Port Switcher Example

Summary

This application note describes the PortSwitcher demonstration project, which can be found in the `\Examples\Reference Designs` folder of the installation.

Application Note
AP0117 (v2.0) February 27, 2008

The Port Switcher project is designed to show how simple application of the BoC system can be used to solve a problem that would require significant logic and PCB routing. The FPGA solution is small and easily adaptable. Concept and FPGA fit and function can be almost fully developed and tested on the NanoBoard. The development FPGA project and the final FPGA project are very similar, showing another advantage of the BoC system. No VHDL has been used, in order to demonstrate that real world problems can be solved with the FPGA while still allowing a traditional discrete chip designer a familiar design paradigm

Some of the software features demonstrated by the design include:

- Processor core embedded in FPGA.
- Traditional "digital" engineer-style design – consisting of basic, well-known building blocks connected in a schematic (no VHDL).
- Mutli-channel features, both within the FPGA and externally on the PCB.
- Design flow between FPGA and PCB.
- Maintaining Nanoboard and custom PCB versions of the source project.
- Use of dedicated input-only pins on the FPGA.
- Manual pin allocation using constraint file.
- Debugging embedded micro-controller and FPGA design through the Nexus JTAG interface.
- Use of User Board headers to configure and debug custom designs.
- Demonstration of a very lightweight RS-232 serial port handler for the TSK51 processor.
- Use of TSK51 embedded processor using only internal RAM (idata) and code memory. That is no additional RAM (either internal or external to the FPGA) or external (to the FPGA) code space is used.

In essence the Port Switcher consists of four ports each with 17 input and 17 output pins. The output pins of a port can be mapped to reflect the state of corresponding input pins of any port. to and drive any of the four output ports. This mapping can be flexible – that is, if Port A's inputs are connected to Port B's outputs, the system does not force Port B's inputs to connect to Port A's outputs. Ports can be configured to loop back to their own outputs, to connect to any other ports outputs (or to connect to more than one output port). Output ports can be disabled.

The system consists of one main printed wiring assembly (PWA) that is designed to plug in to various adapter boards that interface to real world systems. Two example adapter boards are provided. One connects between simple parallel ports (nonbidirectional, so not ECP or EPP ports). The other connects between RS-232 systems.

The parallel port is probably an example of a system where the input of one port would be mapped to the output of another and visa-versa (simple Port X to Port Y mapping). The RS-232 example could, conceivably, be used in more complex switching arrangements where Port X drives port Y, but Port Z drives Port X, or in multi-port situations where Port X might drive Ports W, Y and Z.

Enabling and disabling ports, establishing port to port mappings and reading status and version numbers is done through a dedicated RS-232 serial link using a simple command interpreter that can be accessed via a simple dumb terminal.

Brief Design Description

The Port Switcher main board consists of three linked Altium Designer projects:

- `PortSwitcherPCB1D.PrjPcb`
- `PortSwitcherFPGA1C.PrjFpg`
- `PortSwitcherEmb1C.PrjEmb`

A further FPGA project, `PortSwitcherFPGA1C_Nanoboard.PrjFpg`, is very similar to `PortSwitcherFPGA1C.PrjFpg`. It only differs in the presence of some NanoBoard instruments on the top-level sheet that permit testing on the NanoBoard.

Each of the adapter boards consist of just one PCB project:

- `PortSwitcherParallelPort.PrjPcb`
- `PortSwitcherRS232.PrjPcb`

Design Description - PortSwitcherEmb1C.PrjEmb

The `PortSwitcherEmb1C.PrjEmb` is a firmware project targeting an embedded TSK51 processor. The firmware project is fully contained in one C source file - `portswitcher1c.c`

As well as hardware initialisation the program implements a simple command interpreter allowing a user, via an RS232 terminal, to control the port switching and to query a number of status values. A very lightweight interrupt driven serial port handler is implemented. Since no external RAM is used (only idata) RAM space is very limited. Convenient facilities such as large serial port transmit and receive buffers are **not** implemented.

This project is designed to support the other aspects of the system and does not demonstrate anything particularly unique about the BoC system.

There are a number of points in the code where code will only work for specific number of ports. Despite the presence of #defines that appear to parameterise the code this is not the case. The code in this project will only work as-is for four ports. The places that force this restriction in the code are commented.

The RS232 terminal interface is hard coded to 57600, no parity, 8 bits, 1 stop bit.

