Win10 Driver Manual

PCIe Altera Cyclone IV

Manual Revision 01p1
Revision Date 08/02/2023

Dynamic Engineering
150 DuBois St. Suite B&C
Santa Cruz, CA 95060
(831) 457-8891
www.dyneng.com
sales@dyneng.com
Est. 1988

PCIe Altera Cyclone IV

Cautions and Warnings

The electronic equipment described herein generates, uses, and can radiate radio frequency energy. Operation of this equipment in a residential area is likely to cause radio interference, in which case the user, at their own expense, will be required to take whatever measures may be required to correct the interference.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without express written approval from the president of Dynamic Engineering.

Connection of incompatible hardware is likely to cause serious damage.

Table of Contents

Figures

No table of figures entries found.

Tables

Design Revision History

Table 1: Design Revision History

Revision	Date	Description

Manual Revision History

Table 2: Manual Revision History

Revision	Date	Description

Product Description

The PcieAltBase and PcieAltChan drivers are Windows device drivers for the PCIe Altera Cyclone IV from Dynamic Engineering. These drivers were developed with as Windows Kernel Drivers using the KMDF.

The PCIe Altera Cyclone IV board has a Xilinx Spartan-6-LX100 FPGA to implement a PCI interface, FIFOs and protocol control/status for eight channels. Sixteen byte-wide interfaces send data to and from a reprogrammable Altera Cyclone IV FPGA to implement up to eight full-duplex I/O channels.

There are 40 RS-485/LVDS and twelve bidirectional TTL I/O lines as well as eight programmable PLLs that can create up to 24 clocks for the I/O channels. The PCI bus interfaces with an onboard PCI-to-PCIe bridge that provides a four-lane PCIe interface to the host system.

The Xilinx FPGA has sixteen DMA engines and data FIFOs to provide high-speed input and output data transfers for the eight I/O channels. The Altera can be programmed either from the on-board flash or through the Xilinx programming interface from a configuration file read from host memory. The Altera can be reprogrammed at any time without powering down the system.

Each channel has 8k x 32-bit received data FIFO and an 8k x 32-bit transmit data FIFO implemented with FPGA internal block RAM. These FIFOs can be accessed using either single-word reads or writes or DMA.

When the PCIe Altera Cyclone IV board is recognized by the PCI bus configuration utility it will load the PcieAltBase driver which will create a device object for each board, initialize the hardware, create child devices for the eight I/O channels and request loading of the PcieAltChan driver. The PcieAltChan driver will create a device object for each of the I/O channels and perform initialization on each channel. IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move blocks of data in and out of the I/O channel devices.

Software Description

The drivers consist of two modules, a base driver and a channel driver module, that can be loaded using the .inf files (see below regarding installation instructions).

This package comes with a PcieAltCyc4UserApp which is used to test the hardware as well as provide an example of how to interface with the device through software:

Table 3: Header Files

File Name	Description
ioctl.h	Defines serves as the primary API for the device and demonstrates how to use the IOCTL calls defined in the files below.
PcieAltBasePublic.h	Defines many of the data structures used by the Ioctl calls for the base device.
PcieAltBasePublic.h	Defines many of the data structures used by the Ioctl calls for the channel device.
AlteraCycIIVReg.h	Defines the register map and important bits.

Test Suite

The UserApp is used in-house to validate the hardware and provides a more extensive example of how to interact with the hardware. The UserApp was written in C.

Driver Installation

Driver Installation is simple, first right-click on the PcieAltBase.inf file and click “install”, this will load the base driver automatically and enumerate the channels in the Window’s Device Manager. Once this is installed, follow the same step by right clicking on the PcieAltChan.inf file and click “Install”.

Once the driver is installed on a system, the driver files are automatically copied to the “Window’s Store” (i.e., a special directory where windows stores driver files). The driver will automatically load every time the computer is booted.

Uninstallation

Open the device manager and navigate to the “PcieAltChan” elements, expand the tree down by pressing the “>” arrow. There you will see the base and channel device. Right-click on the channel device and select “uninstall device” (IMPORTANT – once this uninstall is selected a window will pop up, check the box that says “Delete the driver software installed for this device” (this removes the driver from the windows store). For the second channel this pop-up will not offer the same box as the software is already removed from the store. Finally to the same thing with the base driver – again remembering to check the box as the base driver is a separate piece of software that has been copied to the Window’s Store.

