Table of contents

Preface

This documentation is created for a FORK project as a part of a M.Sc.(Eng) in Embedded Systems at the University of Southern Denmark (SDU), Odense, Denmark, autumn 2007.
The FORK project is a 30 ECTS research-project made in preparation for a 30 ECTS master thesis, resulting in a total timeframe of one year (60 ECTS) for FORK and thesis. In this particular case, the 5 ECTS course SSE01 has been taken parallel with the FORK project, thereby restricting the project to only 25 ECTS.
It is assumed that the reader is familiar with FPGAs, and has a general knowledge of electronics and the inner workings of computers and operating systems. An introduction to PCI Express is included as an appendix for readers that are not familiar with this interface.
The project documentation is organized as a short main document, five journals and five applicationguides. This main document summarizes the process of the project, and includes objectives and conclusions for each of the journals. The journals each describe in detail one of the larger tasks in the project. The applicationguides give a quick introduction to each of the example applications created.

Terms and typographical conventions

Throughout the documentation a few terms are commonly used:

- *The system:* Term used in this project to describe the complete, functional combination of an FPGA connected to a PC through a high-speed interface, and a device-driver allowing the PC to communicate with the FPGA.
- *The board:* The Spartan3 PCI Express Starter kit board.
- *The host system:* The PC in which the Spartan3 PCI Express Starter kit is installed.
- *The hardware / BAR interface:* The interface in the FPGA between the PCI Express core and the user-defined hardware modules.
- *The software interface:* The device driver used on the host system.

The following typographical conventions are used:

- *Footnotes:* Footnotes are used for explaining terms, and for linking to sources. Each is marked with a small superscript number[1], referring to a footnote on the current page.
- *Excerpts:* The main document uses excerpts from the journals. These are marked with a large square bracket to the left of the text:

  Excerpt from a journal.

- *Italics:* Italics are used for the following cases:
  - Keywords / headlines in listings (for an example see this list).
  - In text: Words and terms that correspond directly to an object or a function in code, or to a file or application:

    Use the *printf()* function in the *test.cpp* file.

[1] A footnote example.

1 Introduction

During recent years field programmable gate arrays (FPGAs) have gained in popularity. This is both due to their lower pricing and improved capabilities and speed, but also due to their flexibility, which provides system-designers with all new possibilities. Together with the parallel structure of the FPGA, this flexibility makes FPGAs interesting when it comes to accelerating various forms of specialized computation, which are otherwise typically performed on standard PCs. Certain parallelizable operations (for instance mask operations in image processing) can achieve large speed-gains with a parallel implementation in hardware, compared to a sequential software implementation running on a general purpose microprocessor. Of course, in theory this has always been possible with ASICs, but the flexibility of the FPGAs has also made this economically feasible even for small series.
To really take advantage of these possibilities, the possible bottleneck in the interface between FPGA and PC needs to be eliminated, both with regard to speed and usability. One way to do this could be to use the PCI Express interface, which incorporates high transfer speeds and reliability, while still being relatively simple to use from various applications. The possibilities of this will be investigated throughout this project.

1.1 Overall objective

This FORK project is done in preparation for a master thesis project involving FPGAs connected to a PC using PCI Express. The overall objective of this FORK project is to research the possibilities of such a system, and to determine exactly which area(s) to work with during the master thesis.
Additionally, to be able to focus on the main area of interest during the master thesis, and to provide SDU with a working FPGA/PCI Express platform to be used in current and future projects, the FORK project should also result in a usable and working interface between hardware modules in the FPGA and applications running on the PC.

Desired results:
- A specification and list of objectives for the master thesis
- A generally usable and working interface between the FPGA and PC-applications

1.2 Tasks

From the objectives, the following tasks can be found:

- *Research:* An investigation will be conducted to find out if any current projects or ideas at SDU could benefit from the system, and to determine the requirements those projects would impose on the system.
- *Choice of hardware:* A suitable FPGA board has to be found for the project.
- *Interface-specification:* In cooperation with the target audience determined during the research phase, the various interfaces will be specified.
- *Interface-creation:* The specified interfaces will need to be created.
- *Documentation:* As the intention is that the system should be usable and accessible for others also, it is important that it is well-documented.
- *Problem statement for master thesis:* During the project, the area of work for the master thesis should be decided upon, and a problem statement drawn up.

2 Research

An investigation on the possible uses of the system has been conducted at SDU. The complete investigation can be seen in journal 1, and is summed up below.