Use with NanoBoard

- The PortSwitcher1C_Nanoboard.PrjFpg project should be used for demonstrating the project on the NanoBoard. This uses an IOB4x16 instrument to demonstrate the port switching.
- The clock should be set to 32MHz to ensure correct baud rate (in fact the correct clock freq is 33.178MHz, but 32 MHz is close enough to ensure correct UART operation and is a common oscillator frequency).
- Basic Operation. The '?' or 'H' key will give a small help screen. The 'M' command will allow you to map one ports output from another ports input - note that a port can have its outputs driven from a different port than it is driving. A single port can drive multiple outputs. The 'E' command is used to enable a port's outputs (not done automatically when a port map is changed). The 'D' command will dump the current interconnection and port status. To enter a command just the single letter (not case sensitive), any parameters separated by spaces and hit <Enter>. The system does **not** echo characters so you would probably want to set your terminal emulator up to echo characters locally and to append new line to carriage returns. If you connect a 7-segment display to User I/O 2 it will show the number of characters received since the last command. Backspace editing of commands is **not** supported. An incorrect command will generate a 'Huh?' response.
- Use the IOB4x16 instrument to test connections - change the IOB outputs and watch for expected changes on the IOB inputs. Be aware the IOB instrument can be out of synch after the download. Enabling a port may not cause an expected change on the IOB until it has been played with a little after the downloading to the NanoBoard, after changing an output the IOB instrument seems to behave.

Command Description

Commands are entered via an RS232 terminal. The <Enter> key terminates each command. A “cmd>” prompt is presented to the user when the system is ready to accept a command. Backspace is not supported. Unknown commands generate a “Huh?” response. Commands with invalid parameters generate a “Fail” message.

Ports can be referenced as either numbers (1 to 4) or letters (a to b or A to B). Port 1 corresponds to Port A etc.

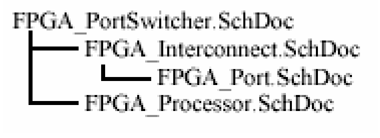
? or H	*Help*: displays a short help page. No parameters
E	*Enable Port Outputs*: takes port and a Boolean parameter. Ports are 1 to 4. Boolean 0 will disable the port; Boolean 1 will enable the port. Example:

	E 1 0 - would disable port 1 (port A) E c 1 - would enable port c (port 3)
D	*Dump Port Status*: dumps the connection and enable state of all four ports
M	*Map Port*: maps a destination port's outputs to be driven by the source port's inputs. Takes two port numbers, destination followed by source. The enable state of the ports is preserved, though the port is turned off while it is being re-mapped. Multiple output ports can be driven from the same input port. Example: M 2 3 - would map the outputs of port 2 (port B) to reflect the inputs of port 3 (port C). M c a - would map the outputs of port C to reflect the inputs of port A.
Q	*Query Port*: dumps the connection and enable state for one nominated port. Example: Q a - would dump the state of port a.
V	*Dump Version*: dumps the version and compile date and time of the application.

Design Description - PortSwitcherFPGA1C.PrjFpg

The `PortSwitcherFPGA1C.PrjFpg` is an FPGA project targeting Xilinx Spartan-IIE (XC2S300E-6PQ208C).

The design quite simple consisting of four schematic sheets in a hierarchy as shown:



`FPGA_PortSwitcher.SchDoc` is the top level sheet and contains ports (mapping to FPGA pins) and two sub-sheets along with a few IBUFG buffers used to signal to the Xilinx fitter that these ports should be connected to dedicated input pins (GCLK pins not being used for global clock inputs) (more on this below).

`FPGA_Interconnect.SchDoc` contains four copies of the FPGA_Port.SchDoc subsheet forming a multi-channel design. Due to limitations with sub-sheets formed into a multi-channel design using the REPEAT keyword1, four individual copies of the sheet are used rather than the REPEAT keyword[1]. The `FPGA_Port.SchDoc` sheet uses a generic 16-bit four channel multiplexer along with a 1-bit four channel multiplexer to implement a 17-bit four channel multiplexer. The output of the multiplexer feeds the output pins. The multiplexers implement an output enable signal.

Bus wiring on `FPGA_Interconnect.SchDoc` connects all the incoming busses to each of the output multiplexers. Enable and multiplexer select signals are routed from the processor sheet (via the top level sheet) to the individual `FPGA_Port.SchDoc` subsheets.

