PCIe-BaseLab

Evaluation-Board für den PCI-Express-Bus

Version 1.02
Juli 2016

Hinweise zum Lieferumfang, zur Installation, zu den Hardware-Eigenschaften und der -Erweiterbarkeit sowie zur Programmierung auf Basis der API-Funktionen

Inhaltsverzeichnis

1. Einleitung

Allgemeine Eigenschaften

PCIe-BaseLab ist ein unentbehrliches Hilfsmittel zur Entwicklung von Zusatzkarten für Personalcomputer und andere Rechnersysteme, die mit dem PCI Express-Bus (PCIe) ausgestattet sind. Diese Karte ermöglicht den schnellen und unkomplizierten Test neu entwickelter Elektronik-Schaltungen am PCIe-Bus.

PCIe-BaseLab arbeitet mit dem universellen PCI Express Endpoint Controller *PEX8311* von *PLX Technology, Inc.,* dessen Peripherie sowie einem leistungsstarken programmierbaren Logikbaustein (PLD), Hersteller: *Lattice Semiconductor*. Mit dem *PEX8311* Controller ist *PCIe-BaseLab* kompatibel zu den Forderungen der PCI Express Spezifikation, Revision 1.0. Die Karte ist komplett bestückt, geprüft und somit sofort betriebsbereit.

Der Schaltungsdesigner kann seine Hardware direkt auf das Lochrasterfeld der aufsteckbaren Tochterkarte montieren und mit den Tests beginnen. Die Beschäftigung mit den Signalspielen und technischen Eigenschaften des PCIe-Bussystems kann auf ein Minimum reduziert werden.

Das PLD (Typ: *ispXPLD5768*, Familie *XPLD5000*) enthält verschiedene vorinstallierte Beispielanwendungen:

- ein synchrones RAM,
- eine FIFO (alternativ zum RAM),
- ein 16 Bit Ein-Ausgaberegister.

Die VHDL-Quellcodes der Beispielanwendungen werden mitgeliefert und können durch den Nutzer modifiziert werden.

Besonders vorteilhaft für Hardware-Entwickler und Messtechniker:
Alle lokalen Anschlusspins des *PEX8311*-Controllers sind vorab mit dem PLD verbunden, auf Pfostenleisten geführt und von dort bequem zugänglich. 106 der 193 I/O-Pins des PLDs sind ebenfalls auf Pfostenleisten gelegt und stehen zur freien Verfügung.

Für *PCIe-BaseLab* sind Tochterboards in verschiedenen Varianten erhältlich. Derzeit mit einem 0.1 Zoll Lochrasterfeld, später auch mit universellen SMD-Footprints oder auch in kundenspezifischer Form. Letztere eröffnet die Möglichkeit, *PCIe-BaseLab* als Fertigmodul zusammen mit applikationsspezifischer Hardware in kleinen und mittleren Serien einzusetzen.

Mit der Nutzung von *PCIe-BaseLab* entstehen keine Lizenzkosten für einen PCIe-IP-Core, auch eine Mitgliedschaft in der PCISIG ist nicht erforderlich. Sub-Vendor und Sub-System-IDs zur eindeutigen Kennzeichnung der eigenen Hardware können vom Hersteller des PCIe-Controllers, von der Firma *PLX Technology, Inc.,* gebührenfrei bezogen werden.

Die Dokumentation von *PCIe-BaseLab* wird durch Schaltungsunterlagen, Bestückungs- und Anschluss-Schemata sowie durch PLD-Quelltexte abgerundet.

Lieferumfang

Der Lieferumfang des Produkts *PCIe-BaseLab* besteht aus folgenden Komponenten:

- *PCI Express Bus Evaluation Board* mit aufgesetzter Tochterkarte (Lötaugenraster 0.1 Zoll),
- *Adapterkabel* zur Stromversorgung der Tochterkarte über SATA-Stromversorgungsstecker,
- *CD-ROM*: Windows-Treiber, Monitor-Programm, Beispielanwendungen inklusive Quelltext und Entwickler-Dateien (Header, Lib, DLL) sowie Schaltpläne, Datenblätter, VHDL-Quellcodes sowie dieses technische Handbuch.

2. Installation

Software

Die *PCIe-BaseLab*-Software kann unter Windows 2000, Windows XP, Windows Vista und Windows 7 verwendet werden. Der Windows-Treiber ist ein WDM-Treiber. Er wird in einer 32-Bit-Version und in einer die 64-Bit-Version geliefert.

ACHTUNG: Zur Installation der Software müssen Sie Administrator-Rechte besitzen!

Zur Installation der Software wählen Sie eine der folgenden Vorgehensweisen (je nach Liefervariante):

- Kopieren Sie die gesamte Verzeichnisstruktur von der CD in das Zielverzeichnis auf Ihrem PC.
- Entpacken Sie die Dateien durch Starten des selbstentpackenden Programmes 'pplab_v1005.exe'.
- Starten Sie die Installation durch Ausführen der 'Setup.exe'.

Im Verzeichnis der *PCIe-BaseLab*-Software finden Sie dann folgende Unterverzeichnisse:

- bin: die Treiber-API-DLL 'PPLABAPI.DLL' (→ Abschnitt 5.) sowie
  die Binärdateien der Beispielprogramme (→ Abschn. 8.),
- dev: die beiden C-Header-Dateien 'PPLABAPI.H' und 'PPLABERR.H' (→ Abschnitt 5.),
- doc: das Handbuch 'PPLAB-v10_Manual.PDF',
- drv: den Windows-Treiber 'PPLAB_E.SYS' sowie 2 zugehörige INF-Dateien (→ folg. Abschnitt),
- lib: die Importbibliothek 'PPLABAPI.LIB' (→ Abschnitt 5.),
- mon: das Monitorprogramm 'PPLABMON.EXE' sowie zugehörige Dateien (→ Abschnitt 7.),
- smp: die Quellcode-Dateien der Beispielprogramme (→ Abschnitt 8.).

Tipp: Es ist sinnvoll, die Software zu installieren, bevor die *PCIe-BaseLab*-Karte in den PC eingebaut wird. Dann kann sofort nach dem Rechner-Neustart die Zuordnung des Windows-Treibers erfolgen.

Hardware

Die Installation der Hardware erfolgt durch Einstecken der Karte in ein x1-, x4-, x8- oder x16-PCI-Express-Slot eines PCs oder eines anderen Rechnersystems mit PCIe-Steckplätzen unter Berücksichtigung der Gesetze der ESD-Sicherheit und des Schutzes vor dem Berühren von Teilen, die unter gefährlicher elektrischer Spannung stehen können.

Der Rechner ist vor der Installation/Deinstallation der *PCIe-BaseLab*-Karte auszuschalten und vom Netz abzutrennen.

Die auf der *PCIe-BaseLab*-Karte integrierte LED (LED1) zeigt nach Einschalten des Rechners das Zustandekommen der physikalischen Link-Verbindung des PCIe-Interfaces an, deren Aufleuchten kann als erste Kontrolle für eine korrekte Hardware-Installation dienen.

Der Betrieb von *PCIe-BaseLab* ist auch über einen PCIe-Bus-Extender möglich. Das erlaubt das Anstecken und Abziehen der Karte bei laufendem PC (Hot-Swapping) ohne Verlust der Konfigurationsdaten (z.B. *PCIe-Bus-Extender* mit *PCFaceSwitch-Software*, Anbieter: *HK Meßsysteme GmbH*).

Zuordnung des Windows-Treibers

Nach dem Neustart des Rechners findet das Betriebssystem die neue Hardware und fordert zur Zuordnung eines geeigneten Treibers auf. Dabei ist Folgendes zu beachten: Der verwendete PLX-Controller PEX8311 besteht intern aus 2 Bausteinen, das sind:

- zum einen die 'PCI-PCI-Bridge' (PCI8111) und
- zum anderen das 'PCI-Device' (PCI9056).

ACHTUNG: Beiden Bausteinen muss ein passender Treiber zugewiesen werden!

Nutzen Sie zur erstmaligen Zuordnung der Treiber nicht die empfohlene 'automatische Installation', sondern verweisen Sie den 'Hardware-Assistenten' in dem entsprechenden Dialog auf das Unterverzeichnis 'drv' im Verzeichnis der *PCIe-BaseLab*-Software. Dort sorgen die beiden INF-Dateien

- PLXBRIDGE.INF und
- PPLAB_E.INF

dafür, dass die richtigen Treiber zugeordnet werden. Das sind

- PCI.SYS – der Microsoft-Standard-Treiber für den PCI-Bus, hier als Treiber für die 'PCI-PCI-Bridge',
- PPLAB_E.SYS – der spezielle Windows-Treiber für das 'PCI-Device' *PCIe-BaseLab.*

Sollte dem 'PCI-Device' ein anderer Windows-Treiber (z.B. der generische PLX-Treiber 'PCI9056.SYS') zugeordnet worden sein, ordnen Sie dem 'PCI-Device' nachträglich den richtigen Treiber zu. Verweisen Sie dazu im Geräte-Manager – Treiber aktualisieren – Verzeichnis 'drv' auf 'PPLAB_E.INF'. Als Ergebnis einer erfolgreichen Treiber-Zuordnung finden Sie im Geräte-Manager die Geräte-Klasse 'PCIe-BaseLab' und dort die installierten 'PCIe-BaseLab'-Geräte.

