



Universidad Autónoma  
de Madrid

**Biblos-e Archivo**  
Repositorio Institucional UAM

**Repositorio Institucional de la Universidad Autónoma de Madrid**

<https://repositorio.uam.es>

Esta es la **versión de autor** del artículo publicado en:  
This is an **author produced version** of a paper published in:

4th Southern Conference on Programmable Logic, Bariloche,  
Argentina, 2008

**DOI:** <https://doi.org/10.1109/SPL.2008.4547750>

**Copyright:** © 2008 IEEE

El acceso a la versión del editor puede requerir la suscripción del recurso

Access to the published version may require subscription

# OPEN AND RECONFIGURABLE SYSTEM ON CHIP ARCHITECTURE WITH HARDWARE AND SOFTWARE PREPROCESSING CAPABILITIES USED FOR REMOTE IMAGE ACQUISITION

*Ricardo Ribalda, Angel de Castro, Guillermo Glez-de-Rivera, Javier Garrido*

Escuela Politécnica Superior - UAM (Spain)

email:(ricardo.ribalda|angel.decastro|guillermo.gdrivera|javier.garrido)@uam.es

## ABSTRACT

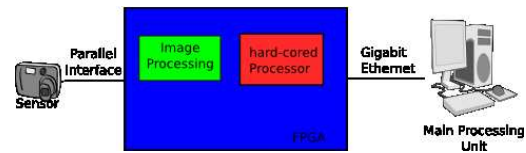
Most of the state of the art image acquisition systems include some kind of image preprocessing, but most of them are just focused on hardware or software preprocessing. This paper describes a working System-on-Chip architecture based on a state-of-the-art FPGA with both hardware and software pre-processing capabilities.

The architecture is based on a Virtex 4 FX FPGA and uses a hard-core PowerPC 405 processor for software processing and a Gigabit Ethernet connection for interfacing the Image Processing System. The whole system is governed by a port of the official branch of Linux Kernel, simplifying the customization of the architecture for specific scenarios.

## 1. INTRODUCTION

Most of present systems for image acquisition/processing use cameras just as sensors with very basic or no processing capabilities. This “classic” approach is enough for environments where the sensor is close to the main processing unit, the images are not very big (less than 1 Megapixel), the time requirements are low and/or there is a reduced set of cameras. When one of the above conditions is missaccomplished the classic approach is problematic.

If the sensor is far away from the main processing unit, the camera needs processing capabilities to send the images through an Ethernet connection or equivalent. Most of the commercial off-the-shelf Ethernet capable cameras act like “black boxes”, with no preprocessing capabilities, and some of them with severe security faults. Without preprocessing capabilities, these cameras must send all the images they capture. A small set of these cameras can collapse a medium network. If the sensor is close to the main processing unit, local interfaces like PCI, USB, Firewire, LVDS... can be used. Unfortunately this interfaces can’t move more than some hundred Megabits per seconds, some of them collapsing the processor. Once the images are located on the processor memory, these images are processed by the processor increasing the whole system latency. Preprocessing can



**Fig. 1.** Proposed Architecture

liberate the main processor and reduce the local interface bandwidth needed.

As shown hereafter, preprocessing is needed for high performance image acquisition systems. Preprocessing can be used to realize online algorithms, which are done during acquisition without external memory, or offline algorithms which are done after acquisition, using external memory.

Most of the state-of-the-art papers describe methods for this purpose which can be classified as hardware or software approaches.

Software approaches [1] consist on processors attached to sensors which pre-process the images and send them to the Main Process Unit (MPU). Beside this architecture reduce the bandwidth needed and MPU requirements, it is only suitable for slow cameras. Processors can not work with big flows of information because they have no way for paralleling the algorithms. Common image processing algorithms as edge detection may require a huge amount of clock cycles. In spite of these inconveniences, these systems can be programmed easily, and they are great for offline algorithms, even though its response is really bad on online algorithms.

