

# Pico Computing Inc.



## **PicoUtil and PicoCommand User's Manual**

**Version 5.0.0.3. Apr 16th, 2010.**

Pico Computing, Inc.  
150 Nickerson, Suite 311  
Seattle, WA, 98109-1634  
(206) 283-2178  
[www.picocomputing.com](http://www.picocomputing.com)

# 1 Product Overview

## Product Overview

The Pico Family of products are high performance computing and FPGA platforms in a very small form factor. Their nominal power consumption is less than one watt, yet they have as much computing power as a desktop computer. For some applications the FPGA can yield performance improvements in excess of 100 times a desktop PC. Hence, the tag line 'Tiny Mighty Machines'.

Pico products are based on Virtex-4 and Virtex-5 chips. These devices have the performance and power consumption of a custom chip (ASIC), but are completely reconfigurable. There are two versions of each Pico Card; they are the "Logic Optimized" (LO) and "Embedded Processor" (EP) versions. The Logic Optimized versions offer the most user-configurable gates, while the embedded processor version trades off some gates for an embedded PowerPC™ processor.

The Pico E-14 is a Cardbus (PCMCIA) credit card sized single board computer. The Pico E-12 is a compact flash card that is three times smaller than the E-14. Even with such a tiny size it packs an impressive feature set such as 64 MB of flash storage, 128 MB of RAM, Gigabit Ethernet and various peripherals. The E-14 has additional Flash and RAM as well as high speed analog converters capable of processing voice, video and radio communications.

Advanced users will enjoy the open source development kits which allow absolute control over the Pico Hardware. Those not interested in programming firmware can use Matlab™ and Simulink™ to implement custom algorithms in hardware with just the click of a button. Board support packages are also available for those who wish to run an Operating System such as Linux or  $\mu$ C/OS.



E12

## 1.1 Guide to Documentation

Other manuals in this help library can be located at [GuideToDocumentation.pdf](#)

NOTE: This link will access the pdf from the PicoComputing.com Website .

## 1.2 Purpose of PicoCommand

PicoCommand.exe is a command line utility which performs many of the functions of picoUtil but from command line arguments. It is a suitable tool for incorporating into scripts. For example, PicoCommand can be used to load a specific FPGA file onto the flash ROM of the Pico Card, load an ELF file onto the flash ROM, link the two files, and start up the new 'system'.

There are several commands available in PicoCommand that cannot be performed in PicoUtil. Notable amongst these are the */i #,#,#* command (which enables actions to be performed against multiple Pico Cards) and the */wo* command.

The Pico Cards are fabricated with a [PCMCIA](#) or a Cardbus connector. The pins on this connector are tied directly into the FPGA. The FPGA shipped with the Pico Card implements this PCMCIA functionality. This means that a Pico Card can be plugged into a PC for maintenance or testing. The FPGA image which implements this dedicated PCMCIA functionality is usually called PrimaryBoot.bit. The PC program designed to control the card over the PCMCIA interface is PicoUtil.exe. The Pico Card can be configured for a standalone application in which the FPGA is not used to serve the PCMCIA interface. PicoUtil.exe may still be useful for loading the user image and files to the Pico Card.

PrimaryBoot.bit is a binary file which defines the setup of the FPGA fabric and the initial block ram code executed by the PPC (PowerPCb). Such files are traditionally called bit image files from the software processes provided by Xilinx to create these files. At power on (see [Power On Sequence](#)) the Primary Boot Image is automatically loaded into the FPGA. If this image doesn't load successfully, the second image - usually called BackupBoot.bit - is loaded. Depending on the option selected additional files may also be loaded.

The Flash ROM may also contain other files. These files are accessible to the PPC or to the PC (through the PCMCIA bus). PicoUtil manages the Flash ROM file system and several other facets of the Pico Card. It can:

- Load the FPGA from any FPGA image.
- Boot software programs on the PowerPCb
- Read and check the CIS read by the OS or embedded in the FPGA image.

