

## Introduction

This document contains information and requirements to demonstrate features and capabilities of the Lattice PCI Master/Target, 33 MHz/32-bit IP core implemented into a LatticeEC™ FPGA. The IP core is compliant to PCI 3.0 specifications as published by the PCI-SIG.

## Requirements

### Demo Package:

- pci\_mt33\_32\_demo\_v1\_0.zip

### Hardware:

- PCI Evaluation Board - LatticeECP/EC Standard Evaluation Board Rev B with LEFC6E -5F484C (fpBGA) device on board
- PC-compatible systems:
  - Minimum of 512Kbytes system memory
  - PCI slot compliant to PCI 2.2 specifications or later

### Operating System:

- Windows 2000 or Windows XP

### Lattice Software and Tools:

- ispLEVER® 6.1
- ispVM® System software
- Download cable

## Demo Package and Installation

The pci\_mt33\_32\_demo\_v1\_0.zip contains the drivers and other files required to run the demo application software. To install:

1. Make a directory in your favorite drive. For example, C:\Lattice
2. Extract pci\_mt33\_32\_demo\_v1\_0.zip into C:\Lattice

After installation, the C:\Lattice\pci\_mt33\_32\_demo\_v1\_0 directory structure and contents should be as shown below:

```
. \Bitstream
| <core_ec06.bit>
  Design implementation containing the Lattice PCI Master/Target,
  33/32 IP Core and Demo Application Module.

. \Doc
| <PCI Demo User's Guide.doc>
  This document.
```

```
. \Driver
|  -- <pcicore.inf>
|  -- \2000
|    -- <pcicore.sys>

|  -- \XP
|    --<pcicore.sys>

. \GUITest
|  --<GUITest.exe>
|  --<Reg.map>
|  --<block.dat>

. \SDK
|  --<pcicore.h>
|  --<pcicore.lib>
```

The software consists of three parts: drivers, SDK and a GUI Test tool. The individual files are listed below:

**Driver Files:**

- Driver install file <pcicore.inf>
- Driver image file <pcicore.sys>

**SDK Files:**

- SDK header file <pcicore.h>
- SDK library file <pcicore.lib>

**GUI Test Tool:**

- Executable file <GUITest.exe>

## How to Install the Lattice PCI Device

In order to run the demo, installation of the Lattice PCI device is required. In the Windows OS context the Lattice PCI device is composed of the PCI Evaluation Board and the device driver.

The following installation sequence is based on a PC running on the Windows XP Operating System.

**Installation Procedure:**

1. With the PC power turned off, plug in the LatticeEC Standard Board into an empty PCI slot.
2. Apply power to the PC and let the system boot-up.
3. Open **Device Manager** and locate **PCI standard RAM Controller**, as shown in figure 1. The PCI Evaluation Board is detected as PCI standard RAM Controller in Windows Device Manager.

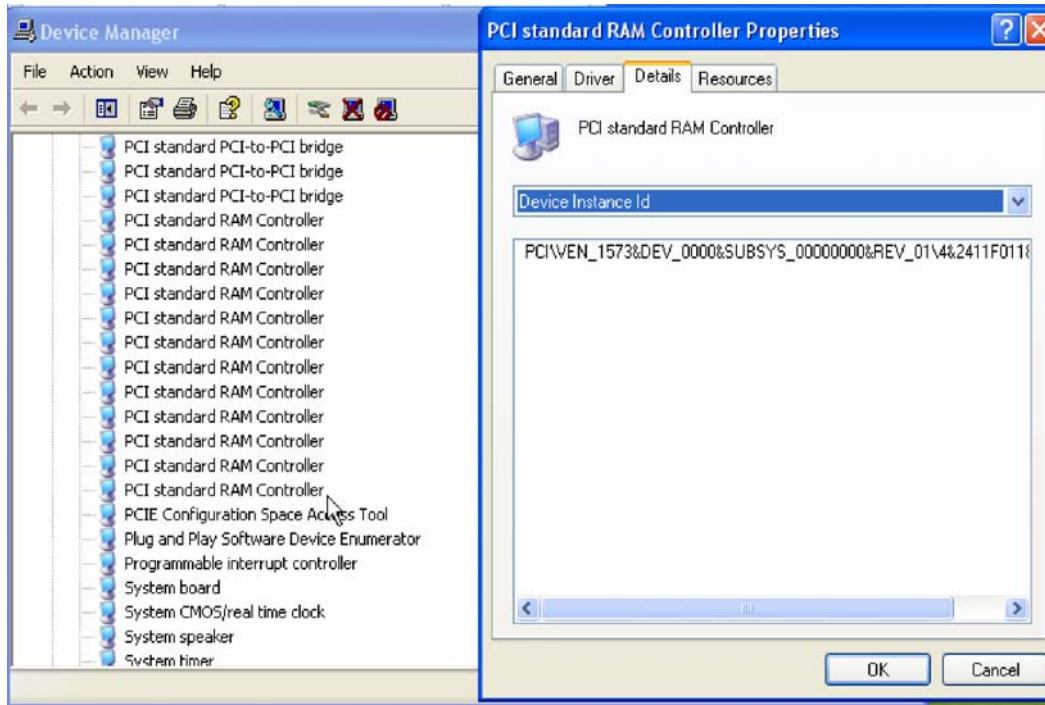
*Note: Depending on the PC system, there could be multiple PCI standard RAM Controllers shown under the Device Manager. There are two ways to properly identify and select the Lattice PCI Evaluation Board the first time. In Figure 1, the last PCI standard RAM Controller in the list corresponds to the Lattice PCI Evaluation Board.*

- a. **Vendor ID:** The Lattice Vendor ID used in this demo package is 1573(h). Therefore, the selected PCI standard RAM Controller Vendor ID must be 1573.

- b. **Location:** A PCI Add-in card, such as the Lattice PCI Evaluation board, is assigned a location. For example:

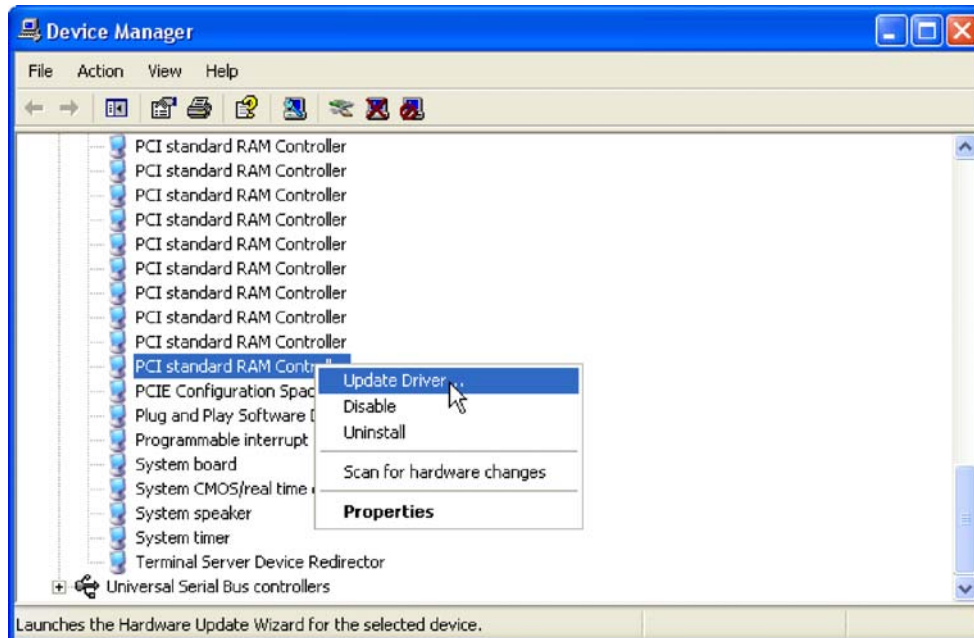
Location: PCI bus 2, device 12, function 0

**Figure 1. Windows XP OS Device Manager**



4. Select **PCI standard RAM Controller** and select **Update Driver...** from the pop-up menu, as shown in Figure 2.

**Figure 2. Update Driver**



5. Select **Install from a list or specific location (Advanced)**, as shown in Figure 3.

Select **Next**.

