# Design of a Secure, Intelligent, and Reconfigurable Web Cam Using a C Based System Design Flow

Diederik Verkest, Dirk Desmet, Prabhat Avasare, Paul Coene, Stijn Decneut, Filip Hendrickx, Théodore Marescaux, Jean-Yves Mignolet, Robert Pasko
IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
diederik.verkest@imec.be

Patrick Schaumont
UCLA, EE Department, Los Angeles CA 90095-1594
schaum@ee.ucla.edu

## Abstract

*This paper describes the design of a reconfigurable Internet camera, Cam-E-leon, combining reconfigurable hardware and embedded software. The software is based on the μClinux operating system. The network appliance implements a secure VPN (Virtual Private Network) with 3DES encryption and Internet camera server (including JPEG compression). The appliance's hardware can be reconfigured at run-time by the client, thus allowing to switch between several available image manipulation functions. This paper focuses on the design process used to implement the appliance starting from a high-level executable specification.*

## 1. Introduction

Future networked appliances should be able to download services from the network and execute them locally. To support this process, the implementation of a network appliance should be flexible. This flexibility is traditionally provided through the incorporation of a programmable instruction-set processor (ISP) of which the behavior can be changed by downloading new software over the network, possibly using JAVA technology. However, computational performance of software based solutions is inadequate for many modern multi-media applications (e.g. image processing) which typically need to run on such a networked appliance. In addition, the high power dissipation of a software based solution is incompatible with the need for portability and wireless Internet connectivity. The advent of large Field-Programmable Gate Arrays (FPGA) has opened up the possibility to offer flexibility through hardware reconfiguration. The power dissipation and computational power of an application implemented on such an FPGA lies between software based implementations and complete custom implementations (ASICs) [1]. In this paper we describe the design of a secure web camera that supports dynamic modification of its image processing capabilities by downloading of services from a configuration server in the network. The objectives of this design exercise were twofold:

- to demonstrate the concept of hardware plug-ins: user initiated run-time dynamic reconfiguration of part of the system functionality. This paper does not further elaborate this aspect. More details can be found in [2].
- to evaluate the use of a software-centric approach to embedded system design where many of the system's components are based on open source software packages and hardware acceleration is introduced where needed by using a C++ based hardware/software co-design environment called OCAPI-xl [3].

The demonstration platform consists of a processor board from Axis Communications [4] running μClinux [5] and a custom designed board with 2 XILINX Virtex800 FPGAs [6]. The board is connected to an IBIS4-camera, a 1.3 megapixel CMOS image sensor developed by IMEC's spin-off FillFactory [7]. The embedded software uses a standard third-party embedded Linux platform. This software is handling the network protocol layers, as well as the (re-)configuration and control of the FPGAs. The use of Linux eases reuse of existing open-source software modules, which allows us to design the full system in a short time.

The design process starts from a full-software reference implementation, re-using a lot of open-source C and C++ based software. The design of the hardware accelerated modules, that are executed on the FPGAs, starts from this full-software reference implementation. The design is carried out using OCAPI-xl, a C++ based embedded system design environment [3]. OCAPI-xl offers the ability to describe the system in C++, using a class library that supports a system model described as a number of concurrent processes that communicate through primitives like semaphores, messages and shared variables. From this system model, a refinement process allows to translate the processes and communication into a mixed hardware-software implementation, all within a traditional C++ development environment. A final automated HDL code generation step makes the link to traditional hardware development environments to complete the FPGA implementation. In addition, OCAPI-xl supports easy integration of external interfaces and of existing software functions through a mechanism called foreign language interface. This mechanism was used here to link the hardware-models and the embedded Linux software.

In Cam-E-leon, FPGAs are used where a software implementation can not meet the performance requirements, but where flexibility is still desired. In our example this is the case for the image acquisition functionality, consisting of the camera interface (the camera is sampled at 10 MHz), the color reconstruction (de-mosaicing), user-dependent image manipulation, JPEG compression and the 3DES encryption, used in the VPN IPSEC security layer.

