# *PCIe-BaseLab*

# Evaluation board for the PCI Express Bus

Version 1.02
July 2016

Notes on scope of delivery, installation, hardware properties and extendibility and on programming using API functions

## Contents

# 1. Introduction

## General properties

*PCIe-BaseLab* is an indispensible aid to the development of add-on cards for PCs and other computer systems equipped with the PCI Express bus (PCIe). The card allows rapid and uncomplicated testing of newly developed electronic circuits on the PCIe bus.

*PCIe-BaseLab* works with the universal PCI Express endpoint controller *PEX8311 from PLX Technology, Inc.,* its peripherals and a high-performance programmable logic device (PLD), manufacturer *Lattice Semiconductor*. With the PEX8311 controller, *PCIe-BaseLab* is compatible with the requirements of the PCI Express specification, Revision 1.0. The card is fully populated, tested and ready for use.

System designers can mount their hardware directly onto the perforated grid of the removable daughterboard and begin testing. Time spent dealing with signal interchanges and the technical properties of the PCIe bus system can be reduced to a minimum.

The PLD includes various pre-installed sample applications:

- a synchronous RAM,
- a FIFO,
- a 16 bit input/output register.

The source code for the sample applications is provided and can be modified by the user.

Particularly useful for hardware developers and measurement engineers:
All of the controller's local connector pins are pre-connected to the PLD and executed as shrouded headers from which they are easily accessible. 106 of the PLD's 193 I/O pins are also laid out on shrouded headers and are freely accessible.

Various types of daughterboard are available for the *PCIe-BaseLab*. Currently these have 0.1 inch perforated grids, future versions will have universal SMD footprints and custom formats can also be produced. The latter offers the option of using *PCIe-BaseLab* as a finished module together with application-specific hardware in small and mid-sized production series.

Use of *PCIe-BaseLab* does not require payment of licensing fees for a core and does not require membership of PCISIG. Sub-vendor and subsystem IDs for explicitly labeling vendor hardware can be obtained free of charge from the PCIe controller vendor.

The documentation for *PCIe-BaseLab* is rounded off by circuit diagrams, population and connection plans and PLD source code.

## Scope of delivery

The product as supplied consists of the following components:

- *PCI Express Bus Evaluation Board* with attached daughterboard (0.1 inch land grid),
- *adaptor cable* for supplying power to the daughterboard via a SATA power supply connector,
- *CD-ROM*: Windows driver, monitor program, sample applications including source text and developer files (Header, Lib, DLL), circuit diagrams, data sheets, VHDL-source code and this technical manual.

# 2. Installation

### Software

The *PCIe-BaseLab* software can be used under Windows 2000, Windows XP, Windows Vista and Windows 7. The Windows driver is a WDM driver. 32-bit and 64-bit operating systems are supported.

WARNING: You must have administrator privileges to install the software!

To install the software, select one of the following procedures (depending on version supplied):

- Copy the entire directory structure from the CD to the destination directory on your PC.
- Unzip the file by executing the self-extracting program 'pplab_v1005.exe'.
- Start the installation by running 'setup.exe'.

The following sub-folders will be present in the *PCIe-BaseLab* software folder:

- bin: the driver API DLL 'PPLABAPI.DLL' (→ section 5) and
  binary files for the sample applications (→ section 8),
- dev: the two C header files 'PPLABAPI.H' and 'PPLABERR.H' (→ section 5),
- doc: the manual 'PPLAB-v10_Manual.PDF',
- drv: the Windows driver 'PPLAB_E.SYS' and two associated INF files (→ following section),
- lib: the import library 'PPLABAPI.LIB' (→ section 5),
- mon: the monitor program 'PPLABMON.EXE' and associated files (→ section 7),
- smp: the source code files for the sample applications (→ section 8).

Tip: We recommend installing the software before installing the *PCIe-BaseLab* card in the PC. This allows the Windows driver to be assigned immediately after restarting the system.

### Hardware

The hardware is installed by inserting the card into a 1x, 4x, 8x or 16x PCI Express slot in a PC or other computer with PCIe slots. Regulations relating to ESD safety and protection from contact with parts which may carry hazardous electrical voltages should be observed.

**The computer must be switched off and disconnected from the mains power supply before installing / removing the *PCIe-BaseLab* card.**

The integrated LED (LED 1) on the *PCIe-BaseLab* card indicates that the physical PCIe interface connection has been established after switching on the computer. If it lights up, this may be taken as an initial indication that the hardware has been correctly installed.

*PCIe-BaseLab* can also be run via a PCIe bus extender. This allows the card to be inserted and removed while the PC is running (Hot-Swapping) without resulting in loss of configuration data. (*PCIe-Bus-Extender* with *PCFaceSwitch-Software*, manufacture: HK Meßsysteme GmbH)

### Assigning the Windows driver

After restarting the computer, the operating system finds the new hardware and requests assignment of a suitable driver. Note that the controller used is made up of two components:

- a 'PCI-PCI bridge' (PCI8111) and
- a 'PCI device' (PCI9056).

WARNING: A suitable driver must be assigned to each component!

In order to carry out driver assignment, do not use the 'Automatic installation' option recommended by Windows. In the 'Add Hardware Wizard' select the 'drv' sub-folder in the *PCIe-BaseLab* software folder when requested to select the driver location. The following two files in this folder:

- PLXBRIDGE.INF and
- PPLAB_E.INF

ensure that the correct drivers are assigned. These are

- PCI.SYS – the standard Microsoft driver for the PCI bus, in this case as a driver for the 'PCI-PCI bridge',
- PPLAB_ESYS – the specific Windows driver for the 'PCI device' *PCIe-BaseLab.*

If another Windows driver (e.g. the generic PLX driver PCI9056.SYS) is assigned to the 'PCI device', reassign the correct driver by selecting 'Update driver' in Device Manager and selecting 'PPLAB_E.INF' in the 'drv' folder. If driver installation has been performed correctly, a device class 'PCIe-BaseLab' and the installed 'PCIe-BaseLab' devices will be displayed in Device Manager.

# 3. Hardware properties and extendibility

The block diagram (see appendix) provides an introductory description. The hardware can be functionally divided into five components:

- PCIe Endpoint Controller PEX8311,
- Serial Configuration EEPROM,
- Programmable Logic Device (PLD), ispXPLD5768,
- 'In-System Programmable' Interface (ISP),
- Pin Header Block (User Interface)

**PCIe endpoint controller PEX8311**

The *PEX8311* is a universal PCIe Endpoint controller for the serial PCIe bus equipped with a 1x link. It functions as a bridge between the PCIe-bus and specific user systems on the local bus. The *PEX8311* processes all typical signals and access mechanisms on the bus. It translates these signals into a universal control, address and data interface to which user specific memory and I/O units can be connected. It has two interfaces which are designated as follows:

- this PCIe Bus Interface
- this Local Bus Interface

The *PEX8311* controller interfaces are of differing significance to *PCIe-BaseLab* card users and are described below.

**PCIe bus interface**

This *PCIe-BaseLab interface* is used to couple the controller to the PCI Express bus. It is fully wired on the PCB, no further action on the part of the user is required.

**Local bus interface**

This *Local Bus interface* is important for the user, as this is where he will connect his system application. It is designed as a universal bus and allows the operation of hardware peripherals with data bus sizes of 8, 16 or 32 bits and a maximum address bus size of 32 bits.

User-specific components control data exchange via this Local Bus interface using classical signals /LHOLD, /HOLDA, /WAIT, /READY or /LW/R.

The connected peripheral can itself consist of a microprocessor system or - the simplest case - be designed as a data latch.

The PCIe Endpoint controller architecture also supports the integration of memory. Up to 4 Gbytes can be addressed per address region; two local address regions are permissible.

The *PEX8311* has an internal register set for saving initialization data, configuring, activating and deactivating operating modes and exchanging data. The register set can be accessed both via the interface and via the bus interface. The PCI controller functions are therefore transparent to and can be used by both sides.

Two independent DMA channels are provided for rapid data transfer with no Host-CPU (Bus Master Transfer Mode) involvement. The start addresses and transfer counters of these channels are configured using registers.

ROMs with parallel interfaces can be connected to the *PEX8311* for BIOS extension.

The PEX8311 controller also allows user-specific generation of interrupts, which can originate from either the local or PCI side.

A complete technical description of the *PEX8311* can be found in the technical manual, which is published and maintained by the manufacturer of the controller. This can be downloaded from the website *PLX Technology, Inc.* and is included on the CD-ROM supplied with this product.

Warning: We cannot guarantee that data sheets produced by other vendors and supplied with this product are up-to-date. They should be used as provisional information only. It is advisable to obtain current data sheets and manuals from the relevant manufacturer and to use these as the basis for your work with these components. Relevant addresses can be found in the appendix to this manual.

### Serial configuration

*PCIe-BaseLab* works with a 2 Kbyte serial EEPROM inserted into a slot on the PCB. It contains essential configuration data which initializes the controller specifically for the *PCIe-BaseLab*. The EEPROM can be edited and overwritten. The program *PPLABMON.EXE*, included in the software package supplied with this product, is a practical editor for the serial EEPROM code.

### Programmable logic device (PLD), ispXPLD5768

The PLD used on this *PCIe-BaseLab* card is a member of the *ispXPLD 5000MX* family from *Lattice Semiconductor*, distinguished by its high performance and flexibility. As well as extensive logic, it is also possible to implement single or dual ported RAM or FIFO memory in the PLD's multi function blocks. The inputs and outputs can operate according to various interface standards (TTL, CMOS, LVCMOS, LVDS, HSTL or SSTL). Integrated PLL systems allow easy clock management. The PLD series has integrated Flash memory for configuration, which makes the PLD ready to use immediately after connecting the supply voltage. The PLD used on the *PCIe-BaseLab* card has 768 macro cells (Flip-flops), a maximum of 384 Kbits of configurable RAM and can be clocked at a maximum system clock frequency of 250 MHz. Its outputs deliver 3.3V. The inputs are 5V compatible. The PLD can be programmed via the integrated interface on the *PCIe-BaseLab* card. No external programming device is required. Further information on the PLD is available from the *Lattice Semiconductor* website. (Web addresses can be found in the appendix to this manual.)