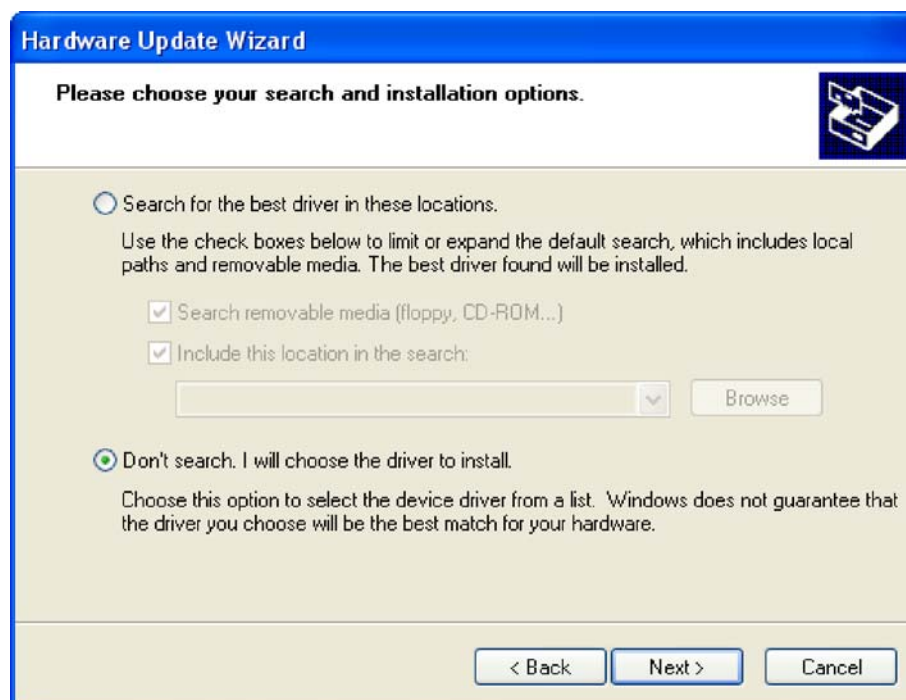
**Figure 3. Driver Installation Option**



6. Select **Don't search. I will choose the driver to install** as shown in Figure 4.

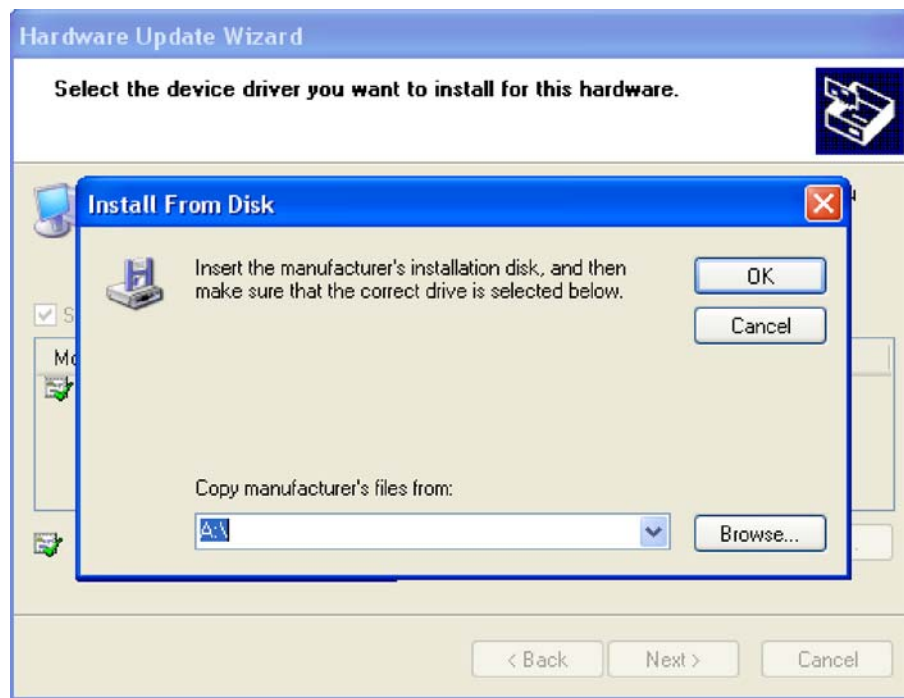
Select **Next**.

**Figure 4. Choose Driver to Install**



7. The **Install From Disk** dialog box appears, as shown in Figure 5.

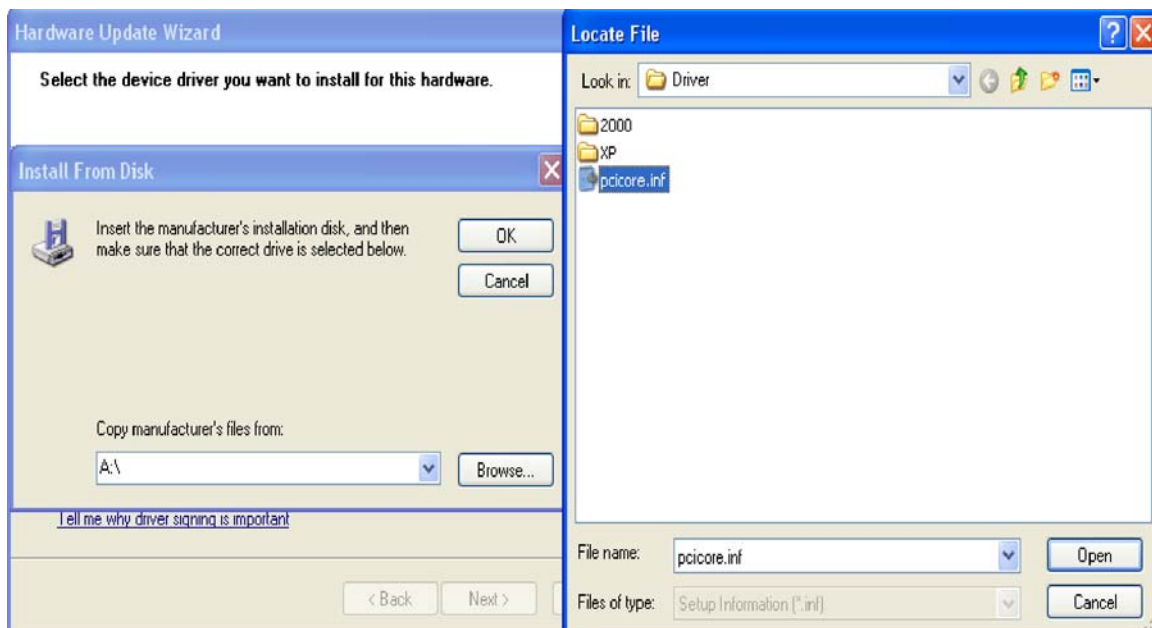
**Figure 5. Install from Disk**



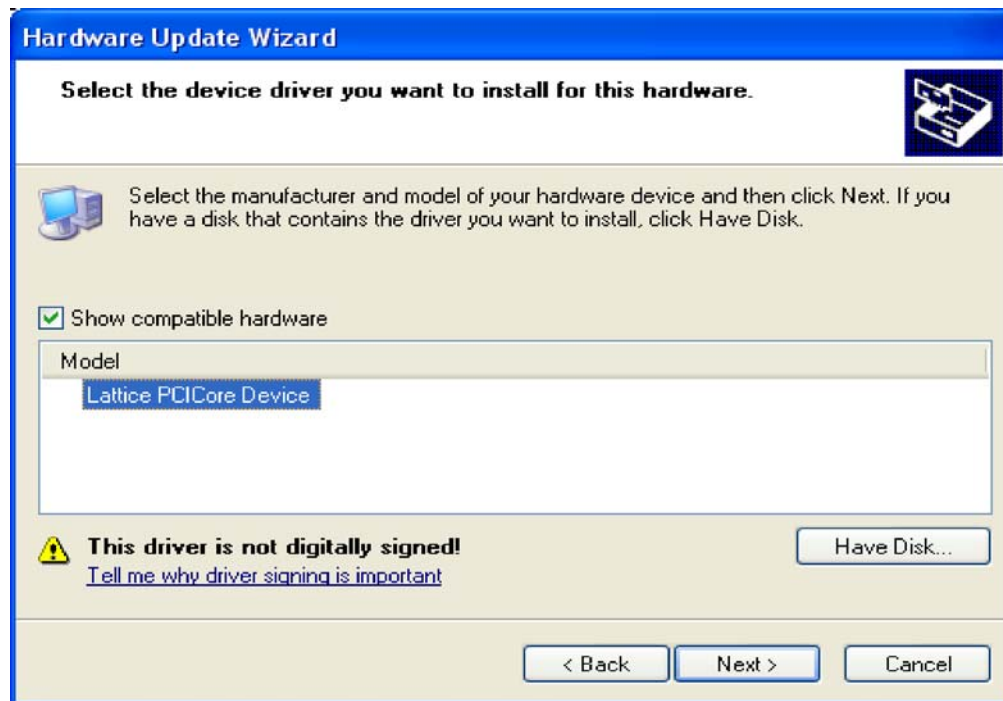
8. Select **Browse**, and then select the file **pcicore.inf**, located in **C:\Lattice\pci\_mt33\_32\_demo\_v1\_0** directory as shown in Figure 6.