A number of image manipulation plug-ins, selectable at run-time by the user from a web browser and downloaded over the network from a reconfiguration server, demonstrate the concept of networked reconfiguration [2]. Specifically for the Cam-E-leon web camera, we developed the Boot-Up Reconfigurable Platforms Protocol (BRPP), a new protocol similar to BootP and DHCP, to allow the camera platform to discover and retrieve the available configurations and services on the network. At boot time the reconfigurable embedded device localizes a neighboring "reconfiguration server", a machine that stores and serves a number of HW/SW configurations to its clients. During a second phase, the reconfigurable appliance negotiates its characteristics and required services with the reconfiguration server. The server will then respond by providing the list of available services. On request, the reconfiguration server uploads new services to the reconfigurable platform that dynamically reconfigures its FPGAs and adapts the HW/SW communication, to interface with the new application.

Section 2 provides details about the system functionality including the network protocol layer, the image capture and compression, and the security aspects. Section 3 provides details about the architecture of the implementation platform. Section 4 explains the design flow that was used to design the Cam-E-leon. Finally, in section 5 we summarize the main points of the paper.

## 2. System functionality

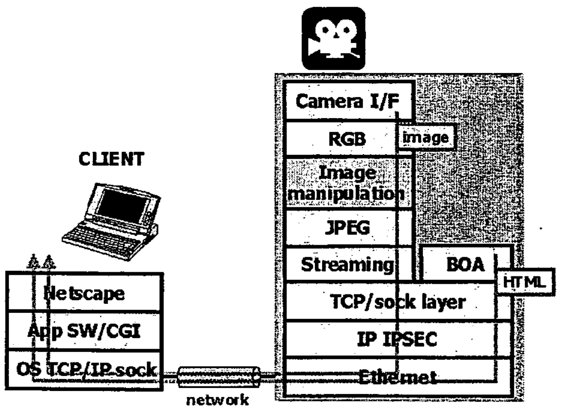


**Figure 1. Cam-E-leon system functionality.**

Figure 1 gives an overview of the system functionality. The left-hand side of the picture shows a user terminal (client) that runs a regular browser and connects via the Internet (TCP/IP, HTTP, ...) to the Cam-E-leon appliance. Some specific application software (CGI and JAVA scripting) allows to control the remote device. The right-hand side of the picture shows the Cam-E-leon functionality. On top of the network protocol stack, we implement the web camera functionality (camera interface, RGB reconstruction, image manipulation, JPEG compression, video streaming), a web server (boa) that serves HTML pages to the client, and - not shown in the picture - some functionality to reconfigure the platform.

The physical connection is done via Ethernet. On top of that we run TCP/IP with IPSEC and VPN functionality. VPN is a technology that allows a secure connection (commonly referred to as a "tunnel") between two network devices. The security is provided through a mechanism of authentication, encryption (in our case using 3DES), and key exchange. In Cam-E-leon we use freeS/WAN [8], a public domain IPSEC library for Linux. The main performance bottleneck in the network functionality consists of the 3DES encryption. As will be explained further on, this bottleneck is removed by integrating a hardware accelerated implementation of 3DES in the system.

The image capture functionality of the system consists of the following steps: the camera interface, the RGB reconstruction (or demosaicing) and the JPEG compression. The IBIS4 CMOS camera we used in Cam-E-leon provides a picture of 1280 x 1024 x 10 bit at a sample rate of 10 MHz. The RGB reconstruction transforms this picture into a 640 x 480 pixel RGB image (3 x 8 bits), suitable for JPEG compression. The JPEG algorithm works on a YUV decimated image (in our case YUV 422) and achieves a compression ratio of a factor 30, from 900 KB/frame to 30 KB/frame. Finally, the compressed frames are streamed to the client using a modified version of Camserv [9], again a public domain software package.

In between the RGB reconstruction and the JPEG compression functionality, optional image manipulation can be performed. This part of the functionality can be downloaded over the network under control of the user.