### In-system programmable (ISP)

This ISP interface (X1, JEDEC Support) allows PLD firmware to be downloaded without the need to edit the PLD on an external programming device. It is able to remain integrated within the system (Therefore ‘in system programmable’, ISP). The required software *ispVMSystem* can be downloaded free of charge from the Lattice Semiconductor website. A parallel port or USB programming adapter can be ordered from your local *Lattice* component dealer.

### Pin header block (User Interface)

The *pin header block* consists of dual row shrouded headers in a 2 mm grid (2 headers with 2x50 pins each, 1 header with 2x20 pins). These are placed at the outer edges of the card and constitute the interface to user-specific hardware the *User Interface*.

This interface provides access to all the *PEX8311* controller's local address, data and control lines and to 106 of the installed PLD's 193 I/O pins. Further connections allow external clock signals to be picked up or fed in. Earth pins are included at regular intervals.

The daughterboard supplied is designed to be able to be inserted onto the shrouded pin header block on the *PCIe-BaseLab* card and screwed to the card. All pins are translated onto the daughterboard and are available for connecting user-specific hardware to the daughterboard. Warning: Power supply and earth connections are not automatically connected to the daughterboard's power supply layers. The additional cable provided is required. This can be connected to a SATA power supply connector from the host computer or to prepared solder connectors on the *PCIe-BaseLab* card as required. All voltage feeds (with the exception of the earth connections) on the daughterboard include replaceable fuses, intended to protect user electronics and the host computer power supply module from overvoltages. Colored LEDs signal that the power supply voltages on the daughterboard are on.

The daughterboard can be separated from the *PCIe-BaseLab* card, but due to the force required to do so should be removed with care and with the aid of a tool (please lever off in parallel using a suitable screwdriver).

# 4. The PLD sample applications

The sample applications are in all cases accommodated on the PLD as 'glue logic'. They are intended to illustrate sample hardware applications with which the *PCIe-BaseLab* can be equipped without requiring the installation of additional components on the daughterboard.

Users can adapt the pre-installed sample applications to their requirements (within the realms of the technically possible) and/or implement their own logic on the PLD.

The source code for the sample applications is included with the product. Software for editing and compiling this code can be downloaded from the *Lattice Semiconductor* website.

The monitor program supplied with this product can be used for communication with the sample applications for test purposes. Further information can be found in section 8.

Further information on the hardware architecture of the pre-installed sample applications is given below.

**Synchronous RAM**

The *RAM* has a capacity of 24 Kbytes and is organized as 6k x 4 bytes. The hardware is laid out for byte, word or dword read and write access. Burst transfers are supported. See section 5 for information on the address range of the RAM.

The integrated RAM can also be written to and read from using the monitor program *PPLABMON.EXE* supplied. For more information see section 8*.*

**FIFO**

The *FIFO buffer* sample application is currently not yet included with the *PCIe-BaseLab* as supplied. It will be made available to our customers in the near future.

**16 bit I/O register**

This sample application consists of a *16 bit data register* for outputting static (latched) data and a separate input data path which can be used to query the current logical state of 16 input lines. The 16 bit I/O register sample application uses a total of 32 pins (16 input pins and 16 output pins) of the available I/Os at the pin header block. These are arranged on the header block such that they can be easily bridged bit-by-bit using a short-circuit plug. This allows the data word output via the output latch to be re-read.

The 16 bit I/O latch is accessible via a read/write address in the card's memory address range (see section 5)*.*

The 16 bit I/O register can also be written to and read from using the monitor program *PPLABMON.EXE* supplied (see section 8)*.*

A table showing the positions of the 16 bit register's I/O pins on the header block is given in the appendix.

# 5. The programming interface

The *PCIe-BaseLab* programming interface provides a range of functions which can be utilized by C/C++ programs to communicate with the *PCIe-BaseLab* card. The following files are provided for using these functions:

| | |
|---|---|
| • PPLABAPI.H: | C header file - defines the API functions, |
| • PPLABERR.H: | C header file - defines the error constants, |
| • PPLABAPI.DLL: | dynamic link library - implementation of API functions |
| • PPLABAPI.LIB: | import library for implicitly loading the DLL from VC applications. |

In order to be able to use these API functions, the header file 'PPLABAPI.H' `included' and the API-DLL drivers 'PPLABAPI.DLL' must be loaded.

**Using the files**

The file must be loaded using a statement in the source code in order to be able to call these functions.

The file can be used to look up error codes and does not usually need to be included.

**Loading the drivers API-DLL**

Drivers API-DLL 'PPLABAPI.DLL' can be loaded either

- explicitly using the Win32 functions 'LoadLibrary' and 'GetProcAddress' or
- implicitly using the import library supplied.

The sample program 'LoadDLL' demonstrates explicit loading of the DLL.

Where Microsoft development environments (VC6, Visual Studio xx) are used, the import library supplied can be used to implicitly load the DLL. Implicit loading is used in all of the sample programs with the exception of 'LoadDLL'.

**Function and error code format**

All interface functions conform to a similar schema. The name of the functions always starts with the prefix '`PPLAB_`'.

The return value of the functions is always of type '`DWORD`' and indicates whether the function was successful. A return value of '0' indicates that no errors occurred when executing the function. A value other than '0' indicates an error and represents an error code which provides information on the source of the error. Possible error codes are defined in the file and utilize values between `0x20010000 and 0x20FF0000`. In addition, operating system error codes are sometimes added. These make use of bits `0 to 11`.

If a function returns data, this is made available via reference parameters (address pointers). This allows multiple pieces of information to be returned simultaneously.

**Memory configuration**

By default the card has 32 Kbytes of memory. This memory is used to demonstrate potential applications. These applications include:

- writing and reading many Kbytes of data
- setting and querying I/O pins
- starting continuous data transfer via DMA

Information on the appropriate card configuration can be found in the accompanying documentation.

Sample configuration 1:

```
0x0000..0x5FFF: RAM (24 kBytes)
0x6000..0x6001: I/O-REGISTER (16 Bits)
0x6002..0x7FFF: unused
```

**Methods for accessing memory**

Three methods for accessing *PCIe-BaseLab* card memory are available:

- KERNEL
- MAPPED
- DMA

The KERNEL method is the standard method for accessing memory areas on an external expansion card. Memory is accessed via the card driver at the kernel level. The driver deals with synchronization of simultaneous memory access. The relevant API functions are `PPLAB_ReadMemory` and `PPLAB_WriteMemory`.

The MAPPED method makes use of the API functions `PPLAB_MapMemory` and `PPLAB_UnmapMemory`. The corresponding memory area on the *PCIe-BaseLab* card is 'shown' in the relevant application's address space. It is then possible to access this memory area directly, bypassing the kernel level. Explicit synchronization is required to prevent simultaneous memory access. The sample program illustrates this method.

The DMA method is used to transfer large quantities of data using the PLX controller's DMA channels. It utilizes the API functions `PPLAB_ReadPlxRegister` and `PPLAB_WritePlxRegister`. The sample program demonstrates the required procedure.

# 6. API Functions - reference

The programming interface encompasses the following API functions:

- Querying the number of active *PCIe-BaseLab* cards
  `PPLAB_GetNumberOfDevices`
- Opening and closing a *PCIe-BaseLab* card
  `PPLAB_OpenDevice`
  `PPLAB_CloseDevice`
- Querying the driver version
  `PPLAB_GetDriverVersion`
- Querying information on the *PCIe-BaseLab* card PCI slot
  `PPLAB_GetDeviceProperties`
- Resetting a *PCIe-BaseLab* card
  `PPLAB_ResetDevice`
- Reading and writing to a *PCIe-BaseLab* card's configuration registers
  `PPLAB_ReadPciRegister`
  `PPLAB_WritePciRegister`
- Reading and writing to a *PCIe-BaseLab* card's PLX controller registers
  `PPLAB_ReadPlxRegister`
  `PPLAB_WritePlxRegister`
- Reading and writing a EEPROM *PCIe-BaseLab* card's content
  `PPLAB_ReadEEPROM`
  `PPLAB_WriteEEPROM`
- Reading and writing to a *PCIe-BaseLab* card's memory area
  `PPLAB_ReadMemory`
  `PPLAB_WriteMemory`
- 'Showing' and 'hiding' a memory area on a *PCIe-BaseLab* card
  `PPLAB_MapMemory`
  `PPLAB_UnmapMemory`
- Querying the physical address of a *PCIe-BaseLab* card's memory areas
  `PPLAB_GetPhysicalAddressOfUserRegion`
- Allocating and releasing continuous memory areas
  `PPLAB_AllocateContMemory`
  `PPLAB_FreeContMemory`

### Querying the number of active *PCIe-BaseLab* cards

DWORD **PPLAB_GetNumberOfDevices** (DWORD* pNumberOfDevices);

This function is used to query the number of *PCIe-BaseLab* cards currently active in the host computer. A `DWORD` variable address should be passed to the function. After the function has successfully executed, this variable `pNumberOfDevices` is set to the number of active cards.

The driver supports up to 16 *PCIe-BaseLab* cards per PC.

This function is usually the first API function called after starting a program in order to determine whether any *PCIe-BaseLab* cards are active in the host computer.

Example:

```
DWORD error, deviceNumber;
error = PPLAB_GetNumberOfDevices(&deviceNumber);
if (error)
… // error handling
else
printf ("Active BaseLab-Cards found: %d\n", deviceNumber);
```

### Opening a *PCIe-BaseLab* card

DWORD **PPLAB_OpenDevice** (HANDLE* pHDevice, DWORD deviceId);

This function is used to start using a *PCIe-BaseLab* card – the device with the parameter `deviceId` is opened. `deviceId` can take any value between `1` and `16`. If only one card is active, its ID will be `1`, if 2 cards are active their IDs will be `1` and `2`, etc. If the function executes successfully, it returns a handle for the relevant device. A `HANDLE` variable address should be passed to the function `pHDevice` for this purpose.

Since `deviceHandle` is required as a parameter for all other API functions (except `PPLAB_GetNumberOfDevices`), one *PCIe-BaseLab* card must always be opened using this function before other functions can be called.

Opened *PCIe-BaseLab* cards should be closed using the `PPLAB_CloseDevice` function before exiting the program.

