PCI 9x56/FPGA AN

June 7, 2004
Version 1.0

Programming FPGA's through PLX PCI 9x56 Devices
Application Note

General Description

This application note discusses two methods for configuring FPGA's connected to the local bus of PLX PCI 9x56 devices.

There are a number of FPGA manufacturers using a variety of systems to program their FPGA's. This application note will outline two different techniques which can be used to configure FPGA devices. The procedures described will allow the FPGA to be configured using software either during the manufacturing process or during the boot of the system. In the case of RAM based FPGA's this may allow the design of a system which does not require the boot EEPROM or Flash memory normally associated with these devices.

The application note assumes that a PC or Compact PCI adapter card is being designed which will have an FPGA connected to the PCI 9x56's local bus. However, the methods shown are not limited to this and may be used in other types of system, or for configuring other types of devices.

The two methods described for configuring FPGA's are:

1. Configure the FPGA by direct slave accesses, with the PCI 9x56 local bus providing I/O signals to the FPGA. This requires a CPLD to interface to the 9x56 control signals.
2. Configure the FPGA using a combination of General Purpose IO and unused PCI 9x56 I/O pins. These pins are connected directly to the FPGA using accesses to the registers of the PCI 9x56 to download the configuration data.

Table of Contents

Figures

TABLES

1. Direct Slave Operation

The fastest method of FPGA configuration is achieved by performing Direct Slave Write transfers to one of the local address spaces. The data bus of the 9x56 is used to provide the majority of the I/O signals required to program the FPGA. However, a small amount of extra glue logic is needed. This glue logic provides the interface to the PCI 9x56 local bus control signals and allows the data lines to provide the appropriate data patterns to the FPGA configuration I/O.

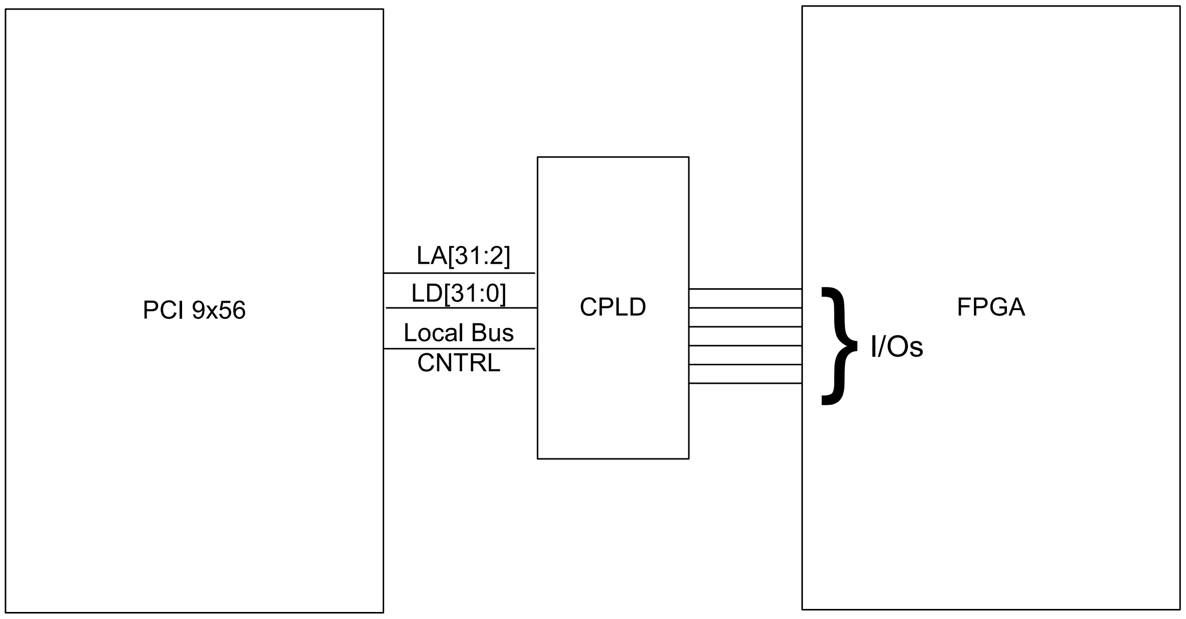

Figure 1. FPGA Programming Using Direct Slave Accesses via a CPLD

The PCI Base Address registers determine the adapter location in PCI Memory and I/O space. In addition, local mapping registers allow address translation from the PCI Address Space to the Local Address Space. Three Local Address Spaces are available:

- Local Address Space 0
- Local Address Space 1
- Expansion ROM

Each Local space can be programmed to operate with an 8, 16, or 32-bit local bus data width.