A 17-bit system was implemented as this used all available pins on the FPGA and maximised the capabilities of the port switcher. The parallel port adapter uses all 17- bits of input and output to implement the simple parallel switch.

Dedicated Input Only Pins

All available pins were required to implement the 17-bits of I/O for all four ports. The FPGA provides four global clock inputs (GCLKn). Only one clock input is required for this application. If a global clock input is not needed the pin can be used as an input-only pin. Three of these GCLKn inputs were configured as input only pins.

This is done by adding a boolean parameter to three of the ports: inhibit_buf=true. In addition three IBUFG buffers had to be placed from the Xilinx Spartan-II & Spartan- IIE FPGA.IntLib library to allow the system to place these input on the input-only pins.

[1] Sheets repeated using the REPEAT construct in the sheet designator and on the sheet entries could not support busses at the time this design was implemented. The multi-channel system detects the four sheets are referencing the same sub-sheet so will still treat this as a multi-channel design – so allowing access to the channel numbering features of schematic and the multi-channel features within PCB.

Manual Pin Allocation

It was found that the automatic pin allocation optimiser did not make the layout as simple as a manual allocation.

A constraint file, PortSwitcherPCB1C(FixedPins).Constraint, was created and edited with all the pin allocations manually assigned to ease the PCB layout.

PortSwitcherFPGA1C_Nanoboard.PrjFpg

The `PortSwitcherFPGA1C_Nanoboard.PrjFpg` FPGA project is very similar to the `PortSwitcherFPGA1C.PrjFpg`. It differs only in the top-level sheet and the constraint file that is configured. This project targets the NanoBoard rather than a custom PCB and so uses the appropriate Xilinx NanoBoard configuration file. The top-level sheet also includes some NanoBoard instruments to allow testing of the FPGA and embedded projects.

Design Description - PortSwitcherPCB1D.PrjPcb

The `PortSwitcherPCB1D.PrjPcb` is a PCB project that implements a custom board including a Xilinx Spartan-IIE XC2S300E-6PQ208C. The FPGA project discussed above implements the FPGA firmware.

This PCB is quite simple. It consists of a central FPGA along with power supply, JTAG interface for debugging and programming and an RS-232 interface circuit. Four 60-pin headers are used to expose the four 17-bit input and output ports. These headers are mounted on the bottom of the PCB, allowing the PWA to be plugged into various adapter boards that interface the system to a specific interface standard.

Multi-Channel Design

The PCB project makes good use of the multi-channel features of Altium Designer. The top-level sheet, `PCB_PortSwitcher1D.SchDoc`, has four copies of the sub-sheet `PCB_PortIO1D.SchDoc`. Four copies, rather than a single copy with a REPEATed designator are used to overcome the problem of the current build not supporting REPEATed busses. See the footnote 4 above. `PCB_PortIO1D.SchDoc` connects the signals from the FPGA to the adapter header, JP400, via the series protection components.

Clock Oscillator

U200 is a 5-volt clock oscillator. A buffer with 5-volt tolerant inputs, U201, is used to level-translate the oscillator output to a suitable level for the FPGA.

U200 is driven directly from the incoming 5-volt power supply. This is not desirable and would almost certainly **not** be appropriate in a production design. Is it important that a 5-volt input supply be used to prevent damage to this device.

To generate the exact 57600 baud serial port rate, the oscillator frequency would be 33.178MHz (assuming the use of an embedded TSK51 processor core). This is not a standard frequency. A common oscillator frequency that is close enough to ensure reliable RS-232 communications is 32MHz – this is what is specified in the design. The baud rate error is 3.6%, which is generally tolerable, at least for short cables. In a production environment this may be considered too high an error.

JTAG Switching

A quad 2-input multiplexer, U500, controlled by a simple jumper, allows a Xilinx configuration Flash memory, XCF04SVO20C, U501, to be included or dropped from the JTAG chain. This device could be left in the JTAG chain at all times. However, by including U500 the JTAG chain can be shortened, so maximising the BoC responsiveness during download and debugging. When a suitable target bit file has been prepared this can be stored permanently in the configuration memory to permit stand-alone operation.

The order of devices on the JTAG chain can, apparently, affect the speed of downloads. Placing the configuration memory at the beginning of the JTAG chain allows maximum speed for production programming. Providing the jumper to allow the configuration memory to be dropped from the JTAG chain permits fastest possible debugging. This is at the small cost of a cheap logic gate (U500).