3. Hardware-Eigenschaften und -Erweiterbarkeit

Zur einführenden Beschreibung dient das Blockschaltbild (siehe Anhang). Funktionell gliedert sich die Hardware in fünf Teile:

- 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

Der *PEX8311* ist ein universeller PCIe Endpoint Controller für den seriellen PCIe Bus, ausgestattet mit einem x1 Link. Er dient als Brücke zwischen dem PCIe-Bus und spezifischen Anwenderschaltungen auf dem lokalen Bus. Der *PEX8311* verarbeitet alle typischen Signale und Zugriffsmechanismen auf dem PCIe Bus. Er übersetzt diese in ein universelles Steuer-, Adress- und Dateninterface, an das sich anwenderspezifische Speicher- sowie auch I/O-Einheiten anschließen lassen. Hierzu besitzt er zwei Interfaces, die folgendermaßen bezeichnet sind:

- das PCIe Bus Interface
- das Local Bus Interface

Die Interfaces des *PEX8311*-Controllers sind für den Anwender der *PCIe-BaseLab*-Karte von unterschiedlicher Bedeutung und werden nachfolgend beschrieben.

PCIe Bus Interface

Das *PCIe Bus Interface* dient zur Ankopplung des Controllers an den PCI Express Bus. Es ist auf der *PCIe-BaseLab*-Leiterkarte bereits voll verdrahtet, der Anwender hat hier keinen weiteren Aufwand zu betreiben.

Local Bus Interface

Das *Local Bus Interface* ist für den Anwender wichtig, da hier seine Schaltungsapplikation angeschlossen wird. Es ist universell ausgelegt und erlaubt den Betrieb von Hardwareperipherie mit Datenbusbreiten von 8, 16 oder 32 Bit sowie einer Adressbusbreite von maximal 32 Bit.

Anwenderspezifische Komponenten steuern den Datenaustausch über das Local Bus Interface zweckmäßig mit Hilfe klassischer Signale wie /LHOLD, /HOLDA, /WAIT, /READY oder /LW/R.

Die angeschlossene Peripherie kann selbst aus einem Mikroprozessorsystem bestehen oder - im einfachsten Fall - auch als Datenlatch ausgebildet sein.

Die Architektur des PCIe Endpoint Controllers unterstützt auch die Einbindung von Speicher. Es lassen sich bis zu 4 GByte pro Adress-Region adressieren, zwei lokale Adress-Regionen sind möglich.

Der *PEX8311* besitzt einen internen Registersatz, um Initialisierungsdaten zu speichern, Einstellung, Aktivierung und Deaktivierung der Betriebsarten zu bewirken oder Daten auszutauschen. Auf den Registersatz kann sowohl über das PCIe Bus Interface als auch über das Local Bus Interface zugegriffen werden. Die Funktionen des PCI-Controllers sind somit für beide Seiten transparent und nutzbar.

Für den schnellen Datentransfer ohne Inanspruchnahme der Host-CPU (Bus Master Transfer Mode) sind zwei unabhängige DMA-Kanäle vorgesehen, deren Startadressen und Transferzähler ebenfalls über Register einzustellen sind.

Für die BIOS- Erweiterung lassen sich ROMs mit parallelem Interface an den *PEX8311* anschließen.

Der Controller *PEX8311* erlaubt auch die anwenderspezifische Generierung von Interrupts, welche sowohl von lokaler als auch von der PCI-Seite ausgehen können.

Der kompletten technischen Beschreibung des *PEX8311* dient ein weiteres technisches Handbuch, welches der Hersteller des Controllers herausgibt und pflegt. Dieses kann von den Webseiten der Firma *PLX Technology, Inc.* heruntergeladen oder von der mitgelieferten Produkt-CD-ROM entnommen werden.

Achtung: Wir können nicht für die Aktualität der mit diesem Produkt gelieferten Datenblätter anderer Hersteller garantieren. Diese sollten nur als erste Information dienen. Es ist jedoch ratsam, immer die aktuellsten Datenblätter und Handbücher der jeweiligen Hersteller zu beschaffen und als Grundlage für die eigene Beschäftigung mit den Komponenten zu nutzen. Entsprechende WEB-Adressen befinden sich im Anhang dieses Handbuchs.

Serial Configuration EEPROM

PCIe-BaseLab arbeitet mit einem seriellen EEPROM von 2kByte Größe, das steckbar in einer Fassung auf der Leiterkarte installiert ist. Es enthält obligatorische Konfigurationsdaten, die den *PEX8311* Controller speziell für die Anwendung *PCIe-BaseLab* initialisieren. Das EEPROM kann editiert und überschrieben werden. Ein praktischer Editor für den Code des seriellen EEPROMS ist im mitgelieferten Softwarepaket in Form des Programms *PPLABMON.EXE* enthalten.

Programmable Logic Device (PLD), ispXPLD5768

Das auf der *PCIe-BaseLab*-Karte eingesetzte PLD gehört zur Familie *ispXPLD 5000MX* des Herstellers *Lattice Semiconductor*. Diese PLD-Familie zeichnet sich durch eine hohe Leistungsfähigkeit und Flexibilität aus. Neben umfangreicher Logik ist es möglich, in den enthaltenen Multi Function Blocks auch Single- oder Dual-Ported RAM bzw. Fifo-Speicher zu implementieren. Die Ein- und Ausgänge können in verschiedenen Schnittstellenstandards wie TTL, CMOS, LVCMOS, LVDS, HSTL oder SSTL arbeiten. Integrierte PLL-Systeme erlauben ein einfaches Taktmanagement. Die PLD-Serie verfügt über einen integrierten Flash-Speicher für die Konfiguration, der das PLD unmittelbar nach Zuschalten der Versorgungsspannung einsatzbereit macht. Das auf der *PCIe-BaseLab*-Karte verwendete PLD besitzt 768 Makrozellen (FlipFlops), maximal 384 kBits konfigurierbaren RAM und kann mit einer Systemtaktfrequenz von maximal 250 MHz getaktet werden. Seine Ausgänge liefern einen Pegelhub von 3,3 V, die Eingänge sind 5-V-kompatibel. Die Programmierung des PLDs kann über das auf der *PCIe-BaseLab*-Karte integrierte ISP-Interface erfolgen, ein externes Programmiergerät ist nicht erforderlich. Mehr Informationen zum PLD findet der Leser auf den Web-Seiten der Firma *Lattice Semiconductor* (Web-Adressen im Anhang dieses Handbuchs).

'In-System Programmable' Interface (ISP)

Das ISP-Interface (Buchse X1, JEDEC-Support) ermöglicht das Downloaden von PLD-Firmware, ohne das PLD auf einem externen Programmiergerät bearbeiten zu müssen. Es kann in der Schaltung eingebaut (deshalb: 'in System Programmable', ISP) verbleiben. Die hierfür erforderliche Software *ispVMSystem* kann kostenfrei von den Webseiten der Firma Lattice Semiconductor herunter geladen werden. Ein Programmieradapter kann als parallele oder USB-Version bei Ihrem lokalen Händler für *Lattice*-Bauelemente geordert werden.

Pin Header Block (User Interface)

Der *Pin Header Block* besteht aus zweireihigen Pfosten-Steckverbindern im 2-mm-Raster (2 Stück mit jeweils 2x50 Pins, 1 Stück mit 2x20 Pins). Diese sind an den äußeren Kanten der *PCIe-BaseLab*-Karte angeordnet und bilden die Schnittstelle zu anwenderspezifischer Hardware, das *User Interface*.

Dieses Interface bietet Zugriff auf alle lokalen Adress-, Daten- und Steuerleitungen des *PEX8311* Controllers sowie auf 106 der 193 I/O-Pins des installierten PLDs *ispXPLD5768*. Einige weitere Anschlüsse erlauben die Abnahme oder das externe Einspeisen von Taktsignalen. In regelmäßigen Abständen sind Masse-Pins zwischengeordnet.

Die mitgelieferte Tochterkarte ist konstruktiv so gestaltet, dass sie auf die Pfosten-Steckverbinder des *Pin Header Blocks* der *PCIe-BaseLab*-Karte aufgesteckt und mit ihr verschraubt werden kann. Dabei werden alle Pins auf die Tochterkarte übertragen und stehen dort zum Anschluss anwenderspezifischer Hardware bereit. Achtung: Stromversorgungen und Masseanschlüsse werden nicht automatisch mit den Versorgungs-Layern der Tochterkarte verbunden. Hierfür ist ein zusätzliches Kabel (ebenfalls im Lieferumfang) erforderlich, welches wahlweise mit einem SATA-Stromversorgungsstecker des Hostrechners oder vorbereiteten Lötanschlüssen auf der *PCIe-BaseLab*-Karte verbunden wird. Alle Spannungsseinspeisungen (mit Ausnahme des Masseanschlusses) sind auf der Tochterkarte über auswechselbare Schmelzsicherungen geführt, welche die Anwenderelektronik und das Stromversorgungsmodul des Hostrechners vor Überströmen schützen sollen. Farbige LEDs signalisieren das Anliegen der Versorgungsspannungen auf der Tochterkarte.