API

A simplified API is provided in the ioctl.h file, however these are merely wrappers around the IOCTL calls listed below. As such, you can integrate software with the device by either incorporating the ioctl.c/h files or directly calling the following IOCTL calls.

IOCTL_PCIE_ALT_BASE_GET_INFO

Function: Returns the device driver revision, FPGA design ID and revision, user switch value, and device instance number.
Input: None
Output: PCIE_ALT_BASE_DRIVER_DEVICE_INFO structure
Notes: The switch value is the configuration of the 8-bit onboard dipswitch that has been selected by the user (see the board silk screen for bit position and polarity). Instance number is the zero-based device number. See the definition of PCIE_ALT_BASE_DRIVER_DEVICE_INFO below.

```
 // Driver/Device information
typedef struct _PCIE_ALT_BASE_DRIVER_DEVICE_INFO {
   UCHAR    DriverRev;     // Base driver revision
   UCHAR    DesignId;      // Xilinx design ID
   UCHAR    DesignRev;     // Xilinx design revision
   UCHAR    MinorRev;      // Xilinx minor revision
   UCHAR    SwitchValue;   // User switch setting
   ULONG    InstanceNum;   // Board instance number
   BOOLEAN  BridgeCnfgd;   // True if bridge conigured successfully
} PCIE_ALT_BASE_DRIVER_DEVICE_INFO, *PPCIE_ALT_BASE_DRIVER_DEVICE_INFO;
```

IOCTL_PCIE_ALT_BASE_LOAD_ALT_FIFO

Function: Writes a single 32-bit word to the Altera programming FIFO.
Input: unsigned long integer
Output: None
Notes: Data from an Altera configuration file (*.rbf) is written to this FIFO. The data will be serialized and clocked into the Altera configuration data input.

IOCTL_PCIE_ALT_BASE_GET_ALT_FIFO_STATUS

Function: Returns the data-count and flags from the Altera programming FIFO.
Input: None
Output: PCIE_ALT_BASE_PRGFF_STAT structure
Notes: See the definition of PCIE_ALT_BASE_PRGFF_STAT below.

```
 // Altera programming FIFO count/status
 typedef struct _PCIE_ALT_BASE_PRGFF_STAT {
   USHORT   FifoCount;
   BOOLEAN  FifoEmpty;
   BOOLEAN  FifoFull;
} PCIE_ALT_BASE_PRGFF_STAT, *PPCIE_ALT_BASE_PRGFF_STAT;
```

IOCTL_PCIE_ALT_BASE_SET_CONFIG

Function: Sets the configuration of the base control register
Input: unsigned long integer
Output: None
Notes: The bits in this register control the loading, reset and interrupt enable for the Altera FPGA. See the bit definitions below.

```
// Base Control
typedef struct _PCIE_ALT_BASE_CONTROL {
   BOOLEAN  AltIntEn;      // Altera interrupt enable
   BOOLEAN  AltFileLdEn;   // Enable loading Altera from a file
   BOOLEAN  AltFlashLdEn;  // Enable loading Altera from flash
   BOOLEAN  AltClearLdDn;  // Clear done signal latch
   BOOLEAN  ResetLoader;   // Reset Altera loader state-machine
   BOOLEAN  ResetAltera;   // Reset Altera device
   BOOLEAN  TestPointEn;   // Enable test points onto Switch pins
   UCHAR    TPChanSel;     // Channel select for test points
} PCIE_ALT_BASE_CONTROL, *PPCIE_ALT_BASE_CONTROL;
```

IOCTL_PCIE_ALT_BASE_GET_CONFIG

Function: Reads and returns the configuration of the base control register.
Input: None
Output: unsigned long integer
Notes: Returns the bits set in the previous call. See the bit definitions above.