Hardware approaches consist on ASICs or FPGAs [2, 3] attached to sensors. These FPGAs extract the images from the sensors, and send them to the MPU using PCI [2], USB [3].... This approach can work with big flows of information in parallel. It supports applying common image algorithms on the fly. A problem is that this kind of architectures must be designed ad-hoc for every different sensor, and programming it can be difficult. These systems are great for online algorithms but fail on offline algorithms.

In order to realize both online and offline algorithms a mixed hardware/software approach is needed. Due to the

big amount of data to move, a multiple chip system composed by an FPGA or an ASIC plus an external Processor is discouraged because it needs a powerful external interface between them. A System On Chip with silicon resources and an internal microprocessor can overcome this problem. First systems using this architecture used soft-core microprocessors [4]. These type of processors can not provide more than 100 MIPS, less than needed for a high performance real-time acquisition system.

Latest FPGA includes one or more hard-core processors which can achieve up to 600 MIPS. Although the use of this systems is becoming part of the state-of-the-art in high-end development network devices [5, 6], up to our knowledge, there are no real image acquisition systems that use this hard-wired processors although there are some descriptions [7, 8] which remark that the use of hard-core processors will increase the performance of the system. This paper describes a real and working image acquisition system with hardware and software preprocessing capabilities composed by a commercial sensor, plus a Virtex 4FX FPGA board with a hard-core processor and hard-core Gigabit network interface (Fig. 1). The whole system is governed by a port of the latest Linux Kernel.

## 2. ARCHITECTURE DESCRIPTION

The final architecture will be described in this section, bottom to top. And then the software that controls it.

### 2.1. Hardware

#### 2.1.1. Virtex-4 FX

A Virtex 4 FX has been used as the main device in the architecture. This family of FPGAs [9] completely fits the related scenarios. They are smaller than an Euro coin and are composed by:

- One or more PowerPC 405 hard-cores
- One or more TEMACs hard-cores
- Big size of re-configurable silicon area.

The PPC<sup>1</sup> is used for controlling the whole system. It can work in stand-alone mode (no Operating System) or governed by an Operating System like Linux. Although the PPC is less powerful than a brand-new processor, it can work together with peripherals to speed-up specific algorithms. The PPC internal architecture supports mapping external devices in the common memory.

TEMACs are used for connecting the System to a Gigabit Ethernet network, giving a bandwidth up to 1000 Mbps which is higher than the limit of USB or Firewire and enough

for high-quality cameras. These TEMACs can also work with more common Fast-Ethernet (10/100Mbps) networks if needed.

The reconfigurable silicon area is used for the rest sub-modules inside the FPGA like buses, connection with the camera and other peripherals. Mixing the power of the PPC and the reconfigurability of the silicon area, this system is ideal for controlling any camera.

#### 2.1.2. External Devices

Some external devices are needed in our architecture.

Because the FPGA has just a small amount of internal memory (called BRAM) which is not enough for our purposes, an external memory should be added to the FPGA. The FPGA is fast enough to control common out-of-the-shelf DDR RAM. Up to 4 GB of memory can be attached to the system.

This family of FPGAs needs also an external non-volatile memory to save the FPGA configuration. Although this memory can be used for other purposes it is usual to add another non-volatile memory save system-specific data.

TEMACs need an external chip for the physical layer. This chip should be able to work in Gigabit networks in order to achieve the higher possible speed.

The last and most important device to add to our system is the camera. For every different camera a data-extractor device must be programmed in the re-configurable silicon area.

#### 2.1.3. Buses & Specific Peripherals

The heart of the architecture is the PPC, but it needs a set of peripherals inside and outside the chip to operate. These peripherals must be added to the processor memory space. The PPC has two buses for this purpose: PLB and OPB. PLB is intended for high speed peripherals, but it has a big set of control lines and a complex state machine. OPB is slower but simpler. Every peripheral needs to be connected to one bus using an IPIF (bus manager). IPIFs for PLB need much more area than those for OPB, so a trade-off between area and performance must be reached depending on the needs of each peripheral.