## 3. System Architecture

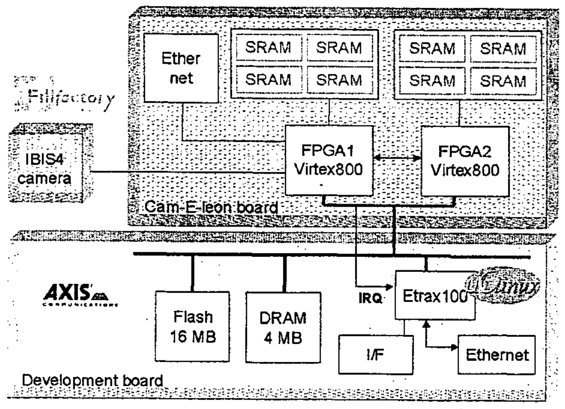


**Figure 2. Cam-E-leon system architecture.**

Figure 2 shows the hardware architecture of the complete system. The system is implemented on three boards. The CMOS image sensor is mounted on a separate board together with some I/O and is clocked at 10 MHz. All the system software, including the $\mu$Clinux OS, runs on an ETRAX100 processor that is mounted on a board obtained from Axis Communications running at 100 MHz. This board contains 4 MB DRAM and 16 MB Flash memory, interfaces, and the Ethernet physical interface that is used to communicate with the client. A third, custom developed, board contains the two Virtex800 FPGAs together with 2 Mbit SRAM memory each for data storage. This board can operate between 20 MHz and 50 MHz[1]. The two Virtex800 FPGAs are connected directly to each other. Each Virtex800 FPGA further has a connection to the bus on the Axis board and a dedicated interrupt line is foreseen from each Virtex800 FPGA to the ETRAX processor.

The software image ($\mu$Clinux , Camserv, IPSEC, drivers, TCP, ...) resides in the Flash memory of the Axis board. The compressed $\mu$Clinux kernel including basic applications and patched with FreeS/WAN requires approximately 1 MByte (uncompressed this becomes 2.5 MB). The Flash memory further contains a file system (/mnt/flash) of about 900 KByte containing the Camserv executable, JPEG image data files, configuration files, default HTML pages, BRPP daemon, etc. Approximately 2MB of the Flash memory are used as a RAM drive to store downloaded Virtex configuration files and HTML pages. The hardware configuration files for the Cam-E-leon board are downloaded (in compressed format) over the network via the Axis board and uncompressed on the ETRAX processor before being used to reconfigure the FPGAs. In our case the configuration file for a single FPGA is approximately 50 KB when compressed with gzip. After uncompressing it becomes approximately 500 KB.

In the experiment described in this paper, one of the FPGAs contains the camera interface, RGB reconstruction, image manipulation and JPEG compression functionality. The other FPGA contains the 3DES encryption functionality. In principle all functionality could have been implemented on a single Virtex800 FPGA. The only concern was bandwidth to the memory and therefore, the JPEG compression writes its results in the second memory attached to the second FPGA.

## 4. Design flow

The design of a complex hardware-software system like the one at hand, necessitates a high-level reference model, from which every component can be refined towards its implementation description (HDL or C). For this design we start from a full software implementation of the system, making use as much as possible of Linux and open-source software. The hardware design process starts from these C/C++ implementations, and uses the OCAPI-xl design flow to gradually refine these C++ implementations to a level from which VHDL code for final implementation can be generated automatically. The design of the large JPEG encoder is explained in more detail in section 4.2. We also show how an existing VHDL module (the 3DES encryption block) can be included in the design.

### 4.1. C++ for hardware design

C++ based design methodologies are among the latest attempts to deal with the complexity of system-on-chip (SoC) designs by introducing the object oriented programming paradigm (OOP) into the design process [10]. The essential idea of all C++ based methodologies is to provide a set of semantic primitives required for the design of hardware (and software), complemented with the necessary simulation and code-generation engines, in a form of an extendible library of classes. The amount of semantic primitives, as well as the underlying computational model(s) can

[1]The Ethernet connection on this board is not used in this particular experiment. Its purpose is to allow a direct download of the hardware (re-)configuration data to the FPGAs without passing via the processor board.

vary from one methodology to another.

The OOP paradigm is very well suited for such approach, since it allows to define and use the new primitives in the same way as the built-in data types and functions. When using a C++ based methodology, the designer must devise the system description using the predefined objects, or his own extensions. Afterwards, the description is compiled using a C++ compiler, resulting in an executable specification providing simulation and/or code generation.

OCAPI-xl can be considered a good example of a C++ based design methodology [3], specifically intended for the design of heterogeneous HW/SW systems. It features a unified approach to hardware and software code, provides parallelism at a process level, and supports communication primitives like messages or semaphores.