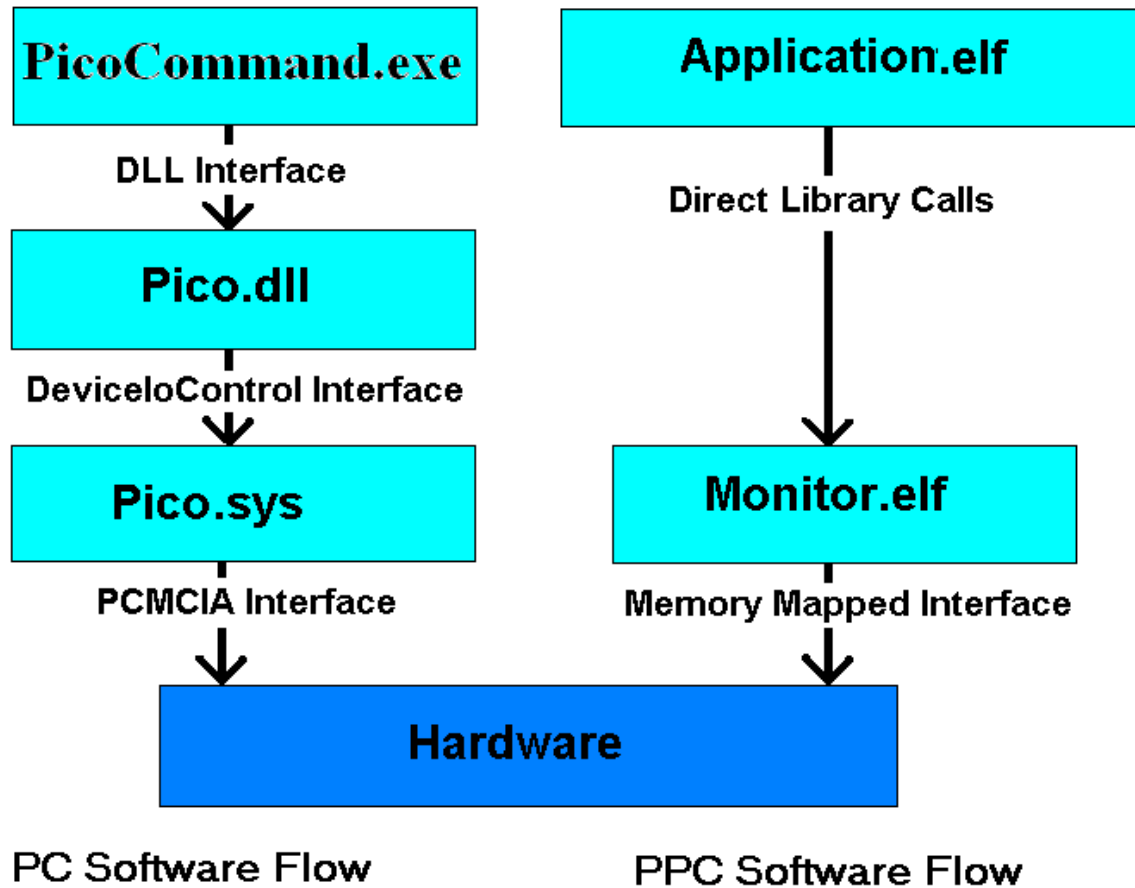
## 1.3 Software Architecture

**PicoCommand depends** three other software modules:

- **Pico.dll**. This DLL has a series of entry points that closely reflect the menu choices in PicoUtil. The DLL is loaded by PicoUtil from the folder defined by the environment variable **picobase** + "\bin". Under Linux this library (DLL) is called Pico.so
- **Pico.sys**. This is a driver (under Windows) which provides the low level interface to the Pico Card. Under Linux this is the "pico" kernel module.
- **Monitor.elf**. This program resides on the Pico Card and is loaded when

PrimaryBoot.bit is loaded into the FPGA.

The software architecture is illustrated below:



Detailed description of these individual components can be found in the [Pico Software Manual](#).

## 2 PicoCommand

PicoCommand.exe is a command line utility which performs many of the functions of picoUtil but from command line arguments. It is a suitable tool for incorporating into scripts. For example, PicoCommand can be used to load a specific FPGA file onto the flash ROM of the Pico Card, load an ELF file onto the flash ROM, link the two files, and start up the new 'system'.

There are several commands available in PicoCommand that cannot be performed in PicoUtil. Notable amongst these are the `/i #,#,#` command and the `/wo` command.

The command 'picoCommand /h' will return the following text.

```

----- PicoCommand 3.5.1.1 -----
Command line utility to manage Pico Cards.
Usage is: PicoCommand /letter fileName [other parameters]
Valid letter codes are:
/a propertyName      display specified property of Pico card
/b flashFile         reboot from specified file
/c flashFile PCfile  changed source file name to PCfile
/d flashFile         delete specified file from flash ROM
/e number            display meaning of specified error code
/f                  defragment flash ROM system
/g                  display flash directory
/h or /?            display this help screen
/h letter or /? letter display help on specific command.
/i #,#,#           specify Pico Card(s), eg /i 1,2
/j power            power cycle card (forced reboot of PrimaryBoot.bit)
/k flashFile1 flashFile2 link flashFile1 to flashFile2
/l PCfilename        load entire flash from PC dump file
/n flashFile "note"  attach note to flashFile
/p PCfile.rom        write primary boot from specified PC file
/q                  inhibit writing 'progress bar' to screen
/rr address         read memory mapped register
/rw address,data,... write memory mapped register
/rt                 read core temperature of FGPA
/s PCfile.rom        save entire flash into PC dump file
/rr /rw address data read/write register.
/t #                run SelfTest # = bit mask of test to run
/u flashFile         update specified file
/u flashFile /b     update specified file and reboot
/w PCfile            write specified file.
/w# PCfile           write multiple copies. eg /w3 will write three
copies
/wo PCfile           write without part number check (see error 7353)
/x flashFile         load ELF file from flash ROM and run
/z simulateFile      run against simulation file instead of Pico Hardware

/i, /q, and /z commands can be followed by other commands.