Example:

```
DWORD error, deviceId;
HANDLE deviceHandle;
deviceId = 1;
error = PPLAB_OpenDevive (&deviceHandle, deviceId);
if (error);
… // error handling
else
printf ("BaseLab-Card %d opened! (handle=%d)\n", deviceId, deviceHandle);
```

### Closing a *PCIe-BaseLab* card

DWORD **PPLAB_CloseDevice** (HANDLE hDevice);

This function is used to close a *PCIe-BaseLab* card. The `deviceHandle` parameter generated using `PPLAB_OpenDevice` must be passed to this function. `deviceHandle` is no longer valid after executing this function. It is therefore not possible to call further API functions for the card in question after calling `PPLAB_CloseDevice`.

Example:

```
…
error = PPLAB_CloseDriver (deviceHandle);
if (error)
… // error handling
```

### Querying the Windows driver version

DWORD **PPLAB_GetDriverVersion** (HANDLE hDevice, DWORD* pDriverVersion);

This function is used to query the version number of the currently installed Windows driver. As well as `deviceHandle`, the address of a `DWORD` variable which will then contain the version number of the Windows driver must be passed to this function.

Example:

```
…
DWORD driverVersion;
error = PPLAB_GetDriverVersion (deviceHandle, &driverVersion);
if (error)
… // error handling
else
printf ("Current driver version: %x\n", driverVersion);
```

### Querying information about *PCIe* card slots

```
DWORD PPLAB_GetDeviceProperties (HANDLE hDevice, DWORD* pSlotNum, DWORD* pBusNum,
                                  DWORD* pDeviceNum, DWORD* pFunctionNum);
```

This function is used to query information about the PCIe card slot in which the relevant *PCIe-BaseLab* card is inserted. Four `DWORD` variable addresses must be passed to this function. After the function has executed these variables will return the values for slot, bus, device and function number. Where more than one *PCIe-BaseLab* card is active, this information can be used for identification.

Example:

```
…
DWORD slotNum, busNum, devNum, funNum;
error = PPLAB_GetDeviceProperties (deviceHandle, &slotNum, &busNum, &devNum,
&funNum);
if (error)
… // error handling
else
printf ("Device properties: slot=%d, bus=%d, dev=%d, fun=%d\n",
         slotNum, busNum, devNum, funNum);
```

### Resetting a *PCIe-BaseLab* card

```
DWORD PPLAB_ResetDevice (HANDLE hDevice, DWORD resetFlags);
```

Calling this function resets the relevant *PCIe-BaseLab* card. No values have been defined for the parameter `resetFlags` at present and a value of `0` should therefore be passed to this function.

Example:

```
error = PPLAB_ResetDevice (deviceHandle, 0);
if (error)
… // error handling
else
printf ("Device reset done!\n");
```

### Reading from the PCI configuration registers

```
DWORD PPLAB_ReadPciRegister (HANDLE hDevice, DWORD offset, DWORD* pValue);
```

This function is used to read the *PCIe-BaseLab* card's 256 byte PCI configuration register. The parameter `offset` to be passed to this function is the offset in bytes (0, 4, 8 up to 252). A `DWORD` variable address must be passed to this `pValue` function.

Example:

```
DWORD value;
DWORD offset = 0;
error = PPLAB_ ReadPciRegister (deviceHandle, offset, &value);
if (error)
… // error handling
else
printf ("Vendor_Device_ID: 0x%X", value);
```

### Writing to the PCI configuration registers

```
DWORD PPLAB_WritePciRegister (HANDLE hDevice, DWORD offset, DWORD value);
```

This function is used to write to the PCI configuration registers. The offset in bytes (0,4,8..252) must be passed to this function as a `value` parameter.

Please note: Only individual PCI configuration registers are writeable. Ensure that you assign meaningful values to these registers!

Example:

```
DWORD subSystemId_subVendorId = 0x12345678;
DWORD offset = 0x2c;
error = PPLAB_ WritePciRegister (deviceHandle, offset,
subSystemId_subVendorId);
if (error)
… // error handling
```

### Reading the *PEX8311*controller registers

```
DWORD PPLAB_ReadPlxRegister (HANDLE hDevice, DWORD offset, DWORD* pValue);
```

This function can be used to read the PEX8311 controller registers. These registers include the local configuration register and runtime and DMA registers (see PLX controller documentation). The parameter `offset` to be passed to this function is the offset in bytes. A `DWORD` variable address must be passed to this `pValue` function.

Example:

```
DWORD intCSR;
DWORD offset = 0x68;
error = PPLAB_ ReadPlxRegister (deviceHandle, offset, &intCSR);
if (error)
… // error handling
else
printf ("Interrupt_Control_Status: 0x%X", intCSR);
```

### Reading the *PEX8311* controller registers

```
DWORD PPLAB_WritePlxRegister (HANDLE hDevice, DWORD offset, DWORD value);
```

This function is used to write to the PEX83111 controller registers. The offset in bytes must be passed to this function as a `value` parameter.

Ensure that only meaningful values are assigned to these PEX8311 registers!

Example:

```
DWORD p2lDoorbell = 0x00000001;
DWORD offset = 0x60;
error = PPLAB_ WritePlxRegister (deviceHandle, offset, p2lDoorbell);
if (error)
… // error handling
```

### Reading data from the serial EEPROM

`DWORD` **`PPLAB_ReadEeprom`** `(HANDLE hDevice, DWORD offset, DWORD* pValue);`

This function is used to read data from the *PCIe-BaseLab* card's EEPROM. The parameter `offset` to be passed to this function is the offset in bytes (0,4,8..0x60). A `pValue` variable address must be passed to this function.

Example:

```
DWORD lat_gnt_pin_line;
DWORD offset = 0x0B;
error = PPLAB_ ReadEeprom (deviceHandle, offset, &lat_gnt_pin_line);
if (error)
… // error handling
else
printf ("MaxLat_MinGnt_IntPin_IntLine: 0x%X", lat_gnt_pin_line);
```

### Writing data to the serial EEPROM

`DWORD` **`PPLAB_WriteEeprom`** `(HANDLE hDevice, DWORD offset, DWORD value);`

This function is used to write data to the EEPROM. The offset in bytes (0,4,8..0x60) must be passed to this function as a `value` parameter.

IMPORTANT:

Before changing this data, you should back-up the original EEPROM data. This can be done using the monitor program PPLABMON. Ensure that you assign meaningful values to these EEPROM registers! Incorrect EEPROM content may lead to *PCIe-BaseLab* card and system errors.

Example:

```
DWORD lat_gnt_pin_line = 0x00000100;
DWORD offset = 0x0B;
error = PPLAB_ WriteEeprom (deviceHandle, offset, lat_gnt_pin_line);
if (error)
… // error handling
```

### Reading the memory area

`DWORD` **`PPLAB_ReadMemory`** `(HANDLE hDevice, DWORD offset, DWORD* pValue);`

This function is used to read from the *PCIe-BaseLab* card memory. The parameter `offset` to be passed to this function is the offset in bytes (0,4,8..). A `pValue` variable address must be passed to this function.

Example:

```
DWORD ioRegister;
DWORD offset = 0x6000;
error = PPLAB_ ReadMemory (deviceHandle, offset, &ioRegister);
if (error)
… // error handling
else
printf ("IORegister: 0x%X", ioRegister);
```

### Writing to the memory area

`DWORD` **`PPLAB_WriteMemory`** `(HANDLE hDevice, DWORD offset, DWORD value);`

This function is used to write to the *PCIe-BaseLab* memory. The offset in bytes (0,4,8..) must be passed to this function as a `value` parameter.

See also the section 'Memory configuration'.

Example:

```
DWORD ioRegister = 0x0000FFFF;
DWORD offset = 0x6000;
error = PPLAB_ WriteMemory (deviceHandle, offset, ioRegister);
if (error)
… // error handling
```

### 'Showing' a memory area in the application address space

DWORD **PPLAB_MapMemory** (HANDLE hDevice, USER_REGION userRegion, DWORD* pVirtAddress);

This function is used to 'show' a memory area on the *PCIe-BaseLab* card in the application's address space. The parameter `userRegion` selects either `USER_REGION_0` or `USER_REGION_1`. The function sets a `DWORD` variable, the address of which is passed to the function `pVirtAddress`, to the virtual address of this memory area. This address can then be used to write to or read from the memory area on the card from the application directly.

WARNING: As currently configured, the *PCIe-BaseLab* card has only one memory area. The user_region parameter should therefore always take the value USER_REGION_0!

Example:

```
DWORD virtAddress;
error = PPLAB_ MapMemory (deviceHandle, USER_REGION_0, &virtAddress);
if (error)
… // error handling
else
{
DWORD* pMem = (DWORD*)(virtAddress + 4);        // set address to offset 4
value = *pMem;                                 // direct read DWORDs
pMem++;                                        // set address to offset 8
*pMem = 0x12345678;                            // direct write DWORDs
}
```

### 'Hiding' a memory area from the application address space

DWORD **PPLAB_UnmapMemory** (HANDLE hDevice, USER_REGION userRegion, DWORD virtAddress);

This function reverses the effect of 'showing' a memory area using the `PPLAB_MapMemory` function. The parameters to be passed to this function are the `userRegion` and the virtual address returned by `PPLAB_ MapMemory`.

After this function has been called, it is no longer possible to access the memory area directly.

Example:

```
error = PPLAB_ UnmapMemory (deviceHandle, USER_REGION_0, virtAddress);
if (error)
… // error handling
```

### Querying the physical address of user regions

DWORD **PPLAB_GetPhysicalAddressOfUserRegion** (HANDLE hDevice, USER_REGION userRegion DWORD* pPhysAddress);

In order to use a *PCIe-BaseLab* card's DMA channels, the physical addresses of the memory areas involved must be available.

This function is used to query the physical address for each user region. The parameter `userRegion` selects either `USER_REGION_0` or `USER_REGION_1`. The function sets a `DWORD` variable, the address of which is passed to the function, to the physical address of this memory area. This address can then be used as a 'local' address when initializing a DMA transfer.

WARNING: As currently configured, the *PCIe-BaseLab* card has only one memory area. The user_region parameter should therefore always take the value `USER_REGION_0`.