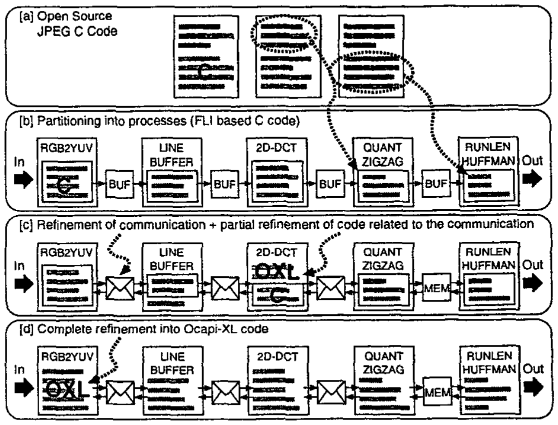
The basic quantum of computation is an instruction. The necessary set of arithmetic, logic, assignment, as well as looping and branching instructions is defined. To support parallel execution, OCAPI-xl provides the notion of a process as the basic level of parallelism and hierarchy. Communication between processes is implemented via three basic communication primitives: messages, semaphores, and shared variables. Finally, to increase flexibility, a direct interface to C++ is implemented via a so-called Foreign Language Interface (FLI). It makes possible to run any snippet of C++ code during an OCAPI-xl simulation.

Finally, the OCAPI-xl description compiles into executable code. In addition, it supports code-generation to other languages: VHDL/Verilog for hardware and C for software. The FLIs are appearing in the generated code as function calls in the C code and black-boxes with appropriate ports in the VHDL/Verilog code.

## 4.2. Design of a JPEG encoder with OCAPI-xl

The design of a JPEG encoder demonstrates how a C++ based methodology allows a step-wise gradual refinement of the target application starting from a high level C code. The complete design effort can be divided into several major phases, as shown in Figure 3.

We start from an openly available JPEG encoder model included in a video conferencing software application [11]. In the first step, the parallel threads inside the encoder were identified and the corresponding C code was partitioned into OCAPI-xl processes using the FLI mechanism, as indicated in Figure 3[b]. The communication between processes was still implemented via C buffers. The following JPEG processes were identified: **Color convertor** transforms the color information from RGB to YUV encoding, **Line buffer** re-groups the camera input into $8 \times 8$ blocks, **2D-DCT** calculates the two-dimensional DCT, **Quantizer** quantizes the DCT output and simultaneously performs the zig-zag re-ordering, **Huffman** performs the run-length and Huffman encoding.



[a] Open Source JPEG C Code
C
[b] Partitioning into processes (FLI based C code)
RGB2YUV
LINE BUFFER
2D-DCT
QUANT ZIGZAG
RUNLEN HUFFMAN
In
Out
BUF
[c] Refinement of communication + partial refinement of code related to the communication
OXL
MEM
[d] Complete refinement into Ocapi-XL code

**Figure 3. JPEG Encoder Design Flow**

**Table 1. Simulation times for JPEG**

| Image size | 32 x 32 | 256 x 256 |
|---|---|---|
| Software reference code | 1 sec | 30 sec |
| High-level OCAPI-xl code | 6 sec | 289 sec |
| Refined OCAPI-xl code | 15 sec | 650 sec |
| Generated VHDL code | 3 min | > 60 min |

In the second step, the communication refinement takes place. This includes introduction of the appropriate communication primitives, i.e. messages and memory buffers instead of the C buffers, as well as writing the OCAPI-xl communication code inside of each process, as shown in Figure 3[c]. However, the core functionality is still implemented via the FLIs, so the communication scheme can be tested before coding of the behavior starts.

Finally, the C code in each FLI is gradually rewritten into OCAPI-xl code, resulting in executable specification, out of which the VHDL code can be generated. The obvious advantage of the presented methodology is the possibility to approach a design in a completely incremental way. At each stage, the complete simulation test benches from previous refinements are available and new code can be cross-checked against any of them.

Table 1 gives an overview of simulation times for different versions (image size, abstraction level) of JPEG models during the OCAPI-xl refinement.

After synthesis of the generated VHDL code, the JPEG block occupies approximately 38 % of the Virtex800 FPGA running at 33 MHz.

## 4.3. Interfacing a VHDL block in the system: the 3DES encryption

The VPN layer is based on the Linux IPSEC implementation of freeS/WAN. The computationally most intensive part in IPSEC is the 3DES encryption. Therefore we opt for an implementation where the IPSEC layer runs mainly in software, with an hardware acceleration of the encryption function.

