A FPGA-based PCI bus interface for real-time log-polar image processing system*

Francisco J. Blasco Fernando Pardo Jose A. Boluda
Dpto. Informática y Electrónica - Universitat de València
C/ Dr. Moliner 50, 46100 Burjassot (Valencia), SPAIN
Email: (Francisco.J.Blasco, Fernando.Pardo, Jose.A.Boluda)@uv.es
Fax: (+34) 963160418 Ph: (+34) 963160411

Abstract

This paper deals about the development of a PCI bus interface for a log-polar image acquisition system. The characteristics of the log-polar camera employed in our system, and its interface to a real-time image processing system, bind an ad-hoc solution that is difficult to achieve with any general purpose image acquisition system. Our main objective was to develop a card allowing log-polar camera image acquisition, camera and image processing system intercommunication, and PC user and other systems interface. The card has a PCI bus interface designed using a complex programmable logic device that holds the interface and other logic control for the system in the same chip.

1 Introduction

At this moment, there are many real-time image-processing frame-grabbers in the market. Usually, most of these systems obtain a sequence of images from a CCD camera; the storage and processing of such amount of information is many times complex and requires extra hardware, especially for real-time applications where fast processing is the key.

With the objective of minimizing the hardware requirements, and speed-up the image processing tasks, among other advantages, a camera based on a log-polar sensor [1, 2] and an In System Reconfigurable FPGA-based real-time image processing system ([3, 4, 5]) have been developed.

The log-polar camera pixel distribution imitates the distribution of the retina photoreceptors in the human eye; there is higher pixel density in the center than in the periphery, using a polar and logarithmic distribution of pixels. One image taken with this log-polar camera has around the same resolution of a 700x700 CCD camera in the center, decreasing toward the periphery (less interesting part) to reduce the amount of data to be processed.

It implies that the use of the camera is better suited for active vision applications, where the camera is dynamically guided to the interesting object. When the object is centered, it fills the area with higher resolution. However, a lesser information is obtained from the object periphery due to the smaller resolution of that area. The result is a 10 KB image that offers similar information as a 256 KB image, since it selectively chooses to have better resolution in the interesting part.

The conventional architecture formed by a single processor and single memory is usually not enough for fast image processing. Other architectures exploiting the inherent parallelism of image data have been proposed in the past. Apart from the realization of the log-polar camera, an image processing system to specifically process these kind of images, has been developed. It is basically based on a multistage pipeline of image processing elements with local memory. Images from the camera are sent to the processing pipeline and this one returns the results to the host computer where higher level image processing will take place. Input and output data to/from the image processing pipeline should be controlled by a specific hardware from the PC (host computer). At the same time, this hardware should also control the log-polar camera for driving the clock signals to read and store the images. The interconnections among the camera, processing pipeline and the host PC through the PCI bus form the system presented in this paper.

The log-polar camera has a maximum throughput of 2 MB/s, that is similar to the data exchange requirements of the image processing pipeline. These performance requirements suggest a control system interface able to have high-speed communication with the PC.

The readiness and the high performance of the PCI bus have been the main issues in order to select it as the best candidate. This bus has been developed to eliminate the classical bottlenecks that take place in the PC I/O transfers and it has become the adopted standard for the x86 and Alpha AXP processor family platforms.

The PCI bus is a general local bus [6] that can be employed in monoprocessor as well as multiprocessor systems, with high I/O bandwidths and/or time constraints. These are the main features of this bus:

*This work has been supported by the Spanish government (CICYT project TIC98-1026) and the Generalitat Valenciana project (GV97-TI-05-27)

- *Configurable implementations.* PCI bus interfaces are able to implement a wide range of the features necessary for an application; other features not necessary for a specific application may be discarded to minimize hardware requirements.
- *High performance.* It works from 32-bit data path at 33 MHz (132 MB/s peak) to 64-bit data path at 66 MHz (528 MB/s peak). It is also capable of full concurrency with processor/memory subsystems.
- *Easy use.* Full auto configuration. Devices contain information registers for device configuration.
- *Low cost.* It has been optimized for direct silicon interconnections. Electrical/driver and frequency specifications are met with standard ASIC technologies and other typical processes.