One local address space is used to program the FPGA. By performing Direct Slave accesses, the host writes data to the local bus or reads data from it. The glue logic is used to decode the local bus transactions and present the data from the local bus data lines to the FPGA.

If all the local address spaces are required during normal operation of the system then the USERo pin can be used as a chip select to control when FPGA programming operations occur.

2. PLX PCI 9x56 I/O pins

Depending on the application it may also be possible to program the FPGA using a combination of the USER I/O pins, EEPROM control signals and other unused I/O signals.

With the FPGA programming pins wired to the PCI 9x56 I/O pins, programming is performed by writing to the appropriate PCI 9x56 registers.

This section will describe the 9x56 I/O pins which can be used as general purpose or USER I/O pins.

	INPUTS	OUTPUTS	CLOCK
I/O Pins	EEDI/EEDO USERi/LLOCKi# PMEREQ#	EEDI/EEDO USERo/LLOCKo# LEDon# LINTo#	EESK

Table 1. PCI 9x56 I/O Pins

The precise pins which can be used will depend on the type of application being developed. The table below shows the pins which can be used in PC adapter cards or Compact PCI Hot Swap cards:

	PC ADAPTER CARD	Compact PCI HOT SWAP CARD
I/O SIGNALS	EEDI/EEDO, EESK USERi/LLOCKi# USERo/LLOCKo# LEDon#, PMEREQ#[1], LINTo#	EEDI/EEDO, EESK USERi/LLOCKi# USERo/LLOCKo# PMEREQ#, LINTo#

Table 2. PCI 9x56 I/O Pins available in PC or CompactPCI Hot Swap Adapter Cards

Note 1: The PMEREQ# signal can only be used as general purpose input in PC adapter card designs if Power management PME# generation is not required.

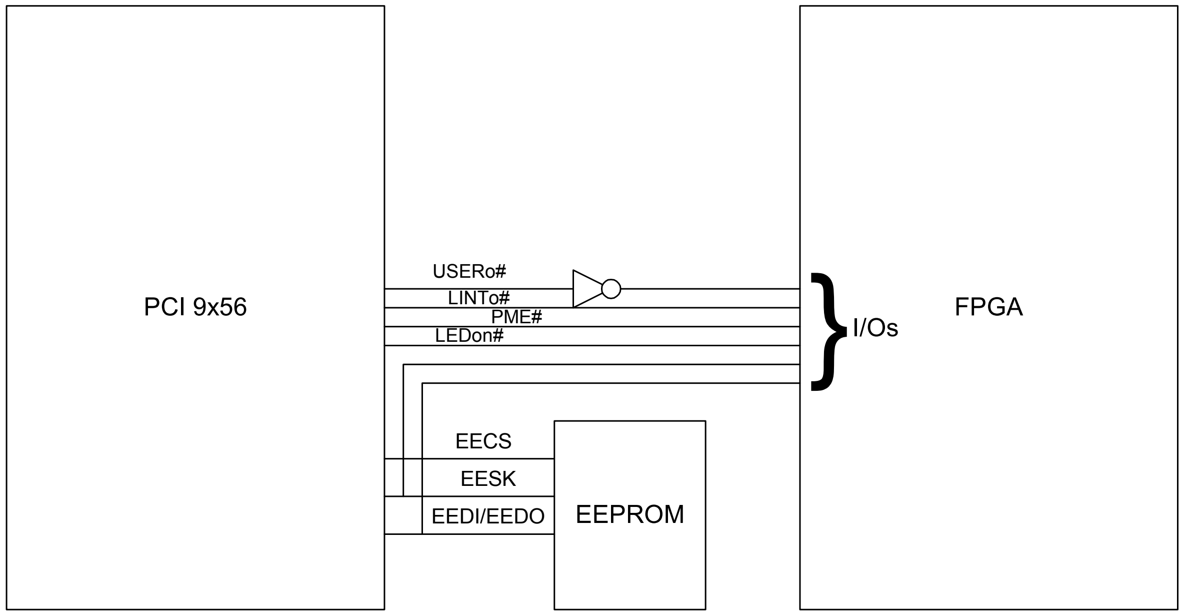

Figure 2. FPGA Programming using PCI 9x56 I/O and EEPROM Signals

2.1 Pin Description

2.1.1 Serial EEPROM Signals (EEDI/EEDO, EESK and EECS)

The EEPROM interface may be used to configure the PCI 9x56 device when the device comes out of PCI reset (PCI RST# de-asserted). Subsequent to this the EEPROM interface is not normally required.

