Nial Stewart Developments Ltd
FPGA Development
High Speed Digital Design

Easy PCI Core and Interface Board Design Description

Rev 1.0

Web: http://www.nialstewartdevelopments.co.uk Email: info@nialstewartdevelopments.co.uk Tel: +44 (0)131 454 0058
Registered Office: 35 Newbattle Gardens, Edinburgh, EH22 3DR Registered in Scotland No: 207181 VAT Reg Number:751 2366 45

1. History

Release	Author	Date	Reason
1.0	NS	2/12/03	Initial Release
1.1	NS	16/12/03	Added FPGA and core size details.

2. Contents

3. Rationale behind the Easy PCI Board/Core development

The PCI bus specification was released in 1992, with the 'standard' revision 2.1 relased in 1995, it has been adopted as the standard PC interfacing bus and is also used in slightly different forms in many embedded products.

Many different products are available for interfacing to the PCI bus; both FPGA/ASIC cores and stand alone ICs dedicated to implementing the interface.

Despite the abundance of devices/approaches available there has been no simple way of quickly and cheaply implementing a PCI based design. Many cores are complex to implement and require a level of understanding of the PCI bus and protocols. They also leave the not insignificant PCI board design requirements and FPGA/ASIC implementation to the engineer. Dedicated PCI interface ICs can be simpler, but add the cost of another IC to component cost, power consumption, manufacturing complexity and testing, and reduce the available board space for custom hardware.

The EASY PCI board and core combination has been designed to make implementing a PCI based design as simple as possible. No significant PCI knowledge required for integrating the core and the PCI aspects of the physical interface have already been designed and tested.

This should allow custom PCI based designs or prototypes to be quickly and more cheaply designed. Removing the expansion headers and substituting custom hardware allows us to quickly develop custom designs. The board can also be used as a generic development platform as daughter boards can be designed to plug onto the expansion headers, allowing the board to be used as the basis for a number of designs.

This design description describes the board and what's required to interface to the local bus. A tutorial will also be provided with an example implementation of the EASY PCI core.

4. Core Details

4.1 PCI Details

Note: An understanding of the following is not necessary to implement designs with the EASY PCI core.

The core is a 32bit, 33MHz target only core (a master/target core will follow). Investigation has failed to reveal a PCI bridge device, which allows target burst accesses. The logic and latency used to implement burst accesses would in 99% (100%?) of cases be wasted, so a decision was made only to support single access cycles.

I/O accesses are not supported, all accesses to the core/board are by memory mapped transfer. PCI Memory Write, and Memory Write And Invalidate are supported and aliased to Memory Write, PCI Memory Read, Memory Read Multiple and Memory Read Line are supported and aliased to Memory Read.

Every access is terminated by a Disconnect With Data.

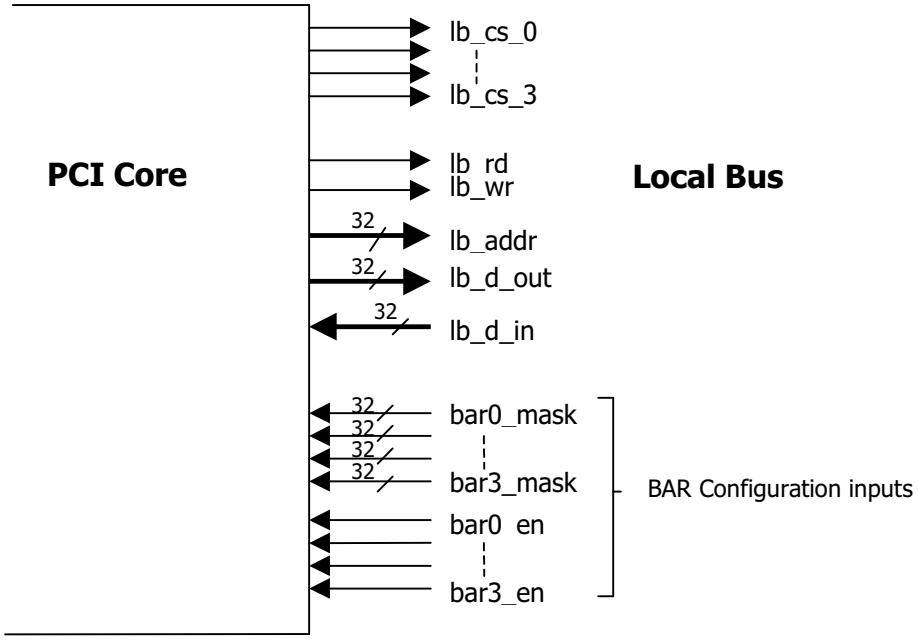
Although the PCI spec allows for 6 BAR registers, all of these are rarely used. This core provides 4 BAR registers, allowing 4 separate memory mapped areas to be defined.

