FIFO data registers

All registers are 32-bits wide. Up to 64-bits of the output data are split into two registers: `PL820_SPIx_OFDT1` and (`PL820_SPIx_OFDT0` or `PL820_SPIx_OFDT0E`).

- For an SPI word consisting of more than 32 bits, a write to `PL820_SPIx_OFDT1` should precede a write to `PL820_SPIx_OFDT0` or `PL820_SPIx_OFDT0E`.
- A write to `PL820_SPIx_OFDT0E` instead of `PL820_SPIx_OFDT0` indicates end of the frame for a multi-word transaction.
- A write to `PL820_SPIx_OFDT0[E]` advances TX FIFO.

B.3.10 PL820_SPI0/1_IFDTx Input Registers Descriptions

0x2120/21A0, 0x2124/21A4 + RD – PL820_SPI0/1_IFDTx Input FIFO data registers

All registers are 32-bits wide. Up to 64-bits of the input data is available via two registers: `PL820_SPIx_IFDT1` and `PL820_SPIx_IFDT0`.

- For SPI words consisting of more than 32 bits, a read from `PL820_SPIx_IFDT1` should precede read from `PL820_SPIx_IFDT0`.
- A read from `PL820_SPIx_IFDT0` advances RX FIFO.

B.4 CPLD/MAX10 Daughter-Board Registers

The CPLD daughter-board houses the user-customizable MAX10 CPLD chip.

The user host application can only access CPLD registers through the FPGA base-board, which uses the internal NIS interface pins NIS-IO2, 4, 8, 12, 16 and 39 in serial mode. Access can be interrupt-driven or polling (based on status bits in LSR register, LSR_IBT bit is set during the transmission of the data from FPGA to CPLD and LSR_IDR bit is set when read from CPLD is completed and data is ready in CLI_ISDR register).

The CPLD address space is 0x3000 through 0x3FFC.

The internal format for the UEI serial interface is as follows:

- Both FPGA-->CPLD and CPLD-->FPGA transmissions start with two start bits.
- 40-bit FPGA-->CPLD transfers:
  - the MSB is a write bit (=1 - write to the register)
  - following 7 bits are the address of the register in the CPLD, shifted 2 bits to the right and ANDed with 16'h7F
  - last 32 bits are data

Bit 39	Bits 38:32 (mapped to 0x3000 through 0x31FC Address Space)	Bits 31:0
=1 – write CPLD register, do not return any data =0 – read CPLD register, return data	CPLD register address DNA address = 0x3000 | (CPLD address << 2)	Data (ignored for the read operation)

Table B-14 FPGA Register Descriptions

- CPLD-->FPGA returns back 32 bits of data.

The table on the next page lists CPLD data and control registers. Bit descriptions for each register are provided in the subsections that follow (Section B.4.1 thru Section B.4.11).

.

Register Address	Read Access	Write Access	Note	Reset State
0x3000	PL820IS_CFG General configuration register	PL820IS_CFG General configuration register	General CPLD configuration register	0x0
0x3004	PL820IS_STS General status register	NA	General CPLD status register	0x0
0x3008	PL820IS_ID Attachment ID register	NA	Return 0x000vA820 where v – PCB revision	0x0
0x3010	NA	PL820IS_IER CPLD Interrupt enable register	CPLD Interrupt enable register	0x0
0x3014	PL820IS_ISR CPLD Interrupt status register	NA	CPLD Interrupt status register	0x0
0x3018	NA	PL820IS_ICR CPLD Interrupt clear register	CPLD Interrupt clear register	0x0
0x3030 0x3034	NA	PL820IS_NISMD0 PL820IS_NISMD1 CPLD NIS pins mode register	NIS IO31-0 mode NIS IO63-32 mode (0-default, 1-test; NIS-IO 2,4,8,12,16,39 are excluded and should be written with 0)	0x0
0x3038 0x303C	NA	PL820IS_NISDIR0 PL820IS_NISDIR1 CPLD NIS pins direction register	NIS IO31-0 direction NIS IO63-32 direction Set direction for the corresponding pin in test mode (1-output)	0x0