```

PicoCommand.exe will return zero if successful, otherwise it returns a negative error code. The commands /i, /q, and /z may be used in conjunction with other commands, for example,

```
Picocommand /q /w PCfile
```

Each command is documented in the following sections.

## 2.1 Show property command (/a)

**picocommand /a [propertyname | null]**

This command will display one or all properties as specified by the propertyName.

Valid PropertyNames are:

- model                    The Pico model, ie E16FX70T, E16SX50T, E16LX50, or E101LX45, etc
- partName                Xilinx part name for the FPGA, eg 5vfx70tff665
- serialNumber            serial number from ee prom, eg 0050C2-442-0019
- debugFlags              hex value
- currentImage            name of current image, eg PrimaryBoot.bit. This information

- `currentImageAddr` will only be reliable when the Pico card has a flashROM. address on flash ROM of the boot image. This information will only be reliable when the Pico card has a flashROM.
- `FlashSize` size of the flashROM in bytes
- `memsize` size of the memory on the Pico Card (excluding the block RAM on the FPGA).
- `LogicalSectorCount` number of logical sectors on flashROM
- `PhysicalSectorCount` number of physical sectors on flashROM
- `PhysicalSectorSize` size of physical sector on flashROM (ie the smallest erasable unit)
- `temperature` core temperature of the FPGA. This is only available on the E16 and E17.
- `internalVoltage` internal voltage of the FPGA. This is only available on the E16 and E17.
- `auxVoltage` auxiliary voltage rail of the FPGA. This is only available on the E16 and E17.

## 2.2 Boot FPGA (/b)

**picocommand /b flashFile.**

This command will reboot the Pico Card from the specified file.

For example:

```
picocommand /b PrimaryBoot.bit.
```

## 2.3 Change Source Flash File Name (/c)

**picocommand /c flashFile PCfile**

This command will change the source file name associated with the flash file name to the specified PC file name.

## 2.4 Delete Flash File (/d)

**picocommand /d flashFile**

This command will delete the specified file(s) from flash ROM. The command will accept wild cards (? and \*) to delete multiple files from the flash ROM.

- ? will match any single character,
- \* will match any number of characters.

For example:

**picocommand /d Mybootfile.bit** remove a single file

**picocommand /d 1meg\*.dat.** remove zero or more files with names like 1meg1.dat, 1meg21.dat, etc

## 2.5 Show Error Information (/e)

**picocommand /e # or picocommand /? #**

This command will display the meaning of the specified error number.

## 2.6 Defragment Flash (/f)

**picocommand /f**

This command will defragment flash ROM system.

## 2.7 Display Flash Directory (/g)

**picocommand /g [/b]**

This command will display the directory of the flashROM including file sizes and other fields. The option /b will limit the display to the filename. Wild cards are permitted using the \* character. For example:

**picocommand /g**

might return

```
01/13/10 00:47:00 3378233 PrimaryBoot.bit
01/14/10 01:47:00 3378233 BackupBoot.bit
01/14/10 02:47:00 3378233 MyBoot.bit
```

and

**picocommand /g/b**

would reasonably be expected to return

```
PrimaryBoot.bit
BackupBoot.bit
MyBoot.bit
```

## 2.8 Show Help Information (/h)

**picocommand /h or picocommand /?**

These command will display generic or command specific information.

For example:

<b>picocommand /h</b>	display generic help information.
<b>picocommand /ha</b>	display help information about the a command.

## 2.9 Specify Pico Card (/i)

**picocommand /i #,#,#**

This command will specify the Pico Card to which the following command applies. In an installation in which there are multiple Pico Cards, the cards have device names \\.\Pico1, \\.\Pico2,... The /i commands specifies \\.\Pico<i>.

When multiple cards are specified, the command will be repeated on each of the specified boards.

For example:

`PicoCommand /i 1,2,4 /p PrimaryBoot.bit` would update the primary boot file on \\.\Pico1, \\.\Pico2, \\.\Pico4.