Example:

```
DWORD physLocalAddr;
error = PPLAB_GetPhysicalAddressOfUserRegion (hDevice, USER_REGION_0,
&physLpcalAddr);
if (error)
… // error handling
else
printf ("psysical address of region0: 0x%08X \n", physLocalAddr);
```

### Allocating a continuous memory area

```
DWORD PPLAB_AllocateContMemory (HANDLE hDevice, DWORD size, DWORD* pVirtAddress,
                                DWORD* pPhysAddress);
```

For DMA transfers, it is useful to be able to use a memory area which is physically continuous. This removes the need to determine the physical addresses of all of the pages of memory involved.

This function requests a physically continuous memory area. The parameter `size` passed to this function is the amount of memory required in bytes – multiples of 4096 (the size of one page of memory) are recommended. The function sets a `DWORD` variable, the address of which is passed to the function `pVirtAddress`, to the virtual address of this memory. This address can then be used to write to or read from this memory area from the application directly. The `DWORD` variable passed to this function `pPhysAddress` contains the physical address of this memory area. This address can then be used as the PCI address when initializing a DMA transfer.

The size of the memory area should not exceed 0x7FFFFF (decimal: 8388607, 8 MB-1) – this is the largest permissible number of bytes for a DMA transfer (Bits 0..22 DMASIZx-Register).

The current version of the Windows driver allows 32 continuous memory areas to be requested.

Example:

```
DWORD memSize, virtPCIAddr, physPCIAddr;
memSize = 1024*1024*2;             // request off 2 MBytes
error = PPLAB_AllocateContMemory (hDevice, memSize, &virtPCIAddr,
&physPCIAddr);
if (error)
… // error handling
else
printf ("virtual: 0x%08X, physical: 0x%08X \n", virtPCIAddr, physPCIAddr);
```

### Releasing a continuous memory area

```
DWORD PPLAB_FreeContMemory (HANDLE hDevice, DWORD virtAddress);
```

This function releases the memory area requested using `PPLAB_AllocateContMemory`. The parameter `virtAddress` passed to the function is the virtual address.

Example:

```
error = PPLAB_FreeContMemory (hDevice, virtAddr);
if (error)
… // error handling
```

# 7. The sample programs (C++)

The sample programs demonstrate the procedure for using the various API functions provided by the *PCIe-BaseLab* programming interface. These are C++ console applications the coding for which has been deliberately kept simple. Each sample program is associated with a file of the same cpp name. Programming can be carried out using any C++ compiler. The import library can be used for Microsoft tools.

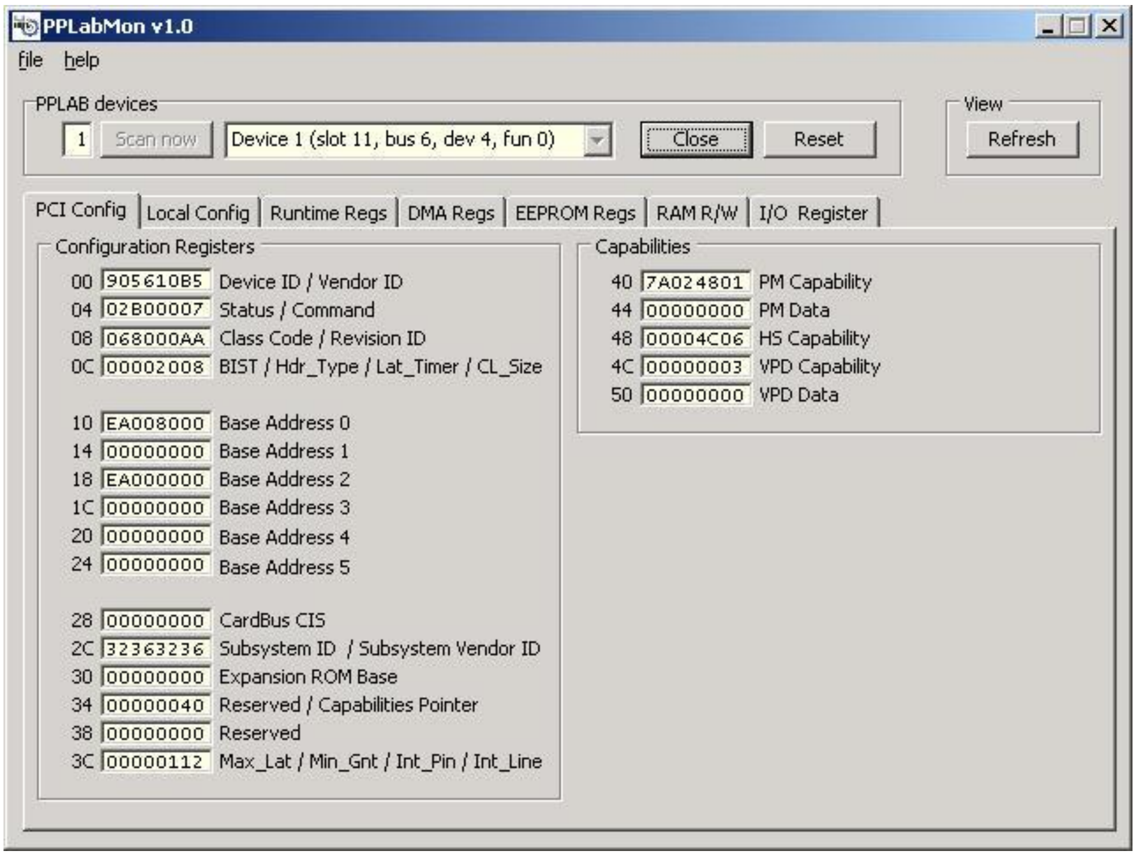
As well as the file, the folders also contain the project files used to generate the programs. (MS Visual C++ 6.0)

The following examples are available:

| | |
|---|---|
| • ApiTest | - calls the basic API functions |
| • LoadDLL | - explicitly loads the API DLL drivers |
| • MapMem | - directly reads from and writes to 'shown' memory areas |
| • RdWrRAM | - reads and writes to memory using the KERNEL method |
| • ReadDMARegs | - reading DMA register |
| • ReadLocalCfg | - reading ‘local config register’ |
| • ReadPCICfg | - reading ‘PCI config register’ |
| • ReadRuntimeRegs | - reading ‘runtime register’ |
| • Reset | - resets the *PCIe-BaseLab* card |
| • TestEEPROM | - reading and writing EEPROM register |
| • UseDMA | - initializes and starts DMA transfers |

# 8. The monitor program PPLABMON.EXE

The monitor program 'PPLABMON.EXE' is a graphical Windows application which provides a comprehensive view of all areas of the *PCIe-BaseLab* card.



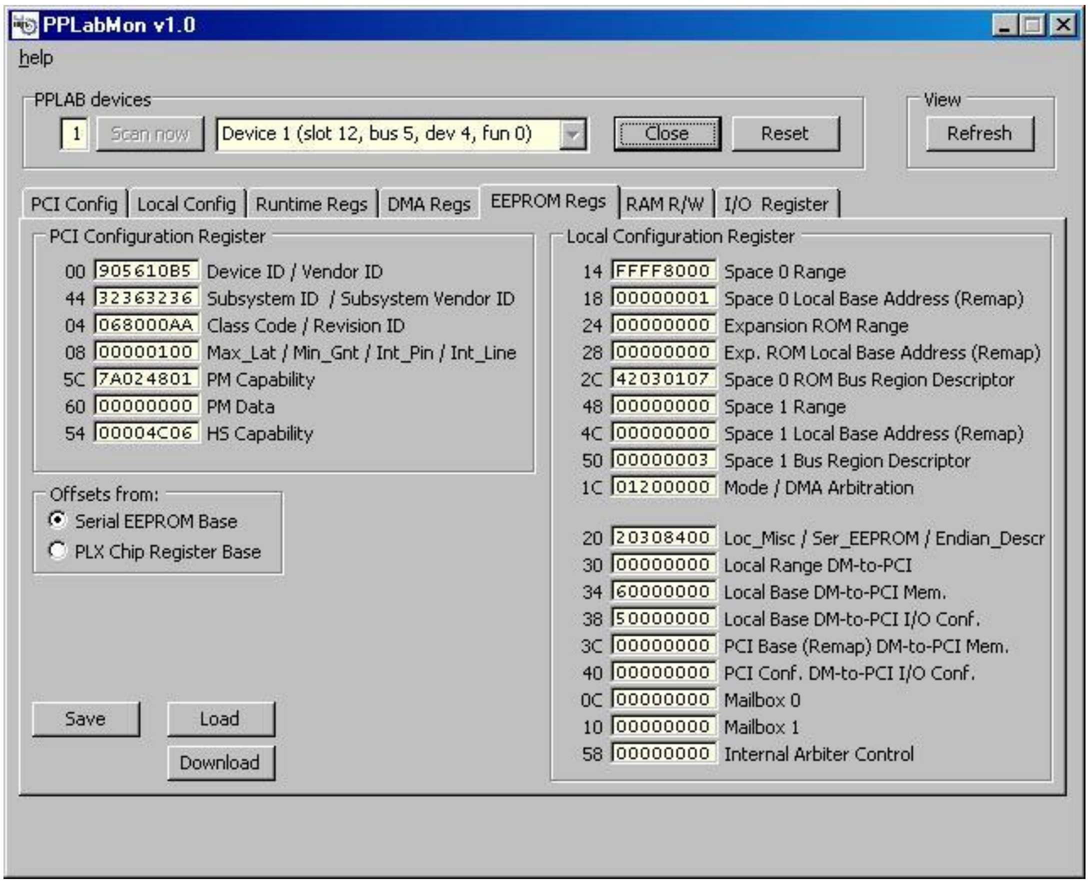
PPLAB devices: 1 — Device 1 (slot 11, bus 6, dev 4, fun 0)

```
Configuration Registers
00 905610B5  Device ID / Vendor ID
04 02B00007  Status / Command
08 068000AA  Class Code / Revision ID
0C 00002008  BIST / Hdr_Type / Lat_Timer / CL_Size

10 EA008000  Base Address 0
14 00000000  Base Address 1
18 EA000000  Base Address 2
1C 00000000  Base Address 3
20 00000000  Base Address 4
24 00000000  Base Address 5

28 00000000  CardBus CIS
2C 32363236  Subsystem ID  / Subsystem Vendor ID
30 00000000  Expansion ROM Base
34 00000040  Reserved / Capabilities Pointer
38 00000000  Reserved
3C 00000112  Max_Lat / Min_Gnt / Int_Pin / Int_Line

Capabilities
40 7A024801  PM Capability
44 00000000  PM Data
48 00004C06  HS Capability
4C 00000003  VPD Capability
50 00000000  VPD Data
```