4.2 Implementation Details

The following sections explain what's involved in using the core in a target design, simulation, performance etc.

4.2.1 Local Bus Interface

The PCI interface core translates the PCI bus transactions to a much simpler local bus, the signals are listed below, the timing diagrams follow.



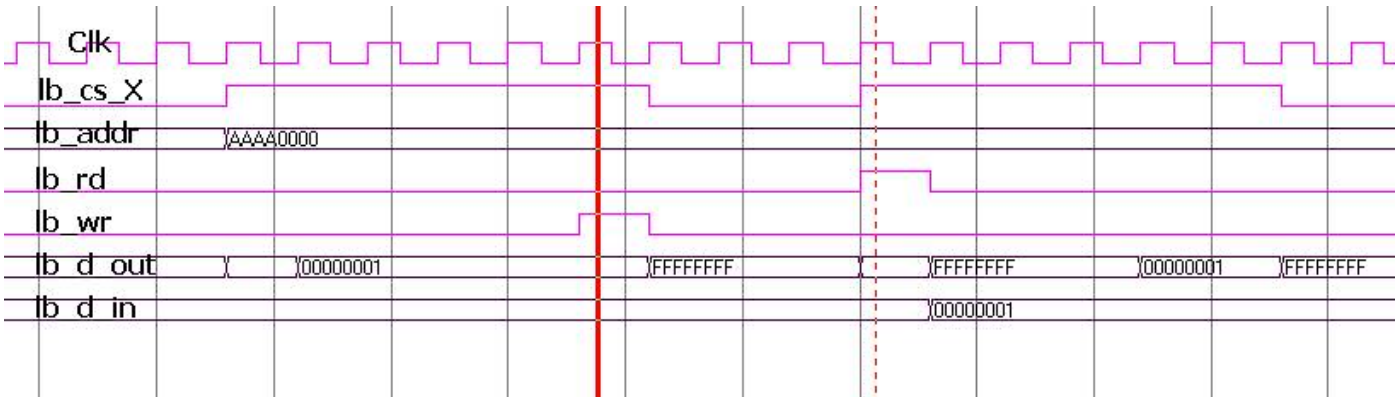
Signal	Direction	Description
lb_cs_0 lb_cs_1 lb_cs_2 lb_cs_2	Out	Local Bus Chip Selects 0 – 3. These signals are used to qualify accesses to one of the 4 available memory mapped address ranges.
lb_wr	Out	Local Bus Write. Used as an enable for lb_d_out to the local bus writes.
lb_rd	Out	Local Bus Read. Enables data in from the local bus onto lb_d_in.
lb_addr(31..0)	Out	32 bit local bus address. Only those bits needed to resolve local register access need be used.
lb_d_out(31..0)	Out	32 Bit local bus write data.
lb_d_out(31..0)	In	32 bit local bus data returned on a read cycle.
bar0_mask bar1_mask bar2_mask bar3_mask	In	Base Address Register X Mask inputs. These are used to define how much memory is used by each chip selectX. They are discussed in more detail later in this document.
bar0_en bar1_en bar2_en bar3_en	In	Base Address X Enable. These are single signal inputs which are set to '1' if the relevant BAR space is used, '0' if it's not. These should be set to '0' even if the BARX_mask is set to all '0's.

Each bus transaction is a simple 32 bit read or write to an address in one of four possible memory mapped regions, these are enabled with the barX_en signal, their size is defined with the barX_mask registers.

All PCI access addresses are aligned to 4 byte boundaries, so the bottom 2 address bits need not be used in local bus address decoding.

The interface is synchronous to the PCI clock, this is nominally 33MHz but bear in mind the PCI specification says this can be slowed to a stop. This is easily interfaced with even if user logic is being driven at a higher clock speed (it's easiest if the user logic is at a significantly higher speed).

A timing diagram is shown here of a PCI write and then read to the same register on the local bus (please excuse the colours, this is a capture of a Modelsim simulation with the colours inverted for clarity).



The write occurs on the cycle indicated by the first solid cursor, the read on the dotted second. The write cycle is writing 0x00000001 to a local address. **The lb_d_in from a read is registered in at the end of the clock cycle following the active lb_rd signal.** It can be seen that the value written during the write cycle is returned. It is recommended this be registered synchronously against the address, chip select and read signals.

Note: The data returned on a read must be valid the following PCI clock cycle. If user defined hardware cannot return data this quickly a separate mechanism must be used to 'pre-read' the data and have it available for reading.