## 2.10 Power Cycle Pico Card (/j)

**picocommand /j power**

This command will power cycle the Pico card. This is a robust way to reboot PrimaryBoot.bit.

## 2.11 Link One Flash File To Another (/k)

**picocommand /k flashFile1 flashFile2**

This command will link two files together. There are three different meaningful linkages between files:

1. flashFile1.elf and flashFile2.bit. This causes flashFile2.bit to be loaded before flashfile2.elf is executed.
2. flashFile1.bit and flashFile2.elf. This causes flashFile2.eld to be loaded when flashfile1.bit is booted (/b command).
3. PrimaryBoot.bit and flashFile2.bit. This causes fileflashFile2.bit to be loaded when the Pico Card is loaded in standalone mode (ie not in a laptop). This is the only situation in which a bit file can be linked to another bit file and will NOT cause flashFile2.bit to be loaded when PrimaryBoot.bit is loaded using the /b command.

## 2.12 Load Entire Flash ROM (/l)

**picocommand /l PCfilename**

This command will rewrite the entire flash ROM from a PC dump file. The dump file typically has a suffix of .dmp and should be create with the /s command or from PicoUtil.exe

### 2.13 Add Note to Flash File (/n)

**picocommand /n flashFile "note"**

This command will (re)write the note field of to flashFile.

### 2.14 Write Primary Boot File (/p)

**picocommand /p PCfile.rom**

This command will write the primary boot from the specified PC file. The file will ONLY be written if the part number of the Pico Card matches the part number specified in the PC file.

### 2.15 Set Quiet Mode (/q)

**picocommand /q**

This command inhibits the '.' characters echoed to the screen when a long operation is in progress.

### 2.16 Read/Write Pico Card (/r)

**picocommand /rr address**

This command reads data from the specified register.

**picocommand /rw address data**

This command write 32bits of data to the register with the specified address. A common method to interface with custom logic on the FPGA is to use memory mapped IO. This technique uses a memory address to select a specific piece of logic implemented in the fabric of the FPGA. These commands will read and write to such memory mapped registers.

For example:

```
picoCommand /rr 0x7000100          reads 32bit register at 0x7000100.  
picoCommand /rw 0x7000100 0x12345678  writes 32bit register at 0x7000100.
```

### 2.17 Save Entire Flash to File (/s)

**picocommand /s PCfile.rom**

This command will write a dump file from the entire flash ROM.

## 2.18 Launch Self Test (/t)

### **picocommand /t #**

This command will run SelfTest. # is a bit mask which defines which tests to run.

test basic registers	0x0001
test flash status	0x0002
test config	0x0004
test cfi	0x0008
test ppc ram	0x0010
test flash read	0x0020
test flash erase	0x0040
test reboot	0x0080
test check jtag	0x0100
test jtag spy	0x0200
test write sector	0x0400
test write sector	0x0800
bus mastering	0x30000
speed test	
bus mastering read	0x10000
test	
bus mastering write	0x20000
test	
repeat, ignoring	0x40000
errors	
repeat, stop on error	
bus mastering speed test	(0x0200 bit)
bus mastering read test	(0x10200 bit)
bus mastering write test	(0x20200 bit)
repeat, ignoring errors	(0x1000000 bit)
repeat, stop on error	(0x2000000 bit)

For example:

### **picocommand /t0x09**

runs basic tests and cfi tests.

## 2.19 Update Specified File (/u)

### **picocommand /u flashFile**

This command will update the specified flash file from the PC file name associated with the flash file.

## 2.20 Display Version Information (/v)

**picocommand /v**

This command will display the version number of the various components:

## 2.21 Write Specified File (/w)

**/w PCfile**

This command will write the specified PC file. The name on the flash ROM will be the last name specified in the PC file name.

For example,

```
picocommand /w c:\directory\file.bit
```

will write the flash file `file.bit`.

The `/w` command (without a numeric count) will overwrite the file if it is already present on the flash ROM.

**picocommand /w# PCfile.**

This variant of the `/w` command accepts a counter which specifies how many copies of the file should be copied from the PC file to the flash ROM. For example `'/w5 1meg.data'` will write five copies of the file `1meg.dat` to the flash ROM. Since the flash ROM requires that the files have unique names the files will be written as **1meg.dat, 1meg0.dat, 1meg1.dat, 1meg2.dat, 1meg3.dat.**

**picocommand /wo PCfile.**

This variant of the write file command will write a file to the flash ROM without a part number check (see error 7353)

## 2.22 Load and Execute Elf File (/x)

**picocommand /x flashFile**

This command loads the specified ELF from flash ROM into RAM and executes it.