Select **Open**.

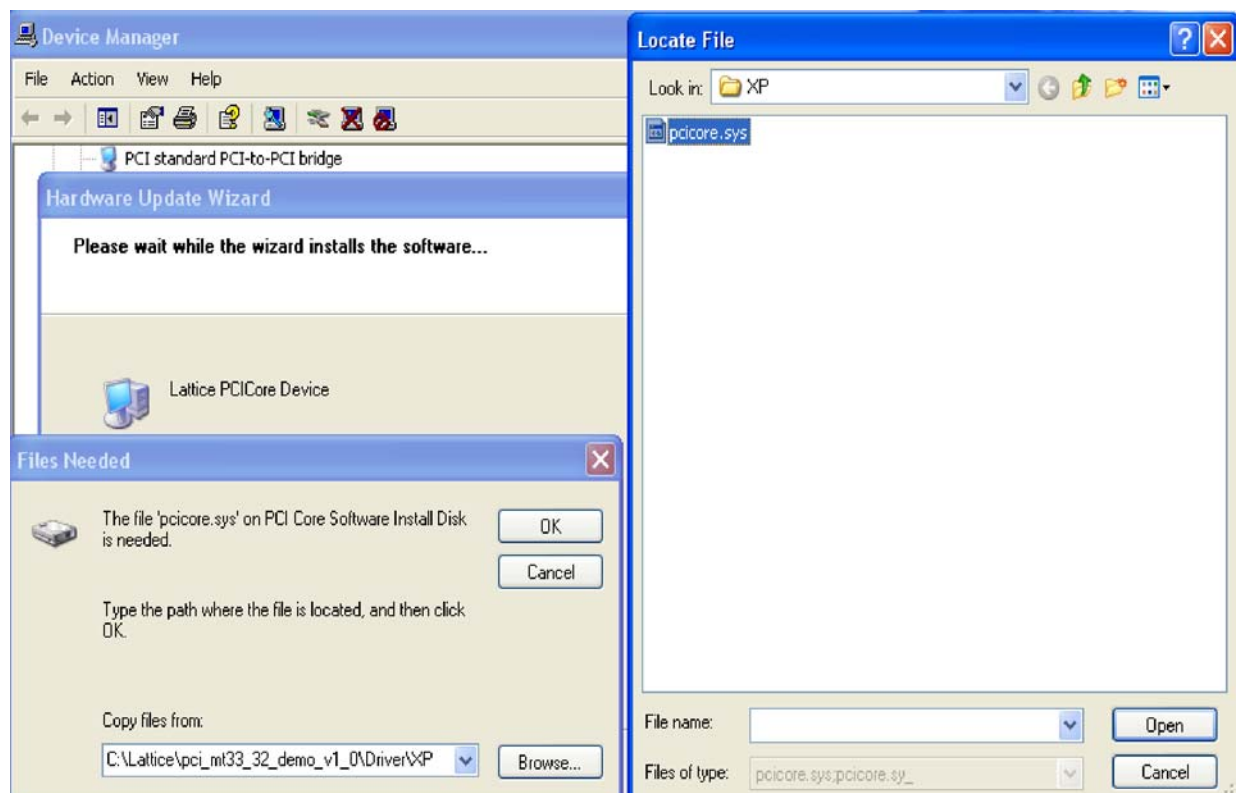
**Figure 6. Locate File pcicore.inf**



9. Choose **Lattice PCICore Device** and select **Next**, as shown in Figure 7.

*Figure 7. Select the Lattice PCI Device*

10. In the **Files Needed** dialog box, select **Browse** and locate the file **pcicore.sys** in the directory **C:\Lattice\pci\_mt33\_32\_demo\_v1\_0** as shown in Figure 8.

*Figure 8. Files Needed/Locate File pcicore.sys*

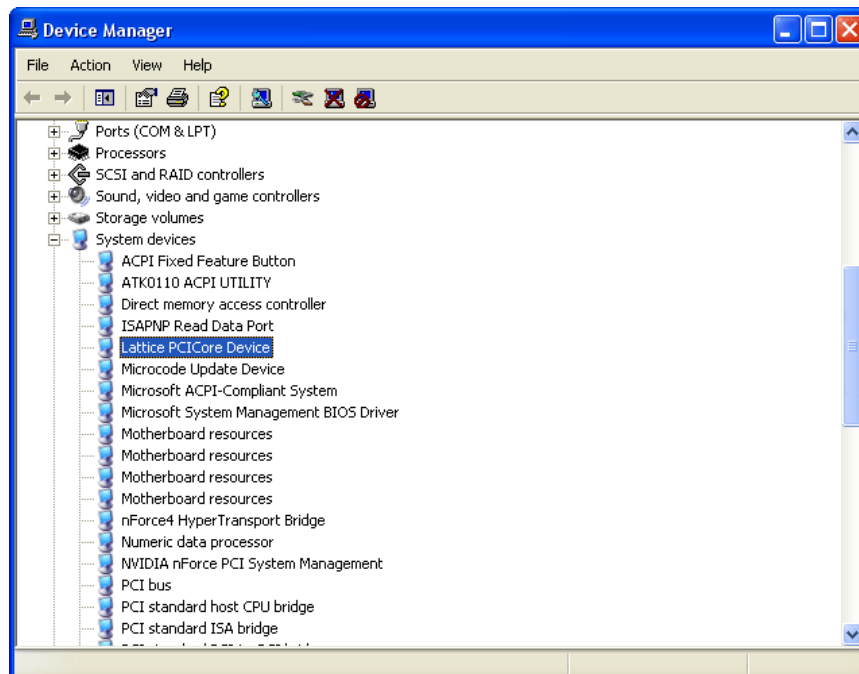
11. In the **Locate File** window, select the **pcicore.sys** file and **Open**, as shown in Figure 9.
12. Select **OK** in the **Files Needed** and **Install from Disk** dialog boxes.
13. The Hardware Update Wizard copies the driver file into the system and completes the driver installation, as shown in Figure 9.

**Figure 9. Hardware Update Wizard Completed**



14. **Lattice PCICore Device** appears in the **System devices** list, showing successful installation.

**Figure 10. The Lattice PCI Device in the Systems Devices List**

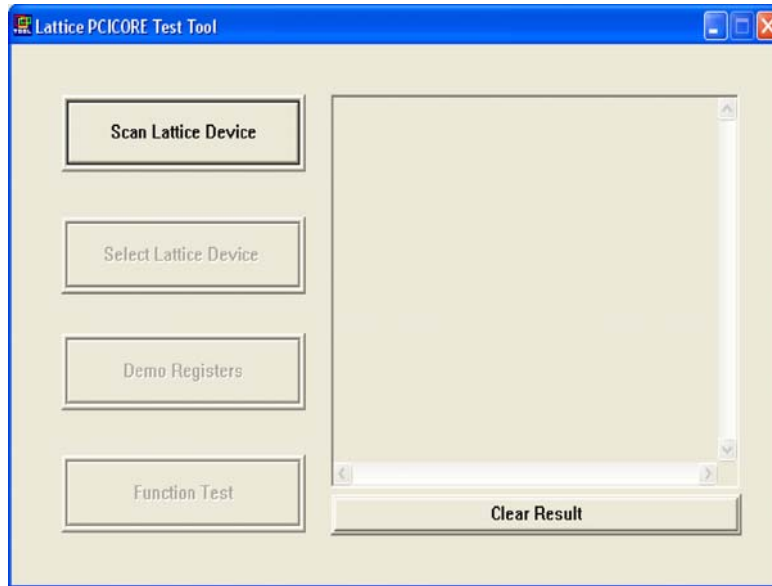


## Running the Demo

After successful installation of the PCI Evaluation Board and the device driver, select/open the application file **GUITest.exe** located in the **C:\Lattice\pci\_mt33\_32\_demo\_v\_1\_0\GUITest** directory.