U500 has 5-volt tolerant inputs, allowing it to safely interface to PC parallel ports.

A small amount of static, and incorrect connection, protection for the JTAG chain is offered by series 100R resistors. Users contemplating using this reference design in a product should determine whether this provides sufficient protection for their needs.

Port Protection

Series resistors on all port inputs and output, and pull-down resistors on port inputs, are used to reduce the likelihood of damage to the FPGA due to incorrect insertion, shorts etc. This protection is quite weak and is probably not satisfactory for a production ready design. The series resistors will also reduce maximum signalling speed.

Users contemplating using this reference design in a product should determine whether this provides sufficient protection for their needs.

EMI

The series resistors, mentioned above, will tend to reduce EMI. Also, the four-layer PCB, along with normal careful consideration of supply and signal routing will also tend to minimise EMI. However, this design has not been laid out or designed to meet any particular EMI standards.

The presence of simple non-impedance controlled and non-shielded connections between the main PCB and the adapter boards would be an area of concern for radiated EMI.

As mentioned above, the clock oscillator is currently driven directly from the 5-volt incoming power supply. Due to the minimal filtering conducted EMI may well be an issue with this design as it stands.

Design Description - PortSwitcherParallelPort.PrjPcb

The `PortSwitcherParallelPort.PrjPcb` is a sample adapter board that allows switching of four input simple parallel ports to any one of four output parallel ports.

Implementation, again, makes use of the multi-channel feature of Altium Designer. This time, however, since there are no busses needing to be REPEATed, the designator REPEAT syntax of the sub-sheet can be used.

Each port consists of bus driver devices ('74LCX244 and '74LCX541) buffering the FPGA from the external parallel port signals. In a simple parallel port (SPP) some signals are driven by the PC toward the printer (or other device) and the printer drives other signals back towards the PC. No signals in the SPP are bi-directional. This design will not work, as-is, with bi-directional parallel ports such as ECP and EPP implementations.

Each adapter port consists of an input port and an output port – that is there is an SPP Input Port A and an SPP Output Port A. Input signals from an input port are directed to the corresponding signal on the mapped output port. Input signals on the output port would generally be mapped back to the same input port – or else the parallel port device connected to the output port would probably not operate correctly. Generally if Input Port A is being mapped to Output Port D (command "M D A") you would expect a reverse mapping to be established (command "M A D") to ensure correct SPP operation (see the section on Mapping Ports, below, for more details).

Pull-downs on the adapter board inputs ensure there are no floating inputs when no parallel port is connected. Low value series resistors offer a small amount of static protection to the adapter board silicon devices. These resistors may prevent portpowered devices from operating correctly when connected to this port switch. The series resistors may also reduce signalling speed, or prevent correct operation in some circumstances. Users considering using this design in a production product should carefully assess the implications of this simple protection network.

Note that the pin numbering of the header pins on the adapter board differs from that of the main board. Pin 1 on the main board goes to pin 2 on the adapter, pin 2 to pin 1, pin 3 to pin 4 and pin 4 to pin 3 etc. This is due to the main boards header being mounted on the bottom of the PCB.

Design Description - PortSwitcherRS232.PrjPcb

The `PortSwitcherRS232.PrjPcb` is a sample adapter board that allows switching of RS-232 ports. There are four DTE and four DCE ports that can be interconnected. The outputs of any DTE (and DCE) port can be mapped to reflect the state of the inputs of any DCE (and DTE) port. This can be simple interconnection such as Port A-DCE driving Port B-DTE, while Port B DCE drives Port A-DTE (mapping command s "M B A" and "M A B"). Alternatively more complex arrangements such as daisy chains or multi-channel broadcasts can be configured.

When a mapping has been established the input signals of the DCE port will drive the corresponding output signals of the mapped port. Also, the input signals of the DTE port will drive the output signals of the mapped DCE port.

Implementation, again, makes use of the multi-channel feature of Altium Designer. The designator REPEAT syntax for the sub-sheet is used.

Each port consists of RS-232 driver/receiver chips along with pull-downs on adapter board inputs to ensure the inputs are not floating when the FPGA is under configuration or being held in reset.