Die Trennung von *PCIe-BaseLab*-Karte und Tochterkarte ist möglich, sollte jedoch aufgrund der starken Kraftwirkung der Steckverbinder vorsichtig und unter Zuhilfenahme eines Werkzeugs geschehen (bitte mit einem geeigneten Schraubendreher gleichmäßig parallel abhebeln).

4. Die PLD-Beispielanwendungen

Die Beispielanwendungen sind ausschließlich als 'Glue Logik' im PLD untergebracht. Sie sollen musterhaft mögliche Hardware-Applikationen zeigen, mit denen *PCIe-BaseLab* ausgestattet werden kann, ohne zusätzliche Komponenten auf der Tochterkarte installieren zu müssen.

Der Anwender kann die vorinstallierten Beispielanwendungen im Rahmen der technischen Möglichkeiten an seine Bedürfnisse anpassen und/oder auch eigene Logik in den PLD implementieren.

Die VHDL-Quellcodes der Beispielanwendungen liegen dem Produktpaket bei. Für die Bearbeitung und Compilierung benötigt man die '*ispLever-Classic*' IDE Software, welche kostenfrei von den WEB-Seiten der Firma *Lattice Semiconductor* herunter geladen werden kann.

Das mit diesem Produkt mitgelieferte Monitorprogramm erlaubt die Kommunikation mit den Beispielanwendungen zu Testzwecken. Weitere Ausführungen hierzu: *'Das Monitorprogramm PPLABMON.EXE'*, Abschnitt 8.

Nachfolgend erhält der Leser weiterführende Informationen über die hardwaretechnische Architektur der vorinstallierten Beispielanwendungen.

Synchrones RAM

Das *synchrone statische RAM* hat eine Kapazität von 24 kByte und ist in der Form 6 k x 4 Bytes organisiert. Die Hardware ist für Byte-, Word- oder DWord-Schreib- und Lesezugriffe ausgelegt. Burst-Transfers werden unterstützt. Zum Adressbereich des RAMs: siehe Abschnitt 5, '*Beschreibung der PCIe-BaseLab Programmierschnittstelle*'.

Das integrierte RAM lässt sich auch mit Hilfe des mitgelieferten Monitorprogramms *PPLABMON.EXE* beschreiben und lesen. Näheres hierzu: siehe Abschnitt 8, '*Das Monitorprogramm PPLABMON.EXE*'*.*

FIFO

Die Beispielanwendung *FIFO-Speicher* ist gegenwärtig im Lieferumfang von *PCIe-BaseLab* noch nicht enthalten. Sie wird unseren Kunden in absehbarer Zeit zur Nutzung bereitgestellt werden.

16Bit I/O-Register

Diese Beispielanwendung besteht aus einem *16-Bit-Datenregister* zur Ausgabe statischer (gelatchter) Daten sowie einem separaten Eingangsdatenpfad, über den sich der momentane logische Zustand von 16 Input-Leitungen abfragen lässt. Insgesamt nutzt die Beispielanwendung '16-Bit-I/O-Register' damit 32 Pins der an den Pfostenleisten (Pin Header Block) frei zur Verfügung stehenden I/Os: 16 Output-Pins und 16 Input-Pins. Diese sind auf den Leisten so angeordnet, dass sie sich mit einem Kurzschluss-Stecker leicht bitweise überbrücken lassen. Damit wird ermöglicht, das über das Output-Latch ausgegebene Datenwort wieder zurückzulesen.

Das 16-Bit-I/O-Latch ist über eine Zugriffsadresse (Schreiben und Lesen) im Memory-Adressbereich der *PCIe-BaseLab*-Karte erreichbar (siehe Abschnitt 5, '*Beschreibung der PCIe-BaseLab Programmierschnittstelle*')*.*

Das 16-Bit-I/O-Register lässt sich auch mit Hilfe des mitgelieferten Monitorprogramms *PPLABMON.EXE* beschreiben und lesen (siehe Abschnitt 8, '*Das Monitorprogramm PPLABMON.EXE*')*.*

Die Lage der I/O-Pins des 16-Bit-Registers auf dem Pin Header Block findet der Leser in tabellarischer Form im Anhang dieses Handbuches.

5. Die *PCIe-BaseLab*-Programmierschnittstelle

Die *PCIe-BaseLab*-Programmierschnittstelle stellt eine Reihe von Funktionen bereit, die von C/C++-Programmen aus verwendet werden können, um mit der *PCIe-BaseLab*-Karte zu kommunizieren. Zur Nutzung dieser Funktionen stehen die folgenden Dateien Bereit:

- PPLABAPI.H: C-Header-Datei - Definition der API-Funktionen,
- PPLABERR.H: C-Header-Datei - Definition der Fehlerkonstanten,
- PPLABAPI.DLL: dynamische Linkbibliothek - Implementierung der API-Funktionen,
- PPLABAPI.LIB: Import-Bibliothek zum impliziten Laden der DLL von VC-Applikationen aus.

Um die API-Funktionen nutzen zu können, muss die Header-Datei 'PPLABAPI.H' 'includet' und die Treiber-API-DLL 'PPLABAPI.DLL' geladen werden.

Die Nutzung der Header-Dateien

In den C/C++-Quelldateien muss die Header-Datei 'PPLABAPI.H' mit einer '#include'-Anweisung geladen werden, um die API-Funktionen aufrufen zu können.

Die Header-Datei 'PPLABERR.H' kann zum Nachschlagen der Fehlercodes verwendet werden und muss in der Regel nicht 'includet' werden.

Das Laden der Treiber-API-DLL

Das Laden der Treiber-API-DLL 'PPLABAPI.DLL' kann entweder

- explizit mit Hilfe der Win32-Funktionen 'LoadLibrary' und 'GetProcAddress' oder
- implizit durch Nutzung der mitgelieferten Import-Bibliothek

geschehen.

Das Beispielprogramm 'LoadDLL' demonstriert das explizite Laden der DLL.

Bei der Nutzung der Microsoft-Entwicklungsumgebungen (VC6, Visual Studio xx) kann die mitgelieferte Importbibliothek zum impliziten Laden der DLL verwendet werden. In allen Beispielprogrammen – außer 'LoadDLL' – wird das implizite Laden verwendet.

Format der API-Funktionen, Fehlercode

Alle API-Funktionen der *PCIe-BaseLab*-Schnittstelle haben einen ähnlichen Prototyp. Der Funktionsname beginnt immer mit dem Präfix '`PPLAB_`'.

Der Rückgabewert der API-Funktionen ist immer vom Typ '`DWORD`' und signalisiert, ob die Funktion erfolgreich war. Der Rückgabewert '0' zeigt an, dass die Funktion ohne Fehler ausgeführt werden konnte. Ein Wert ungleich '0' signalisiert einen Fehler und stellt einen Fehlercode dar, der Auskunft über die Ursache des Fehlers gibt. Die möglichen Fehlercodes sind in der Header-Datei 'PPLABERR.H' definiert und nutzen Werte zwischen `0x20010000..0x20FF0000`. Zusätzlich werden manchmal noch Fehlercodes des Betriebssystems hinzugefügt, die die Bits `0...11` verwenden.

Liefert eine API-Funktion Daten, so werden diese über Referenzparameter (Adress-Zeiger) bereitgestellt. Dadurch können auch mehrere Informationen gleichzeitig geliefert werden.

Speicher-Konfigurationen

Die *PCIe-BaseLab*-Karte verfügt standardmäßig über 32 kBytes Speicher. Dieser Speicher wird verwendet, um mögliche Applikationen zu demonstrieren. Zu diesen Applikationen zählen z.B.:

- RAM – Schreiben und Lesen von mehreren kBytes Daten
- IO-REGISTER – Setzen und Abfragen von I/O-Pins
- FIFO – Starten von kontinuierlichen Datentransporten per DMA

Die jeweilige Konfiguration Ihrer Karte(n) entnehmen Sie bitte den beigefügten Unterlagen.

Beispiel-Konfiguration 1:

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

Methoden für den Speicher-Zugriff

Um auf den Speicher der *PCIe-BaseLab*-Karte zuzugreifen, stehen drei Methoden zur Verfügung:

- KERNEL
- MAPPED
- DMA

Die KERNEL-Methode ist die Standard-Vorgehensweise zum Zugriff auf Speicherbereiche einer externen PCI-Steckkarte. Dabei erfolgt der Speicherzugriff durch den Windows-Treiber der *PCIe-BaseLab*-Karte auf Kernelebene. Der Treiber übernimmt dabei auch die Synchronisierung von gleichzeitigen Speicherzugriffen. Die dazugehörigen API-Funktionen sind `PPLAB_ReadMemory` und `PPLAB_WriteMemory`.

Für die MAPPED-Methode stehen die API-Funktionen `PPLAB_MapMemory` und `PPLAB_UnmapMemory` zur Verfügung. Der entsprechende Speicherbereich der *PCIe-BaseLab*-Karte wird in den Adressraum der jeweiligen Applikation 'eingeblendet'. Danach sind direkte Zugriffe auf diesen Speicherbereich ohne 'Umweg' über die Kernelebene möglich. Um gleichzeitige Zugriffe zu verhindern ist eine explizite Synchronisierung nötig. Das Beispielprogramm *'MapMem'* veranschaulicht diese Methode.