The main GUI for running the demo is shown in Figure 11.

**Figure 11. Main GUI for PCI Demo**



Then, follow this sequence:

1. Select **Scan Lattice Device**. This verifies the presence of the Lattice PCI Evaluation Board as a valid system device. Notice the number of devices, BARs and their properties.
2. Select **Select Lattice Device**.

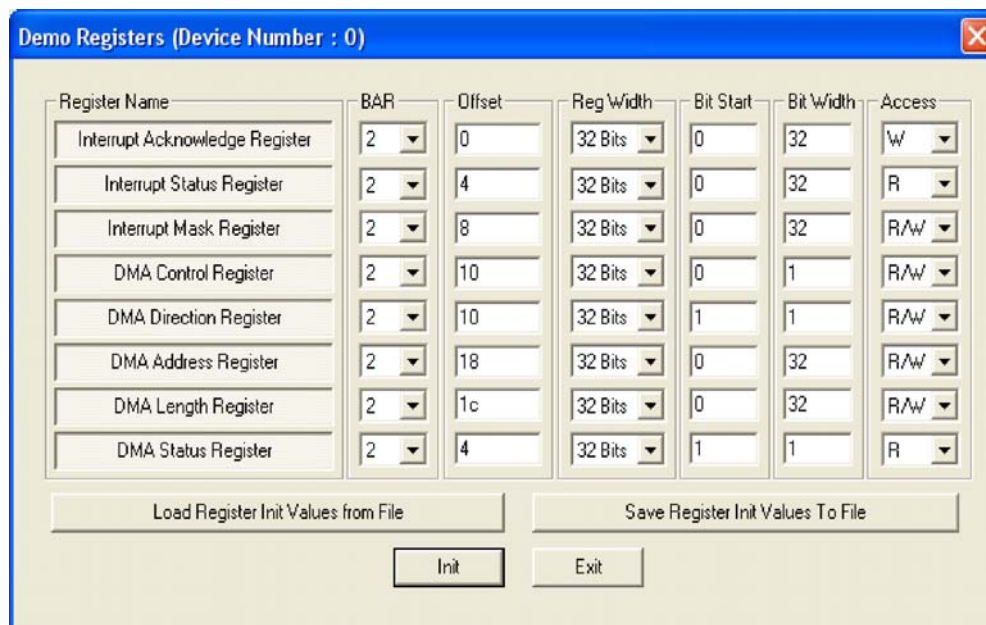
**Figure 12. Select Lattice Device**



Select **OK**.

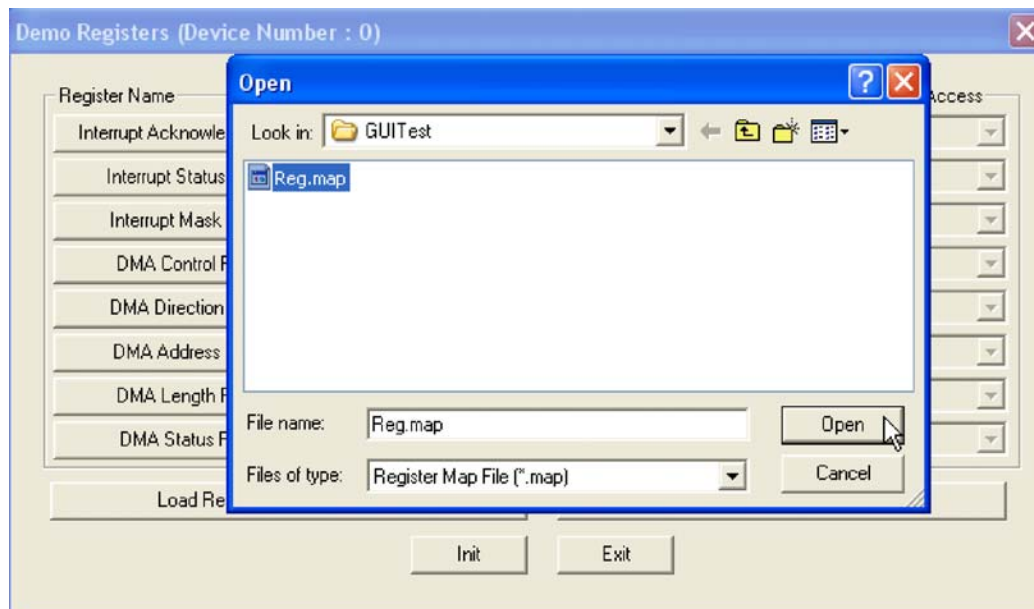
3. Select **Demo Registers**.

**Figure 13. Demo Registers**



- a. Select **Load Register Init Values from File**. Interrupt and DMA registers must be set prior to testing these functions. A pop-up window will bring up a file to set initial register values.

**Figure 14. Loading Reg.map File**



- b. Select the **Reg.map** file.
- c. Select **Open**.
- d. Select **Init**. This loads the initial values to all the demo registers.

4. Select **Function Test**. The Function Test GUI is shown below. To support the corresponding demo functions, the GUI is divided into the following sections:
- **Configuration Space:** Read/Write access to the Lattice PCI Device CSR
  - **I/O or Memory Space:** I/O or Memory Space single Read/Write transaction
  - **DMA:** DMA Read/Write Transfers from/to PC system memory. Lattice PCI Device stores data read from system memory.
  - **Data:** Select and Generate the Data Pattern for DMA transfer
  - **Block Write/Read:** Supports file transfer to/from disk drive
  - **Interrupt:** Software generated interrupt

Figure 15. Function Test GUI

Function Test (Device Number : 0)

**Configuration Space**

Offset (hex): 00000004

Bits Width : 32 bits

Data (hex): 0480075F

Read Write

Scan Configuration Space

**I/O or Memory Space**

BAR (dec): 0

Offset (hex): 00000000

Bits Width : 32 bits

Data (hex): A5A5A5A5

Read Write

**DMA**

Length (hex): 00000000

DMA Read DMA Write

**Interrupt**

Generate the interrupt

**Data**

Set DMA Data

**Block Write/Read**

BAR (dec): 0 Offset (hex): 00000000 Length (hex): 00000070

Write file : Browse View

Read file : Browse View

Block Read Block Write

Clear Result

Exit

- a. **Configuration Space:** Read/Write PCI Configuration Space Registers (CSR)

Select **Scan Configuration Space**. Examine contents of Results Window, Vendor ID, etc.

Figure 16. Scanning Lattice PCI Device CSR

Function Test (Device Number : 0)

**Configuration Space**  
 Offset (hex): 00000000  
 Bits Width : 32 bits  
 Data (hex): 00000000  
 Read Write  
 Scan Configuration Space

**I/O or Memory Space**  
 BAR (dec): 0  
 Offset (hex): 00000000  
 Bits Width : 32 bits  
 Data (hex): 00000000  
 Read Write

**DMA**  
 Length (hex): 00000000  
 DMA Read  
 DMA Write

**Interrupt**  
 Generate the interrupt

**Data**  
 Set DMA Data

**Block Write/Read**  
 BAR (dec): 0 Offset (hex): 00000000 Length (hex): 00000000  
 Write file :  
 Read file :  
 Block Read Block Write

**Results**  
 Vendor ID : 0x1573  
 Device ID : 0x0000  
 Command : 0x0007  
 Status : 0x0480  
 Revision ID : 0x01  
 Class Code : 0x050000  
 Cache Line Size : 0x10  
 Latency Timer : 0x40  
 Header Type : 0x00  
 BIST : 0x00  
 Base Address Register 0 : 0xfeafc000  
 Base Address Register 1 : 0x0000a801  
 Clear Result

Exit

Enter the following:

Offset (hex): 00000004

Bits Width: 32 bits

Data (hex): 0000075f

Select **Write**. The PCI Command register contents are updated.

Select **Read**. Read/verify the contents of the PCI Command register.

b. **I/O or Memory Space**: Read/write to I/O or memory space.