Table B-15 CPLD Register Descriptions

Register Address	Read Access	Write Access	Note	Reset State
0x3040 0x3044	NA	PL820IS_NISOUT0 PL820IS_NISOUT1 CPLD NIS pins output register	NIS IO31-0 output value NIS IO63-32 output value Set output value of the corresponding pin when test mode is enabled and direction is output; read returns last written value (1-output)	0x0
0x3048 0x304C	PL820IS_NISIN0 PL820IS_NISIN1 CPLD NIS pins input register	NA	NIS IO31-0 current state NIS IO63-32 current state	0x0
0x3050 0x3054 0x3058 0x305C 0x3060	NA	PL820IS_ADC0 PL820IS_ADC1 PL820IS_ADC2 PL820IS_ADC3 PL820IS_ADCTC	ADC access is reserved for future revisions. ADC Channel 0-4 data (3+16LSB valid)	0x0
0x3080 0x3084 0x3088 0x308C	NA	PL820IS_USRDIR0 PL820IS_USRDIR1 PL820IS_USRDIR2 PL820IS_USRDIR3 CPLD user I/O pins direction register	USER IO 31-0 direction USER IO 50-32 USER IO 82-51 direction USER IO 104-83 direction (1-output)	0x0

Table B-15 CPLD Register Descriptions (Cont.)

Register Address	Read Access	Write Access	Note	Reset State
0x3090 0x3094 0x3098 0x309C	NA	PL820IS_USROUT0 PL820IS_USROUT1 PL820IS_USROUT2 PL820IS_USROUT3 CPLD user I/O pins output register	USER IO 31-0 value USER IO 50-32 value USER IO 82-51 value USER IO 104-83 value Set output value of the corresponding pin when direction is output; read returns last written value (1-output)	0x0
0x30A0 0x30A4 0x30A8 0x30AC	PL820IS_USRIN0 PL820IS_USRIN1 PL820IS_USRIN2 PL820IS_USRIN3 CPLD user I/O pins input register	NA	USER IO 31-0 current state USER IO 50-32 current state USER IO 82-51 current state USER IO 104-83 current state	0x0
0x3100 0x3180	NA	PL820IS_SPI0_CFG PL820IS_SPI1_CFG Slave SPI configuration register	SPI0/1 configuration register. Set number of bits and other SPI parameters, SPI module should be reset every time when parameters are changed	0x0
0x3104 0x3180	PL820_SPI0_STS PL820_SPI1_STS Slave SPI Status register	NA	SPI0/1 status register. Reserved for future use	0x0

Table B-15 CPLD Register Descriptions (Cont.)

Register Address	Read Access	Write Access	Note	Reset State
0x3108 0x3188	NA	PL820IS_SPI0_ODT0 PL820IS_SPI1_ ODT0 Slave SPI LSB output data register	SPI0/1 LSBs (31-0)/ MSBs(63-32) output data register – should be pre-loaded with the value that will be replied via SPI during the next transaction. In multi-word mode LSBs can be replaced with value taken from the SPI register pool	0x0
0x310C 0x318C	NA	PL820IS_SPI0_ODT1 PL820IS_SPI1_ ODT1 Slave SPI MSB output data register		0x0
0x3110 0x3190	PL820IS_SPI0_IDT0 PL820IS_SPI1_ IDT0 Slave SPI LSB input data register	NA	SPI0/1 LSBs (31-0)/ MSBs (63-32) input data register, data received from SPI is stored in these registers. In multi-word mode LSBs can be stored to the SPI register pool	0x0
0x3114 0x3194	PL820IS_SPI0_IDT1 PL820IS_SPI1_ IDT1 Slave SPI MSB input data register	NA		0x0
0x3118 0x3198	PL820IS_SPI0_IADDR PL820IS_SPI1_ IADDR Slave SPI input address register	NA	SPI0/1 input address register, address/ command portion of the incoming SPI transfer stored to this register	0x0
0x311C 0x319C	NA	PL820IS_SPI0_ACFG PL820IS_SPI1_ACFG Slave SPI address configuration register	SPI0/1 address configuration register – used as a part of sample code that shows how to process up to 32-bit (data)/8bit (address/command) multiple data word SPI transmissions	0x0
0x3120- 0x313C 0x31A0- 0x31BC	PL820IS_SPI0_REGx PL820IS_SPI1_REGx Slave SPI register pool	PL820IS_SPI0_REGx PL820IS_SPI1_REGx Slave SPI register pool	Register pool is used as a part of sample code that allows reading or writing multiple SPI words (up to 32 bits data).	0x0

Table B-15 CPLD Register Descriptions (Cont.)

B.4.1 PL820IS_CFG Configuration Register Description

0x3000 + RD/WR – PL820IS_CFG CPLD configuration register

General configuration register used to enable/disable major subsystems on the PL-820 attachment daughter-board.