In the above diagram one of the I/O signals connected to the FPGA enables/disables configuration accesses to the FPGA and ensures that EEPROM accesses do not influence the FPGA during boot.

The EECS signal is the serial EEPROM chip select and is asserted when the PCI 9x56 reads from or writes to the EEPROM. When using the EEDI/EEDO and EESK signals to configure the FPGA the EECS signal remains de-asserted ensuring that the EEPROM contents are not accessed.

Another approach may be to connect the EECS signal to both the EEPROM and the FPGA. Depending on the sense of the FPGA chip select or configuration control an inverter may be required between EECS and the FPGA so that only one device is enabled at any time. However, care must be taken to ensure that the FPGA is not configured accidentally. To avoid this, an alternative method is to switch the EECS signal between the EEPROM and the FPGA by gating this signal with the LRESETo# output. This method is the same as that detailed in the 9x56 design notes as part of the sample VPD implementation circuit – see design note 2 in the PCI 9056 and PCI 9656 design notes.

In order to program the EEPROM or the FPGA the PCI 9x56 provides the ability to manually access the serial EEPROM control signals. This is accomplished by using CNTRL[31, 27:24] register bits. CNTRL[24] is used to generate EESK (clock), CNTRL[25] enables or disables the EEPROM chip select, CNTRL[26] controls the EEDI output logic level, and CNTRL[31] enables the EEDO input buffer. When read, CNTRL[27] will return the value of EEDO. Therefore, by performing a series of accesses to the CNTRL register, the FPGA clock and serial data can be output from the EESK and EEDI/EEDO lines.

Details on using the CNTRL register to access a device interfaced to the EEPROM control signals can be found in sections 2.4.3 and 4.4.3 of the PCI 9x56 data book.

2.1.2 USERo/USERi

The PCI 9x56 provides a user input pin and a user output pin: USERi and USERo. Both pins are multiplexed with other signals. By default, the PCI 9x56 configures these pins to be USERi and USERo. USERi is selected when CNTRL[18]=1, and USERo is selected when CNTRL[19]=1. User output data is controlled by writing to the General-Purpose Output bit (CNTRL[16]). User input data can be read from the General-Purpose Input bit (CNTRL[17]).

USERi is also used to configure the PCI 9x56 for the desired PCI Bus behavior during chip initialization and should be pulled high or low depending on the application (see sections 2.4.2 or 4.4.2 of the PCI 9x56 data book).

During hardware reset, the PCI 9x56 will drive the USERo pin low. If the USERo pin is used to control an FPGA's configuration pin, an inverter may be required between the FPGA and USERo.

2.1.3 LEDon#:

For non Hot Swap applications the LEDon# can be used as an output.
The LEDon# pin is controlled by the LED Software On/Off Switch bit (HS_CSR[3]). When HS_CSR[3] is set to 1, the LEDon# pin will be driven high.

The PCI 9x56 will assert the LEDon# output (low) during PCI a reset (RST# asserted).

The LEDon# pin is an open drain output so a pull-up in the range of 3K to 10K will be required on this output.

2.1.4 PMEREQ#:

For non power management applications PMEREQ# can be used as a general purpose input.

Before using PMEREQ# as a general purpose input PRESENT_DET must be pulled high. The value of PRESENT_DET is ANDed with the value in PMCSR[15] so if PRESETM_DET is low then the PMEREQ# input is effectively disabled.

The PMEREQ# pin is controlled by the PMCSR register. Asserting PMEREQ# will cause the PCI 9x56 to set the PME_Status bit PMCSR[15]=1. The value in PMCSR[15] is a sticky bit. Therefore, de-asserting PMEREQ# will not immediately change the value of PMCSR[15].

To clear PMCSR[15] the host processor must write one (1) to PMCSR[8] and PMCSR[15]. On writing one (1) to PMCSR[15] if PMEREQ# is still asserted (low) then PMCSR[15] will remain set to one (1). However, if PMEREQ# is de-asserted (high) then PMCSR[15] will be cleared to zero (0) and will remain at zero (0) until the next assertion of PMEREQ#.