The following options are available:

- PCI Config - reading 'PCI config register'
- Local Config - reading 'local config register'
- RuntimeRegs - reading 'runtime register'
- DMARegs - reading 'DMA register'
- EEPROMRegs - edits the registers
- RAM R/W - reading and writing RAM
- I/O Register - sets and queries the 16 bit I/O register pins

**Editing the EEPROM registers**



PPLAB devices: 1 — Device 1 (slot 12, bus 5, dev 4, fun 0)

PCI Configuration Register

| | | |
|---|---|---|
| 00 | 905610B5 | Device ID / Vendor ID |
| 44 | 32363236 | Subsystem ID / Subsystem Vendor ID |
| 04 | 068000AA | Class Code / Revision ID |
| 08 | 00000100 | Max_Lat / Min_Gnt / Int_Pin / Int_Line |
| 5C | 7A024801 | PM Capability |
| 60 | 00000000 | PM Data |
| 54 | 00004C06 | HS Capability |

Offsets from: [✓ Serial EEPROM Base] PLX Chip Register Base

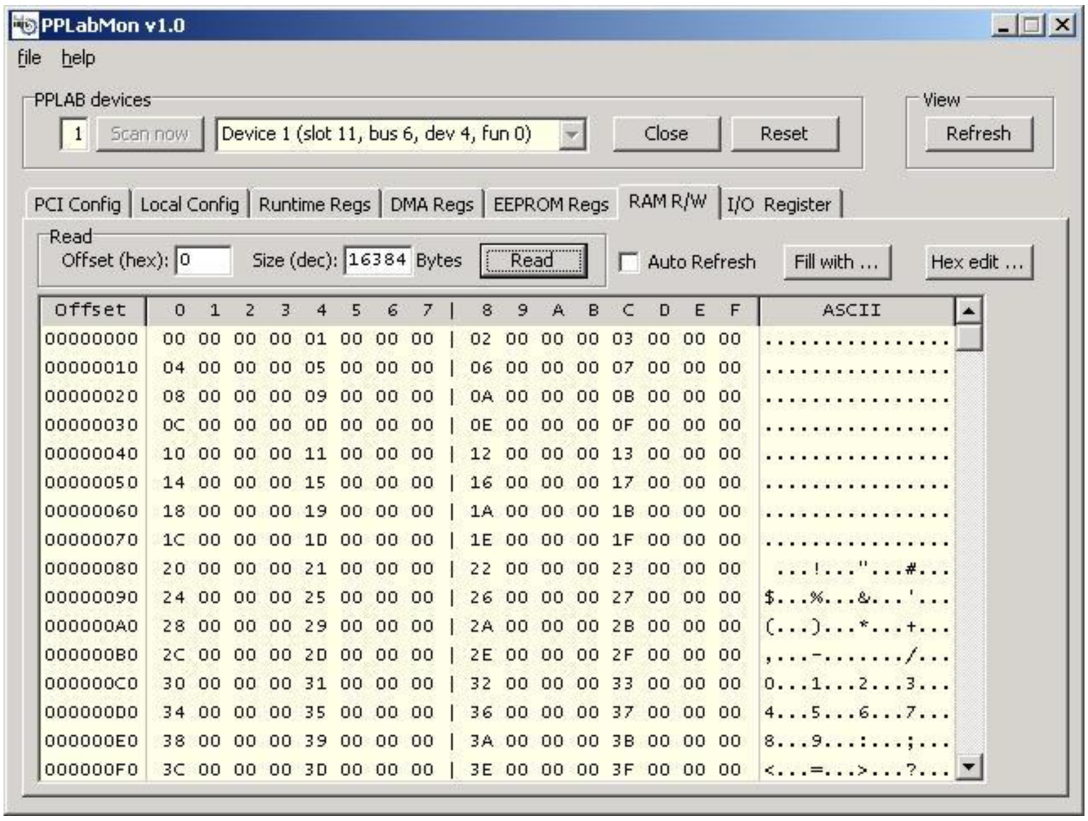
Local Configuration Register

| | | |
|---|---|---|
| 14 | FFFF8000 | Space 0 Range |
| 18 | 00000001 | Space 0 Local Base Address (Remap) |
| 24 | 00000000 | Expansion ROM Range |
| 28 | 00000000 | Exp. ROM Local Base Address (Remap) |
| 2C | 42030107 | Space 0 ROM Bus Region Descriptor |
| 48 | 00000000 | Space 1 Range |
| 4C | 00000000 | Space 1 Local Base Address (Remap) |
| 50 | 00000003 | Space 1 Bus Region Descriptor |
| 1C | 01200000 | Mode / DMA Arbitration |
| 20 | 20308400 | Loc_Misc / Ser_EEPROM / Endian_Descr |
| 30 | 00000000 | Local Range DM-to-PCI |
| 34 | 60000000 | Local Base DM-to-PCI Mem. |
| 38 | 50000000 | Local Base DM-to-PCI I/O Conf. |
| 3C | 00000000 | PCI Base (Remap) DM-to-PCI Mem. |
| 40 | 00000000 | PCI Conf. DM-to-PCI I/O Conf. |
| 0C | 00000000 | Mailbox 0 |
| 10 | 00000000 | Mailbox 1 |
| 58 | 00000000 | Internal Arbiter Control |

The following options are available on the EEPROM register page:

| | |
|---|---|
| • Read register | - reads the current EEPROM content ('Refresh') |
| • Amend register | - amends and downloads an EEPROM register (double-click) |
| • Save to file | - saves all registers to a file in ASCII format – editable ('Save') |
| • Load from file | - loads all register content from a file ('Load') |
| • Download all | - downloads all registers to the EEPROM ('Download') |

**Reading and writing to the RAM**



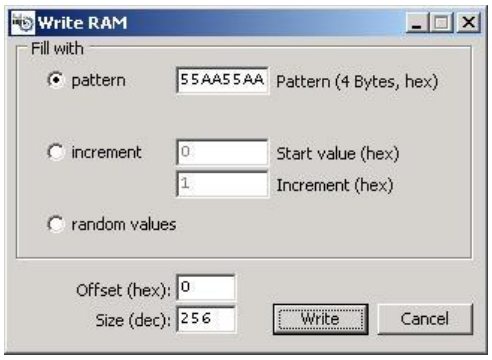
1 Device 1 (slot 11, bus 6, dev 4, fun 0)

Offset (hex): 0 Size (dec): 16384 Bytes

```
Offset     0  1  2  3  4  5  6  7 |  8  9  A  B  C  D  E  F    ASCII
00000000  00 00 00 01 00 00 00 00 | 02 00 00 00 03 00 00 00   ................
00000010  04 00 00 00 05 00 00 00 | 06 00 00 00 07 00 00 00   ................
00000020  08 00 00 00 09 00 00 00 | 0A 00 00 00 0B 00 00 00   ................
00000030  0C 00 00 00 0D 00 00 00 | 0E 00 00 00 0F 00 00 00   ................
00000040  10 00 00 00 11 00 00 00 | 12 00 00 00 13 00 00 00   ................
00000050  14 00 00 00 15 00 00 00 | 16 00 00 00 17 00 00 00   ................
00000060  18 00 00 00 19 00 00 00 | 1A 00 00 00 1B 00 00 00   ................
00000070  1C 00 00 00 1D 00 00 00 | 1E 00 00 00 1F 00 00 00   ................
00000080  20 00 00 00 21 00 00 00 | 22 00 00 00 23 00 00 00    ...!..."...#...
00000090  24 00 00 00 25 00 00 00 | 26 00 00 00 27 00 00 00   $...%...&...'...
000000A0  28 00 00 00 29 00 00 00 | 2A 00 00 00 2B 00 00 00   (...)...*...+...
000000B0  2C 00 00 00 2D 00 00 00 | 2E 00 00 00 2F 00 00 00   ,...-......./...
000000C0  30 00 00 00 31 00 00 00 | 32 00 00 00 33 00 00 00   0...1...2...3...
000000D0  34 00 00 00 35 00 00 00 | 36 00 00 00 37 00 00 00   4...5...6...7...
000000E0  38 00 00 00 39 00 00 00 | 3A 00 00 00 3B 00 00 00   8...9...:...;...
000000F0  3C 00 00 00 3D 00 00 00 | 3E 00 00 00 3F 00 00 00   <...=...>...?...
```

The following options are available on this page:

| | |
|---|---|
| • Change offset | - defines the start address |
| • Change size | - sets the displayed size |
| • Read | - reads the current content |
| • Refresh automatically | - the content is read and updated automatically |
| • Write to RAM | - fills the RAM with various values |
| • Change individual bytes | - edits individual bytes |



(● pattern) 55AA55AA Pattern (4 Bytes, hex)

○ increment 0 Start value (hex)
1 Increment (hex)

○ random values

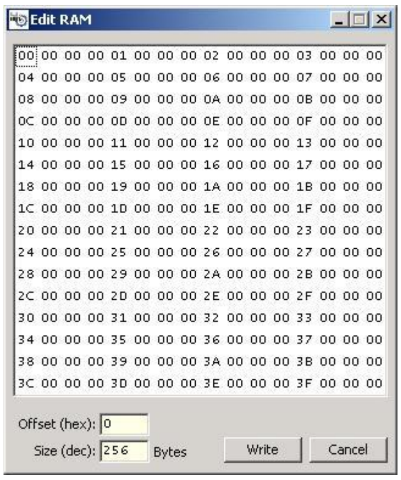
Offset (hex): 0
Size (dec): 256

Write RAM

The following options for writing to the RAM are available:

- fill RAM with a pattern (‘pattern’)
- fill RAM incrementally (‘increment’)
- fill RAM with random values (‘random values’)