Die DMA-Methode dient dazu, größere Datenmengen durch Nutzung der DMA-Kanäle des PLX-Controllers zu transportieren. Dabei werden die API-Funktionen `PPLAB_ReadPlxRegister` und `PPLAB_WritePlxRegister` verwendet. Das Beispielprogramm *'UseDMA'* demonstriert die nötige Vorgehensweise.

6. API-Funktionen - Referenz

Die Programmier-Schnittstelle umfasst die folgenden API-Funktionen:

- Abfrage der Anzahl aktiver *PCIe-BaseLab*-Karten
  `PPLAB_GetNumberOfDevices`
- Öffnen und Schliessen einer *PCIe-BaseLab*-Karte
  `PPLAB_OpenDevice`
  `PPLAB_CloseDevice`
- Abfrage der Treiber-Version
  `PPLAB_GetDriverVersion`
- Abfrage von Informationen über den PCI-Steckplatz einer *PCIe-BaseLab*-Karte
  `PPLAB_GetDeviceProperties`
- Rücksetzen einer *PCIe-BaseLab*-Karte
  `PPLAB_ResetDevice`
- Lesen und Schreiben der Register im PCI-Konfigurationsbereich einer *PCIe-BaseLab*-Karte
  `PPLAB_ReadPciRegister`
  `PPLAB_WritePciRegister`
- Lesen und Schreiben der Register des PLX-Controllers einer *PCIe-BaseLab*-Karte
  `PPLAB_ReadPlxRegister`
  `PPLAB_WritePlxRegister`
- Lesen und Schreiben des EEPROM-Inhaltes einer *PCIe-BaseLab*-Karte
  `PPLAB_ReadEEPROM`
  `PPLAB_WriteEEPROM`
- Lesen und Schreiben im Speicherbereich einer *PCIe-BaseLab*-Karte
  `PPLAB_ReadMemory`
  `PPLAB_WriteMemory`
- 'Einblenden' und 'Ausblenden' eines Speicherbereiches einer *PCIe-BaseLab*-Karte
  `PPLAB_MapMemory`
  `PPLAB_UnmapMemory`
- Abfragen der physischen Adresse der Speicherbereiche einer *PCIe-BaseLab*-Karte
  `PPLAB_GetPhysicalAddressOfUserRegion`
- Allokieren und Freigeben von zusammenhängenden Speicherbereichen
  `PPLAB_AllocateContMemory`
  `PPLAB_FreeContMemory`

Abfrage der Anzahl aktiver *PCIe-BaseLab*-Karten

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

Mit dieser Funktion kann abgefragt werden, wie viele *PCIe-BaseLab*-Karten in diesem Rechner gerade aktiv sind. Dazu ist die Adresse einer `DWORD`-Variable als Parameter `pNumberOfDevices` zu übergeben. Nach erfolgreicher Ausführung der Funktion enthält diese Variable die Anzahle der aktiven Karten.

Der *PCIe-BaseLab*-Treiber unterstützt bis zu 16 *PCIe-BaseLab*-Karten pro PC.

Üblicherweise wird diese Funktion beim Start eines Programmes als erste API-Funktion aufgerufen um festzustellen, ob überhaupt *PCIe-BaseLab*-Karten in diesem Rechner aktiv sind.

Beispiel:

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

Öffnen einer *PCIe-BaseLab*-Karte

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

Mit dieser Funktion wird die Nutzung einer *PCIe-BaseLab*-Karte begonnen – das 'Gerät' mit dem Parameter `deviceId` wird geöffnet. Als `deviceId` kann ein Wert zwischen `1` und `16` übergeben werden. Ist nur eine Karte aktiv, hat diese das Id `1`, 2 Karten heißen immer `1` und `2` usw.. Die Funktion liefert bei Erfolg ein 'Handle' für das betreffende Gerät zurück. Dazu ist die Adresse einer `HANDLE`-Variable als Parameter `pHDevice` zu übergeben.

Da das `deviceHandle` als Parameter bei allen weiteren API-Funktionen benötigt wird (außer bei `PPLAB_GetNumberOfDevices`), muss eine *PCIe-Baselab*-Karte immer als erstes mit dieser Funktion geöffnet werden, bevor weitere Funktionen aufgerufen werden können.

Eine geöffnete *PCIe-Baselab*-Karte sollte mit der Funktion `PPLAB_CloseDevice` wieder geschlossen werden, bevor das betreffende Programm beendet wird.

Beispiel:

```
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);
```

Schliessen einer *PCIe-BaseLab*-Karte

```
DWORD PPLAB_CloseDevice (HANDLE hDevice);
```

Mit dieser Funktion wird eine *PCIe-BaseLab*-Karte wieder geschlossen. Als Parameter ist das `deviceHandle` zu übergeben, das mit `PPLAB_OpenDevice` erzeugt wurde. Das `deviceHandle` ist damit nicht länger gültig. Nach dem Aufruf von `PPLAB_CloseDevice` kann somit für diese Karte keine weitere API-Funktion mehr aufgerufen werden.

Beispiel:

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

Abfrage der Version des Windowstreibers

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

Diese Funktion dient dem Abfragen der Version des installierten Windowstreibers. Neben dem `deviceHandle` ist als Parameter die Adresse einer `DWORD`-Variable zu übergeben, die dann die Versionsnummer des Windowstreibers enthält.

Beispiel:

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

Abfrage von Informationen über den PCIe-Steckplatz

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

Mit dieser Funktion werden Informationen über den PCIe-Steckplatz der betreffenden *PCIe-BaseLab*-Karte abgefragt. Dazu sind die Adressen von vier `DWORD`-Variablen zu übergeben, die nach Ausführung der Funktion die Werte für Slot-, Bus-, Geräte- sowie Funktions-Nummer wiedergeben. Anhand dieser Informationen ist bei mehreren aktiven *PCIe-BaseLab*-Karten eine Identifizierung möglich.

Beispiel:

```
…
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);
```

Rücksetzen einer *PCIe-BaseLab*-Karte

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

Durch Aufruf dieser Funktion wird auf der betreffenden *PCIe-BaseLab*-Karte ein 'Reset' ausgelöst. Für den Parameter `resetFlags` sind bisher keine Werte definiert, daher sollte der Wert `0` übergeben werden.

Beispiel:

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

Lesen der Register im PCI-Konfigurationsbereich

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

Mit dieser Funktion kann der 256 Bytes große PCI-Konfigurationsbereich der *PCIe-BaseLab*-Karte ausgelesen werden. Als Parameter `offset` ist der Adress-Offset in Bytes (0,4,8..252) anzugeben, für den Wert ist die Adresse einer `DWORD`-Variablen als `pValue` zu übergeben.

Beispiel:

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

Schreiben der Register im PCI-Konfigurationsbereich

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

Diese Funktion dient zum Beschreiben die Register im PCI-Konfigurationsbereich. Als Parameter `offset` ist der Adress-Offset in Bytes (0,4,8..252) anzugeben, der Wert als Parameter `value`.

Bitte beachten Sie: Es sind nur einzelne Register im PCI-Konfigurationsbereich beschreibbar! Weisen Sie diesen Registern nur sinnvolle Werte zu!

Beispiel:

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

Lesen der Register des *PEX8311*-Controllers

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

Mit dieser Funktion können die Register des PEX8311-Controllers ausgelesen werden. Zu diesen Registern zählen die 'Local Configuration'-Register sowie die Runtime- und DMA-Register (siehe Dokumentation zum PLX-Controller). Als Parameter `offset` ist der Adress-Offset in Bytes anzugeben, für den Wert ist die Adresse einer `DWORD`-Variablen als `pValue` zu übergeben.

Beispiel:

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

Schreiben der Register des *PEX8311*-Controllers

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

Diese Funktion dient zum Beschreiben der Register des PEX8311-Controllers. Als Parameter `offset` ist der Adress-Offset in Bytes anzugeben, der Wert als Parameter `value`.

Bitte achten Sie darauf, nur sinnvolle Werte den entsprechenden PEX8311-Registern zuzuweisen!

Beispiel:

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

Lesen von Daten aus dem seriellen EEPROM

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

Mit dieser Funktion können Daten aus dem EEPROM der *PCIe-BaseLab*-Karte ausgelesen werden. Als Parameter `offset` ist der Adress-Offset in Bytes (0,4,8..0x60) anzugeben, für den Wert ist die Adresse einer `DWORD`-Variablen als `pValue` zu übergeben.

Beispiel:

```
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);
```

Schreiben von Daten in den seriellen EEPROM

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

Diese Funktion dient zum Schreiben von Daten in den EEPROM. Als Parameter `offset` ist der Adress-Offset in Bytes (0,4,8..0x60) anzugeben, der Wert als Parameter `value`.

WICHTIG - bitte beachten Sie:

Sichern Sie vor dem Ändern von EEPROM-Daten den ursprünglichen Inhalt, z.B. mit dem Monitorprogramm PPLABMON. Weisen Sie den EEPROM-Registern nur sinnvolle Werte zu! Fehlerhafte EEPROM-Inhalte können zu Fehlern der *PCIe-BaseLab*-Karte und des Gesamtsystems führen!