2.1.5 LINTo#:

If the local interrupt output LINTo# is not used during normal operation of the card, then it may be used as a general purpose output. The Local Interrupt output (LINTo#) is enabled by setting INTCSR[16]=1 and is asserted (low) by one of the following:

- PCI-to-Local Doorbell register Write access.
    - A PCI Bus master can assert the Local interrupt output (LINTo#) if the doorbell interrupt is enabled (INTCSR[17]=1) by writing a one (1) to any of the PCI-to-Local Doorbell bits (P2LDBELL[31:0]). The only way to clear this interrupt and de-assert the LINTo# output is for a local bus device to write to the appropriate bit in the P2LDBELL register. This facility may be useful to ensure that the FPGA cannot be reconfigured once programmed.
- PCI-to-Local Mailbox register (MBOX0, MBOX1, MBOX2, and/or MBOX3) Write access.
    - A PCI write to one of these four Mailbox registers sets a corresponding status bit in INTCSR[31:28]. If the Mailbox interrupt is enabled (INTCSR[3]=1) then this will cause the assertion of LINTo#. The only way to clear this interrupt and de-assert the LINTo# output is for a local bus device to read each of the Mailbox registers that were written to. This facility may be useful to ensure that the FPGA cannot be reconfigured once programmed.
- PCI Built-In Self-Test interrupt
    - A PCI Bus master can assert a Local interrupt by performing a PCI Configuration write that sets the PCI Built-In Self-Test Interrupt Enable bit (PCIBISTR[6]=1). The Local interrupt and INTCSR[23] remain set as long as PCIBISTR[6]=1. Clearing PCIBISTR[6] will de-assert LINTo# and clear INTCSR[23].

2.1.6 LINTi#:

The local interrupt input may be used as a general purpose input.

If the PCI interrupt is enabled (INTCSR[8]=1) and the Local Interrupt Input is enabled (INTCSR[11]=1) then asserting LINTi# will cause the PCI 9x56 to assert a PCI interrupt (INTA#). The PCI Host processor can read the PCI 9x56 Interrupt Control/ Status register (INTCSR) to determine whether an interrupt is pending as a result of the LINTi# assertion (INTCSR[15]=1). The interrupt remains asserted as long as LINTi# is asserted and the Local Interrupt input is enabled.

When using the LINTi# pin to generate a PCI interrupt the interrupt service routine should de-assert the PCI interrupt by either; disabling the PCI interrupt by clearing INTCSR[8], disabling the local interrupt by clearing INTCSR[11], or by performing an access to the local bus which then de-asserts the LINTi# input.

To use the LINTi# pin as a general purpose input without generating PCI interrupts the Local Interrupt should be enabled (INTCSR[11]=1) but the PCI Interrupt Enable bit should be cleared (INTCSR[8]=0). This will ensure that the PCI interrupt (INTA#) remains de-asserted, regardless of the LINTi# pin state. User software should then poll INTCSR[15] to determine the state of the LINTi# pin.

It should be noted that the LINTi# pin may have to be asserted for some time in order for INTA# to be asserted and the PCI interrupt service routine to be called, or for the polling routing to determine the state of the LINTi# pin.

3. Altera FPGA configuration examples

This section provides two worked examples utilizing the methods detailed in the previous sections. Although the examples use Altera FPGAs, the same principle can be used for FPGAs produced by other manufacturers.

Altera FPGAs can be configured using a number of different configuration schemes. In this example, we will use Passive Serial Configuration (PS). At the time of writing the Altera FPGAs which support the PS configuration mode include the following device families: Stratix, Stratix GX, Cyclone, APEX II, APEX 20K, Mercury, ACEX 1K, FLEX 10K, and FLEX 6000.

PS configuration can be performed by using an Altera download cable, an Altera enhanced configuration device or configuration device, or an intelligent host such as a microprocessor. During PS configuration, configuration data is transferred from a storage device, such as a configuration device or flash memory, to the FPGA on the DATA (FLEX 6000) or DATA0 (Stratix, Stratix GX, Cyclone, APEX II, APEX 20K, Mercury, ACEX 1K, and FLEX 10K) pin. This configuration data is latched into the FPGA on the rising edge of DCLK. Configuration data is transferred one bit per clock cycle.

During device operation, Altera FPGAs store configuration data in SRAM cells. Because SRAM memory is volatile, the SRAM cells must be loaded with configuration data each time the device powers up. After the device is configured, its registers and I/O pins must be initialized. After initialization, the device enters user mode for in-system operation.

3.1 Programming an FPGA via a CPLD (see Section 1)

Figure 3 and the CPLD code show one possible method for using direct slave accesses to program the FPGA with the PCI 9x56 configured to operate in C mode. It is up to the user to generate the appropriate data pattern to download to the FPGA.