All the fields in this window can be edited.



```
00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00
04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00
08 00 00 00 09 00 00 00 0A 00 00 00 0B 00 00 00
0C 00 00 00 0D 00 00 00 0E 00 00 00 0F 00 00 00
10 00 00 00 11 00 00 00 12 00 00 00 13 00 00 00
14 00 00 00 15 00 00 00 16 00 00 00 17 00 00 00
18 00 00 00 19 00 00 00 1A 00 00 00 1B 00 00 00
1C 00 00 00 1D 00 00 00 1E 00 00 00 1F 00 00 00
20 00 00 00 21 00 00 00 22 00 00 00 23 00 00 00
24 00 00 00 25 00 00 00 26 00 00 00 27 00 00 00
28 00 00 00 29 00 00 00 2A 00 00 00 2B 00 00 00
2C 00 00 00 2D 00 00 00 2E 00 00 00 2F 00 00 00
30 00 00 00 31 00 00 00 32 00 00 00 33 00 00 00
34 00 00 00 35 00 00 00 36 00 00 00 37 00 00 00
38 00 00 00 39 00 00 00 3A 00 00 00 3B 00 00 00
3C 00 00 00 3D 00 00 00 3E 00 00 00 3F 00 00 00
```

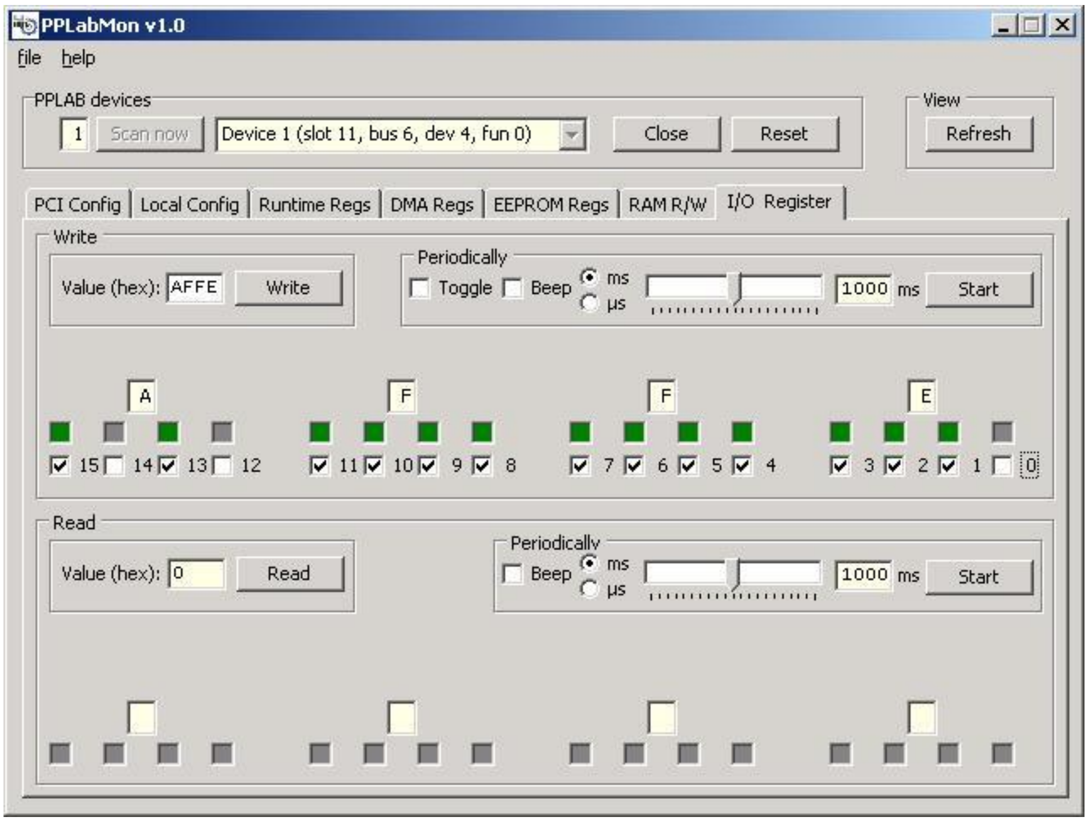
Offset (hex): 0
Size (dec): 256 Bytes

Change individual bytes

The edit window displays 256 bytes starting at the previously selected offset. To make changes:

- select the byte you wish to edit
- start mode - double-click or ‘Enter’
- enter the new value and press 'Enter'
- Click on 'Write' to write the changed value 256 Byte HEX Dumps to the RAM

**Setting and querying the register pins**

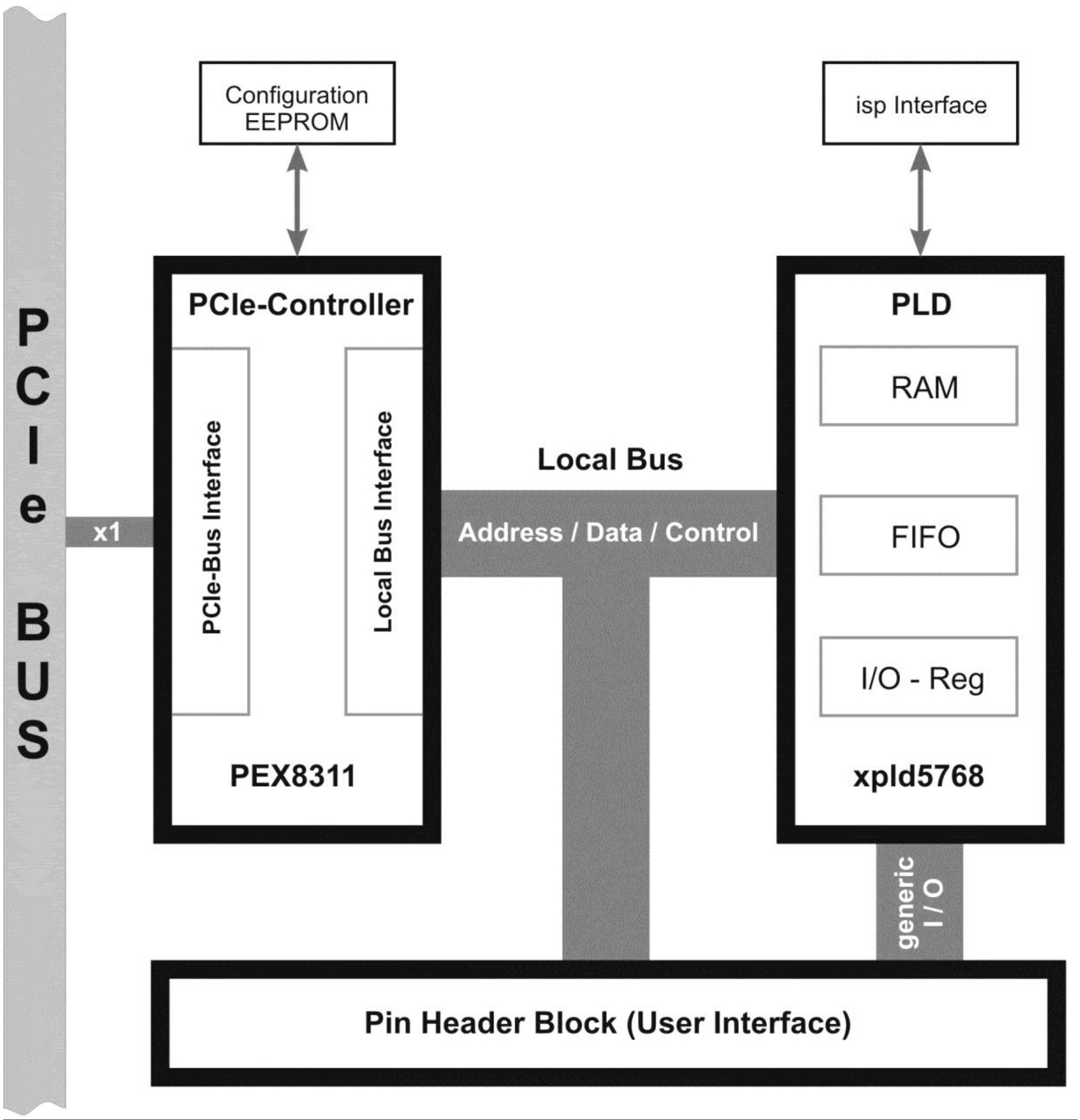


The following options are available on this page:

| | |
|---|---|
| • Set the 16 Bit I/O register | - The pins can be set using the hex value or bit-by-bit. Click on 'Write' to transfer the settings to the register. |
| • Query 16 Bit I/O register | - queries the pins (‘Read’) |
| • Periodic write/read | - The slider can be used to set a variety of periods. Click on 'Start' to activate periodic write/read (‘Periodically’) |
| • Signal tone | - a beep will be emitted to signal a read/write process (‘Beep’) |
| • Bit-by-bit negation | - toggles the logical state of individual bits (‘Toggle’) |