Most programmable logic device manufacturers have designed logic devices that are PCI compliant. These devices work with electric, time and frequency PCI specifications. FPGA register capacity allows the implementation of PCI cores that can be employed to design PCI devices. FPGA devices allow the reconfigurability of PCI interface implementations. The idea is very simple: the FPGA and the PCI bus form the physical part, while the program stored in the FPGA is the firmware, that can be upgraded in every revision of the HDL (Hardware Description Language) description of the interface. There are some commercial solutions [6],[7],[8], but they are expensive for making small prototyping designs.
The speed requirements and camera and image processing control complexity impose a system that must be reconfigured in order to solve possible bugs, without the need of repeating the process of design-simulation-fabrication-back again.

2 System architecture

The overall system is shown in figure 1. The card is connected to three devices:

- *Log-polar camera.* It is autonomous. The camera sends the image as a block through the data link that connects the camera and the acquisition card. This link is also employed to program the camera configuration registers (latency, speed, windowing, etc.). With these features the camera frame rate can be adjusted to the system requirements.
- *Image processing system.* This is the set of processing elements of the pipelined architecture. The input data bus to the pipeline is fed with the camera data and the output returns processing data that must be stored. Also, the system process could be reconfigured through a specific bus designed for this function. The acquisition card manages all the interfaces of this system, and it can also work just as a frame-grabber bypassing the pipeline of processing elements.
- *PCI acquisition card.* The acquisition card is connected to the computer PCI bus. It has an internal memory buffer for storing images from the camera and results from the processing elements. It also serves to allow the computer programs to read these images and results.

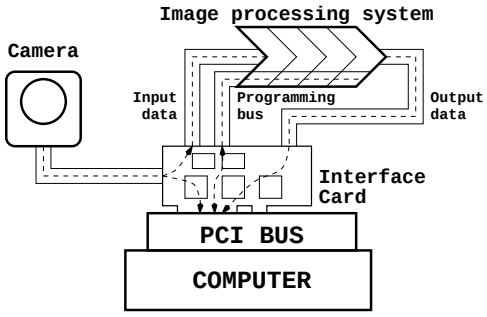

Figure 1: System architecture

The image acquisition card performs the following tasks:

1. Reading of the camera sent images and storage of these images to the card local memory.
2. Sending of the stored images to the image processing system, collecting of the output data and store it in a local memory.
3. Reading of the local memory (images + output data) through the PCI bus.
4. Controlling of the programming bus of the image processing system.

3 Image acquisition card architecture

The card architecture implemented is shown in figure 2 and the card after manufacturing is shown in figure 3. All programmable logic has been programmed using three devices. Two of them (namely A and B) are an EPF8820A-3 from ALTERA, and the other (device C) is an EPM7096LC84-12 from ALTERA too. It has not been an arbitrary decision to select these devices. We had to share these resources with others projects having a trade-off between global cost and number of logic devices.