Most of the peripherals of our architecture are generic and can be obtained from Xilinx. But the interface with the camera and any peripheral to speed-up algorithms must be defined by the final user. These devices can work on DMA or non DMA modes. As they will work with a huge amount of data in a very short time, it is recommended that they support DMA data transfers. Using this method, PPC won't be used in data extraction, freeing the processor from this task.

Image manipulation can be easily accelerated using an FPGA [10]. Most of the operation consists on matrix oper-

---

<sup>1</sup>PowerPC

ations, which can be easily parallelized in a peripheral and done in a single clock cycle.

Final implementation is shown in Fig. 2

## 2.2. Software

### 2.2.1. Operating System and Drivers

As said on section 2.1.1, the PPC can run governed by an Operating System or without it. The benefits from an Operating System are clear: multiple processes, memory management, network protocols like TCP or UDP,...

Since Linux 2.6 [11], the Virtex FX is supported officially by the Vanilla branch of the kernel, but Xilinx peripherals like Temac, System-Ace, etc, are not supported by the Vanilla branch. There are some commercial solutions supporting all the peripherals by Xilinx but as they are not Open Source, ad-hoc drivers have been developed for them.

Also, some modifications have been done to the boot process to speed it up and improve its debugging. Any of these improvements can be provided upon request. They have been developed under Open Source licenses.

Linux has been chosen because its stability, great support by its community, adaptability, services in the kernel, and huge amount of applications developed for it [12]. Any application that runs on a desktop Linux can be ported to this architecture easily.

Some of the services used in the architecture, provided by the kernel are:

- DHCP client. During boot-up every camera gets its IP from an external DHCP server.
- NFS client. The file system of the cameras is obtained from a boot server. Cameras File System can be changed even during operation and there is no need for external memory.
- Early debugging console. An UART has been developed inside the camera interface to debug the whole system.
- Install and remove modules (drivers) during operation.

For every peripheral created in the architecture a driver must be developed. Linux 2.6 provides an easy and well documented API [13] for developing new drivers easily.

A driver for the image extraction device has been developed. This driver can be easily ported to support other image extraction devices.

### 2.2.2. Data Flow

Hardware and software work closely during image extraction and pre-processing. The system works following these steps:

1. The control application asks the image extractor driver for an image.
2. The driver contacts the image extractor, which takes a photo.
3. Once the photo is taken, the image extractor moves the image via DMA from the CMOS to the DDR Memory. During this step, the image extractor device applies online image algorithm to the image.
4. The device launches an interrupt.
5. The interface driver captures the interrupt and notifies the user level.
6. The control application gets the image from the driver (it is mapped via mmap on its memory space) and process it applying offline algorithms.
7. The control application decides if this image or a part of it should be sent to the main PC.
8. The application sends the image to the server using the network stack from the kernel.

Other devices can be implemented on the silicon fabric to perform other algorithms if needed.

### 2.2.3. Protocol

There are two communication paths between the main PC and the proposed system.

The first is the control path. This path is used to set the camera/CMOS parameters and also the system behaviour. For this path a TCP communication scheme is recommended. Latency is not important, and also speed is not relevant.

The second is the data path. It will be used to send the images or parts of them from the system to the PC. As the PPC is not very fast, protocol overhead cannot be ignored. There is no sense in making a system that has an input bandwidth of hundreds of Megabits if its output bandwidth is much smaller. For this reason, a TCP transmission is completely discouraged. The kernel overhead+TCP overhead reduces the transmission speed to a fraction of the potential one. For this path UDP with a very simple data flow control is recommended.

The reason of this not complex data flow is that all the cameras will be in a controlled network.

To reduce the kernel and network stack overflow it is also recommended to increase the MTU<sup>2</sup> as much as possible.