Appendix

## Block diagram



## Pin configuration

| Pin | Name | Function | Pin | Name | Function |
|---|---|---|---|---|---|
| 01 | GND | GND | 02 | GND | GND |
| 03 | LCLK_JP | local clock output | 04 | GCLK1 | pld clock 1 input |
| 05 | GCLK2 | pld clock 2 input | 06 | GCLK3 | pld clock 3 input |
| 07 | P30/D1 | pld GPI/O | 08 | P28/E1 | pld GPI/O |
| 09 | P26/F4 | pld GPI/O | 10 | P24/F5 | pld GPI/O |
| 11 | P22/E2 | pld GPI/O | 12 | P20/CLK_OUT0/F2 | pld GPI/O / pll0 CTL |
| 13 | P18/F1 | pld GPI/O | 14 | P16/G1 | pld GPI/O |
| 15 | P14/F3 | pld GPI/O | 16 | P12/G5 | pld GPI/O |
| 17 | P10/H5 | pld GPI/O | 18 | P8/PLL_RST0/G4 | pld GPI/O / pll0 CTL |
| 19 | P6/G3 | pld GPI/O | 20 | P4/PLL_FBK0/H3 | pld GPI/O / pll0 CTL |
| 21 | GND | GND | 22 | GND | GND |
| 23 | P2/G2 | pld GPI/O | 24 | P0/H1 | pld GPI/O |
| 25 | O30/C4 | pld GPI/O | 26 | O28/E4 | pld GPI/O |
| 27 | O26/B1 | pld GPI/O | 28 | O24/C1 | pld GPI/O |
| 29 | O22/D3 | pld GPI/O | 30 | O20/C2 | pld GPI/O |
| 31 | O18/E3 | pld GPI/O | 32 | O16/D2 | pld GPI/O |
| 33 | N5/A2 | pld GPI/O | 34 | N4/B2 | pld GPI/O |
| 35 | K30/D12 | pld GPI/O | 36 | K28/B14 | pld GPI/O |
| 37 | K26/C13 | pld GPI/O | 38 | K24/A14 | pld GPI/O |
| 39 | K22/A13 | pld GPI/O | 40 | K21/B13 | pld GPI/O |
| 41 | GND | GND | 42 | GND | GND |
| 43 | K20/D11 | pld GPI/O | 44 | K18/B12 | pld GPI/O |
| 45 | K16/C12 | pld GPI/O | 46 | K14/E11 | pld GPI/O |
| 47 | K4/E10 | pld GPI/O | 48 | K2/A12 | pld GPI/O |
| 49 | K0/A11 | pld GPI/O | 50 | J14/C16 | pld GPI/O |
| 51 | J12/B16 | pld GPI/O | 52 | J10/C15 | pld GPI/O |
| 53 | J8/B15 | pld GPI/O | 54 | J6/E14 | pld GPI/O |
| 55 | J4/D14 | pld GPI/O | 56 | J2/E13 | pld GPI/O |
| 57 | J0/A15 | pld GPI/O | 58 | G8/M13 | pld GPI/O |
| 59 | LD1/ | PEX8311/Data | 60 | LD0 | PEX8311/Data |
| 61 | GND | GND | 62 | GND | GND |
| 63 | LD3 | PEX8311/Data | 64 | LD2 | PEX8311/Data |
| 65 | LD5 | PEX8311/Data | 66 | LD4 | PEX8311/Data |
| 67 | LD7 | PEX8311/Data | 68 | LD6 | PEX8311/Data |
| 69 | LD9 | PEX8311/Data | 70 | LD8 | PEX8311/Data |
| 71 | LD11 | PEX8311/Data | 72 | LD10 | PEX8311/Data |
| 73 | LD13 | PEX8311/Data | 74 | LD12 | PEX8311/Data |
| 75 | LD15 | PEX8311/Data | 76 | LD14 | PEX8311/Data |
| 77 | LD17 | PEX8311/Data | 78 | LD16 | PEX8311/Data |
| 79 | LD19 | PEX8311/Data | 80 | LD18 | PEX8311/Data |
| 81 | GND | GND | 82 | GND | GND |
| 83 | LD21 | PEX8311/Data | 84 | LD20 | PEX8311/Data |
| 85 | LD23 | PEX8311/Data | 86 | LD22 | PEX8311/Data |
| 87 | LD25 | PEX8311/Data | 88 | LD24 | PEX8311/Data |
| 89 | LD27 | PEX8311/Data | 90 | LD26 | PEX8311/Data |
| 91 | LD29 | PEX8311/Data | 92 | LD28 | PEX8311/Data |
| 93 | LD31 | PEX8311/Data | 94 | LD30 | PEX8311/Data |
| 95 | DP2 | PEX8311/Parity | 96 | DP3 | PEX8311/Parity |
| 97 | DP0 | PEX8311/Parity | 98 | DP1 | PEX8311/Parity |
| 99 | GND | GND | 100 | GND | GND |

**Pin-side pin configuration JP4**

| Pin | Name | Function | Pin | Name | Function |
|---|---|---|---|---|---|
| 01 | GND | GND | 02 | GND | GND |
| 03 | M30/B7 | pld GPI/O | 04 | M28/A7 | pld GPI/O |
| 05 | M26/D7 | pld GPI/O | 06 | M24/C7 | pld GPI/O |
| 07 | M22/B6 | pld GPI/O | 08 | M21/E7 | pld GPI/O |
| 09 | M20/E6 | pld GPI/O | 10 | M18/A6 | pld GPI/O |
| 11 | M12/B5 | pld GPI/O | 12 | M14/A4 | pld GPI/O |
| 13 | M8/B4 | pld GPI/O | 14 | M10/A3 | pld GPI/O |
| 15 | M5/C5 | pld GPI/O | 16 | M6/B3 | pld GPI/O |
| 17 | M2/D5 | pld GPI/O | 18 | M4/C6 | pld GPI/O |
| 19 | L30/B11 | pld GPI/O | 20 | M0/D6 | pld GPI/O |
| 21 | GND | GND | 22 | GND | GND |
| 23 | L26/B10 | pld GPI/O | 24 | L28/C11 | pld GPI/O |
| 25 | L22/C10 | pld GPI/O | 26 | L24/A10 | pld GPI/O |
| 27 | L20/C9 | pld GPI/O | 28 | L21/D10 | pld GPI/O |
| 29 | L12/A9 | pld GPI/O | 30 | L18/E9 | pld GPI/O |
| 31 | L8/E8 | pld GPI/O | 32 | L14/F9 | pld GPI/O |
| 33 | L5/B9 | pld GPI/O | 34 | L10/F8 | pld GPI/O |
| 35 | L2/B8 | pld GPI/O | 36 | L6/A8 | pld GPI/O |
| 37 | I30/H14 | pld GPI/O | 38 | L4/D8 | pld GPI/O |
| 39 | I26/G15 | pld GPI/O | 40 | L0/C8 | pld GPI/O |
| 41 | GND | GND | 42 | GND | GND |
| 43 | I22/PLL_RST1/H12 | pld GPI/O / pll1 CTL | 44 | I28/G16 | pld GPI/O |
| 45 | I18/F16 | pld GPI/O | 46 | I24/PLL_FBK1/F15 | pld GPI/O / pll1 CTL |
| 47 | I14/G13 | pld GPI/O | 48 | I20/G14 | pld GPI/O |
| 49 | I10/F14 | pld GPI/O | 50 | I16/E16 | pld GPI/O |
| 51 | M16/VREF0/A5 | pld GPI/O / VREF0 Input | 52 | I12/G12 | pld GPI/O |
| 53 | D10/VREF1/L9 | pld GPI/O / VREF1 Input | 54 | I8/CLK_OUT1/E15 | pld GPI/O / pll1 CTL |
| 55 | E20/VREV2/T14 | pld GPI/O / VREF2 Input | 56 | EECS# | PEX8311/SPI_EPROM |
| 57 | L16/VREF3/D9 | pld GPI/O / VREF3 Input | 58 | EECLK | PEX8311/SPI_EPROM |
| 59 | LRESET# | PEX8311/local rst output | 60 | EEWRDATA | PEX8311/SPI_EPROM |
| 61 | GND | GND | 62 | GND | GND |
| 63 | M_RST# | pld reset input | 64 | EERDDATA | PEX8311/SPI_EPROM |
| 65 | LHOLD | PEX8311/Control | 66 | DMPAF/EOT# | PEX8311/Control |
| 67 | LHOLDA# | PEX8311/Control | 68 | BIGEND# | PEX8311/Control |
| 69 | ADS# | PEX8311/Control | 70 | USERO/LLOCK0# | PEX8311/Control |
| 71 | BLAST# | PEX8311/Control | 72 | USERI/LLOCK1# | PEX8311/Control |
| 73 | READY# | PEX8311/Control | 74 | DREQ0# | PEX8311/Control |
| 75 | WAIT# | PEX8311/Control | 76 | DREQ1# | PEX8311/Control |
| 77 | LW/R# | PEX8311/Control | 78 | DACK0# | PEX8311/Control |
| 79 | BTERM# | PEX8311/Control | 80 | DACK1# | PEX8311/Control |
| 81 | GND | GND | 82 | GND | GND |
| 83 | CCS# | PEX8311/Control | 84 | MODE0 | PEX8311/Control |
| 85 | LINTO# | PEX8311/Control | 86 | MODE1 | PEX8311/Control |
| 87 | LINTI# | PEX8311/Control | 88 | I0/D15 | pld GPI/O |
| 89 | LSERR# | PEX8311/Control | 90 | I2/D16 | pld GPI/O |
| 91 | BREQI | PEX8311/Control | 92 | I4/F13 | pld GPI/O |
| 93 | BREQO | PEX8311/Control | 94 | I6/F12 | pld GPI/O |
| 95 | A6/J4 | pld GPI/O | 96 | PMEIN# | PEX8311/Control |
| 97 | not connected | - | 98 | not connected | - |
| 99 | GND | GND | 100 | GND | GND |

**Pin-side pin configuration JP5**

| Pin | Name | Function | Pin | Name | Function |
|---|---|---|---|---|---|
| 01 | GND | GND | 02 | GND | GND |
| 03 | LA31 | PEX8311/Address | 04 | LA30 | PEX8311/Address |
| 05 | LA29 | PEX8311/Address | 06 | LA28 | PEX8311/Address |
| 07 | LA27 | PEX8311/Address | 08 | LA26 | PEX8311/Address |
| 09 | LA25 | PEX8311/Address | 10 | LA24 | PEX8311/Address |
| 11 | LA23 | PEX8311/Address | 12 | LA22 | PEX8311/Address |
| 13 | LA21 | PEX8311/Address | 14 | LA20 | PEX8311/Address |
| 15 | LA19 | PEX8311/Address | 16 | LA18 | PEX8311/Address |
| 17 | LA17 | PEX8311/Address | 18 | LA16 | PEX8311/Address |
| 19 | LA15 | PEX8311/Address | 20 | LA14 | PEX8311/Address |
| 21 | LA13 | PEX8311/Address | 22 | LA12 | PEX8311/Address |
| 23 | LA11 | PEX8311/Address | 24 | LA10 | PEX8311/Address |
| 25 | LA9 | PEX8311/Address | 26 | LA8 | PEX8311/Address |
| 27 | LA7 | PEX8311/Address | 28 | LA6 | PEX8311/Address |
| 29 | LA5 | PEX8311/Address | 30 | LA4 | PEX8311/Address |
| 31 | LA3 | PEX8311/Address | 32 | LA2 | PEX8311/Address |
| 33 | GPIO0 | PEX8311/GPI/O | 34 | LBE0# | PEX8311/Control |
| 35 | GPIO1 | PEX8311/GPI/O | 36 | LBE1# | PEX8311/Control |
| 37 | GPIO2 | PEX8311/GPI/O | 38 | LBE2# | PEX8311/Control |
| 39 | GPIO3 | PEX8311/GPI/O | 40 | LBE3# | PEX8311/Control |