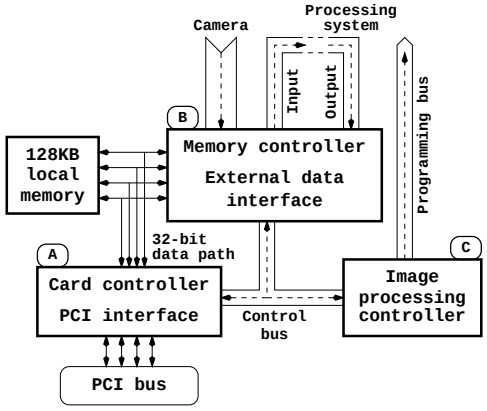

Figure 2: Card architecture

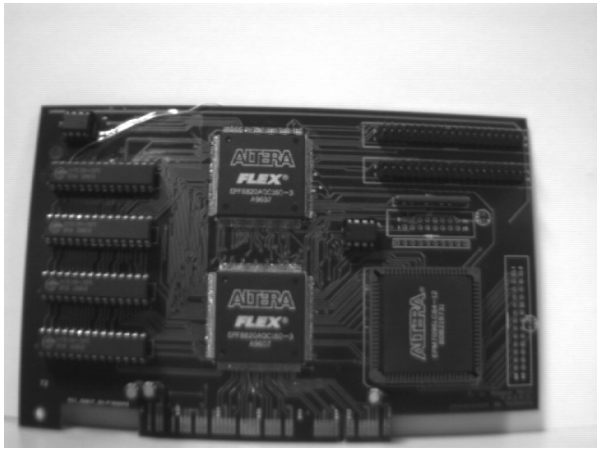

Figure 3: Acquisition card

The whole functionality was divided among the devices to use trying to have the best partition we could achieve. After programming these devices were not used at their maximum capacity, but it is good since it provides enough flexibility to be reprogrammed. Each device realize the following tasks:

- *Device A*. This is the PCI interface and card controller. It has the PCI interface logic, explained later, and the control of other card devices. Due to performance needs and pin number constrains, it is connected to the memory data path and it has a two-word memory buffer. A control bus transmits all needed information among the devices. This design use 58% of the EPF8820A-3 device capacity (398 logic cells for all logic, 228 logic cells for the PCI interface logic).
- *Device B*. This device is the memory controller, the camera data interface and the data processing system interface. The local memory is formed by four single 32 KB SRAM. It is organized into 32K words of 32 bits. The memory controller has a smart time-slicing algorithm to divide the memory time into PCI reads, camera writes (image storage) and processing system reads (images sent to the system) and writes (output data). The architecture is shown in figure 4.
  Each interface has a different protocol to communicate with its own device. All interface buses have an 8-bit data path, but each interface has a 32-bit buffer. This last one reduces the total number of memory accesses by a factor of 4. The PCI interface is connected directly to the memory, and some control signals indicate when this device must read the data bus. The time-slice algorithm leaves one out of two clock data cycles for the PCI interface without losing time requirements, and with a PCI peak throughput of 66 MB/s. This design use 62% of the EPF8820A-3 device capacity (421 logic cells).
- *Device C*. This device is the simplest one. Its task is to collect the received data through the card control bus and translate it to the programming bus protocol of the processing system. For this reason, it is enough an EPM7096LC84-12 at 63% of its total capacity (61 logic cells).

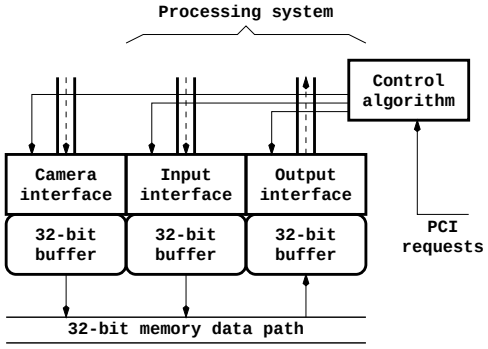

Figure 4: Memory controller

4 PCI interface

This section describes the PCI interface implemented for the card. It is a simple interface that can be employed as a PCI core base for target devices. It was designed using Altera HDL because the design was described at RTL level, and we checked that the logic synthesis was better using AHDL than VHDL. This model was available through the web, but now, also a VHDL version is available. We ported the original description to VHDL to make it freely available; the advantages of VHDL is that it can be employed to synthesize almost any programmable device of the market thanks to the tools availability and standardization. In the near future, a more refined VHDL version will permit on-line tailoring for the specific needs of most applications.

4.1 Features

- PCI V2.1 compliant 32-bit, 33 MHz target.
- It was optimized for Altera FLEX architecture, but it can be ported to other devices.
- The back-end functionality can be modified.
- Supported target functions:
  - Type 0 configuration space header, with specific device configuration registers.
  - One base address register, slow decode speed. This feature will change very soon to support up to 6 BARs.
  - Parity generation (PAR) is supported (it is necessary). Parity error detection pins are not implemented. This interface is defined for small applications that do not usually need these pins.
  - The supported commands are Memory Read, Configuration Read, Configuration Write and Interrupt Acknowledge. There is no really a need for others commands, though it is easy

to upgrade them.
  - 32-bit burst transfers with linear address ordering.
  - Target Disconnect and Target Abort.
  - The Command/Status register is defined for the functionality of the back-end device.
- **It is now freely available on the web**: http://tapec.uv.es/research/PCI

4.2 Applications

This PCI interface is defined for small applications. The implementation of this interface has been done in a small and not very fast device. The initial idea was to have a little free interface for student designs or small PCI cards. Obviously, it can be upgraded with more features. Current work is the creation of a VHDL design for a target device with all target features available. All this work can be accessed through the web, our future work is creating a web page where anybody could configure a target device automatically generating the VHDL source code.