IOCTL_PCIE_ALT_BASE_GET_STATUS

Function: Reads and returns the value of the base status register.
Input: None

Output: unsigned long integer
Notes: This register reports the interrupt status for the eight Xilinx I/O channels and the Altera. See the bit definitions below.

```
// Status bits
#define STATUS_INT_CHAN_0     0x00000001  // Channel 0 interrupt is active
#define STATUS_INT_CHAN_1     0x00000002  // Channel 1 interrupt is active
#define STATUS_INT_CHAN_2     0x00000004  // Channel 2 interrupt is active
#define STATUS_INT_CHAN_3     0x00000008  // Channel 3 interrupt is active
#define STATUS_INT_CHAN_4     0x00000010  // Channel 4 interrupt is active
#define STATUS_INT_CHAN_5     0x00000020  // Channel 5 interrupt is active
#define STATUS_INT_CHAN_6     0x00000040  // Channel 6 interrupt is active
#define STATUS_INT_CHAN_7     0x00000080  // Channel 7 interrupt is active
#define ALTERAINT             0x00000100  // Altera interrupt is active
#define ALTERAINTMASKED       0x00000200  // Altera masked interrupt is active
#define XA_REFCLK_LOCKED      0x00001000  // Clock for Xil->Alt xfers is locked
#define AX_REFCLK_LOCKED      0x00002000  // Clock for Alt->Xil xfers is locked
#define ALTERA_NSTATUS        0x00020000  // Readback of Altera Programming Status line
#define ALTERA_PROG_IDLE      0x00100000  // Altera programmer idle
#define ALTERA_STRT_CNFG_ERR 0x00200000  // Altera start configuration error
#define ALTERA_END_CNFG_ERR   0x00400000  // Altera end configuration error
#define STATUS_MASK           0x007233FF
```

IOCTL_PCIE_ALT_BASE_SET_ALT_REG

Function: Writes a 32-bit word to the Altera memory space.
Input: ALTERA_MEM_ACCESS structure
Output: None
Notes: Memory offset is relative to the Altera base address (0x8000 relative to the board base address). See the definition of ALTERA_MEM_ACCESS below.

```
 // Longword Access to Altera memory space
typedef struct _ALTERA_MEM_ACCESS {
   ULONG    MemOffset;
   ULONG    Data;
} ALTERA_MEM_ACCESS, *PALTERA_MEM_ACCESS;
```

IOCTL_PCIE_ALT_BASE_GET_ALT_REG

Function: Reads and returns a 32-bit word from the Altera memory space.
Input: Memory offset (unsigned long integer)
Output: Data (unsigned long integer)
Notes: As in the previous call, memory offset is relative to the Altera base address (0x8000 relative to the board base address).

The IOCTLs defined for the PcieAltChan driver are described below:

IOCTL_PCIE_ALT_CHAN_GET_INFO

Function: Returns the driver revision and instance number of the channel device.
Input: None
Output: PCIE_ALT_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of PCIE_ALT_CHAN_DRIVER_DEVICE_INFO below.

```
 // Driver/Device information
typedef struct _PCIE_ALT_CHAN_DRIVER_DEVICE_INFO {
   UCHAR    DriverRev;     // Channel driver revision
   UCHAR    Channel;       // Channel number
   UCHAR    DesignId;      // These fields
   UCHAR    DesignRev;     // passed
   UCHAR    MinorRev;      // from
   UCHAR    SwitchValue;   // base
   ULONG    InstanceNum;   // device
} PCIE_ALT_CHAN_DRIVER_DEVICE_INFO, *PPCIE_ALT_CHAN_DRIVER_DEVICE_INFO;
```