Figure 17. I/O Space - Single R/W Transaction

The screenshot shows the 'Function Test (Device Number : 0)' window. It is divided into several sections:

- Configuration Space:** Offset (hex): 00000000, Bits Width: 32 bits, Data (hex): 00000000. Buttons: Read, Write, Scan Configuration Space.
- I/O or Memory Space:** BAR (dec): 1, Offset (hex): 00000048, Bits Width: 08 bits, Data (hex): 000000BD. Buttons: Read, Write.
- DMA:** Length (hex): 00000000. Buttons: DMA Read, DMA Write.
- Interrupt:** Generate the interrupt.
- Data:** Set DMA Data.
- Block Write/Read:** BAR (dec): 0, Offset (hex): 00000000, Length (hex): 00000000. Fields for Write file and Read file with Browse and View buttons. Buttons: Block Read, Block Write.
- Results:**
  - Read I/O or Memory Space: Device Number: 0, Bar: 1, Offset: 0x00000048, Bits Width: 8, Data: 0x00000000.
  - Write I/O or Memory Space: Device Number: 0, Bar: 1, Offset: 0x00000048, Bits Width: 8, Data: 0xffffffffbd.
  - Read I/O or Memory Space: Device Number: 0, Bar: 1, Offset: 0x00000048, Bits Width: 8, Data: 0x000000bd.
- Buttons:** Clear Result, Exit.

**I/O Space:** Single read/write transaction (refer to Figure 17).

**Read** - Enter the following:

BAR (dec): 1  
Offset (hex): 00000048  
Bits Width: 32 bits  
Data (hex):

Select **Read**. Examine result. Should be all 0's

**Write** - Enter the following.

BAR (dec): 1  
Offset (hex): 00000048  
Bits Width: 08 bits  
Data (hex): FFFFFFFBD

Select **Write**. This executes the write.

**Read/Verify** - Select **Read**.

BAR (dec): 1  
Offset (hex): 00000048  
Bits Width: 08 bits  
Data (hex): 000000BD

Question: Why is the data read back as 000000BD(h)?

**Memory Space:** Single read/write transaction.

**Figure 18. Memory Space - Single R/W Transaction**

**Read** - Enter the following:

BAR (dec): 0  
Offset (hex): 00000074  
Bits width: 32 bits  
Data (hex):

Select **Read**. Examine result. Should be all 0's.

**Write** - Enter the following:

BAR (dec): 0  
Offset (hex): 00000074  
Bits width: 32 bits  
Data (hex): 98C5B7D3 (or other 32-bit data)

Select **Write**. This executes the write.

**Read/Verify** - Select **Read**.

Verify data read back.  
BAR (dec): 0  
Offset (hex): 00000074  
Bits Width: 32 bits  
Data (hex):

Try reading with different Bits Width setting.

### c. DMA Write/Read

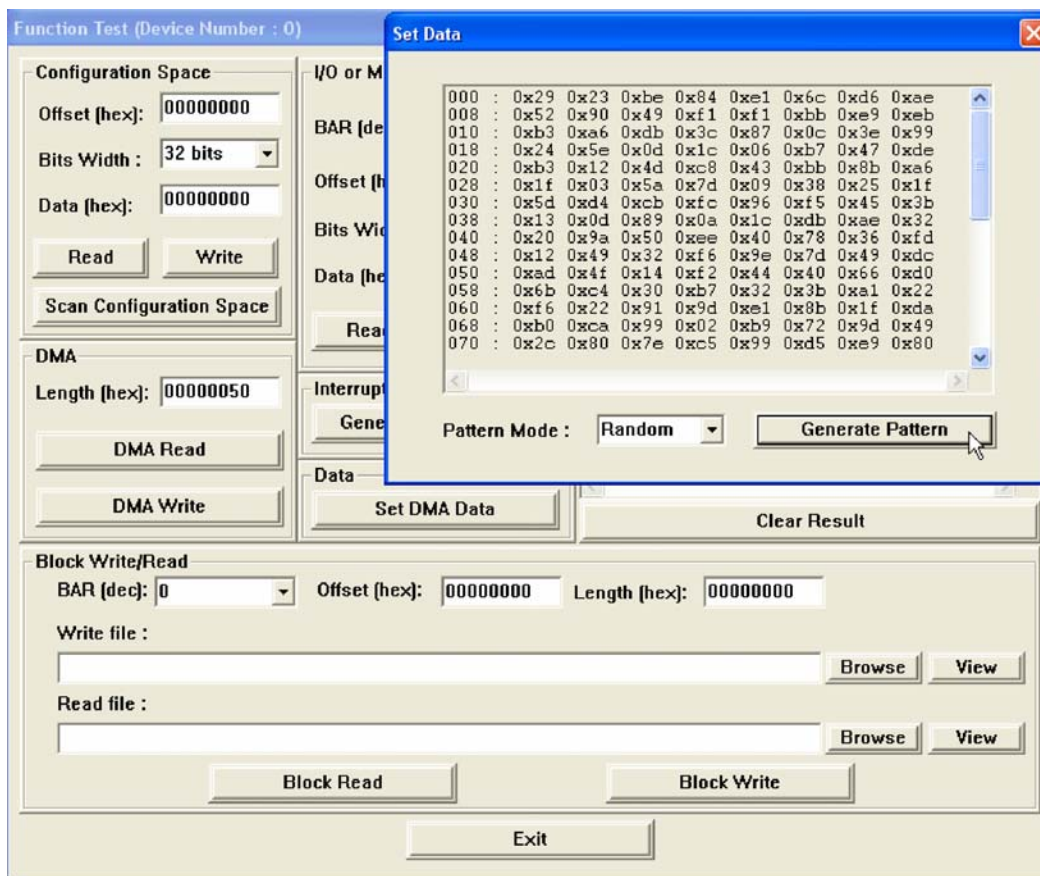
Select **Exit** to go to main GUI

As described above, Select **Demo Registers**, re-load **Reg.map** file and **Exit**.

*Note: If any of the Demo Registers are modified before starting the DMA transfer test, the Demo Registers must be reinitialized, otherwise the application software will no longer function.*

Select **Function Test**.

**Figure 19. Random Data Pattern for DMA Transfer**



Select **Set DMA Data >Random>Generate Pattern** and close the window.

To specify size of data to be transferred, enter the following in the **DMA section**:  
Length (hex): 00000050

Select **DMA Write**. This writes data, with Length: 50(hex), to system memory.

Select **DMA Read**. This reads data from the system memory and stores it in the BAR0 memory space (EBRs in the LatticeEC6 device). The result is shown in Figure 20.

Figure 20. Result after DMA Read

The screenshot shows the 'Function Test (Device Number : 0)' window. The 'I/O or Memory Space' section is active, displaying the results of a DMA Read operation. The 'DMA Read' section shows the device number as 0 and the length as 0x00000050. The data readback is displayed in a table format:

|       | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |
|-------|----|----|----|----|----|----|----|----|----|----|
| 000 : | 29 | 23 | be | 84 | e1 | 6c | d6 | ae | 52 | 90 |
| 010 : | b3 | a6 | db | 3c | 87 | 0c | 3e | 99 | 24 | 5e |
| 020 : | b3 | 12 | 4d | c8 | 43 | bb | 8b | a6 | 1f | 03 |
| 030 : | 5d | d4 | cb | fc | 96 | f5 | 45 | 3b | 13 | 0d |
| 040 : | 20 | 9a | 50 | ee | 40 | 78 | 36 | fd | 12 | 49 |

The 'Configuration Space' section shows the offset (hex) as 00000000, bits width as 32 bits, and data (hex) as 00000000. The 'DMA' section shows the length (hex) as 00000050. The 'Block Write/Read' section shows the BAR (dec) as 0, offset (hex) as 00000000, and length (hex) as 00000000. The 'Write file' and 'Read file' sections are empty. The 'Block Read' and 'Block Write' buttons are visible at the bottom.

Verify data in BAR0 memory space.