---

<sup>2</sup>Maximum Transfer Unit

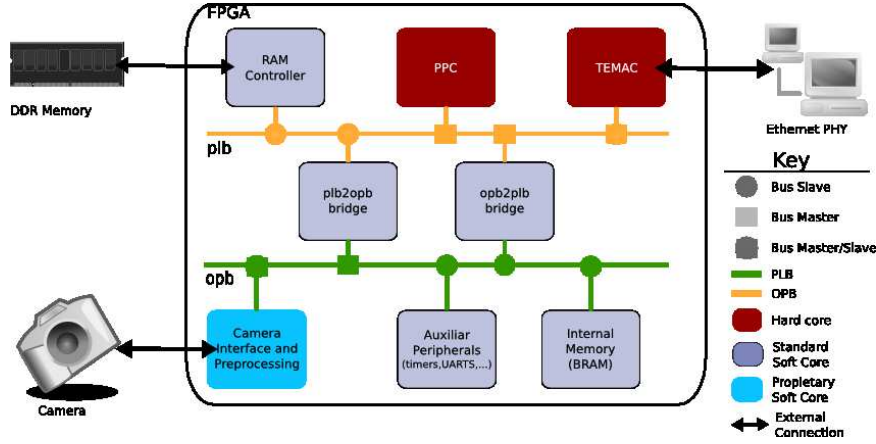


Fig. 2. Final Architecture

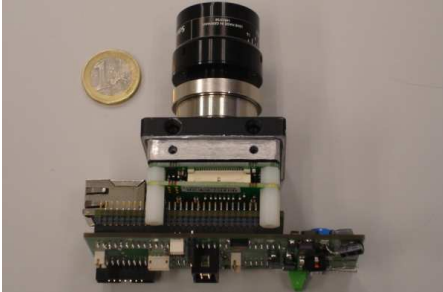


Fig. 3. Hardware used

### 3. VERIFICATION AND RESULTS

#### 3.1. Hardware selected

For the implementation of this architecture a commercial board with a Virtex FX-12 has been chosen to simplify the design. In particular, the Virtex FX12 Mini-Module by Avnet has been selected [14]. Remarkable features from this board are:

- Affordable: less than 200\$ buying a single unit
- Small size : 30 mm x 65.5 mm
- Big amount of User Definable I/O pins: 76
- Big External DDR Memory: 64 MB
- 4 MB ROM for the system and a separated platform flash (for FPGA configuration)

The chosen camera has been BCI4-6600 [15], from Vector International. It is a CMOS black and white 12bpp, 6.6 Mpixels camera. An image of the complete hardware set, including the camera and the FPGA board, is shown in Fig. 3.

#### 3.2. Communication Speed

Some experiments have been done in order to measure the transfer speed between the main PC and the proposed system, using different transmission methods.

All the tests consist on sending 20 times 8 MB from the system to the main PC with only one switch between them. The main PC is a high-end dual core computer. Results from these experiments are shown in Table 1.

##### 3.2.1. UDP vs. TCP

On this test, two different network stacks have been measured in order to test the relevance of the network stack in the speed transmission. UDP is a light protocol without control flow, and TCP is a complex protocol with a big state machine and complete flow control.

The tests show that in the same conditions UDP is significantly faster than TCP, up to 2.5 times faster.

##### 3.2.2. MMAP vs. READ

On this test, two memory copy methods have been tested: the common read-write method (Fig. 4) and memory access through segment mapping (Fig. 4) (mmap). If common read-write is used, every package should be copied, from a kernel buffer to a user buffer and then to a kernel buffer again (the network stack). If mmap method is used, the memory is copied between two kernel buffers.

The test shows that under the same conditions, transfers using Mmap are up to 20 % faster.