IOCTL_PCIE_ALT_CHAN_SET_CONFIG

Function: Sets the channel's control configuration.
Input: PCIE_ALT_CHAN_CONFIG structure
Output: None
Notes: Specifies the enabled interrupt sources, DMA preemption behavior and other control parameters. See the definitions of PCIE_ALT_INTS, PCIE_ALT_DMA_PRMPT and PCIE_ALT_CHAN_CONFIG below.

```
typedef struct _PCIE_ALT_INTS {
   BOOLEAN  TxAmtInt;      // Transmit FIFO almost empty interrupt
   BOOLEAN  RxAflInt;      // Receive FIFO almost full interrupt
   BOOLEAN  RxOvflInt;     // Receive FIFO overflow interrupt
   BOOLEAN  TxAmtLvlInt;   // Transmit FIFO almost empty level interrupt
   BOOLEAN  RxAflLvlInt;   // Receive FIFO almost full level interrupt
} PCIE_ALT_INTS, *PPCIE_ALT_INTS;

 // Channel DMA priority (use sparingly)
typedef enum _PCIE_ALT_DMA_PRMPT {
   PCIE_ALT_NONE,    // No priority
   PCIE_ALT_READ,    // Read DMA has priority
   PCIE_ALT_WRITE,   // Write DMA has priority
   PCIE_ALT_RDWR     // Read and Write DMA have priority
} PCIE_ALT_DMA_PRMPT, *PPCIE_ALT_DMA_PRMPT;

typedef struct _PCIE_ALT_CHAN_CONFIG {
   BOOLEAN              FifoBypassEn;  // Enables auto tx->rx FIFO transfer
   BOOLEAN              TxEnable;      // Set to enable Tx operation
   BOOLEAN              RxEnable;      // Set to enable Rx operation
   PCIE_ALT_INTS        IntConfig;     // Interrupt condition enables
   PCIE_ALT_DMA_PRMPT   DmaPriority;   // DMA preemption control
} PCIE_ALT_CHAN_CONFIG, *PPCIE_ALT_CHAN_CONFIG;
```

IOCTL_PCIE_ALT_CHAN_GET_CONFIG

Function: Reads and returns the channel configuration set in the previous call.
Input: None
Output: PCIE_ALT_CHAN_CONFIG structure
Notes: See the definitions of PCIE_ALT_INTS, PCIE_ALT_DMA_PRMPT and PCIE_ALT_CHAN_CONFIG above.

IOCTL_PCIE_ALT_CHAN_GET_STATUS

Function: Reads and returns the channel's status register bit values.
Input: None
Output: Value of the channel's status register (unsigned long integer)
Notes: See the status bit definitions below.

```
// Chan Status bits
#define CHAN_STAT_TX_FF_MT          0x00000001
#define CHAN_STAT_TX_FF_AMT         0x00000002
#define CHAN_STAT_TX_FF_FL          0x00000004
#define CHAN_STAT_RX_FF_MT          0x00000010
#define CHAN_STAT_RX_FF_AFL         0x00000020
#define CHAN_STAT_RX_FF_FL          0x00000040
#define CHAN_STAT_TX_AMT_INT        0x00000100
#define CHAN_STAT_RX_AFL_INT        0x00000200
#define CHAN_STAT_TX_AMT_INT_LAT    0x00000400
#define CHAN_STAT_RX_AFL_INT_LAT    0x00000800
#define CHAN_STAT_WR_DMA_ERR        0x00001000
#define CHAN_STAT_RD_DMA_ERR        0x00002000
#define CHAN_STAT_WR_DMA_INT        0x00004000
#define CHAN_STAT_RD_DMA_INT        0x00008000
#define CHAN_STAT_RX_OVFL_INT       0x00080000
#define CHAN_STAT_RD_DMA_IDLE       0x00400000
#define CHAN_STAT_WR_DMA_IDLE       0x00800000
#define CHAN_STAT_LOC_INT           0x40000000
#define CHAN_STAT_INTSTAT           0x80000000
```

IOCTL_PCIE_ALT_CHAN_CLEAR_STATUS

Function: Clears the specified latched status bits.
Input: Latched channel status bits to clear (unsigned long integer)
Output: None
Notes: Only CHAN_STAT_TX_AMT_INT_LAT, CHAN_STAT_RX_AFL_INT_LAT, CHAN_STAT_WR_DMA_ERR, CHAN_STAT_RD_DMA_ERR and CHAN_STAT_RX_OVFL_INT can be cleared with this call. No other status bits are latched.