To configure the FPGA one of the PCI 9x56 address spaces should be configured such that direct slave accesses can occur at a local address with LA[31] asserted. The data pattern should be written to the FPGA using a series of direct slave writes.

The CPLD only makes use of LD[0] so programming data from the PCI bus will always be present on the least significant bit of an access. If the local address space is configured as an 8 bit wide region then, 4 bits of data will be transferred per PCI LWord transfer with the programming data being in bits 0, 8, 16 and 24.

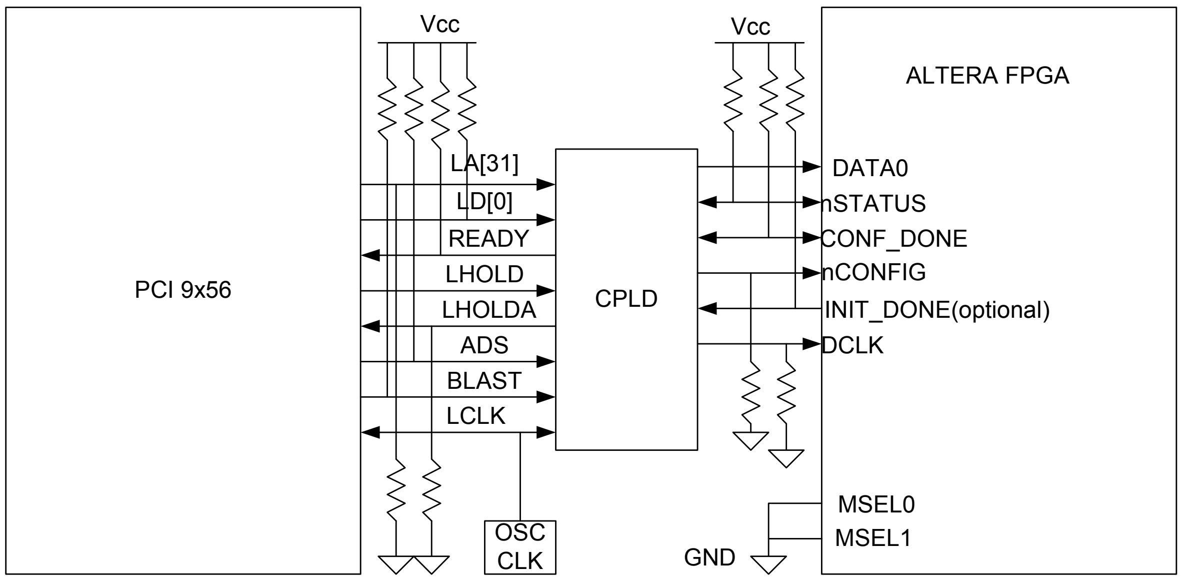

Figure 3. Altera FPGA programming using Direct Slave accesses via a CPLD

3.1.1 CPLD Code

The following pages contain the ABEL Code for the CPLD.

```
CPLD ABEL CODE
//=================================================================
// 03/18/2004
// Direct slave accesses provide I/O signals to the FPGA using a CPLD to
// interface to the 9x56 control signals.
//=================================================================
Module fpgacnfg
Title 'FPGA configuration'

Declarations
"inputs
 LCLK, LHOLD, ADS, BLAST, LA[31], LD[0], INIT_DONE;
"outputs
 LHOLDA, READY, DCLK, DATA0, nCONFIG;
"input/output
nSTATUS, CONFIG_DONE;

SREG state_register;
S0..S3 state;
equations
SREG.clk = LCLK;

// LHOLD & LA[31] are used to start the configuration of the FPGA. The Local address LA[31] is used
as a chip select to assert the nCONFIG pin.

nCONFIG = (LHOLD & !ADS & LA[31]) # CONF_DONE; // A low to high transaction starts a
configuration cycle.
LHOLDA = LHOLD & !ADS & LA[31]; // Asserts LHOLDA in response to LHOLD.

State_diagram SREG;
state S0: if (nCONFIG & nSTATUS & !ADS) then S1 with { READY = 0;
                                                       DCLK=LCLK;
                                                       DATA0=LD[0]; } // nCONFIG
 and nSTATUS must be at logic high level in order for the configuration stage to begin.

          else S0 with { READY = 1;
                         DCLK=0;}    // Configuration does not begin.

state S1: if (!BLAST) then S2 with {READY = 1;
                                    DCLK=LCLK; }   // Direct Slave cycle is complete,
 release READY.

          else if (!nSTATUS) then S0;        // If an error occurs during configuration, nSTATUS is
pulled low again by the Altera device.

           else S1 with {READY = 0;          // Direct Slave cycle is not complete.
                         DCLK=LCLK;
                         DATA0=LD[0]; }
```

state S2: if (INIT_DONE & CONFIG_DONE) then S3 with DCLK=0;   // A low to high transaction on CONF_DONE indicates configuration is complete. INIT_DONE signals the end of the initialization and the start of user-mode.