##### 3.2.3. MTU

On this test, the data is transferred using different MTUs. A bigger MTU means less packages in the network, but also more chances for this package to be lost.

```

/* Common Read-Write */
BYTE buffer[SIZE];
read(camera,buffer,SIZE);
write(ethernet,buffer,SIZE);

/* Mmapped Read */
BYTE *pointer;
pointer=mmap(camera,SIZE);
write(ethernet,pointer,SIZE);

```

**Fig. 4.** Transfer Modes

Protocol	MMap	MTU	DMA	Gigabit	Mbps
TCP	Read	1500	No	No	32
TCP	Read	1500	Yes	No	40
TCP	Mmap	8500	No	Yes	40
TCP	Read	8500	No	Yes	40
UDP	Read	1500	No	Yes	56
UDP	Mmap	1500	No	Yes	64
UDP	Read	8500	No	Yes	80
TCP	Read	1500	Yes	Yes	88
UDP	Mmap	8500	No	Yes	104
UDP	Read	1500	Yes	Yes	192
TCP	Read	8500	Yes	Yes	200
UDP	Read	8500	Yes	Yes	256

**Table 1.** Communication Speed Experiments Results

Test shows that a big MTU (also called Jumbo Pack-ages) provides much more performance in UPD transmis-sions over Gigabit, up to 2 times faster.

#### 3.2.4. DMA

On this test, the Ethernet Interface works using DMA or pro-cessed transfer. If DMA is enabled, the processor is inter-rupted when the package is in the memory, ready for the network stack. In processed transfer, the processor should read the packages from the interface FIFOs and send them to the memory.

Test shows that DMA improves up to 4 times the perfor-mance. We must be aware that a DMA interface consumes much area of the FPGA, so in some cases DMA can't be used.

#### 3.2.5. Gigabit vs Fast Ethernet

On this test, the transmission speed is measured for Gigabit Ethernet and Fast Ethernet.

Test shows that Gigabit is up to two times faster than Fast Ethernet, not ten times. We must realize the device will consume much less power in Fast Ethernet Method.

### 3.3. Acquisition Speed

Acquiring images from the CMOS/Camera can be done us-ing DMA or not, depending on the internal memory of the CMOS/Camera. If the CMOS/Camera used has internal mem-ory, the timing during data extraction is not critical, but if it has no memory, the internal timing must be respected.

If the timing is critical, a DMA method must be imple-mented. In this method, the interface takes control of the memory bus avoiding any interruption from the processor. If the timing is not critical DMA is not needed, reducing the complexity of the camera interface.

For a 40MHz, 6.6 Megapixels camera without memory (the proposed one), a DMA interface in the proposed archi-tecture was able to retrieve all its pixels without losses, but it just could retrieve a 10% of its pixels without DMA.

For a 4 Megapixels camera with memory, a non-DMA interface could retrieve all the image pixels in 540 ms.

The acquisition speed for the DMA system was 480 Megabits per second and for the non DMA system just 88 Megabits per second.

### 3.4. Pre-processing

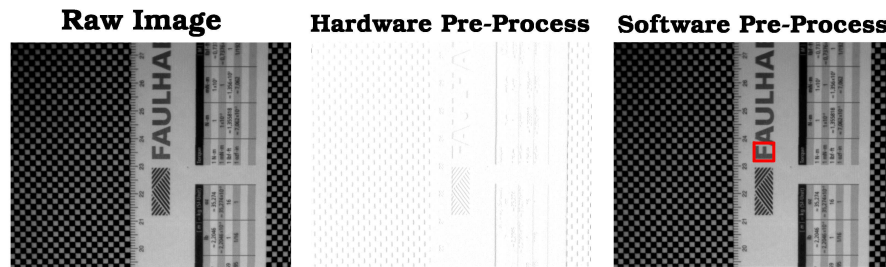
The implemented architecture has preprocessing algorithms implemented in software (as processor programs) and in hard-ware (as part of the image acquisition device). Both can be implemented easily.

Hardware algorithms are developed in VHDL using the standard Xilinx tools. On the proposed architecture some simple image algorithms were implemented to be executed during image extraction (online). On figure 5 a simple verti-cal edge detection is shown. It has been accomplished with-out any extra time consume.

Software algorithms can be implemented in almost any language thanks to the recent Linux Kernel that governs the system. It could even run java or other high level languages, reducing the skills needed to develop the system. On figure 5 a pattern recognition algorithm is shown, It looks for the letter 'F'. It is executed as a simple program in the micropro-cessor, accessing the images from the sensor as simple files, thanks to the Linux Kernel. It is done after the acquisition is completed using extra time.