IOCTL_PCIE_ALT_CHAN_SET_FIFO_LEVELS

Function: Sets the threshold levels for the transmitter almost empty and receiver almost full pulse and level interrupts for the channel.
Input: PCIE_ALT_CHAN_FIFO_LEVELS structure
Output: None
Notes: The pulse almost empty and full interrupts are latched while the level interrupts are not. Also if DMA priority is enabled, the pulse FIFO level values are used to determine when to give priority to an output or input DMA channel that is running out of data or room to store data. See the definition of PCIE_ALT_CHAN_FIFO_LEVELS below.

```
typedef struct _PCIE_ALT_CHAN_FIFO_LEVELS {
   USHORT   PulseAlmostEmpty;
   USHORT   PulseAlmostFull;
   USHORT   LevelAlmostEmpty;
   USHORT   LevelAlmostFull;
} PCIE_ALT_CHAN_FIFO_LEVELS, *PPCIE_ALT_CHAN_FIFO_LEVELS;
```

IOCTL_PCIE_ALT_CHAN_GET_FIFO_LEVELS

Function: Returns the pulse and level transmitter almost empty and receiver almost full levels for the channel.
Input: None
Output: PCIE_ALT_CHAN_FIFO_LEVELS structure
Notes: Returns the values set in the previous call. See the definition of PCIE_ALT_CHAN_FIFO_LEVELS above.

IOCTL_PCIE_ALT_CHAN_GET_FIFO_COUNTS

Function: Returns the number of data words in the transmit and receive data FIFOs.
Input: None
Output: PCIE_ALT_CHAN_FIFO_COUNTS structure
Notes: There is an 8k-1 data FIFO for the transmit data-path and four pipe-line latches and a two 4k-1 data FIFOs for the receive data-path. These are counted in the FIFO counts. That means the transmit count can be a maximum of 8191 32-bit words and the receive count can be a maximum of 8194 32-bit words. See the definition of PCIE_ALT_CHAN_FIFO_COUNTS below.

```
typedef struct _PCIE_ALT_CHAN_FIFO_COUNTS {
   USHORT   TxCount;
   USHORT   RxCount;
} PCIE_ALT_CHAN_FIFO_COUNTS, *PPCIE_ALT_CHAN_FIFO_COUNTS;
```

IOCTL_PCIE_ALT_CHAN_RESET_FIFOS

Function: Resets the transmit or receive or both FIFOs for the channel.
Input: PCIE_ALT_FIFO_SEL enumeration type
Output: None
Notes: Resets the transmit or receive FIFO or both depending on the input parameter selection. See the definition of PCIE_ALT_CHAN_FIFO_SEL below.

```
typedef enum _PCIE_ALT_FIFO_SEL {
   PCIE_ALT_TX,
   PCIE_ALT_RX,
   PCIE_ALT_BOTH
} PCIE_ALT_FIFO_SEL, *PPCIE_ALT_FIFO_SEL;
```

IOCTL_PCIE_ALT_CHAN_WRITE_FIFO

Function: Writes a 32-bit data-word to the transmit FIFO.
Input: FIFO word (unsigned long integer)
Output: None
Notes: Used to make single-word accesses to the transmit FIFO instead of using DMA.

IOCTL_PCIE_ALT_CHAN_READ_FIFO

Function: Reads and returns a 32-bit data word from the receive FIFO.
Input: None
Output: FIFO word (unsigned long integer)
Notes: Used to make single-word accesses to the receive FIFO instead of using DMA.

IOCTL_PCIE_ALT_CHAN_REGISTER_EVENT

Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts do not cause this event to be signaled.

IOCTL_PCIE_ALT_CHAN_ENABLE_INTERRUPT

Function: Enables the channel master interrupt.
Input: None
Output: None
Notes: This command is run to allow the board to respond to user interrupts. The master interrupt enable is disabled in the driver interrupt service routine when a user interrupt is serviced. This command must be run after each user interrupt occurs.

IOCTL_PCIE_ALT_CHAN_DISABLE_INTERRUPT

Function: Disables the channel master interrupt.
Input: None
Output: None
Notes: This call is used when user interrupt processing is no longer desired.

IOCTL_PCIE_ALT_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the channel master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_PCIE_ALT_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (unsigned long integer)
Notes: Returns the status that was read while servicing the last interrupt caused by one of the user-enabled channel interrupt conditions. The interrupts that deal with the DMA transfers do not affect this value.