## 2.23 Specify Simulation File (/z)

**/z simulateFile**

This command specifies a simulation file instead of actual hardware. This is a holdover from the original development of PicoUtil and PicoCommand. Using a simulation file allows one to assemble a flashROM image that can then be written in its entirety to the flashROM on the Pico Card.

Certain commands (such as reboot) are not valid on a simulation file.

## 3 Theory of Operation

### 3.1 Power On & Bootup Process

#### Theory of Operation: Power On & Bootup Sequence

The Pico Card can be rebooted in a variety of ways:

- The Pico Card is booted at power on from the primary boot image.
- The Pico Card may be rebooted on command from the PC to any image file.
- The Pico Card may be rebooted on command from the PPC to any image file.

Except for the choice of FPGA file, the boot sequence is as follows:

1. The TurboLoader loads FPGA file from the Flash ROM. After bootup, the TurboLoader is inactive.
2. Primary.bit programs the FPGA so that it can be accessed from the PCMCIA bus. The FPGA provides a CIS which identifies the card to the PC operating system.
3. The FPGA starts a program called PicoMonitor.elf which can field requests through the keyhole to the Pico Card.
4. If the card is plugged into a PC (ie the PCMCIA bus is active) then the monitor program waits for commands from the PC.
5. The OS reads the CIS or accesses the functions on the card to determine what kind of card this is.
6. The OS loads the appropriate drivers for this card. In this case the drivers are Pico.sys and Parport.sys (under Windows)
7. The Pico Card is now ready to be accessed as a device. For example the program PicoUtil.exe could now be used to examine the Flash ROM, change or delete files from the Flash ROM, and provoke additional reboots from the PCMCIA bus.
8. The parallel port provided by primary.bin is connected to the JTAG chain on the Pico Card. For hardware debugging tools such as EDK or ISE (from Xilinx) a special driver called XilinxPC4Driver.sys is loaded on the PC to provide bidirectional access to the parallel port and thus the JTAG chain. This is required for debug support of the PPC.

#### Reboot Under Command from the PC

When primary.bin is loaded into the FPGA, the Pico Card is entirely under the control of the PC. As noted above the PC can manipulate the Flash ROM, for example to add a new .bin and .elf file, or even to replace primary.bin. The primary utility for this purpose is PicoUtil.exe. Using this program (or any other program capable of accessing a device – DeviceloControl(...) under Windows), the PC can trigger a reboot of a tertiary .BIN and .elf file stored on the Flash ROM. The sequence is as follows:

1. The program (PicoUtil) calls the following function:
2. Pico.sys receives this IO control command and generates the following set of commands to the Pico Card:

- a. The value `0xFE00 + imageNo` is written to address `0x0216000` (defined as `REBOOT_CMD`).
- b. `Pico.sys` then waits for the program loader to ask which elf file to load.
- c. The program loader (in `Secondary.bin`) then loads the appropriate ELF file and performs the proper relocation of code/data segments.

## Reboot Under Command from the PPC

`Secondary.bin` contains a program in addition to the FPGA code which is loaded into the FPGA. The program contains the following functions:

1. Initial startup code (at address `0xFFFFFFF0`)
2. An ELF program loader (at address `0xFFFFFFF8`).
3. Flash ROM file access routines (at address `0xFFFFFFF4`).
4. Reboot code (at address `0xFFFFFFF0`).  
These addresses are fixed and independent of any operating system.
5. A keyhole debugger, ie a single byte bidirectional port through which commands can be sent (`PC_TO_PPC_KEYHOLE = 0x021600062`) and from which information can be retrieved (`PPC_TO_PC_KEYHOLE = 0x2160004`).

Typically the device driver `Pico.sys` will receive this information and make it available to an application program using the commands:

- a. `Pico_PPC_TO_PC_KEYHOLE`. Send a command block from the PPC running on the Pico to the PC.
- b. `Pico_PC_TO_PPC_KEYHOLE`. Send a command block from the PC to the PPC

When the PPC calls one of these addresses it is able to specify which image file and/or ELF file to load.

## Standalone Boot

After the initial boot, `primary.bin` will determine whether the card is plugged into a PCMCIA slot. If it is not, `primary.bin` will proceed with a standalone boot. This is similar to a reboot from the PPC, but the specification for the bit file image and the ELF file are obtained from the configuration control information on sector zero.

## Ad Hoc Boot

The user may at their discretion replace `primary.bin` and `backup.bin` on the Flash ROM and allow the Pico Card to boot directly into the application FPGA and program code. This is perfectly acceptable, however, it would require access through the JTAG port to rewrite an image. Proceed as follows:

1. Attach a JTAG cable to the Pico Card and a JTAG support port. These are typically connected to a PC through the parallel port.