4.3 Interface implementation

In figure 5 the device A architecture is shown. The card control and the memory interface are the logic connected to the PCI back-end. All remaining blocks comprise the PCI interface logic.

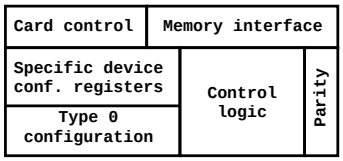

Figure 5: Device A architecture

This interface comprise these four blocks:

- *Logic control*: This block holds the state machines that controls all interface signals and the global working interface.
- *Parity*: This is the parity generator logic. This part is independent of the rest of the circuits and is continuously running.
- *Type 0 configuration header*: This block generates the standard type 0 configuration space header. Some PCI interface implementations do not generate this block, and it remains connected to an external memory through the PCI back-end. We decide to generate hard-wired logic and to reduce the pin number of the device.
- *Specific device configuration registers*: Some control registers are defined on the configuration space for device configuration. Since this control registers are only accessed during the initial steps of the acquisition card, they are located at this space and it is not necessary the use of another BAR register or the use of I/O space commands.

This implementation is specially oriented to the physical device employed in the design. All information about FLEX architecture [9],[10] can be obtained from the web page *http://www.altera.com*. The logic description has been specifically targeted to the FLEX logic cell that has one register and four input LUTs for combinatorial purposes.

4.3.1 Parity generation

The logic circuit for the parity generation and check is the simplest of the designed blocks. Nevertheless, if the design of this circuit is made general, without thinking on the specific architecture, the result can occupy a lot of space in the device. Parity calculation has been designed using XOR gates, if the device logic do not have a special treatment for this kind of gates, the resulting logic could be bigger. If inputs are grouped in sets of four, each of these groups parity is calculated in one logic cell. The logic circuit has been designed to have the parity result in two clock cycles. In the first cycle inputs are grouped in sets of 4 and the partial result is stored in an intermediate register. For the next cycle, the results stored in these registers are grouped in sets of 4 again obtaining in this way the final result. Using a natural way of describing parity calculation (without thinking of different stages, just putting the XOR at all inputs) the number of logic cells is two or three times larger than the proposed solution, that is also faster. We decided not to use the parity error signaling pins since a data error just implies an incorrect read image. All this logic is running concurrently along with the other logic.

4.3.2 Type 0 configuration space header

The type 0 header of the PCI configuration space is defined as 32-bits 16 words and must be implemented at any PCI compliant device. There are some of these registers (say the manufacturer or device identification) that must be implemented obligatorily. Nevertheless, most of the other registers are optional depending on the target application. For our design, the number of implemented registers has been the minimum required for the application since the space left on the programmable device was not very high. This entire header has been implemented in 64 logic cells. All implemented registers are shown in figure 6.
All the configuration space has been implemented using registers and wired logic due to the specific programmable device employed. Every read/write register (Interrupt Line and Base Address Register 0) have been implemented with D-type registers, while the remaining registers have been designed using gate-logic. Figure 7 shows the type 0 header design schematic of the configuration space.
We only need one base address register because we only have one memory address range. If one wants to use more BARs there is an increasing of the decode address logic, and this can be a critical time path. The