Note that the pin numbering of the header pins on the adapter board differs from that of the main board. Pin 1 on the main board goes to pin 2 on the adapter, pin 2 to pin 1, pin 3 to pin 4 and pin 4 to pin 3 etc. This is due to the main boards header being mounted on the bottom of the PCB.

Mapping Ports

Port mapping is quite simple, but can be a little difficult to understand when combined with signal routing on an adapter board.

Within the FPGA, a 4-channel 17-bit multiplexer switches one of the four sets of 17 inputs (the inputs from ports A, B, C and D) through to the 17 outputs of the port. Each input channel of the multiplexer consists of 17 input signals; there are four such input channels. The output of the multiplexer consists of 17 output signals. An enable signal, when inactive, can be used to force all outputs, on that port, to zero.

The output pins of each port are driven by such a multiplexer – there are four multiplexers in the design. A 17-bit input bus can drive more than one 17-bit output bus. Internally, all four input busses are presented to each port's multiplexer – independent select (and enable) signals to each multiplexer allow the embedded processor to control each multiplexer fully independently.

An adapter input port may contain both input and output signals. An adapter output port would (generally) contain the inverse set of output and input pins – a signal that was an input on the Adapter Input Port would be matched by a signal that was an output on the Adapter Output Port.

Figure 1 below shows the basic interconnection between FPGA inputs, FPGA multiplexers and the FPGA outputs. It also shows the signal flow for adapter input and output pins on the Adapter Input and Output Ports.

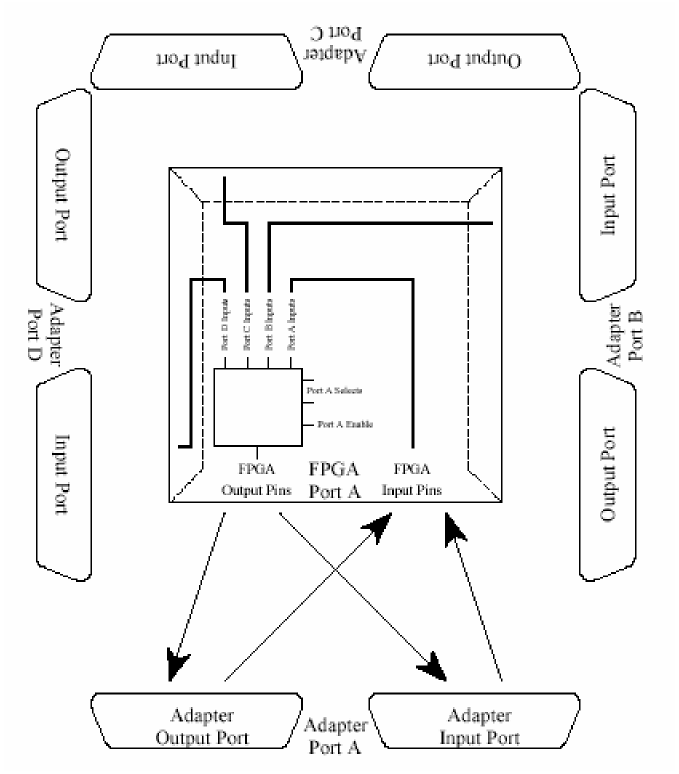


Figure 1 Signal routing between Adapter and FPGA Ports

Simple Mapping

The most common form of switching would be one adapter port (as distinct from a port on the FPGA multiplexer) being mapped to another adapter port. Since adapter ports will often contain both input and output signals, adapter boards will usually need to route some FPGA input pins and some FPGA output pins to the adapter input port and similarly some FPGA output pins and some FPGA input pins to the adapter output port.

The implication for this sort of system is that a user would generally want Port X to map to Port Y and Port Y to map to Port X. This is "Simple Mapping". If the user is attempting to map an input connection on Port X to the output on Port X this will only require a single loopback mapping command – for example "M A A", mapping port A to Port A. If the user is attempting to map from Port X to Port Y this would require a matched pair of mapping commands such as "M A B" and "M B A" – which would map port A outputs to be driven by port B inputs and port B outputs to be driven by port A inputs.

Figure 2 below shows a simple interconnection, and the required mapping, using the Simple Parallel Port (SPP) adapter connecting a single PC to two different printers as an example.