Bit Number	Name	Description	Reset State
31-4	RSV	Reserved	0
3	ADCR	=1 - reset ADC(reserved)	0
2	SPI1R	=1 - reset slave SPI1. SPI should be reset every time when SPI configuration is changed	0
1	SPI0R	=1 - reset slave SPI0. SPI should be reset every time when SPI configuration is changed	0
0	LED	=1 - turn ON user LED on the attachment	0

Table B-16 PL820IS_CFG CPLD Configuration Register Bit Descriptions

B.4.2 PL820IS_STS Status Register Description

0x3004 + RD – PL820IS_STS CPLD status register

General status register for the CPLD attachment daughter-board.

Bit Number	Name	Description	Reset State
31-19	RSV31_19	Reserved	0
Sticky (latched) status bits, auto-cleared after read			
18	ADCDR	1 - data ready from ADC (reserved)	0
17	SPI1DR	1 - data ready from SPI1	0
16	SPI0DR	1 - data ready from SPI0	0
Current (static) status bits			
15-10	RSV15_10	Reserved	0
9	SPI1BSY	=1 - SPI1 busy	0
8	SPI0BSY	=1 - SPI0 busy	0
7-0	LVER	Report logic version	0

Table B-17 PL820IS_STS CPLD Status Register Bit Descriptions

B.4.3 PL820IS_ADCx ADC Registers Description

0x3050-0x3060 + RD – PL820IS_ADCx CPLD ADC data register

Reserved: refer to the PL-820 Logic Interface document.

B.4.4 PL820IS_USRx User Configurable IO Pins (JI01-TOP/ J101-BOT)

0x3080-0x308C + WR – PL820IS_USRDIRx CPLD User I/O pins direction register

0x3090-0x309C + WR – PL820IS_USROUTx CPLD User I/O pins output register

0x30A0-0x30AC + RD – PL820IS_USRINx CPLD ADC User I/O pins input register

The CPLD on the daughter-board attachment provides an interface to two DB-62F connectors, JIO1-TOP and JIO1-BOT.

All I/O lines are fully programmable and can be assigned to be ether input or output.

- `PL820IS_USRDIRx` registers are used to set the direction of each pin and when the pin is selected as an output.
- `PL820IS_USROUTx` is used to set value of the output pin.
- `PL820IS_USRINx` register report status (current value) of each pin regardless of the mode.

NOTE: All `PL802IS_USRx` registers above are 32-bit wide; however, only bits 8:0 are used for xx_USRxx3 registers.

B.4.5 PL820IS _SPI0/1_CFG Configuration Register Descriptions

0x3100/3180 + WR – PL820IS_SPI0/1_CFG CPLD SPI Configuration registers

SPI configuration registers on the CPLD attachment are used to set up the desired Slave SPI mode. Slave SPI can be reconfigured at any time; however, after reconfiguration the SPI reset should be issued by toggling corresponding `PL820IS_CFG_SPIxR` bit.

The CPLD has two matching slave SPI modules corresponding to the FPGA's two master SPI interfaces, which can be configured for the 4-64 bit command/ data transfers over SPI bus.

Bit Number	Name	Description	Reset State
31	CSI	=1 - Inverted SCS	0
30	RSV30	Reserved	0
29-24	ABE	Address bit MSB#, set to "PL820_SPI_CFG_DBE" if address is unused	0
23	PSR	=1 - receive data is valid on rising edge of the clock	0
22	RSV22	Reserved	0
21-16	ABS	Address bit LSB#, set to 0 if address is unused	0

Table B-18 PL820IS_SPIx_CFG CPLD Register Bit Descriptions

Bit Number	Name	Description	Reset State
15	PSW	=1 - transmit data is valid on rising edge of the clock	0
14	RSV14	Reserved	0
13-8	DBE	MSB data bit# in the SPI word	0
7	MWE	=1 multi-word mode. In multi-word mode address/command part of the SPI word transmitted once followed by multiple data words. Internally SPI target address is auto-incremented every time once RX SPI data is latched; first time address is incremented after initial address is latched	0
6	RSV6	Reserved	0
5-0	DBS	LSB data bit# in the SPI word	0

Table B-18 PL820IS_SPIx_CFG CPLD Register Bit Descriptions (Cont.)

B.4.6 PL820_SPI0/1_STS Status Register Descriptions

0x3104/3184 + RD – PL820_SPI0/1_STS CPLD Status registers

Reserved for future use.