2.1 Objectives

The overall objective is to investigate the possible uses of an FPGA-board in a PC. This is done to lay out the further course of the project, and to constrain it with regard to functionality, operating systems and similar. The investigation is done mainly on project-ideas and projects that are already running at SDU.
The following four possible uses have been found:

- Multipurpose hardware interface
- Network interface card
- Vision
- Collision detection

2.2 Conclusion

Four cases have been investigated. These concern using the FPGA board as a multipurpose hardware interface, as a network interface, and using it for processing vision algorithms and collision detection. From these it can be seen that there are some rather varying requirements for the system, depending on the application. The following can be summed up:

- The software interface (between FPGA and applications on the host PC) and the hardware interface (inside the FPGA) have to be flexible, and need to be designed to allow for easy add-ons and modifications at a later time.
- The interfaces need to be well-documented and easy to use.
- The system must be able to handle large amounts of data being transferred each way at high speeds. The target is about 180 MB/s each way.
- The system needs to be able to store 100 MB of data.
- The system needs to work on both Windows and Linux machines.

3 Choice of hardware

It was quickly decided to buy a readymade FPGA-board with a PCI Express interface for the project. This was done to avoid spending too much time with high-speed hardware design.
For the FPGA-chip, a Xilinx chip is chosen. This is done as chips from this manufacturer are already in wide use across SDU, and so projects and substantial knowledge on these already exists.
Xilinx have a few PCI Express based FPGA development boards available. Two of those are described below.

- *Xilinx Spartan3 PCI Express Starter Kit*
  Price: ~349 USD, http://www.xilinx.com/products/devkits/HW-S3PCIE-DK-PROMO1.htm
  This kit features a Spartan3 XC3S1000 FPGA, 128 MB onboard DDR SDRAM, and a PCI Express 1x interface. Additionally, there are a number of user pushbuttons, leds, and external interfaces (VGA, RS232, EXP connectors). The card has a relatively low price, and Spartan3 based kits are commonly used at SDU, resulting in a lot of available experience.
  The Spartan3 lacks some of the more advanced FPGA features available though, like being able to reprogram itself (unlike the Virtex family, there is no internal configuration access port (ICAP)), support for PowerPC soft-cores, etc.
  However, it is cheap, and could be a good place to start to achieve some experience with PCI Express based devices.

- *Xilinx Virtex5 LXT ML555 FPGA Development Kit*
  Price: ~2200 USD, http://www.xilinx.com/products/devkits/HW-V5-ML555-G.htm
  This kit features a Virtex5 LXT XC5VLX50T-1FF1136 FPGA, a RAM socket with support for standard SODIMM DDR2 modules and a large amount of various external interfaces (USB, SATA, etc). It also has both a PCI Express x8 interface, and a PCI / PCI-X interface.
  The Virtex5 has a lot of features, including support for the aforementioned Power-PC cores, self-reconfiguration through ICAP, and many more.
  All this comes at a price though, and the kit is much more expensive than the Spartan3 Starter Kit.

The Xilinx Spartan3 PCI Express Starter Kit is chosen as a starting point, as this has the necessary features to demonstrate the principles of PCI Express. A Virtex5 kit can be acquired at a later point, if the features of this are needed.
For testing, a Dell Optiplex GX280 PC is used. The specifications are:

- Intel Pentium 4 (Prescott) running at 3 GHz (HyperThreading[1] enabled)
- Intel i915P/i915G based, Dell brand motherboard
- 1024 MB DDR2 RAM
- SATA harddrive
- 1 PCI Express x1 slot, 1 PCI Express x16 slot, 3 PCI slots
- Xilinx USB programming cable

The software used is:

- Dualboot Windows XP SP2 / Slackware Linux 12.0 (using a 2.6 kernel and kde)
- Xilinx 9.2 (on Windows) with all current updates (nov. 2007 – PIPE 1.7)
- Windows Driver Kit v.6001 / GCC (for building Windows / Linux drivers)
- Dev C++ editor / KWrite (code editor on Windows / Linux)
- MingW / GCC (application compiler on Windows / Linux)

The Windows Driver Kit can be found on CD2, or the newest version can be obtained from the Microsoft Connect[2] program (requires a free Microsoft Live or Passport login). A Dev C++ installation, including MingW, can be found in the *Tools* folder on CD1. GCC and KWrite are included in the standard Slackware installation (and in most other Linux installation also).

[1] More info: http://www.intel.com/technology/platform-technology/hyper-threading and http://en.wikipedia.org/wiki/Hyper-threading
[2] See http://www.microsoft.com/whdc/DevTools/WDK on how to obtain the Windows Driver Kit.

4 Creation of FPGA designs

After having decided on an FPGA board, some designs need to be created for it. This is further described in journal 2, and is summed up below.