In **I/O or Memory Space**, enter/select:

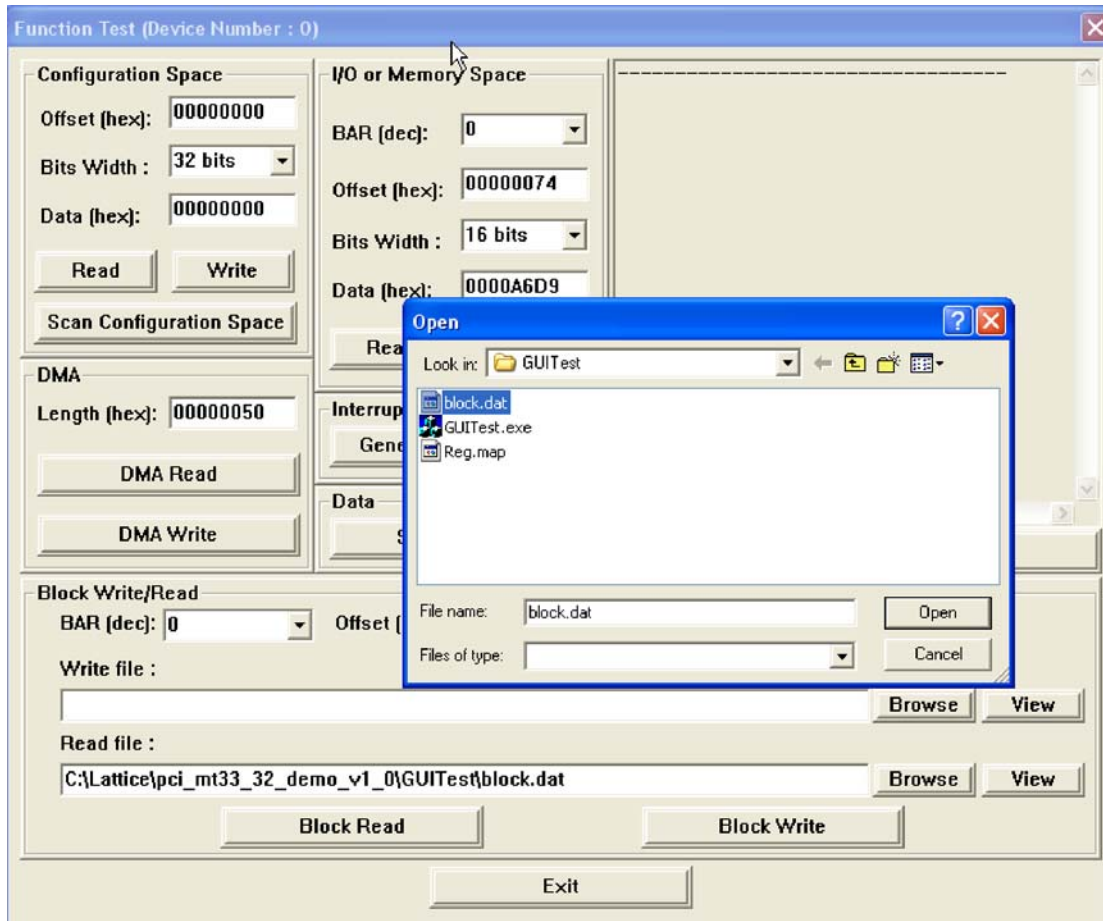
BAR (dec): 0  
Offset (hex): 00000000  
Bits Width: 32 bits  
Data:

Select **Read** and note the data readback. Compare the data with the Random Data Pattern starting at Offset 00000000(h). The two should match.

- d. **Block Write/Read.** A segment of a “data” file is read from the PCI Demo package installation directory and written to the BAR0 memory space. The contents of the BAR0 memory space are modified and then saved, with a different filename, in the same directory as the original “data” file.

Read the “data” file and write into BAR0 memory (EBR in LatticeEC6).

Figure 21. Locating and Opening block.dat File



In **Block Write/Read** section:

Go to **Read file:** > **Browse** to locate the file **block.dat** in the following directory:  
**C:\Lattice\pci\_mt33\_32\_demo\_v1\_0\GUI Test**

To set segment size, enter the following:  
Length (hex): 00000070

Select **Block Write**. This writes the file segment into the LatticeEC6 EBR.

Verify the data written into the EBR and modify the data in location 0x0.

In **I/O or Memory Space** section, enter the following:

BAR (dec): 0  
Offset (hex): 00000000  
Bits Width: 32 bits  
Data (hex):

*Note: Offset value can be within the range Length (hex): 00000070.*

Select **Read**.

Modify the data by entering the following:  
Data (hex): "Different 32-bit data"

Select **Write** to change the contents of BAR0, Memory address 0x0.

Select **Read** to verify the data changed.

Read the contents of the EBR and save it as a file. In Block Write/Read Section:

Select **Write->Browse** and find the directory called  
**C:\Lattice\pci\_mt33\_32\_demo\_v\_1\_0\GUI\Test**.

In the **Save As** window enter as the file name: **block\_mod.dat**.

Select **Save**.

Select **Block Read**. This saves the contents of the EBR as a file named **block\_mod.dat**.

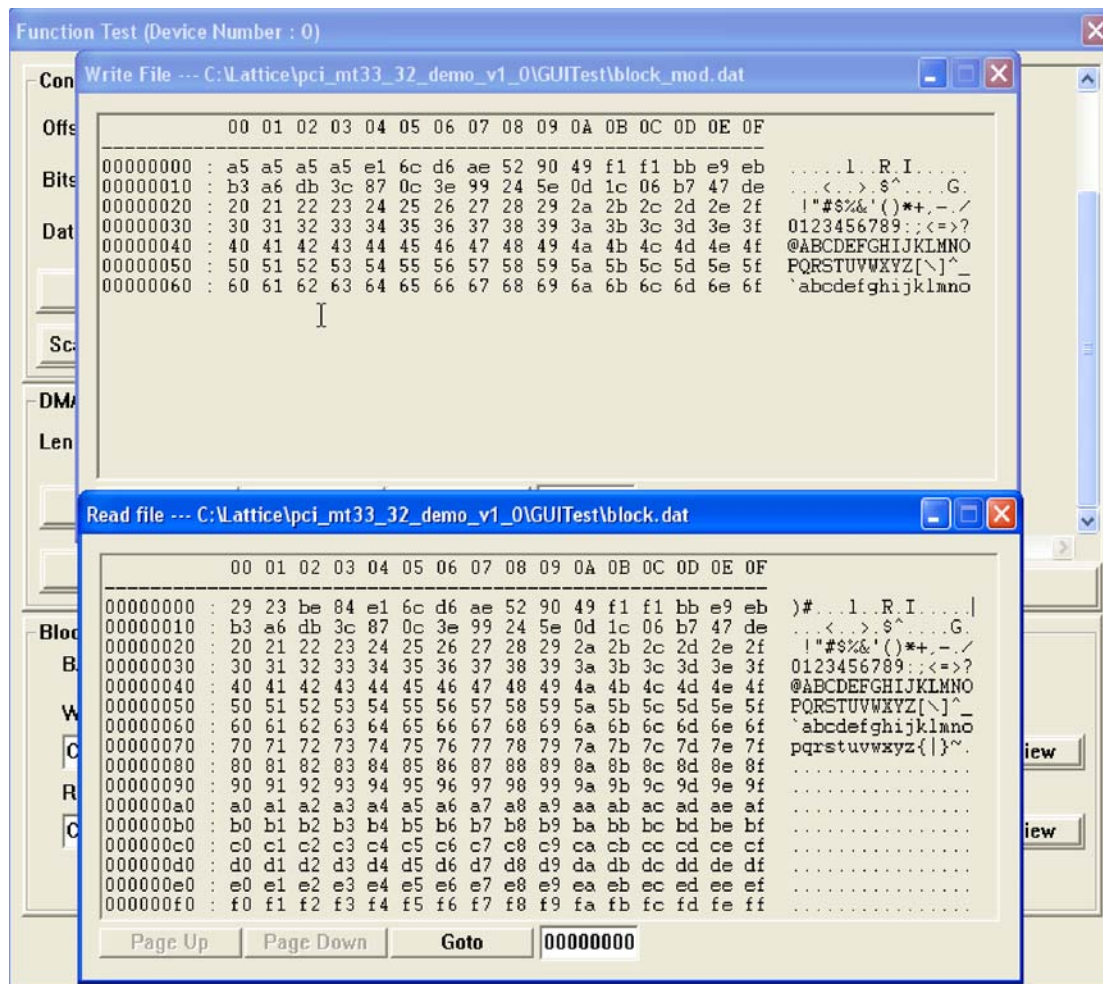
Compare the original data file and the modified data file.

Select **View** under **Read file**. This displays the original data file block.dat.

Select **View** under **Write file**. This displays the modified data file block\_mod.dat.

Compare the contents of the two files from offset 00(hex) to 03(hex). The result should look like Figure 22.

**Figure 22. Compare Original and Modified/Saved Data File**



- e. **Interrupt.** In this demo, an interrupt is initiated by the software. This is accomplished by selecting **Generate the Interrupt** in the Function Test GUI. The Lattice PCI device drives the physical INTx line to logic low. Software also acknowledges the interrupt. This transaction is shown in Figure 23.