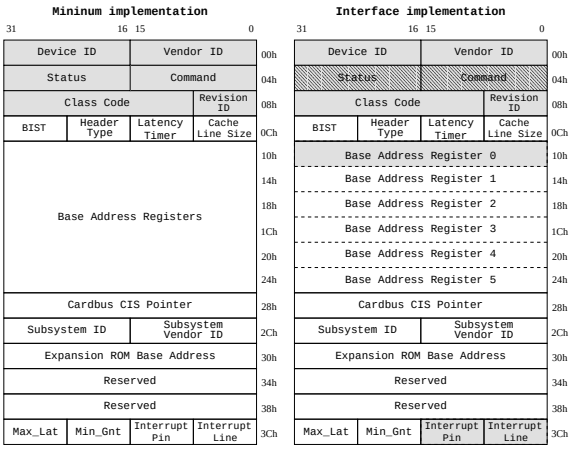

Figure 6: Type 0 configuration space header implementation

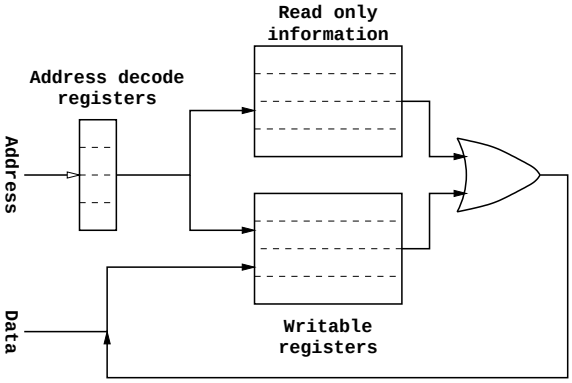

Figure 7: Type 0 configuration space header implementation

status register and the **command** register do not implement all their bits. They only implement the basic target control bits. The **Interrupt Pin** register and the **Interrupt Line** register are not the basic implementation, but we need they to signal the image read completion.

4.3.3 Specific device configuration registers

Apart from the standard registers shown so far, other registers comprise the configuration space. The PCI specification says that the configuration space is 256 bytes wide; the first 64 bytes, in the type 0 header, are predefined as standard, but the following 192 may be employed by the designer to introduce specific registers for his specific application. In this case, this space has been employed to accommodate the configuration registers for the log-polar camera and the pipeline of image processing elements. The implementation of this circuitry is similar to the head of the configuration space, maintaining these two areas separated to adapt this same design to other applications or interfaces. The recommendation says that this is not the best way to implement this registers, but if we want to introduce another BAR to create a memory segment with this control registers, the decoding logic can be out of timing specifications. The use of IO space commands implies the use of byte swapping logic. The capacity of our devices are little and we decided it was preferable a smaller interface.

4.3.4 Logic control

This part of the total circuitry is what controls the whole interface. Basically its main component is a finite state machine than controls the data flow through the different components that comprises the image acquisition card. In figure 8 it is possible to see a simplified view of this subsystem.

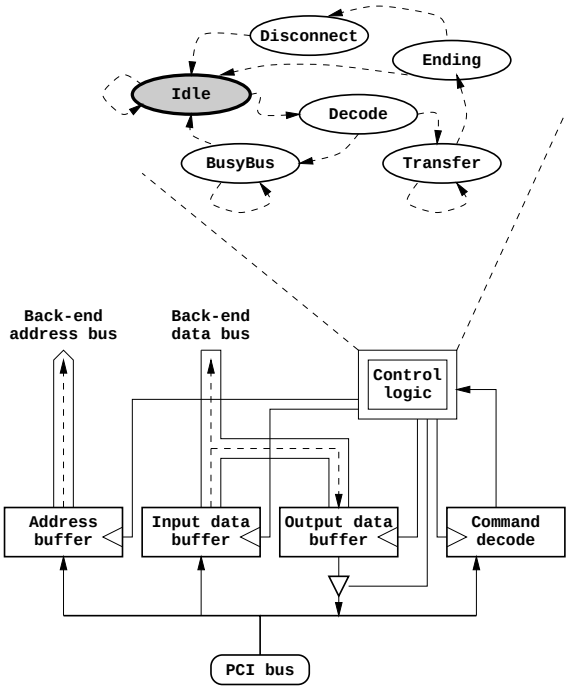

Figure 8: PCI interface control

The state machine has six states. When the device is doing nothing the state machine is in the **Idle** state if there is nothing going on the bus, or **BusyBus** if there is a transaction in the bus but it is being attended by other device. During the **Decode** state the address decoding is carried out. For speed reasons this task is performed in one clock cycle since the decoding logic is designed in two stages (two clock cycles). In the state **Transfer** is where the data transfer takes place. Finally, the states **Ending** and **Disconnect** are employed to finish the device bus operations.
The state machine controls the device buffers. These buffers store the input data and address, and also store the output data. The PCI configuration space circuitry connects the interface back-end. In the same way, the PCI interface is connected to the rest of the frame-grabber. The result of this has been the modularity of