Each of the 4 chip select signals represents a separate area of PC memory and each may or may not be used. The chip select signals should be used to qualify the write and read signals to control access to different parts of implemented designs if required.

The memory area covered by each chip select signal is defined using the barX_mask inputs, these are explained below.

4.2.2 BAR Enabling

Each of the memory mapped regions which is used must be enabled by setting it's respective barX_en control line to '1'. This could have been automatically disabled when the BAR mask is set to all '0's, but this would have used more FPGA resource, and as it doesn't dynamically change this is easily set at design build time.

4.2.3 BAR Mask Configuration

In order to use the 4 different chip select signals of the local bus, the amount of PC memory space that is needed for each must be defined. This allows the Plug and Play mechanism to allocate the required memory when the OS boots up. Each of the 4 chip selects has a 32 bit 'barX_mask' associated with it that defines the amount of memory needed for that chip select. If the barX_mask is set to '0's the associated chip select will be disabled.

The memory size is restricted to sizes that are powers of 2, and to save resources the smallest memory space that can be reserved for a chip select line is 1K. To set the memory size needed to X, where X is 2^Y, bits 31 to 'Y' in the barX_mask are set to '1' (where the barX_mask bits are numbered 31……0), the rest '0'.

As an example, an area of 4K is needed for chip select 0. $4K = 2^{12}$, so bar0_mask bit 12 and all above it are set to '1', the rest are cleared so bar0_mask = "11111111111111111110000000000000".

To reserve 1Mbyte for chip select 1, $= 2^{20}$, so bits 31 to 20 are set to '1', the rest '0's, bar1_mask = "11111111111100000000000000000000".

If the BAR enable control line is active '1', the respective BAR mask register **must not** be set to all '0's as this will confuse the operating system plug and play mechanism, probably with disasterous results.

4.2.4 Interrupt Generation / Clearing

In order to allow flexible interrupt responses, a separate interrupt source "intrpt_srce" and interrupt clear "intrpt_clear" signals are used.

When intrpt_srce is active high the PCI core latches this in and signals a PCI interrupt to the host CPU. The interrupt is only cleared when intrpt_clear is driven active. This allows the interrupt to be cleared by whatever mechanism the user wishes (i.e. intrpt_clear can be driven during a read of a register, but a specific register write etc).

NOTE: If interrupts are used the intrpt_clear signal must be implemented, otherwise the host PC may lock up.

4.2.5 PCI Pin Locations

The PCI spec recommends ordering the pins on a device perimeter in the same order that they come off the PCI connector; this simplifies routing and reduces crosstalk between signals. This rule has been broadly followed except for a few critical signals, TRDY#, IRDY#, STOP# and RST# which have been routed to Cyclone dual-purpose clock pins to help meet the set up and hold requirements.

4.2.6 Core Distribution

It is currently proposed to distribute the EASY PCI core as a Quartus *.vqm file. The core should be incstantiated in projects as a black box, Quartus will insert the contents of the *.vqm file.

4.2.7 Functional Simulatio n

A pre-compiled Modelsim 5.3d library containing the core can be downloaded for simulation. This can be 'refreshed' for use with later versions of Modelsim, see the Modelsim help files for details.

Alternately a post synthesis, place and route simulation can be done of the final chip design incorporating the EASY PCI core (although this will be much slower).

4.2.8 Driver Generation

An example driver with example software in C++ is provided with the core.

We recommend the Jungo Windriver tools from …

http://www.jungo.com

..for custom driver development. This tool suite makes it fast and simple to generate drivers for custom applications.

There is a fully functional 30 day trial download available.

4.2.9 Vendor ID/Device ID etc

The Vendor ID, Device ID, Revision ID, Subsystem ID and Subsystem Vendor ID are embedded in the configuration header of a PCI interface. These are used by the operating system to identify the board and load up the correct drivers. There are provided as generics with the EASY PCI core and so can be configured at build time.

The PCI Special Interest Group (PCI SIG) distributes Vendor IDs to members of the PCI SIG; membership costs $3000 annually.

If products are designed round the EASY PCI card, customers should join the PCI SIG to obtain a Vendor ID that is guaranteed to be unique. For experimentation purposes a random Vendor ID can be chosen and checked on the PCI SIG web site that it hasn't previously been assigned….

http://www.pcisig.com/membership/vid_search/

For evaluation purposes the provided driver used a Vendor ID of 0xED04 and Device ID of 0x1CE0.

4.2.10 Performance

Tests have shown that a PC running a tight software loop can perform 2.5 million 32 bit writes per second to the core, 1.6 million reads. This equates to a write bandwidth of approx. 10MBytes/second and a read bandwidth of approx. 6.6MBytes/second.