2. Load the image primary.bin into the FPGA.
3. Use the program PicoUtil to replace the files primary.bin, etc.

It is possible that an image file is so corrupted that it will not even provide access to the Pico Card through the JTAG port. In this case the JTAG clock must be stopped by placing a small magnet near the board – refer to the application note [Raising a Pico Card from the Dead](#).

### **Rebooting the Pico Card.**

If the Pico-E12 is plugged into a standalone slot (such as our little brother board) the hosting hardware should pull the wait pin down with a 470 Ohm resistor. This is detected Monitor.elf in PrimaryBoot.bit and initiates 'standalone mode'. When standalone mode is enabled the file linked to PrimaryBoot is loaded. The program loaded by PrimaryBoot.bit can do several things. But let me backup to explain program load and linked files.

As you are aware, when the Pico Card is powered on the CPLD loads the first image it encounters on flash ROM which happens to have the name 'PrimaryBoot.bit'. This file is present on the flash ROM because it was written there by the maintenance utilities running on the pc-host. The host accesses the Pico Card through the PCMCIA connector.

The software on the host side comprises a driver called Pico.sys and an application mode interface called Pico.dll. These modules are driven by our GUI (PicoUtil.exe) or our command line app (PicoCommand.exe), or from a user program which call the various entry points in the dll.

When the Pico card is plugged into a PCMCIA slot PrimaryBoot.bit is loaded by the CPLD and NOTHING ELSE happens. This is appropriate for 'maintenance mode'. A user / user program may initiate loading other bit files into the FPGA or programs into RAM.

The user mode applications (PicoUtil and PicoCommand) also allow files to be linked so that loading one will also load the other. The mechanics of writing such linkage to the flash ROM is handled by Pico.dll (see WriteFile & ChangeFileProperties).

An elf file may be linked to a specific bit image (or vica versa - a bit file may be linked to a particular elf file). This means that when a bit image is loaded the linked elf file will also be loaded (or when a request to load the elf file the linked the bit file will be loaded first). This linkage is managed by Pico.dll (and can be disabled). In other words the linkage is managed from the pc-host. Pico.dll looks at the linked files and issues the commands to the Pico Card to load a particular bit file and / or a particular elf file. These options can be disabled by a parameter to the dll call.

PrimaryBoot.bit adheres exactly to this model. PrimaryBoot could be linked to an elf file in which case a command originating in PicoUtil to reboot the primary image (menu startup / boot selected file) would also load the linked elf file. However, PrimaryBoot.bit is usually loaded when power was applied to the card (ie when the card was inserted) so the linked elf

file is not loaded in the normal power on sequence - because pico.dll was not present to manage the process.

However, an elf file linked to PrimaryBoot.bit WILL be started if the card detects the standalone mode switch - ie is plugged into a standalone connector. In fact the load of ANY bit file will cause its corresponding linked elf file to be loaded by the Monitor program. PrimaryBoot.bit has one other wrinkle – it may be linked to a bit file. The second bit file will be loaded automatically in standalone mode. This is functionality that we are putting the finishing touches to right now. For our own application the sequence of events is as follows:

- 1) Power on loads the first image from the flash ROM (ie PrimaryBoot.bit).
- 2) Monitor.elf embedded in the block ram of PrimaryBoot.bit looks to the standalone switch and loads the bit file linked to PrimaryBoot. In our case this is a program called foobar.bit. NOTE: PrimaryBoot.bit could be linked to an ELF file which could then intelligently determine which image to load. I will not belabor the point since this sequence of ELF and bit file loading could be repeated indefinitely.
- 3) Foobar.bit is a specialized image for our standalone application. It too has the monitor program embedded in block RAM which in turn looks to the standalone switch and loads the file linked to foobar.bit - in this case Linux-2.6.15.elf. In our case the Linux system has a read only file system which enables it to access the flash ROM as a file system. We use this to read different drivers from flash ROM and then load these into Linux after the main Linux kernel (including its usual cluster of drivers) has been loaded.

The advantage of this three step process is that the Pico Card may still be plugged into a laptop and will still be automatically recognized. Maintenance of the flash ROM and some level of testing of the specialized bit file (foobar.bit) and the application (Linux-2.6.15.elf) can be executed in maintenance mode. PrimaryBoot.bit includes a parallel port emulation of the JTAG port to facilitate this testing.