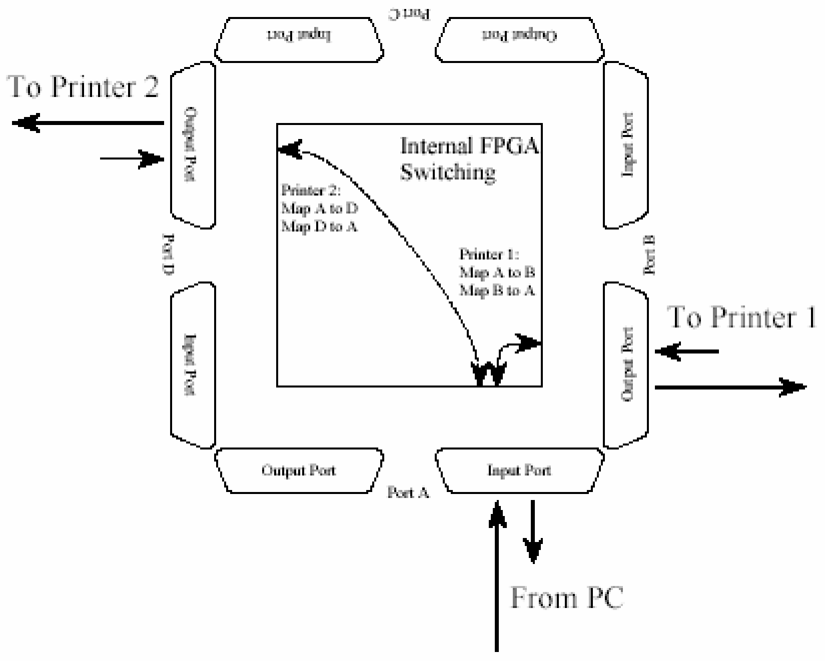


Figure 2 Simple Interconnections between ports

Complex Mapping

The system can support a more complex form of mapping where the signal path in one direction is different from that in another. Figure 3 below shows a somewhat contrived example of this. Such a scheme may be useful in some fail-safe or data sniffing applications.

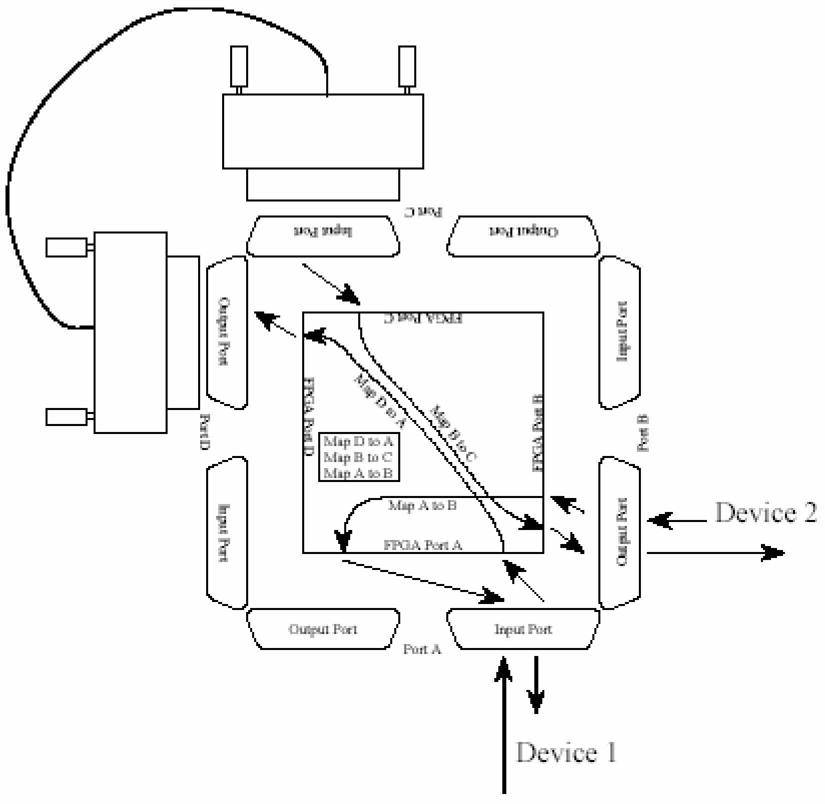


Figure 3 Complex Mapping

Revision History

Date	Version No.	Revision
30-Jan-2004	1.0	New product release
09-Nov-2004	1.1	File references updated to match updated design files
27-Feb-2008	2.0	Updated for Altium Designer Summer 08