B.4.7 PL820_SPIx_ODTx SPI Slave Data Output Register Descriptions

0x3108/3188, 0x310C/318C + WR – PL820_SPIx_ODTx SPI slave output data registers

Slave SPI returns data every time in reply to the master SPI transmission.

These registers are used to pre-load data that will be returned to the master SPI in single word mode. In multi-word mode, the SPI register pool should be used instead.

Bit Number	Name	Description	Reset State
31-0	SPI_DOUT	xx_ODT0 registers are used for the bits 31-0 xx_ODT1 registers are used for the bits 63-32	0

Table B-19 PL820_SPIx_Slave Data Output Register Bit Descriptions

B.4.8 PL820_SPIx_IDTx SPI Slave Data Input Register Descriptions

0x3110/3190, 0x3114/3194 + RD – PL820_SPIx_IDTx SPI slave input data registers

Single-word SPI data is processed based on the settings in SPI_CFG register and split into two parts: the address/command part and data part.

Data is saved into the xx_SPIx_IDTx registers.

Bit Number	Name	Description	Reset State
31-0	SPI_DIN	xx_IDT0 registers are used for the bits 31-0 xx_IDT1 registers are used for the bits 63-32	0

Table B-20 PL820_SPIx_IDTx SPI Slave Data Input Register Bit Descriptions

B.4.9 PL820_SPIx_IADDR SPI Slave Input Address Register Descriptions

0x3118/3198 + RD – PL820_SPIx_IADDR SPI slave input address registers

Single-word SPI data is processed based on the settings in SPI_CFG register and split into two parts: address/command part and data part.

Addresses are saved into the xx_SPIx_IADDR registers.

Bit Number	Name	Description	Reset State
31-0	SPI_AIN	Up to 32 address/command bits are supported	0

Table B-21 PL820_SPIx_IADDR SPI Slave Address Input Register Bit Descriptions

B.4.10 PL820_SPIx_ACFG SPI Slave Address Configuration Register Descriptions

0x311C/319C + WR – PL820_SPIx_ACFG SPI slave address configuration registers

Slave SPI allows sequential read and write of multiple words.

As an example, we can access eight 32-bit registers this way. The SPI address/command field should be at least 8-bit wide and 8 LSBs will be used to determine whether the incoming read or write register address matches the address of the register pool access. If the SPI is configured with a data word size greater than 32 bits upper bits, it would be truncated, for the data word size below the upper 32 bits of the data word will be undefined.

Bit Number	Name	Description	Reset State
31	REN	=1 - enable register pool read access. Usually only read or write access is enabled at the time	0
30-24	RSV30_24	Reserved	0
23	WEN	=1 - enable register pool write access. Usually only read or write access is enabled at the time	0
22-16	RSV22_16	Reserved	0
31-0	RAD	8-bit read address start, should match eight LSBs of the SPI address/command	0
31-0	WAD	8-bit write address start, should match eight LSBs of the SPI address/command	0

Table B-22 PL820_SPIx_IADDR SPI Slave Address Configuration Register Bit Descriptions

B.4.11 PL820_SPIx_REGx SPI Slave Register Pool Register Descriptions

0x3120-0x313C, 0x3120-0x31BC + RD/WR – PL820_SPIx_REGx SPI slave register pool registers

The SPI register pool consists of eight 32-bit registers. Registers can be accessed via SPI and also via the UEI serial interface.

NOTE: Note that the PL820IS_SPI_CFG_MWE should be set on both sides of the SPI for proper operation.

When 8 LSBs of the SPI address/command portion match the PL820IS_SPI_ACFG_RADx field in the PL820_SPIx_ACFG register and PL820IS_SPI_ACFG_REN is set, data from the registers will be transferred via SPI back to the master. When 8 LSBs of SPI address/command portion match PL820IS_SPI_ACFG_WADx and PL820S_SPI_ACFG_WEN is set, data from the registers will be written via SPI back to the master.

The first register address for a read access is defined by the three LSBs of the PL820IS_SPI_ACFG_RAD and will be incremented for the consequential data words, if more than eight words will be read during multi-word transaction, data from the registers will be recycled.

The first register address for write access is defined by the three LSBs +1 of the PL820IS_SPI_ACFG_WAD and will be incremented for the consequential data words, if more than eight words will be written during multi-word transaction, registers will be recycled (overwritten).

Index