This three step process can be reduced one or two steps depending upon circumstances:

- 1) If PrimaryBoot is adequate to run the application, the loading is only steps 1 & 2. In the terminology used above Linux-2.6.15.elf is linked to PrimaryBoot.bit.
- 2) If the entire software application can be embedded in block RAM within the bit image, only step 1 need occur. In this case, Primaryboot.bit is replaced by the application firmware including the PPC code in block RAM.
- 3) Unless the application firmware also included the interface to the PCMCIA (eg was derived from PrimaryBoot.bit) the card would NOT be able to interact with a laptop over the PCMCIA port. In this case maintenance of the card would require an external JTAG cable. We proceed as follows: Power up the card and plug in an external JTAG cable.
  - a) Load PrimaryBoot.bit onto the FPGA.
  - b) Load PrimaryBoot.bit onto the FPGA.
  - c) Insert the card into a laptop and use PicoUtil, or PicoCommand to change files on the flashROM.

I also should note that the one step version is the only one that works on the LX part (logic only - no PPC processor). We may embed a soft processor on this part but have not done so

heretofore.

## Software interfaces:

### Host side:

#### Linking two files.

- In PicoUtil specify a link file when the file is written to flash ROM, use right click / change file properties to add a link file.
- 'PicoCommand /k bitFileName elfFileName'. The command 'PicoCommand /k elfFileName bitFileName' is NOT useful in the context of loading the Pico Card in standalone mode.

Both programs use one of two calls to Pico.dll:

```
PicoXfaceP->WriteFile
    (const uint8_t *dataInP, int dataSize, //size of data block to write, NULL,0 if not provided
    const char *pcFileNameP, //name of pc file. Will be used if dataInP is null
    const char *flashNameP, //name on the flash file system
    const char *noteP, //notes associated with the file
    const char *linkedFileNameP, //elf file or bit file
    uint32_t writeOptions, //
    int *sectorsP) //sector map used (internally) when updating a file.

PicoXfaceP->ChangeFileProperties(
    FLASHROM_ADDR addr,
    const char *pcFileNameP,
    const char *flashNameP,
    const char *linkedFileNameP,
    const char *noteP)
```

### PPC side

#### Loading a new image

To load a new image call

```
KernelLinksP->LoadBitFile(const char *flashFileNameP)
```

this call culminates in two accesses to firmware registers in PrimaryBoot.bit

```
volatile uint32_t *accessP=(uint32_t*) 0x70000020,
                  *rebootP=(uint32_t*) 0x70000024;

*accessP = 1; //grants the PPC access to the PCMCIA
bus
*rebootP = fileAddress; //reboots with specified image.
```

#### Loading a ELF image

To load an Elf file call

```
KernelLinksP->LoadElfFile(const char *elfFileNameP)
```

NOTE: both of these calls are in the 3.5.0.x release of the firmware and software.

## 3.2 Raising a Pico Card from the Dead

### Theory of Operation: Raising a Pico Card from the Dead

The primary boot image shipped with the Pico Card provides the logic to present a valid Card Information Structure to the operating system at card insertion time. If this facility is compromised the Pico Card cannot be recognized by an operating system.

The Pico Card may be perfectly functional, and can be loaded using JTAG external hardware, or may operate in standalone mode. But it cannot be simply plugged into a PCMCIA slot for routine maintenance.

There are at least three conditions which will give rise to this problem:

- Deleting the primary boot and secondary boot file from the Flash ROM.
- Loading a corrupted primary boot image to the Pico Card.

The Pico Card can be resurrected from this condition with the following equipment:

- A JTAG controller.
- An external PCMCIA card carrier which can control the power to the Pico Card.
- An external power supply.
- A Pico JTAG cable.
- A copy of Xilinx Impact, or some other program to load the Pico Card through the JTAG port.
- PicoUtil.exe and a valid boot image.

Proceed as follows:

1. Plug the Pico Card into the external PCMCIA card carrier but not into the PC host.
2. Provide external power (3.3 volts) to the card carrier.
3. Plug the JTAG controller into the Pico Card (using a JTAG cable).
4. Start Impact and load the FPGA file over the JTAG port.
5. Push the external PCMCIA card carrier (with the Pico Card still plugged into it) into the PC card slot.
6. Remove external power.
7. Use PicoUtil to write the same primary boot image to the Pico Card. PicoUtil will usually volunteer to write a backup image at this time. This is not strictly necessary in this situation.
8. Eject the card carrier and Pico Card from the PC slot.
9. Remove the Pico Card from the card carrier.
10. Re-insert the Pico Card (alone) into the PC card slot - it should now be recognized by the operating system and behave normally.

It is possible that the (corrupted) primary image loaded on the Pico Card can prevent the JTAG controller from accessing the card. In this case a small magnet must be attached to the Pico Card to disable the FPGA clock so that the JTAG load (step 2) will be successful. The magnet should be placed on the 'o' of Pico. When the JTAG connection is established the magnet can be removed.