These rates correspond to expected maximum rates for a Target only PCI core.

4.2.11 Core Size

The Easy PCI core with a minimal test design around it (which generates variable interrupt rates, drives the LEDs and contains a couple of dummy registers) occupies 399 Logic Elements (6% of the EP1C6), of which 234 are the Easy PCI core.

5. Board Details

The board is the universal 5V/3.3V PCI card form factor and has been designed to operate in both environments. All testing and verification has been performed on systems with 5V PCI card slots, a 3.3V system was not available. Although this has not been tested no problems are envisaged, as all voltage levels are 3.3V compliant.

The board will also operate stand alone on the bench when powered by a 5V supply via connector J1. The on board voltage regulators accept an input to 7V and can supply 3A to the 1.5V and 3.3V sections of the board.

It's recommended that stand-offs are used in the 4* 3mm corner holes if the board is used on the bench to avoid accidental shorting of the back of the board.

5.1 FPGA

An EP1C6Q240C FPGA has been used in this design. This contains 5980 Logic Elements, 20 M4K Block Rams (128x36 bits configurable), 2 PLLs (the PCI clock is routed to one, a 20MHz clock to the other), and a **maximum** of 185 user pins (all aren't available in every configuration, 49 are lost to the PCI interface).

5.2 PCI compliance

The board is PCI compliant except for clause **4.4.3.4 Signal Loading** that stipulates that only one load is allowed per PCI net.

In order to use modern cheap high density FPGA families that don't have 5V tolerant input pins, protection must be provided from the raw 5V PCI signals. This is provided by IDT Quickswitch devices that operate as bi-directional voltage limiters. They are not voltage clamps but variable impedance buffers. When the voltage at a pin approaches a threshold the impedance of the device rises sharply from almost zero. This impedance rise stops a voltage higher than the threshold being transmitted across the device without clamping it at the source side. In this application the threshold is biased to clamp to 3.1V or so, the switching threshold for 5V PCI is 2.4V. The delay across the Quickswitch device is 120 ps in both directions.

The Quickswitch devices also provide the overshoot and undershoot protection required by the specification.

Inspection has shown that all transitions on the bus are monotonic and glitch free. The rising impedance of the Quickswitch when a source signal passes 3.1V will not cause spurious glitches or reflections on the 5V side of the PCI bus.

More details on the Quickswitch devices can be found here…

http://www.xilinx.com/bvdocs/appnotes/xapp646.pdf
Xilinx App. note 646 "Connecting Virtex-II devices to a 3.3V/5V PCI bus".

and here…

http://www.idt.com/docs/AN_11.pdf
IDT App note AN-11A "Bus Switches provide 5V and 3V Logic Conversion with Zero Delay"

The board meets the spirit of the spec. w.r.t. 4.4.3.4 Signal Loading, but not quite the word of the spec.

The board meets all other spec. stipulations, including a maximum capacitance of 10pF per line, the restriction on physical track lengths, plating and connection of all connector contacts and the decoupling of unused power pins.

5.3 Expansion Headers

The expansion headers allow access to 78 unused FPGA IO pins. Each connector contains multiple Gnd and power connections (typically every 3rd pin is a ground) to allow easy transfer of high speed signals with fast rising edges to user logic on daughterboards. There are 4* 3mm mounting holes that are widely spread to allow secure mounting of such cards.

The connectors are currently high density surface mount parts, AMP part number 179031-1 . We would recommend using the corresponding AMP 177983-1 to give a spacing of 8mm between the boards.

We will supply mechanical drawings of the board to aid daugherboard design on request.

An 'educational' version of the board with simpler 0.1" headers may be provided in the future.

The on-board 3.3V regulator can provide 3A each for the PCI interface board and any custom daughterboards. Raw 5V from the PCI connector is available for daughterboards.

5.4 FPGA Configuration

The FPGA configuration is held in an EPCS4. This is currently programmed using an Altera ByteBlasterII cable via connector J2. The EPCS4 can be re-programmed when the board is plugged into a running PC, the FPGA configuration is then updated the next time the machine is powered down and up again.

Altera provide IP that allows a NIOS CPU core to re-programme the EPCS4. We are currently investigation the feasibility of providing a minimal NIOS design which would allow the EPCS4 to be configured over the PCI bus without the ByteBlasterII cable. This would potentially allow customers the ability to offer field upgradeable PCI boards.

5.5 Clocking

The PCI clock is routed to PLL1. This is used to clock the PCI interface core and is available for user logic.

A 20MHz oscillator is also provided, connected to PLL2. This allows internal logic to be clocked at speeds of **20MHz * M/(N * post_scale_counter)** where M = 2 to 32, N and post_scale_counter = 1 to 32.