4.1 Objectives

The main objective is to create an FPGA design that supports the PCI Express interface, and features a usable, generic interface for connecting this to other hardware modules inside the FPGA. The design should also be created with the requirements regarding transfer speed and storage possibilities specified in journal 1 in mind.
Additionally, it could be interesting to demonstrate some of the possibilities of the Spartan3 PCI Express Starter kit, such as usage of the onboard DDR RAM memory and the VGA output port.

4.2 Conclusion

An FPGA design structure with a working PCI Express interface and an interface for hardware modules has now been created. Up to six independent hardware modules are supported at a time, through a multiplexed bus. The Xilinx Endpoint PIPE for PCI Express IPcore is used, which implements a complete PCI Express protocol.
Using this structure, three designs have been created – a base design, *Empty*, without hardware modules, to be used as a starting point for new designs, the *IOControlDemo* with modules implemented to control the onboard leds and buttons and an 8 kB blockram, and the *VGADemo* design demonstrating the use of the VGA port. The possibilities of using the onboard DDR RAM memory for storage have also been investigated, and a test design, *DDRControllerTest*, has been created. This has been brought up and running, and can successfully access the onboard DDR memory. It is not implemented together with the PCI Express interface though. All the designs are further described in the applicationguides.
The performance will be evaluated when a driver has been created.

5 Creation of drivers and applications

The creation of drivers and applications are described in detail in journal 3. The objectives and conclusion of the journal are listed below.

5.1 Objectives

To use the Spartan3 PCI Express Starter kit, some device drivers are needed. The objective of this journal is to describe the creation of such drivers for both Windows XP and Linux 2.6 that allows other applications to read from and write to registers on the board.

5.2 Conclusion

Drivers and simple applications for both Windows XP and Linux 2.6 have now been created. The functionality is similar on both platforms. Each provides a way of writing or reading a single DWORD register on the board. This can be done from a user-application through very similar calls, *DeviceIoControl()* on Windows, and *ioctl()* on Linux.
This has been tested, and the functionality works on both platforms.

6 Performance test

Journal 4 details a test of the performance of the drivers created in journal 3. The objectives and conclusion are summed up below.

6.1 Objectives

The performance of the created set of drivers and applications for both Windows and Linux needs to be tested. This will be done on both platforms under various environments.

6.2 Conclusion

The performance of the created drivers and applications has now been tested. As there is a substantial amount of overhead present in both the host OS and in the PCI Express transfer itself, the performance is far away from the theoretical maximum of 250 MB/s. By using the driver-controlled approach and thus minimizing the overhead in the host OS, a best-case transfer speed of ~30 MB/s when writing to and ~2.5 MB/s when reading from the board has been achieved on both platforms. This only works for large transfers where all data is transferred to/from the driver in one request. When transferring single DWORDS, the additional overhead of calling the driver has a higher influence, and drops the transfer speeds to well below 1 MB/s in both directions on Windows, and to around 7 MB/s writing and 1.8 MB/s reading on Linux.
Sadly this is far away from the desired transfer speed of 180 MB/s. It should therefore be investigated if it is possible to improve on the current performance.

7 Improving performance

Journal 5 discusses a number of ways to improve the performance of the drivers, and describes the implementation of one of these. The journal is summed up below.

7.1 Objectives

As seen in the performance test, the current transfer speeds of the system are far away from the desired. The lack of speed is most probably due to excessive overhead in the type of transfer currently used, where driver requests are used to read and write single registers.
Alternatives could be:

- Transfer arrays of data to the driver and let the driver transfer several DWORDs per request.
- Map the device memory/registers to the user space, to allow for direct access without going through the driver.
- Use bus mastering/DMA to let the board perform the read/write operations.

7.2 Conclusion

Three ways of improving the transfer speeds have now been investigated. One of these, mapping device memory to user space, has been implemented. This effectively removes the overhead present in the host OS when performing a driver request, as now only two requests are needed – one to map the memory, and one to unmap it when done. This makes sure that the transfer speed for all sizes of transfers, is very close to the best-case 30 MB/s / 2.5 MB/s (write / read respectively) achieved in the previous performance test in journal 4.
To further improve on this performance, DMA functionality will need to be implemented. This has some good potential, as it will allow the PCI Express transactions to have a data payload of more than 1 DWORD, while also offloading the CPU in the host system. As this will require large modifications to both the FPGA design and the drivers, it is left to a future project to implement and prove.

8 Problem statement for master thesis

Together with the advisor on the project, Anders S. Sørensen, the following problem statement for the master thesis has been agreed upon. It is based on the network interface card project researched in journal 1. The full problem statement including requirements, tasks and a preliminary time-schedule can be found in the appendices.