the design specification, although there is a loose in the idea of having an interface to the back-end since the connection logic to the back-end and part of the PCI interface are very tightthe advantage is the reduction of the logic. In future VHDL versions this has been taken into account and the interface specification will be more general and independent.
In the logic control there are others state machines that control the generation of the output signals. It is good to do it this way since output delays are an important constrain in the PCI specification, and if each output was controlled by a particular state machine, the synthesized logic cells could be closer to the outputs. In fact, all time constrain problems can be eliminated using a faster device, but if our design works for a slower device, surely it will also work for a faster one.

4.3.5 Some commercial bus interfaces implementations

There are bus interfaces implementations from many vendors. Companies like ALTERA, Xilinx, Actel and others have now some cheap bus interfaces implementations. There are cheap but not yet freely available. Possibly, theirs products are more sophisticated and better. Third party companies like Eureka Technology or PLD Applications have a wide variety of products. When this project begun, the number of these products was little, and they were more expensive.
Another approach for our frame-grabber implementation was the use of generic PCI interfaces. There were several reasons why we finally implemented the PCI interface in the card using a programmable logic device. In the one hand, we just needed few of these cards so buying many of these "interface on a chip" devices was more expensive than using our stock of programmable devices. In the other hand, the use of programmable logic for the interface allowed us to implement the whole system with a few number of chips with the different elements of the architecture (camera, PCI, memory and processing elements) tight together. Finally, we also wanted to have our own PCI interface model for academic and future projects purposes.

5 Conclusions

The main objective of this work was to design and implement a simple PCI interface for a specific application. Using this first step, a more generic interface, freely available, has been designed to be employed in more generic purpose applications. There are commercial models to implement generic PCI interfaces using programmable logic, but these models result expensive if the target application is very specific or the objective is just to make a non commercial prototype. There are also integrated circuits that implements a simple PCI interface to same extend, this can be also a cheap solution for those that want to make a very high performance, big volume production, PCI card.

References

[1] F. Pardo, B. Dierickx, and D. Scheffer, "CMOS foveated image sensor: Signal scaling and small geometry effects," *IEEE Transactions on Electron Devices*, vol. 44, pp. 1731–1737, Oct. 1997.

[2] F. Pardo, B. Dierickx, and D. Scheffer, "Space-variant non-orthogonal structure CMOS image sensor design," *IEEE Journal of Solid State Circuits*, vol. 33, pp. 842–849, June 1998.

[3] J. Boluda, F. Pardo, and J. Pelechano, "Reconfigurable architecture for machine perception. an approach for autonomous vehicle navigation," in *Workshop on European Scientific and Industrial Collaboration on promoting*, (Girona, Spain), pp. 359–363, June 1998.

[4] F. Blasco, F. Pardo, and J. Boluda, "Sistema de adquisición de imágenes log-polares con alta velocidad basado en bus PCI," in *XIX Jornadas de Automática*, (Madrid, Spain), pp. 277–280, Sept. 1998.

[5] J. Boluda, F. Pardo, F. Blasco, and J. Pelechano, "Una arquitectura segmentada para el cálculo del tiempo al impacto con visión log-polar," in *XIX Jornadas de Automática*, (Madrid, Spain), pp. 281–285, Sept. 1998.

[6] PCI Special Interest Group, http://www.pcisig.com, *PCI Local Bus Specification - Revision 2.1*, June 1995.

[7] Altera Corporation, http://www.altera.com, *PCI Bus Target Interface Megafunction*. Solution Brief 25.

[8] Xilinx Inc., http://www.xilinx.com, *PCI32 SpartanXL Master & Slave Interface*. LogiCORE.

[9] Altera Corporation, http://www.altera.com, *FLEX 8000 Programmable Logic Device Family*. Datasheet.

[10] Altera Corporation, http://www.altera.com, *Configuring FLEX 8000 Devices*. Application Note 33.