## 4. CONCLUSIONS

This paper has proposed an intelligent and reconfigurable architecture for remote image acquisition. The architecture includes a Power PC processor, muti-purpose silicon area and Gigabit Ethernet connection in a board much smaller than a PC board (30 x 65.5 mm). Regarding its software, it includes Linux 2.6 for easy development of applications and services, giving it a huge potential. Different tests have shown the relation between configuration options (such as



**Fig. 5.** Pre-Processing Algorithms

protocol, memory copy, packet size or DMA) and communication speed.

Its flexibility allows connection to any camera and adding any pre-processing algorithm necessary in a specific application developed either in hardware or software. The proposed system is specially suitable for remote and high bandwidth applications, obtaining a good trade-off between size-performance and cost.

## Acknowledgements

This work has been supported by the TEC2006-13141-C03-03 project of the Spanish Ministry of Science and Technology. Ricardo Ribalda is supported by a FPU Fellowship from the Ministerio de Educacion y Ciencia (Spanish Ministry of Science).

## 5. REFERENCES

- [1] S. Fleck and W. Strasser, "Adaptive Probabilistic Tracking Embedded in a Smart Camera," *Computer Vision and Pattern Recognition, 2005 IEEE Computer Society Conference on*, vol. 3, 2005.
- [2] K. Shimizu and S. Hirai, "CMOS+ FPGA Vision System for Visual Feedback of Mechanical Systems," *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2060–2065, 2006.
- [3] N. Lepisto, B. Thornberg, and M. O'Nils, "High-performance FPGA based camera architecture for range imaging," *Proceedings of the 23rd NORCHIP Conference*, pp. 165–168, 2005.
- [4] J. Gaisler, "The LEON-2 Processor User's Manual," *Gaisler Research*, November, 2003.
- [5] M. van den Braak and S. Wong, "FPGA implementation of Voice-over IP," *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing*, pp. 338–342, 2005.
- [6] S. Heithecker and R. Ernst, "Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements," *Proceedings of the 42nd annual conference on Design automation*, pp. 575–578, 2005.
- [7] B. Fiethe, H. Michalik, C. Dierker, B. Osterloh, and G. Zhou, "Reconfigurable system-on-chip data processing units for space imaging instruments," *Proceedings of the conference on Design, automation and test in Europe*, pp. 977–982, 2007.
- [8] Y. Ren, J. Zhu, X. Yang, and S. Ye, "The Application of Virtex-Pro FPGA in High-Speed Image Processing Technology of Robot Vision Sensor," *Journal of Physics: Conference Series*, pp. 373–378, 2006.
- [9] XilinxInc., "Virtex-4 family overview," <http://www.xilinx.com/bvdocs/publications/ds112.pdf>, 2007.
- [10] C. T. Johnston, K. T. Gribbon, and D. G. Bailey, "Implementing image processing algorithms on FPGAs," in *ENZCON '04: Proceedings of the 11th Electronics New Zealand conference*, 2004, pp. 118–123.
- [11] L. Torvalds, "Kernel home page," <http://kernel.org>.
- [12] J. W. Williams and N. Bergmann, "Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip," in *ERSA*, 2004, pp. 163–169.
- [13] K.-H. G. Corbet J, Rubini A, *Linux Device Drivers*, 3rd ed. O'Reilly & Associates, Inc., 2005.
- [14] AvnetInc., "Virtex-4 fx12 mini-module product brief," [http://avnet.co.jp/products/kits/docs/Xilinx\\_Virtex-4\\_FX\\_Mini-Module-Product%20Brief.pdf](http://avnet.co.jp/products/kits/docs/Xilinx_Virtex-4_FX_Mini-Module-Product%20Brief.pdf), 2005.
- [15] VectorInternational, "Bci4-6600 datasheet," <http://www.vector-international.be/C-Cam/doc/Bci4-6600.pdf>, 2006.