> The master thesis will concern the design and development of a network system for use with distributed robot-controllers and similar setups.
> The idea is that each separate part of the robot-controller is connected to the network through a "node". A node provides a number of registers to the robot-controller, and automatically and reliably mirrors these across the other nodes in the network. The nodes should be self-contained, and consist of a network interface, an interface for the registers, an FPGA, and the necessary glue components. An additional interface for subnodes, for example SPI to a slave-part of the robot-controller, can be implemented if time allows for it.
> To avoid spending too much time on board design, an existing FPGA-board (such as a Zefant board) will be used.
> During the FORK project, the possible uses of an FPGA-board with a PCI Express interface have been investigated. To build on the FORK project, the used FPGA-board will be implemented as a kind of optional master-node, with additional control- and supervision-functionality to be used through a PC system.
> The main tasks of the project will be the boarddesign, and the design and implementation of a reliable network protocol and registers in an FPGA.

9 Discussion

It was quickly discovered that the project included some rather large and extensive topics (PCI Express and device driver development for instance) which have taken some time to get familiar with. Due to this there has only been time to "scratch the surface" with regards to some of the more advanced possibilities of the system, such as bus mastering. Hopefully this project can provide a good starting point for future work on the subjects though.
A number of issues have also obstructed the project. Amongst others could be mentioned that all Xilinx projects that use the PIPE IPcore take at least 10 minutes to synthesize and implement. This made the initial experimentations very time-consuming. Also, the fact that an error in a Windows device driver will usually crash the whole system, meant that the driver debugging process became rather tedious too.

10 Conclusion

The objectives of this FORK project was to arrive at a problem statement for a master thesis involving a FPGA connected to a PC through a PCI Express interface, and to create working hard- and software-interfaces for this. Both these overall goals have been achieved.
During the project an investigation was conducted to find projects at SDU, where the described system could be used. This helped set some of the goals and requirements for the interface. Additionally one of these projects, the network interface card, has been chosen for the master thesis.
The hard- and software-interfaces have been created for use in the master thesis, but also with usage for other projects at SDU in mind. Due to this, some work has been put into creating thorough documentation and working demo-applications.

10.1 Evaluation of system requirements

The requirements specified for the system during the research phase in journal 1, can now be compared to the achieved results:

- The software interface (between FPGA and applications on the host PC) and the hardware interface (inside the FPGA) have to be flexible, and need to be designed to allow for easy add-ons and modifications at a later time.
- The interfaces need to be well-documented and easy to use.
- The system needs to work on both Windows and Linux machines.

This has been achieved. A hardware-interface inside the FPGA and a software-interface for a host PC have been created, and should provide flexibility enough for most uses. The hardware-interface uses the Endpoint PIPE for PCI Express IPcore, and supports up to six independent hardware modules at a time, through a multiplexed bus. The software-interface consists of device drivers with functionality for reading and writing 32 bit registers in the hardware modules, and for mapping the memory space of the hardware modules to a simple array in a user mode application. Drivers with this functionality have been created for both Windows and Linux.
The interfaces have been created with simplicity in mind, and the created documentation and example applications should make it easy to get started using the system.

- The system must be able to handle large amounts of data being transferred each way at high speeds. The target is about 180 MB/s each way.

This has not been achieved. The maximum transfer speed is 30 MB/s when writing to the board, and about 2.5 MB/s when reading from the board. Theoretically, the desired target speed should be obtainable if bus mastering is implemented. This should make it possible for the FPGA to carry out transfers by itself, without the restrictions of the x86 architecture (which restricts the payload of each transfer to 1 DWORD) and of a multitasking operating system (which imposes extra overhead during driver requests).

- The system needs to be able to store 100 MB of data.

This has been partially achieved. A test design that provides access to the 128 MB of DDR SDRAM on the Spartan3 PCI Express Starter kit has been brought up and running, but not yet together with the PCI Express functionality.

10.2 Future work and perspectives

As stated throughout the report, there are a number of things that could be done to improve the functionality and performance of the system.

- *Bus mastering / DMA:* Implementing bus mastering should give a large performance boost. In theory, transfer speeds above 200 MB/s in both directions should be possible for "large" transfers, where the amount of overhead used to set up the transfer can be neglected. This will require some substantial changes to the PIO module in the FPGA to enable transfers with payloads larger than 1 DWORD, and to enable the use of interrupts. Support for this in the device drivers will also need to be added.
- *DDR SDRAM controller:* The DDR SDRAM controller could be integrated with the PCI Express interface in the FPGA design.
- *Advanced functionality:* Some of the more advanced features supported in PCI Express could be implemented. For instance delayed responses, power saving and similar.

There are many possible applications for the system. Both for projects that can take advantage of the parallel nature of the FPGA to accelerate certain operations and calculations, and for projects that can use the system to interface external hardware to a PC. In these cases the FPGA can be used to adapt whatever interface might be used on the external hardware, to the memory/register interface used for the system.