Beispiel:

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

Lesen des Speicherbereiches

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

Mit dieser Funktion kann der Speicher der *PCIe-Baselab*-Karte ausgelesen werden. Als Parameter `offset` ist der Adress-Offset in Bytes (0,4,8..) anzugeben, für den Wert ist die Adresse einer `DWORD`-Variablen als `pValue` zu übergeben.

Beispiel:

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

Schreiben des Speicherbereiches

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

Diese Funktion dient zum Beschreiben des *PCIe-BaseLab*-Speichers. Als Parameter `offset` ist der Adress-Offset (0,4,8..) in Bytes anzugeben, der Wert als Parameter `value`.

Siehe dazu auch Abschnitt 'Speicher-Konfigurationen'.

Beispiel:

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

'Einblenden' eines Speicherbereiches in den Anwendungs-Adressraum

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

Mit dieser Funktion wird ein Speicherbereich der *PCIe-BaseLab* Karte in den Adressraum der Anwendung 'eingeblendet'. Mit dem Parameter `userRegion` wird entweder `USER_REGION_0` oder `USER_REGION_1` ausgewählt. Als Ergebnis der Funktion enthält eine `DWORD`-Variable, deren Adresse als Parameter `pVirtAddress` zu übergeben ist, die virtuelle Adresse dieses Speicherbereiches. Über diese Adresse kann danach direkt von der Anwendung in den Speicherbereich der *PCIe-BaseLab* Karte geschrieben bzw. aus ihm gelesen werden.

ACHTUNG: In der aktuellen Konfiguration der *PCIe-BaseLab*-Karte ist nur ein Speicherbereich vorhanden. Als Parameter `userRegion` ist daher immer `USER_REGION_0` anzugeben!

Beispiel:

```
DWORD virtAddress;
error = PPLAB_ MapMemory (deviceHandle, USER_REGION_0, &virtAddress);
if (error)
 … // error handling
else
{
 DWORD* pMem = (DWORD*)(virtAddress + 4);       // Adresse an Offset 4
einstellen
 value = *pMem;                                 // direktes Auslesen eines
DWORDs
 pMem++;                                  // Adresse an Offset 8 einstellen
 *pMem = 0x12345678;                      // direktes Schreiben eines DWORDs
}
```

'Ausblenden' eines Speicherbereiches aus dem Anwendungs-Adressraum

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

Mit diese Funktion wird die 'Einblendung' eines Speicherbereiches mit `PPLAB_ MapMemory` wieder rückgängig gemacht. Als Parameter ist wieder die betreffenden `userRegion` sowie die virtuelle Adresse, die `PPLAB_ MapMemory` lieferte, zu übergeben.

Nach Aufruf dieser Funktion sind keine direkten Zugriffe zu diesem Speicherbereich mehr möglich!

Beispiel:

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

Abfragen der physischen Adresse der 'User-Regions'

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

Um die DMA-Kanäle der *PCIe-BaseLab*-Karte nutzen zu können, müssen die physischen Adressen der beteiligten Speicherbereiche zur Verfügung stehen.

Mit dieser Funktion kann die physische Adresse der jeweiligen 'user region' abgefragt werden. Mit dem Parameter `userRegion` wird entweder `USER_REGION_0` oder `USER_REGION_1` ausgewählt. Als Ergebnis der Funktion enthält eine `DWORD`-Variable, deren Adresse als Parameter `pPhysAddress` zu übergeben ist, die physische Adresse dieses Speicherbereiches. Diese Adresse kann danach als die 'lokale' Adresse beim Initialisieren eines DMA-Transfers verwendet werden.

ACHTUNG: In der aktuellen Konfiguration der *PCIe-BaseLab*-Karte ist nur ein Speicherbereich vorhanden. Als Parameter `userRegion` ist daher immer `USER_REGION_0` anzugeben!

Beispiel:

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

Allokieren eines zusammenhängenden Speicherbereiches

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

Bei einem DMA-Transfer ist es vorteilhaft, einen Speicherbereich nutzen zu können, der physisch zusammenhängend allokiert wurde. Dadurch ist es nicht nötig, die physischen Adressen aller beteiligten Speicher-Seiten zu ermitteln.

Mit dieser Funktion wird ein physisch zusammenhängender Speicherbereich angefordert. Als Parameter `size` ist die Größe des Speichers in Bytes anzugeben – hier sind Vielfache von 4096 (Größe einer Speicherseite) sinnvoll. Als Ergebnis der Funktion enthält die `DWORD`-Variable, deren Adresse als Parameter `pVirtAddress` zu übergeben ist, die virtuelle Adresse dieses Speichers. Über diese Adresse kann danach direkt von der Anwendung aus in diesen Speicherbereich geschrieben bzw. aus ihm gelesen werden. Die `DWORD`-Variable, deren Adresse als Parameter `pPhysAddress` übergeben wird, enthält die physische Adresse dieses Speicherbereiches. Diese Adresse kann danach als die 'PCI'-Adresse beim Initialisieren eines DMA-Transfers verwendet werden.

Die Größe des Speicherbereiches sollte einen Wert von 0x7FFFFF (dezimal: 8388607, 8 MB - 1) nicht überschreiten – das ist die größtmögliche Bytezahl für einen DMA-Transfer (Bits 0..22 des DMASIZx-Register, siehe PEX8311-Handbuch)!

In der aktuellen Version des Windows-Treibers können 32 solcher zusammenhängenden Speicherbereiche angefordert werden!

Beispiel:

```
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, psysical: 0x%08X \n", virtPCIAddr, physPCIAddr);
```

Freigeben des zusammenhängenden Speicherbereiches

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

Mit dieser Funktion wird der Speicherbereich, der mit `PPLAB_AllocateContMemory` angefordert wurde, wieder freigegeben. Als Parameter `virtAddress` ist die virtuelle Adresse zu übergeben.

Beispiel:

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

7. Die API-Beispielprogramme (C++)

Die Beispielprogramme demonstrieren die Vorgehensweise bei der Nutzung der verschiedenen API-Funktionen der *PCIe-BaseLab*-Programmierschnittstelle. Es sind C++-Konsolenanwendungen, deren Programmierung bewusst einfach gehalten wurde. Zu jedem Beispielprogramm gehört eine gleichnamige `cpp`-Quellcode-Datei. Die Programmierung sollte mit jedem C++-Compiler möglich sein. Bei Microsoft-Werkzeugen kann die Import-Bibliothek verwendet werden.

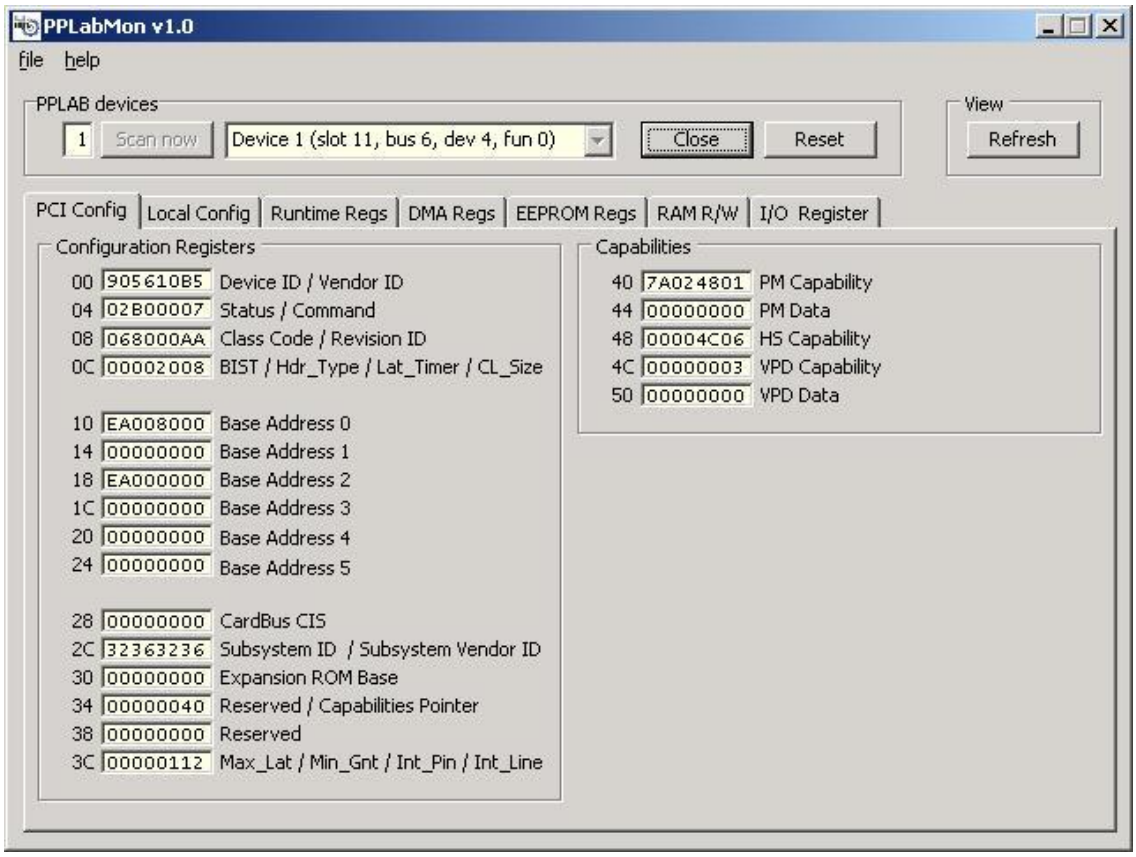
In den Verzeichnissen befinden sich neben der Quellcode-Datei auch die zur Erstellung der Programme verwendeteten Projektdateien (MS Visual C++ 6.0).