### 3.3 Flash File System

#### Theory of Operation: Flash File System

The Flash ROM is used to manage three different kinds of files:

1. **Image Files.** The FPGA is a vast array of logic cells that are tied together with active matrix. The interconnections defined by this matrix is what gives the FPGA its personality. This data is stored in .BIT or .BIN files. There are at least three image files on any Pico:
  - a. **primary.bin.** This file programs the FPGA to act as a PCMCIA client, provides the tuple data, and fundamental access to Flash ROM from the PC.
  - b. **backup.bin.** This file is an exact copy of primary.bin. In the event that primary.bin becomes corrupted, or does not load the FPGA at startup, the backup.bin file will be used by the TurboLoader.
  - c. **secondary.bin.** This file contains a standalone image which provides an RS-232 port, an Ethernet port, a parallel port, and a boot loader. In other words it contains logic and code.
2. **ELF Files.** PPC executable files are stored in the Flash ROM in ELF format. This is a standard relocatable format for PPC code. The boot loader in secondary.bin will relocate ELF files to run from off chip RAM.
3. **Other Data Files.** These are general purpose files with no presumed structure. Such files can be used for specific applications, for example, they might contain a representation of the human genome. Note that an operating system such as Linux may define its own file structures within such a data file.

The .BIN files are stored as contiguous blocks of data as required by the TurboLoader. The ELF and data files are stored as linked lists of sectors and are loaded by the program loader in secondary.bin. Each ELF sector has an eight byte link to the next sector in the file.

All file types have an ASCII header that identifies the file, its size, and any supporting notes about the file. All file types can be read or written from the PC using the utility PicoUtil. All file types can be accessed from the PPC at run time.

## 3.4 Debugging Support



### JTAG Debug Interface

The JTAG interface is an industry standard serial interface. The JTAG interface allows real-time debugging of hardware, firmware, and software. A wide variety of user programs (such as Impact and EDK (from Xilinx) use the JTAG interface.

The Pico Card is equipped with two JTAG diagnostic ports - an internal port and

external port.

- External port. The external port requires an external cable, a separate JTAG debugging interface, and a hardware interface to the laptop such as parallel port, USB port, or 1394 port. When the JTAG connector of the Pico Card is plugged in the soft JTAG port described below is disabled.
- Internal port. PrimaryBoot.bit contains an embedded JTAG diagnostic port. This simulates a parallel port. There are two ways to access this interface
  - Using a standard parallel port driver provided by the operating system. Under Windows this interface is serviced by parport.sys.
  - Directly from the driver Pico.sys (refer to [the Pico Software manual](#) for details of this interface). The JTAG interface is also a key element of the Viva interface to the Pico Cards.

**NOTE:** Some JTAG programs require the length of the instruction register (IR). The IR length is listed below for all devices in the JTAG chain.

Device	IR length	IdCode
FPGA / PPC FX12	10 bits	0x01E58093
FPGA / PPC LX25	10 bits	0x9167C093
FPGA / PPC FX20	10 bits	0x9167C093
FPGA / PPC FX60	10 bits	0x01EB4093
Ethernet MAC	8 bits	0x000003D3
TurboLoader	8 bits	0x06E5D093

**NOTE:** The internal JTAG cannot be used to reprogram the FPGA on the Pico Card. However, the The JTAG port is also the only way to reprogram the TurboLoader (CPLD).

**NOTE:** Windows has not historically provided high level access to the parallel port and therefore a number of applications have crafted their own technologies for obtaining and access the port address of the parallel port. These typically required a low level device driver and upper level DLL's to interpret the access to these drivers. Such products include Port95NT, and XilinxPc4Driver. Known bugs in the XilinxPc4Driver will prevent the proper 'surprise removal' of the Pico Card. In all probability this driver will cause your machine to 'blue screen'. We strongly recommend that any machine on which the Xilinx EDK or ISE is installed add a startup procedure 'net stop XilinxPc4Driver' to unload this driver except when it is actually in use.

## 4 Appendices

### Appendices

## 4.1 Glossary

### Glossary

**PCMCIA.** Personal Computer Memory Card International Association. A standard promulgated by this organization for a plug in card for Laptop computers (typically). <http://www.pcmcia.org>.

**Cardbus.** A 32bit version of the PCMCIA standard.

**Ethernet address.** This is a six byte address designed to be unique over the world. All network cards have an Ethernet address (whether they are Ethernet cards or not) and for this reason the address is frequently referred to as the MAC (Media Access Control) address.

**PCI.** The Peripheral Computer Interface is an ad-hoc standard adopted by some 900 companies who develop differentiated, inter operable products based on the PCI specifications. see [PCI SIG](#).