Write

DMA data is written to the referenced I/O channel device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually written, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Read

DMA data is read from the referenced I/O channel device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually read, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Warranty and Repair

Please refer to the warranty page on our website for the warranty and options that are currently offered.

www.dyneng.com/warranty

Service Policy

Before returning a product for repair, verify to the best of your ability, that the suspected unit is as fault. Then call the Dynamic Engineering Customer Service Department for a Return Material Authorization (RMA) number. Carefully package the product, in the original packaging if possible, and ship prepaid and insured with the RMA number clearly written on the outside of the package. Include a return address and the telephone number of a technical contact. For out-of-warranty repairs, a purchase order for repair charges must accompany the return. Dynamic Engineering will not be responsible for damages due to improper packaging of returned items. For service on Dynamic Engineering products not purchased directly from Dynamic Engineering, contact your reseller. Products returned to Dynamic Engineering for repair by anyone other than the original customer will be treated as out-of-warranty.

Out-of-Warranty Repairs

Out-of-warranty repairs will be billed on a material and labor basis. Customer approval will be obtained before repairing any item if the repair charges will exceed one half of the list price for one of that kind of unit. Return transportation and insurance will be billed as part of the repair in addition to the minimum RMA charge.

Contact:

Customer Service Department
Dynamic Engineering
150 DuBois St. Suite B&C
Santa Cruz, CA 95005
(831) 457-8891
support@dyneng.com

Glossary

Baud	Used as the bit period when talking about UARTs; Not strictly correct, but is the common usage when talking about UARTs.
CardID	Unique number assigned to a design to distinguish between all designs of a particular vendor
CFM	Cubic feet per minute
FIFO	First In First Out memory
Flash	Non-volatile memory used on Dynamic Engineering boards to store FPGA configurations or BIOS
JTAG	Joint Test Action Group – a standard used to control serial data transfer for test and programming operations.
LFM	Linear feet per minute
LVDS	Low Voltage Differential Signaling
MUX	Multiplexor – multiple signals multiplexed to one with a selection mechanism to control which path is active.
Packed	When UART characters are always sent/received in groups of four, allowing full use of host bus/FIFO bandwidth.
Packet	Group of characters transferred. When the characteristics of the group of characters is known, the data can be stored in packets and transferred as such; the system is optimized as a result. Any number of characters can be transferred.
PCI	Peripheral Component Interconnect – parallel bus from host to this device
PIM	PMC Interface Module (PIM). Provides rear I/O in cPCI systems. Mounts to PIM Carrier
PIM Carrier	PIM Mounting Device. Mounts on rear of cPCI backplane.
PMC	PCI Mezzanine Card – establishes common connectors, connections, size and other mechanical features.
TAP	Test Access Port – basically a multi-state port that can be controlled with JTAG [TMS, TDI, TDO, TCK]. The TAP States are the states in the State Machine that are controlled by the commands received over the JTAG link.
TCK	Test Clock provides synchronization for the TDI, TDO, and TMS signals

TDI	Test Data in – this serial line provides the data input to the device controlled by the TMS commands. For example, the data to program the FLASH comes on the TDI line while the commands to the state machine to move through the necessary states comes over TMS. Rising edge of TCK valid.
TDO	Test Data Out is the shifted data out. Valid on the falling edge of the TCK. Not all states output data.
TMS	Test Mode State – this serial line provides the state switching controls. '1' indicates to move to the next state, '0' means stay put in cases where delays can happen; otherwise, 0,2 are used to choose which branch to take. Due to the complexity of state manipulation, the instructions are usually precompiled. Rising edge of TCK valid.
UART	Universal Asynchronous Receiver Transmitter. Common serialized data transfer with start bit, stop bit, optional parity, optional 7/8 bit data. Can be over any electrical interface. RS232 and RS422 are most common.
Unpacked	When UART characters are sent on an unknown basis requiring single character storage and transfer over the host bus
VendorID	Manufacturers number for PCI/PCIe boards. DCBA is Dynamic Engineering's VendorID
VME	Versa Module European
VPX	Family of standards based on the VITA 46.0
XMC	Switched mezzanine card (PMC with PCIe)