                else if (!nSTATUS) then S0;   // If an error occurs during configuration, nSTATUS is pulled low again by the Altera device.

                 else S2 with DCLK=LCLK; // INIT_DONE is not complete.

state S3: if (!nSTATUS) then S0 with DCLK=0;   // If an error occurs.

                else S3 with DCLK=0;   // configuration and initialization are complete.

END cnfgfpga

3.2 Programming an FPGA using PLX I/O in a PC adapter card or CPCI card (see Section 2)

Figure 4 shows one possible method using the PLX 9x56 I/O pins to program the FPGA. It is up to the user to generate the appropriate data pattern to download to the FPGA.

The USERo pin is used to set the FPGA into programming mode. Data to program the FPGA is then downloaded to the device using a series of accesses to the CNTRL register.

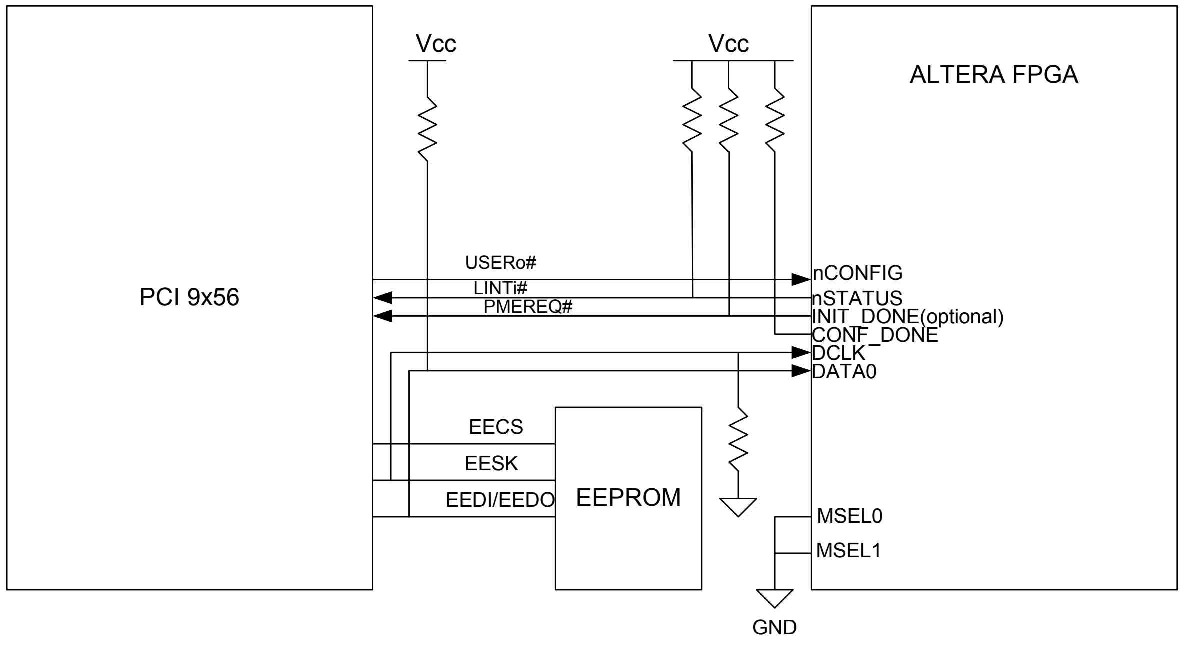

Figure 4. Altera FPGA Programming using PCI 9x56 I/O and EEPROM Signals

4. References

The following is a list of additional documentation to provide the reader with further information on PLX products.

PCI 9656 Data Book, Version 1.1
PCI 9056 Data Book, Version 1.1
PCI 9656 Design Notes, Revision 1.2 or higher
PCI 9056 Design Notes, Revision 1.2 or higher

PLX Technology, Inc.
870 Maude Ave.
Sunnyvale, CA 94085 USA
Tel: 408-774-9060, 800-759-3735
Fax: 408-774-2169
http://www.plxtech.com