**Figure 23. Software Initiated/Acknowledged Interrupt**

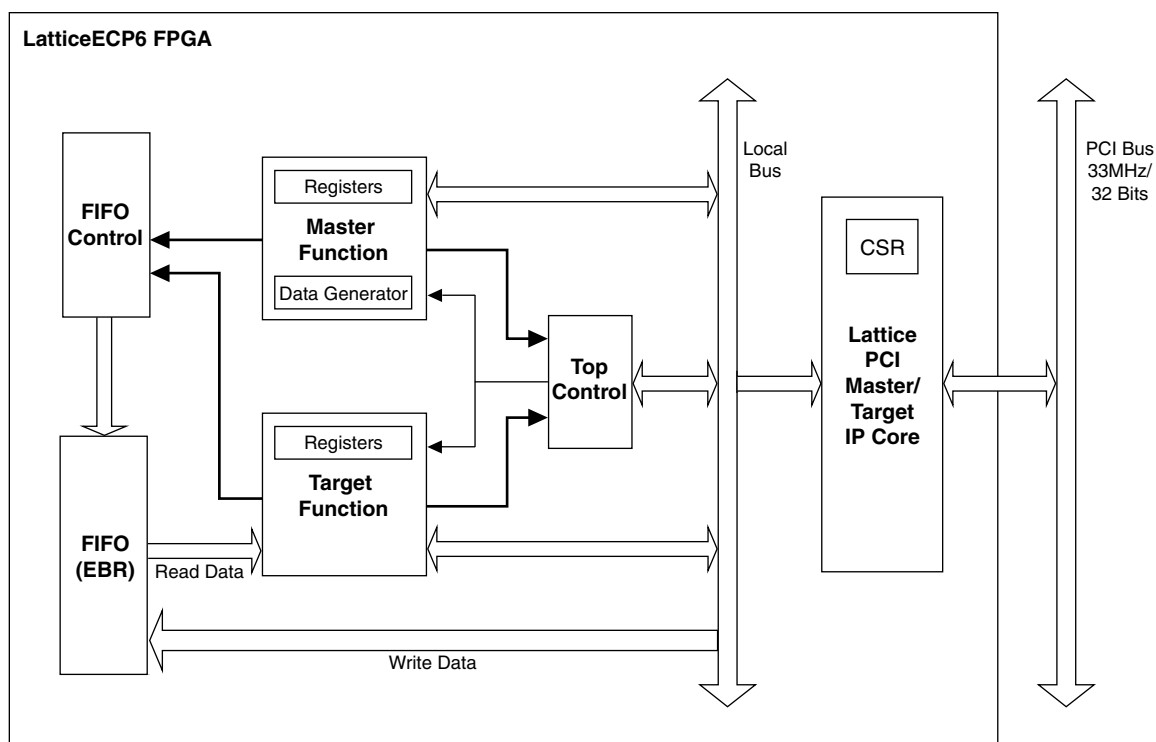
The screenshot shows the 'Function Test (Device Number : 0)' GUI. It is divided into several sections:

- Configuration Space:** Includes fields for Offset (hex): 00000000, Bits Width: 32 bits, and Data (hex): 00000000. Buttons for Read, Write, and Scan Configuration Space are present.
- I/O or Memory Space:** Includes fields for BAR (dec): 0, Offset (hex): 00000000, Bits Width: 32 bits, and Data (hex): 00000000. Buttons for Read and Write are present.
- DMA:** Includes a field for Length (hex): 00000000 and buttons for DMA Read and DMA Write.
- Interrupt:** Includes a button for 'Generate the interrupt' and a 'Data' section with a 'Set DMA Data' button.
- Block Write/Read:** Includes fields for BAR (dec): 0, Offset (hex): 00000000, and Length (hex): 00000000. It has sections for 'Write file' and 'Read file', each with a 'Browse' and 'View' button. Buttons for 'Block Read' and 'Block Write' are also present.
- Results:** A large text area on the right shows 'Generate Interrupt' and 'Interrupt Acknowledge'. A 'Clear Result' button is at the bottom right.
- Exit:** A button at the bottom center.

## Demo Logic Functions

Figure 24 shows the Lattice PCI Device Demo Logic block diagram. The Demo Logic is composed of logic functions supporting the PCI Master/Target IP core. For further information on these functions, please refer to the *PCI Core User's Guide* on the Lattice website at [www.latticesemi.com](http://www.latticesemi.com).

**Figure 24. PCI Master/Target IP Core + Demo Logic Block Diagram**



## PCI Master/Target 33 MHz/32-bit IP Core BARs

The three Base Address Registers implemented in the PCI Master/Target IP Core are described in Table 1.

**Table 1. PCI Master/Target IP Core BARs**

| BAR | Memory / I/O Space | Size   | Access | Description                                                                                                                                                                                                                              |
|-----|--------------------|--------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0   | Memory             | 0x4000 | R/W    | Represented as 32-bit wide FIFO in Figure 24. Physically, the FIFO is implemented as EBRs in the LatticeEC6 FPGA. Data is stored in the FIFO during DMA transfer. Each location is also read/write accessible via the Test Function GUI. |
| 1   | I/O                | 0xFF   | R/W    | General-purpose I/O.                                                                                                                                                                                                                     |
| 2   | I/O                | 0xFF   | R/W    | Demo Registers. Implements the registers to support interrupt and DMA functions.                                                                                                                                                         |

## Demo Registers Description

The Demo Registers and supporting hardware/logic modules are used by the driver/software to run the demo applications.

**Table 2. Demo Registers**

| BAR | Offset | Name                       | Access | Bits   | Description                                                                                                                                                             |
|-----|--------|----------------------------|--------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2   | 0x0    | Interrupt Control Register | R/W    | [0]    | When 0, the demo logic will generate the interrupt. When 1, the demo logic will clear the interrupt.                                                                    |
|     |        |                            |        | [1]    | When 1, the demo logic will clear the DMA interrupt                                                                                                                     |
|     |        |                            |        | [31:2] | Reserved                                                                                                                                                                |
| 2   | 0x4    | Interrupt Status Register  | R      | [0]    | Software-initiated interrupt<br>1: Interrupt Generated<br>0: None                                                                                                       |
|     |        |                            |        | [1]    | DMA-generated interrupt<br>1: Interrupt Generated<br>0: None                                                                                                            |
|     |        |                            |        | [31:2] | Reserved                                                                                                                                                                |
| 2   | 0x8    | Interrupt Mask Register    | R/W    | [0]    | 1: Mask the software interrupt. Software-initiated interrupt.is not recognized.<br>0: None                                                                              |
|     |        |                            |        | [1]    | 1: Mask the DMA interrupt. Interrupt generated during DMA transfers is not recognized.<br>0: None                                                                       |
|     |        |                            |        | [31:2] | Reserved                                                                                                                                                                |
| 2   | 0x10   | DMA Control Register       | R/W    | [0]    | 1: Starts the DMA transfer.                                                                                                                                             |
|     |        |                            |        | [1]    | 1: DMA read mode<br>0: DMA write mode.                                                                                                                                  |
|     |        |                            |        | [31:2] | Reserved                                                                                                                                                                |
| 2   | 0x14   | DMA Status Register        | R      | [31:0] | This register contains the status bits of the DMA process.<br>Bit [0] = 1. DMA transfer is in progress.<br>Bit [1] = 1. Indicates there is an error in the DMA process. |
| 2   | 0x18   | DMA Address Register       | R/W    | [31:0] | This register contains the starting address for the DMA transfer.                                                                                                       |
| 2   | 0x1C   | DMA Length Register        | R/W    | [31:0] | The data length for the DMA transfer. Currently limited to maximum of 0x100(hex).                                                                                       |
| 2   | 0x20   | Reserved                   |        | [31:0] | Reserved for future use.                                                                                                                                                |

## Interrupt Generation and Status

Interrupts can be masked or un-masked. The Interrupt Mask Register controls this function as described in Table 2.

If un-masked, there are two ways an interrupt is initiated:

1. By software
2. End of DMA transfer

The software method is used to verify the interrupt function and simulate the hardware to generate the interrupt. When the software sets bit [0] of the Interrupt Control Register (BAR2/Offset 0) 0, the Demo Logic will generate the hardware interrupt. The interrupt is cleared when the software writes a 1 to this same bit location. The status for software-initiated interrupt is indicated via the Interrupt Status Register, bit[0] as described in Table 2.

When the DMA transfer is completed, an interrupt is also generated. In this case it is purely hardware. This interrupt is cleared when the bit [1] of the Interrupt Control Register is set to 1.

## DMA Transfers

The following registers must be initialized with the appropriate values.