Rather than implementing the 3DES function ourselves, we have chosen to integrate an existing hardware implementation of this block, thus demonstrating the feasibility of IP integration. The DES hardware module was obtained from [12]. In order to connect this block in our system a hardware wrapper module needs to be written. This wrapper maps the (relevant) I/O ports of the block on the available HW/SW communication primitives: memory-mapped registers and interrupts. For Cam-E-leon only memory mapped registers were used. With this interface an efficient driver can be easily written in software. Since it works fully without interrupts (polling), this driver can be easily used in the IPSEC routines, which are implemented as interrupt handlers in Linux.

When using this hardware accelerated encryption together with freeS/WAN on the processor, we can observe an important speedup. Indeed, while the elapsed time measured to transmit a 20 KB packet over the network in an all software version was 550 ms, the time with the hardware accelerated 3DES was 130 ms. This can be compared to 75 ms, when using clear text (without encryption).

## 5. Conclusions

In this paper we described the design of a smart networked camera the behavior of which can be changed over the network. The camera implements video streaming functionality using motion-JPEG over a secure IPSEC/VPN network link. Both image processing functionality and encryption functionality are accelerated through FPGA hardware implementations. In addition, extra image manipulation services (plug-ins) which are available on the network can be selected by the user and dynamically downloaded on the platform where they reconfigure both hardware and software aspects. Figure 4 shows a picture of the Cam-E-leon platform.

The Cam-E-leon platform serves about 5 frames/second in normal operation (with images of 640 by 480 pixels) and uses about 6.5 Watt. Compared to a software JPEG solution, the hardware accelerated version results in a speed-up of a factor 10, improving the energy efficiency of the platform also with a factor of 10[2]

[2]The power dissipation does not change significantly by hardware acceleration of JPEG. Software related power remains unchanged as the processor is 100 % busy with $\mu$Clinux and other software anyhow. Hardware related power increases only slightly (from 3.78 Watt to 4.05 Watt) by the hardware acceleration of JPEG.

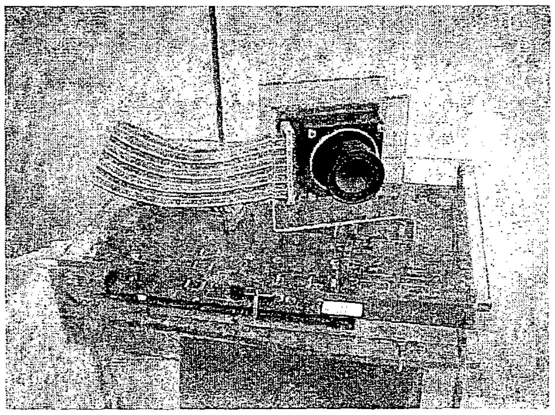


**Figure 4. The Cam-E-leon boards.**

## References

[1] A. DeHon, "The Density Advantage of Configurable Computing", *IEEE Computer*, pp. 41- 49, April 2000.

[2] D. Desmet et al., "Design of Cam-E-leon, a Run-time Reconfigurable Web Camera", to appear in "Simulation, Architecture and Modeling of Systems", LNCS, Springer-Verlag.

[3] G. Vanmeerbeeck et al., "Hardware/Software Partitioning of Embedded Systems in OCAPI-xl", *Proceedings of Ninth International Symposium on Hardware/Software Co-design (CODES-2001)*, Copenhagen, Denmark, pp. 30-35, April 2001.

[4] Axis Communications, http://www.axis.com/

[5] $\mu$Clinux, http://www.uclinux.org/

[6] Xilinx, http://www.xilinx.com/

[7] FillFactory, http://www.fillfactory.com/

[8] LINUX FreeS/WAN, http://www.xs4all.nl/~freeswan/

[9] Camserv, http://cserv.sourceforge.net/

[10] D. Verkest, J. Kunkel, F. Schirrmeister, "System Level Design Using C++", Proc. of Design, Automation and Test in Europe conference (DATE-2000), Paris, France, pp. 74-81, March, 2000.

[11] UCB/LBNL Video Conferencing Tool (vic), http://www-nrg.ee.lbl.gov/vic/

[12] http://www.ra.informatik.uni-stuttgart.de/~stankats/pg99.html