**Pin-side pin configuration JP6**

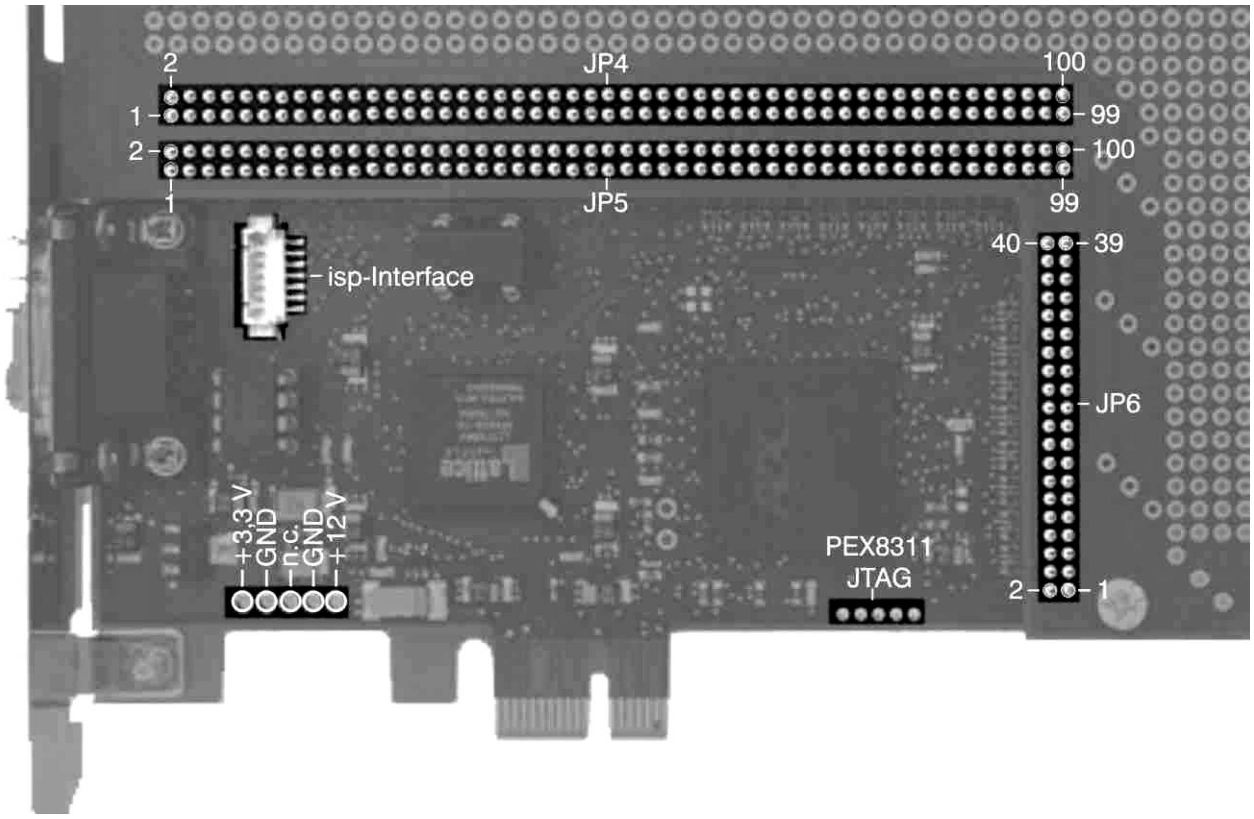
## Pin configuration ISP-Interface (X1)

| Pin | Name |
|---|---|
| 01 | GND |
| 02 | TDO |
| 03 | TDI |
| 04 | TMS |
| 05 | TCK |
| 06 | TOE |
| 07 | Not connected |

## Pin configuration JTAG Support PEX8311 (JP2)

| Pin | Name |
|---|---|
| 01 | TDO |
| 02 | TRST# |
| 03 | TDI |
| 04 | TMS |
| 05 | TCK |

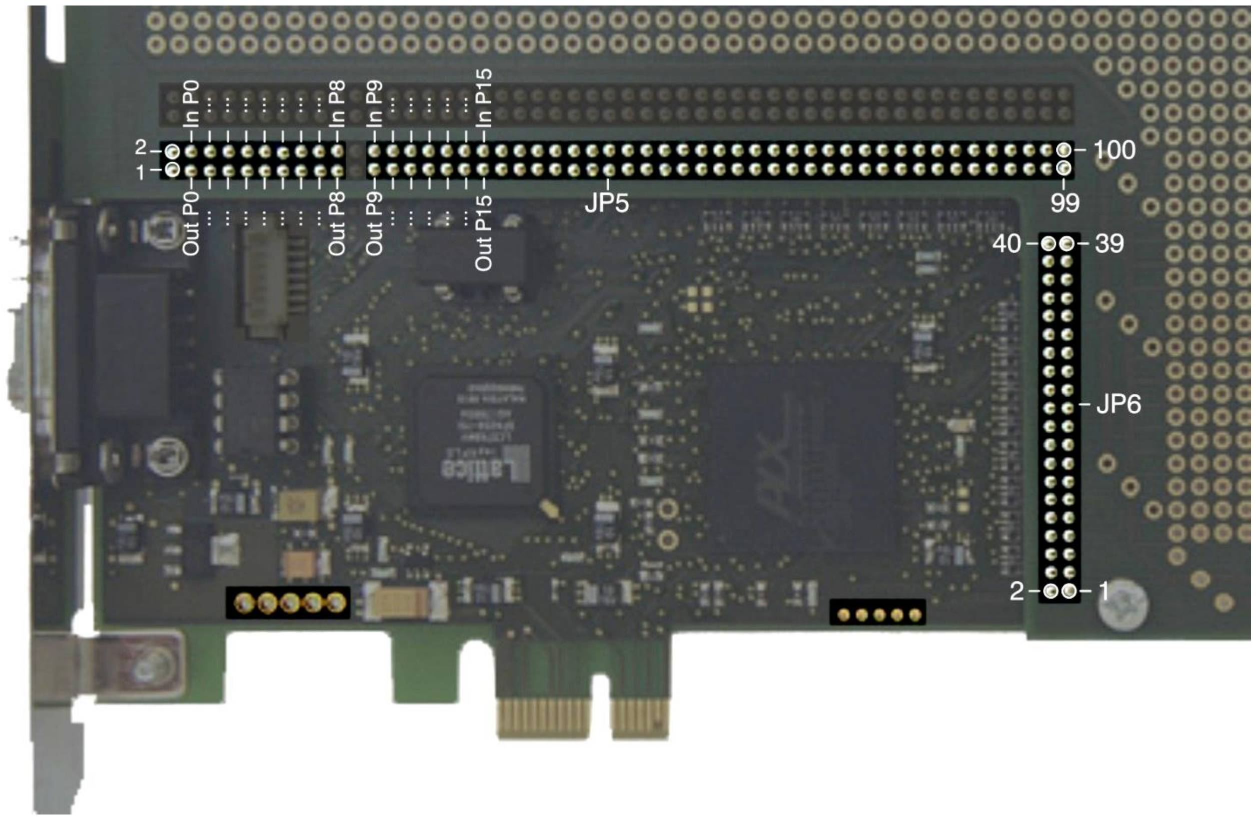
## Position of shrouded headers / connectors



## Sample application, position of connecting pins JP5

| Pin | Name | Function | Pin | Name | Function |
|---|---|---|---|---|---|
| 01 | GND | GND | 02 | GND | GND |
| 03 | M30/B7 | Out P0 | 04 | M28/A7 | In P0 |
| 05 | M26/D7 | Out P1 | 06 | M24/C7 | In P1 |
| 07 | M22/B6 | Out P2 | 08 | M21/E7 | In P2 |
| 09 | M20/E6 | Out P3 | 10 | M18/A6 | In P3 |
| 11 | M12/B5 | Out P4 | 12 | M14/A4 | In P4 |
| 13 | M8/B4 | Out P5 | 14 | M10/A3 | In P5 |
| 15 | M5/C5 | Out P6 | 16 | M6/B3 | In P6 |
| 17 | M2/D5 | Out P7 | 18 | M4/C6 | In P7 |
| 19 | L30/B11 | Out P8 | 20 | M0/D6 | In P8 |
| 21 | GND | GND | 22 | GND | GND |
| 23 | L26/B10 | Out P9 | 24 | L28/C11 | In P9 |
| 25 | L22/C10 | Out P10 | 26 | L24/A10 | In P10 |
| 27 | L20/C9 | Out P11 | 28 | L21/D10 | In P11 |
| 29 | L12/A9 | Out P12 | 30 | L18/E9 | In P12 |
| 31 | L8/E8 | Out P13 | 32 | L14/F9 | In P13 |
| 33 | L5/B9 | Out P14 | 34 | L10/F8 | In P14 |
| 35 | L2/B8 | Out P15 | 36 | L6/A8 | In P15 |

**Pin-side pin configuration JP5**



## Our Hotline:

| | | |
|---|---|---|
| HK Meßsysteme GmbH | Telephone: | ++49/30/633 75 114 |
| Straße am Heizhaus 1 | Fax: | ++49/30/633 75 116 |
| D-10318 Berlin /Germany | E-Mail: | support@pci-tools.de |
| | Web: | http://www.pci-tools.com |
| | | http://www.pci-tools.de |

| | | |
|---|---|---|
| DriverFactory | Telephone: | ++49/30/5304 2020 |
| Ostendstr. 25 | Fax: | ++49/30/5304 2021 |
| D-10318 Berlin /Germany | E-Mail: | info@driverfactory.de |
| | Web: | http://www.driverfactory.de |

## Manufacturer PEX8311:

| | | |
|---|---|---|
| Broadcom Limited Company | Phone: | ++1-877-673-9442 |
| 1320 Ridder Park Drive | Web: | http://www.broadcom.com |
| San Jose, CA 95131 | | |
| U.S.A. | | |

## Manufacturer xpld5768:

| | | |
|---|---|---|
| Lattice Semiconductor Corporation | Phone: | ++1-503-268-8000 |
| 5555 N.E. Moore Court | Fax: | ++1-503-268-8347 |
| Hillsboro, Oregon 97124-6421 | Web: | http://www.latticesemi.com |
| U.S.A. | | |

## Web addresses

http://www.pci-tools.com

http://www.pci-tools.de

http://www.driverfactory.de

http://www. broadcom.com

http://www.latticesemi.com

http ://www.pcisig.com