- DMA Address Register
- DMA Length Register
- DMA Direction Register
- DMA Control Register

The DMA Control Register should be the last one accessed and the only to start the DMA transfer.

If the demo logic is used, the registers' configuration can be completed by loading the default file Reg.map.

## Reg.map File

The Reg.map file contains the initial Demo Register values to execute the DMA transfer successfully when running the demo via the GUI Test applications software.

### The Register Config File (Reg.map):

```
Lattice
;Lattice PCICORE Test Tool Register Map File
;
; Register Width
; (B)yte --- 8 bits
; (W)ord --- 16 bits
; (D)word --- 32 bits
;
; Access Method
; R --- Read Only
; W --- Write Only
; A --- Read and Write

;Interrupt Status Register
Reg:0 Bar=2,Offset=0x4,RegWidth=D,BitStart=0,BitWidth=32,Access=R

;Interrupt Acknowledge Register
Reg:1 Bar=2,Offset=0x0,RegWidth=D,BitStart=0,BitWidth=32,Access=W

;Interrupt Mask Register
Reg:2 Bar=2,Offset=0x8,RegWidth=D,BitStart=0,BitWidth=32,Access=A

;DMA Address Register
Reg:3 Bar=2,Offset=0x18,RegWidth=D,BitStart=0,BitWidth=32,Access=A

;DMA Length Register
Reg:4 Bar=2,Offset=0x1c,RegWidth=D,BitStart=0,BitWidth=32,Access=A

;DMA Direction Register
Reg:5 Bar=2,Offset=0x10,RegWidth=D,BitStart=1,BitWidth=1,Access=A

;DMA Control Register
Reg:6 Bar=2,Offset=0x10,RegWidth=D,BitStart=0,BitWidth=1,Access=A

;DMA Status Register
Reg:7 Bar=2,Offset=0x4,RegWidth=D,BitStart=1,BitWidth=1,Access=R
```

---

## How to Use SDK

1. **Initialize the device.** Before the device can be initialized, the **EnumDevice( )** function must be invoked to determine how many devices exist in the system. You can initialize the device by invoking the function **InitDevice( )**.
2. **Access the Configuration Space.** Below are the functions that have access to the configuration space.

|                       |                                                  |
|-----------------------|--------------------------------------------------|
| ReadConfigB( )        | Read a byte from the configuration space.        |
| ReadConfigW( )        | Read a word from the configuration space.        |
| ReadConfigD( )        | Read a double word from the configuration space. |
| ReadDataFromConfig( ) | Read a data buffer from the configuration space. |
| WriteConfigB( )       | Write a byte to the configuration space.         |
| WriteConfigW( )       | Write a word to the configuration space.         |
| WriteConfigD( )       | Write a double word to the configuration space.  |
| WriteDataToConfig( )  | Write a data buffer to the configuration space.  |

*Note: Before you can make the above function calls, you must initialize the device by invoking the InitDevice( ) function.*

3. **Access the I/O Space or Memory Space.** Below are the functions that have access to the I/O space or memory space.

|                    |                                                        |
|--------------------|--------------------------------------------------------|
| ReadBarB( )        | Read a byte from the I/O space or memory space.        |
| ReadBarW( )        | Read a word from the I/O space or memory space.        |
| ReadBarD( )        | Read a double word from the I/O space or memory space. |
| ReadDataFromBar( ) | Read a data buffer from the I/O space or memory space. |
| WriteBarB( )       | Write a byte to the I/O space or memory space.         |
| WriteBarW( )       | Write a word to the I/O space or memory space.         |
| WriteBarD( )       | Write a double word to the I/O space or memory space.  |
| WriteDataToBar( )  | Write a data buffer to the I/O space or memory space.  |

*Note: Before you can make the above function calls, you must initialize the device by invoking the InitDevice( ) function.*

4. **Process the Interrupt.** You can install the ISR (interrupt service routine) by invoking the **RegISRForApp( )** function. The ISR prototype reads:

```
void ApplISR(void *data)
```

Before the ISR can be installed, the interrupt registers must be set. The following functions can be used to set the registers:

|                     |                                                                                |
|---------------------|--------------------------------------------------------------------------------|
| SetIntrStatusReg( ) | Set the interrupt status register to identify whether the interrupt generates. |
| SetIntrMaskReg( )   | Set the interrupt mask register to disable or enable interrupt.                |
| SetIntrAckReg( )    | Set the interrupt acknowledge register to clear the interrupt.                 |

5. **DMA Operation.** The DoDMA( ) function is used for DMA operation. When the DMA operation completes, the device generates an interrupt message. Therefore, the function DMADone( ) must be invoked in the ISR. Otherwise, the function DoDMA( ) will be blocked. Before DMA operation, DMA registers must be set. The following functions can be used to set the registers:

|                       |                                 |
|-----------------------|---------------------------------|
| SetDmaAddrReg( )      | Set the DMA address register.   |
| SetDmaLenReg( )       | Set the DMA length register.    |
| SetDmaDirectionReg( ) | Set the DMA direction register. |

|                    |                               |
|--------------------|-------------------------------|
| SetDmaCtrlReg( )   | Set the DMA control register. |
| SetDmaStatusReg( ) | Set the DMA status register.  |

*Note: For the register map, see the Demo Registers Description section of this document.*

## Design Example

```
#include "pcicore.h"

void AppIsr(void *data)
{
    unsigned long value, status;
    ReadBarD(0, 2, 4, &status);

    if(status)
    {
        value = 3;
        WriteBarD(0, 2, 0, &value);

        if(status & 0x2)
            DMADone();
    }
}

int main(int argc, char* argv[])
{
    unsigned long devnum, I, value;
    unsigned char buf[256];

    devnum = EnumDevice();
    if(devnum == 0)
    {
        printf("Device can not be found\n");
        return 1;
    }

    InitDevice(0);

    // Access the configuration space
    ReadDataFromConfig(0, 0, buf, 256);
    for(i = 0; i < 256; i++)
    {
        printf("0x%02x ", *(buf + i));
        if(!((i + 1) % 4))
            printf("\n");
    }

    for(i = 0; i < 256; i++)
        *(buf + i) = (unsigned char)i;

    // Access the I/O or memory space
    WriteDataToBar(0, 0, 0, buf, 256);

    ReadDataFromBar(0, 0, 0, buf, 256);
}
```

---

```

for(i = 0; i < 256; i++)
{
    printf("0x%02x ", *(buf + i));
    if(!((i + 1) % 8))
        printf("\n");
}

// Config the register
SetIntrAckReg      (0, 2, 0x00, REG_WIDTH_32, REG_WRITE_ONLY, 32, 0);
SetIntrStatusReg   (0, 2, 0x04, REG_WIDTH_32, REG_WRITE_READ, 32, 0);
SetIntrMaskReg     (0, 2, 0x08, REG_WIDTH_32, REG_WRITE_READ, 32, 0);
SetDmaCtrlReg      (0, 2, 0x10, REG_WIDTH_32, REG_WRITE_READ, 1, 0);
SetDmaDirectionReg (0, 2, 0x10, REG_WIDTH_32, REG_WRITE_READ, 1, 1);
SetDmaAddrReg      (0, 2, 0x18, REG_WIDTH_32, REG_WRITE_READ, 32, 0);
SetDmaLenReg       (0, 2, 0x1C, REG_WIDTH_32, REG_WRITE_READ, 32, 0);
SetDmaStatusReg    (0, 2, 0x14, REG_WIDTH_32, REG_WRITE_READ, 32, 0);

RegISRForApp(0, AppIsr, NULL);

// Generate the software interrupt
value = 2;
WriteBarD(0, 2, 0, &value);

// DMA operation
for(i = 0; i < 256; i++)
    *(buf + i) = 0xFF - (unsigned char)i;

DoDMA(0, buf, 256, DMA_WRITE);

DoDMA(0, buf, 256, DMA_READ);

for(i = 0; i < 256; i++)
{
    printf("0x%02x ", *(buf + i));
    if(!((i + 1) % 8))
        printf("\n");
}

UnregISRForApp(0);

UninitDevice(0);

}

```

## Technical Support Assistance

Hotline: 1-800-LATTICE (North America)  
           +1-503-268-8001 (Outside North America)  
 e-mail: techsupport@latticesemi.com  
 Internet: [www.latticesemi.com](http://www.latticesemi.com)

---

---

## Revision History

| Date          | Version | Change Summary   |
|---------------|---------|------------------|
| February 2007 | 01.0    | Initial release. |