Folgende Beispiele stehen zur Verfügung:

- ApiTest - Aufruf der Basis-API-Funktionen
- LoadDLL - explizites Laden der Treiber-API-DLL
- MapMem - direktes Lesen und Schreiben im 'eingeblendeten' Speicherbereich
- RdWrRAM - Lesen und Schreiben im Speicher mit 'KERNEL'-Methode
- ReadDMARegs - Lesen der DMA-Register
- ReadLocalCfg - Lesen der 'local config register'
- ReadPCICfg - Lesen der 'PCI config register'
- ReadRuntimeRegs - Lesen der 'runtime register'
- Reset - Rücksetzen der *PCIe-BaseLab*-Karte
- TestEEPROM - Lesen und Schreiben der EEPROM-Register
- UseDMA - Initialisieren und Starten von DMA-Transfers

8. Das Monitorprogramm PPLABMON.EXE

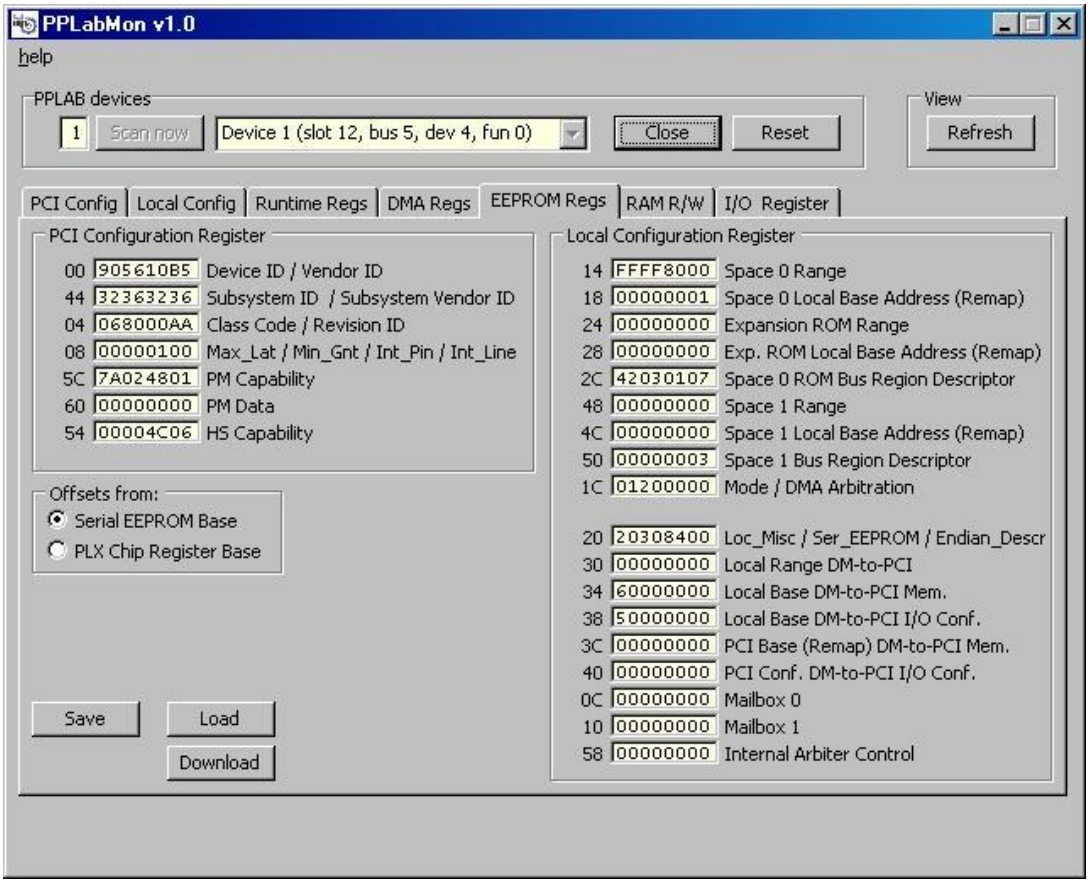
Das Monitorprogramm 'PPLABMON.EXE' ist eine grafische Windows-Anwendung, die einen umfassenden Einblick in alle Bereiche der der *PCIe-Baselab*-Karte gestattet.



Folgende Möglichkeiten stehen zur Verfügung:

- PCI Config - Lesen der 'PCI config register'
- Local Config - Lesen der 'local config register'
- RuntimeRegs - Lesen der 'runtime register'
- DMARegs - Lesen der 'DMA register'
- EEPROMRegs - Bearbeiten der EEPROM-Register
- RAM R/W - Lesen und Schreiben im RAM
- I/O Register - Setzen und Abfragen der Pins des 16-Bit-I/O-Regsiters

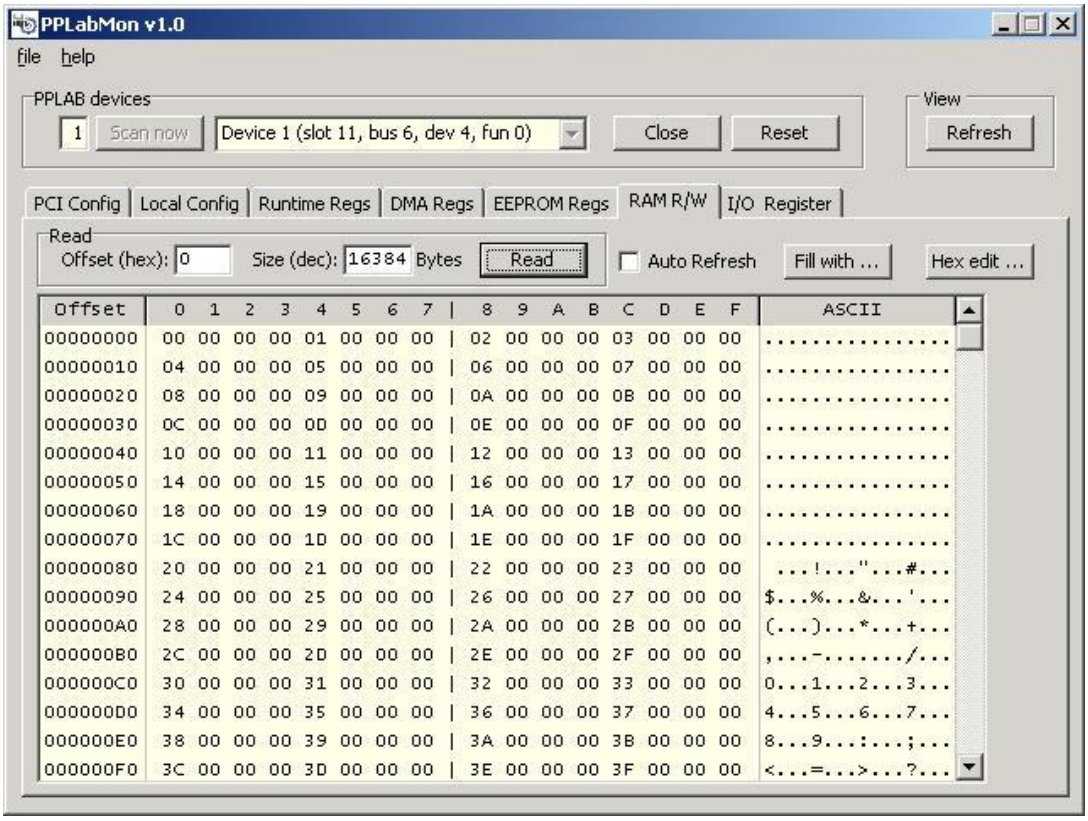
Bearbeiten der EEPROM-Register



Auf der 'EEPROM-Register'-Seite stehen folgende Möglichkeiten zur Verfügung:

- Register auslesen - Auslesen des aktuellen EEPROM-Inhaltes ('Refresh')
- Register ändern - Ändern und 'Download' eines EEPROM-Registers (Doppelklick)
- Speichern in Datei - Speichern aller Register in eine Datei in ASCII-Format – editierbar ('Save')
- Laden aus Datei - Laden aller Registerinhalte aus einer Datei ('Load')
- Gesamt-Download - Download aller Register ins EEPROM ('Download')

Lesen und Schreiben im RAM



Auf der 'RAM R/W'-Seite stehen folgende Möglichkeiten zur Verfügung:

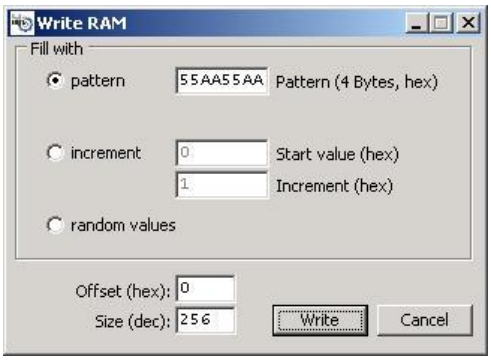
- Offset ändern - Bestimmen der Startadresse
- RAM-Größe ändern - Die angezeigte RAM-Größe einstellen
- RAM auslesen - Auslesen des aktuellen RAM-Inhaltes ('Read')
- Automatisches auffrischen - Der RAM-Inhalt wird automatisch gelesen und aktualisiert
- Schreiben in den RAM - Den RAM mit verschiedenen Werten füllen ('Fill with …')
- Ändern einzelner Bytes - Einzelne Bytes editieren ('HEX edit …')

<u>Schreiben in den RAM</u>

Es stehen folgende Möglichkeiten zur Verfügung, um in das RAM zu schreiben:

- RAM mit einem Muster füllen ('pattern')
- RAM inkrementierend füllen ('increment')
- RAM mit zufälligen Werten füllen ('random values')

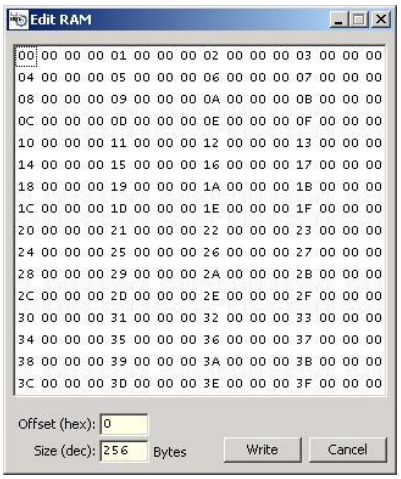
In diesem Fenster sind alle Edit-Felder editierbar.



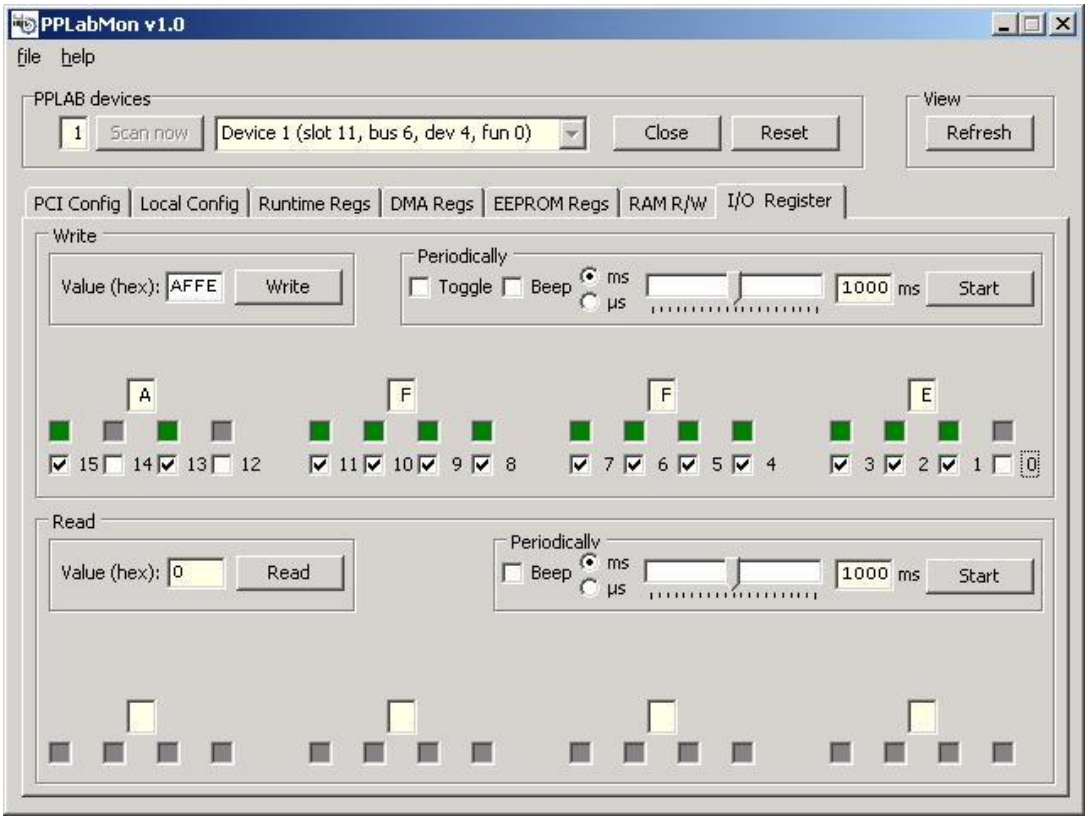
<u>Ändern einzelner Bytes</u>

Dieses editierbare HEX-Dump-Fenster zeigt immer 256 Bytes an, beginnend mit dem vorher eingestellten Offset. Zum Ändern sind folgende Schritte zu tun:

- das zu editierende Byte auswählen
- 'Edit'-Modus starten - Doppelklick oder 'Enter'
- neuen Wert eingeben und mit 'Enter' bestätigen
- mit 'Write' geänderten Wert des 256-Byte-HEX-Dumps in den RAM schreiben



Setzen und Abfragen der Pins des 16-Bit-I/O-Regsiters

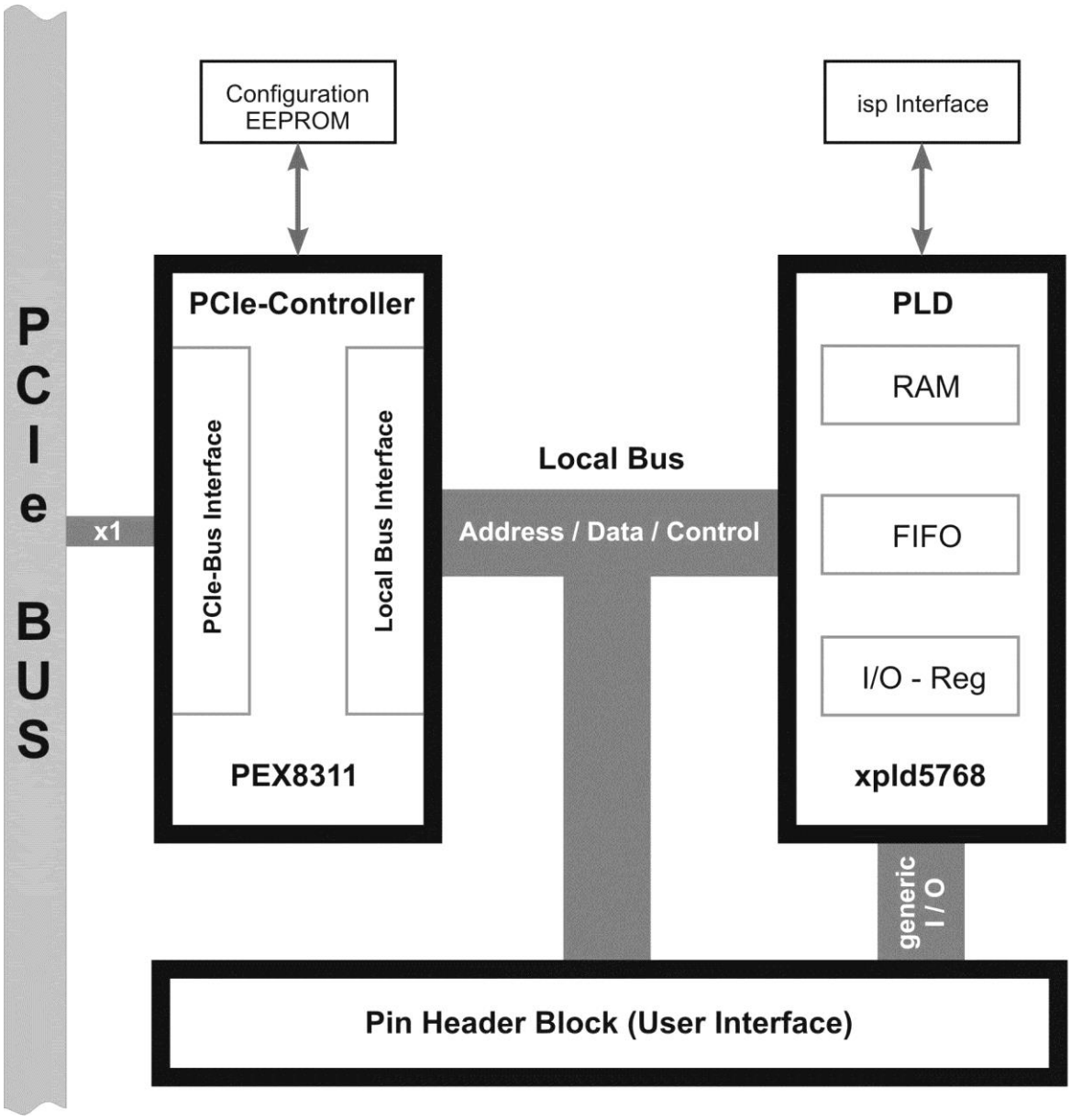


Auf der 'I/O Register'-Seite stehen folgende Möglichkeiten zur Verfügung:

- Setzen des 16-Bit-I/O-Register - Die Pins können als Hex-Wert oder bitweise gesetzt werden, mit 'Write' werden die Einstellungen in das Register übertragen
- Abfrage des 16-Bit-I/O-Register - Abfrage der Pins ('Read')
- Periodisches Schreiben/Lesen - Mit dem Schieberegler können verschiedene Zeiten von 50µs bis 2s eingestellt werden, 'Start' aktiviert das periodische Schreiben/Lesen ('Periodically')
- Signalton - Es ertönt ein Signalton, der das Schreiben/Lesen signalisiert ('Beep')
- bitweises Negieren der Werte - Wechseln der logischen Zustände der einzelnen Bits ('Toggle')

Anhang

Blockschaltbild



Anschlussbelegung Pin Header Block (User Interface)

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

Anschlussbelegung Pfostenseite 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

Anschlussbelegung Pfostenseite 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

Anschlussbelegung Pfostenseite JP6

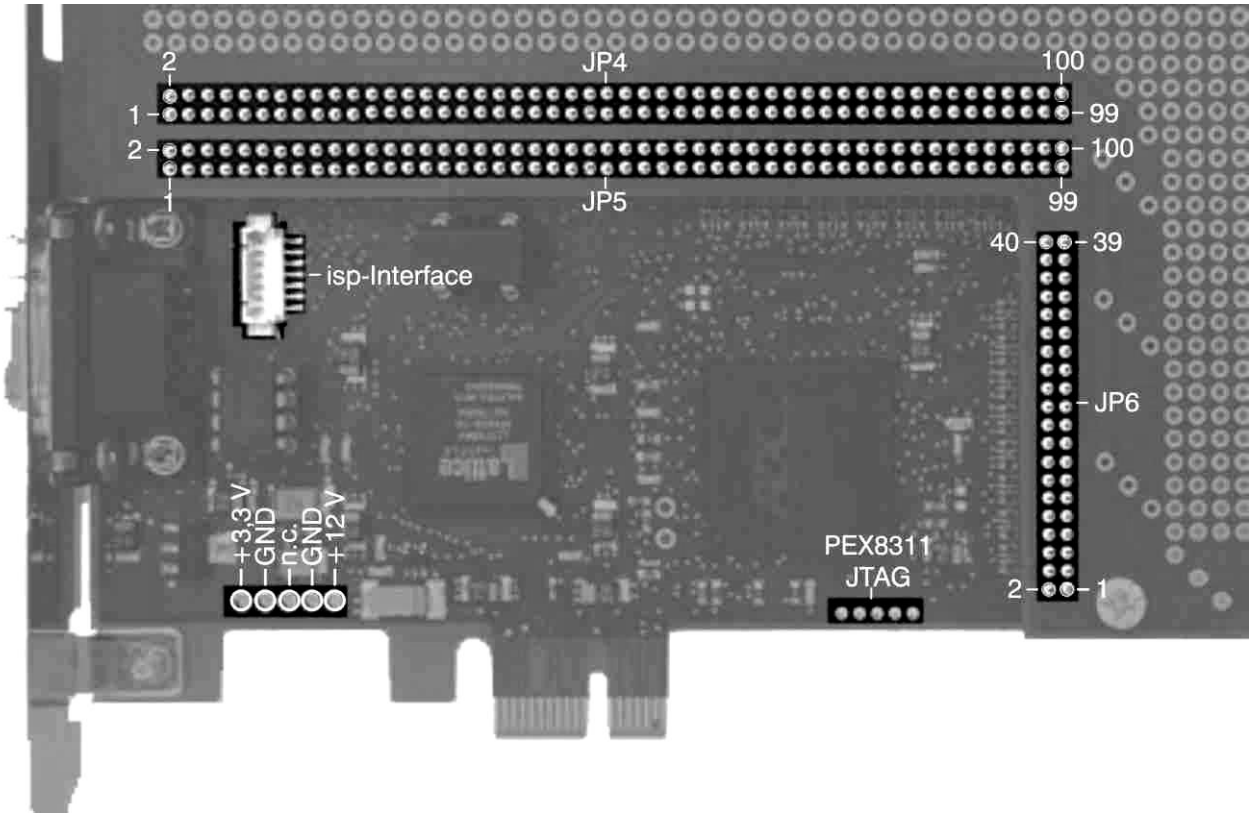
Anschlussbelegung ISP-Interface (Buchse X1)

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

Anschlussbelegung JTAG Support PEX8311 (Stecker JP2)

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

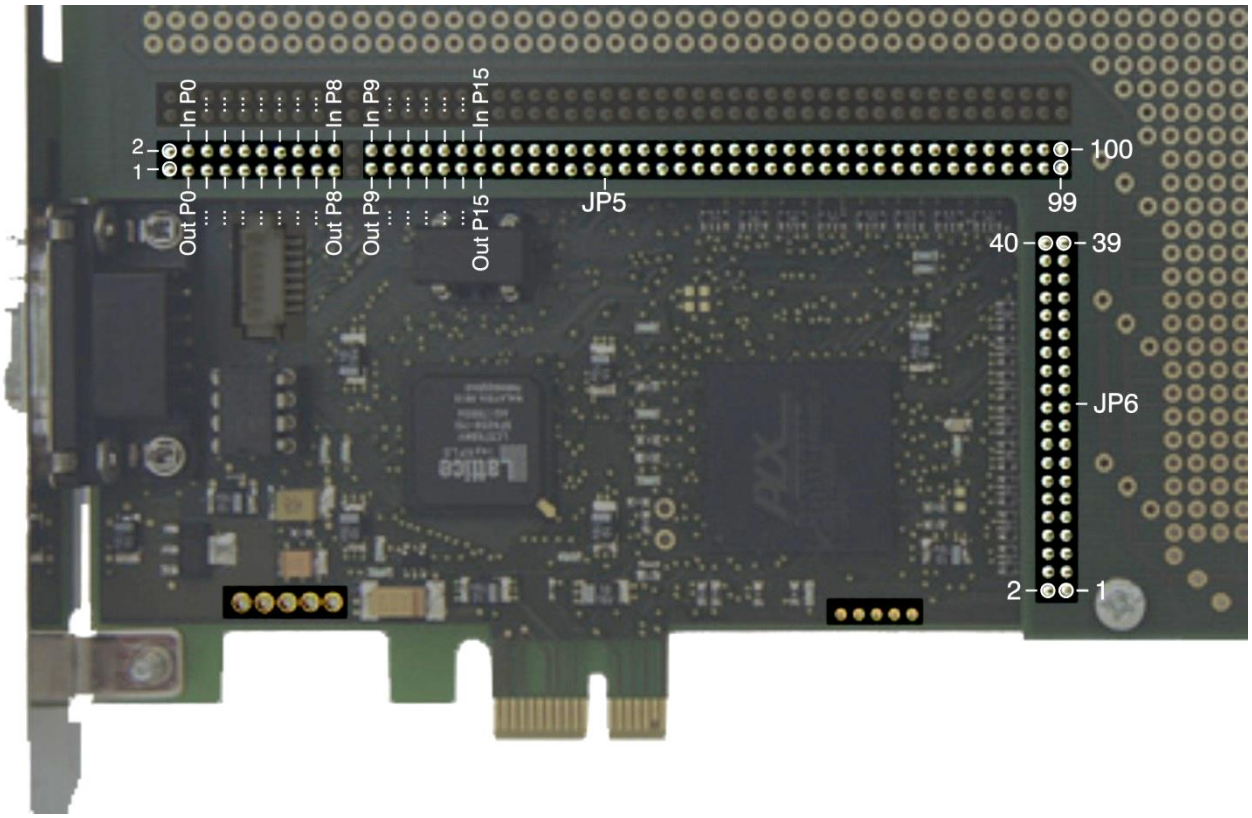
Lage der Pfostenleisten / Steckverbinder



Beispielanwendung ‚16Bit I/O-Register', Lage der Anschlusspins an 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

Anschlussbelegung Pfostenseite JP5



Unsere Hotline:

HK Meßsysteme GmbH
Straße am Heizhaus 1
D-12557 Berlin /Germany

Telefon: ++49/30/633 75 114
Fax: ++49/30/633 75 116
E-Mail: support@pci-tools.de
Web: http://www.pci-tools.com
http://www.pci-tools.de

DriverFactory
Ostendstr. 25
D-12459 Berlin /Germany

Telefon: ++49/30/5304 2020
Fax: ++49/30/5304 2021
E-Mail: info@driverfactory.de
Web: http://www.driverfactory.de

Hersteller PEX8311:

Broadcom Limited Company
1320 Ridder Park Drive
San Jose, CA 95131
U.S.A.

Phone: ++1-877-673-9442
Web: http://www.broadcom.com

Hersteller xpld5768:

Lattice Semiconductor Corporation
5555 N.E. Moore Court
Hillsboro, Oregon 97124-6421
U.S.A.

Phone: ++1-503-268-8000
Fax: ++1-503-268-8347
Web: http://www.latticesemi.com

Webadressen

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