Design Technology for Networked Reconfigurable FPGA Platforms

S. Guccione, Xilinx, San Jose, CA, Steven.Guccione@xilinx.com
D. Verkest, IMEC, Leuven, Belgium, diederik.verkest@imec.be
I.Bolsens, Xilinx, San Jose, CA, ivo@xilinx.com

Abstract

Future networked appliances should be able to download new services or upgrades from the network and execute them locally. This flexibility is typically achieved by processors that can download new software over the network, using JAVA technology. This paper demonstrates that FPGAs are a realistic implementation platform for thin server or client applications. FPGAs can offer the same end-user experience as software based systems, combined with more computational power and lower cost.

1. Introduction

Internet has become a driving force for the deployment of embedded systems. Software embedded systems often do not offer the best solution in terms of cost, speed and power. It will be demonstrated in this paper that FPGA platforms create a good compromise between high performance and maintaining the capability of networked reconfiguration.

First, we will highlight the state-of-the-art of programmable logic platforms.
During the second section we will discuss a user scenario of networked reconfigurable hardware. An appliance equipped with a reconfigurable FPGA platform localizes a reconfiguration server. The reconfigurable appliance negotiates its characteristics and required services with the reconfiguration server. On request, th reconfiguration server will upload new services to the reconfigurable FPGA platform. The FPGA will then be dynamically reconfigured and will allow for a flexible implementation of new services provided by the appliance.

Finally, the underlying technology that has been applied to make the run time reconfiguration possible, JBits, will be explained . JBits is a set of Java classes which provide an API to access the FPGA bitstream. This approach allows for reconfigurable HW resources to be individually configured under software control. In addition, the object oriented support in the Java programming language allows for the development of a library of parameterizable cores.

2. Platform FPGAs

Today one can purchase FPGA devices with up to 6 million system gates [1] and within three to four years processing technology will allow us to build 50 million gate devices, enough logic to build very complex , high performance systems. In addition these devices operate at internal clock speeds above 300MHz, the equal of many ASICs. As device densities keep increasing, a wide variety of hard and soft intellectual property cores are being made available on these platforms. Hardcores are fixed hardware designs that are incorporated into the FPGA architecture (examples are processor cores such as PowerPC/ARM and DSP datapaths such as Booth multipliers) Softcores are flexible IP building blocks that take full advantage of an efficient and flexible implementation in the FPGA fabric (examples are I/O busses such as PCI-cores, processor cores such as MicroBlaze/Nios and DSP functions such as Viterbi decoders) [2].
Memory is a critical part of most designs. The ratio of block RAM and distributed RAM to logic gates in FPGAs will continue to increase over time. Current devices support up to 3.5 Mb of embedded block RAM and 2Mb of distributed RAM [1]. The demand of high performant systems requires the use of gigabit-per-second serial I/O capability for interconnecting devices, backplanes and systems.

These great platforms are creating headaches for the EDA teams that are trying to capture this ever growing design complexity and heterogeneity in new modeling languages, co-design flows, verification techniques and IP-re-use strategies. Research teams from different fields are joining forces to master the problem of such large complexity.

FPGA platforms will have to be accompanied by high level design tools that allow to capture the growing complexity and flexibility (in time and space) of these components
Next to this, SoC platforms will have to further integrate solutions that free the system engineer from dealing with the effects that result from smaller line-widths, higher speeds and higher power.
Providing the design community with such field-programmable hardware platforms that contain millions of gates, running at clock frequencies of several hundred MHz, communicating with integrated processors and distributed memories at GByte/s data rates is definitely breaking down barriers to build tomorrows products.
The next section will highlight such an innovative approach to use FPGAs in the context of a web based appliance, Cam-E-Leon allowing to dynamically update functionality by downloading and reconfiguring the 'soft' hardware and software over the network.

3. A webcam with hardware plug-ins

Cam-E-leon is a webcam, combining reconfigurable hardware and embedded software. This network appliance implements a secure VPN (Virtual Private Network) connection with 3DES encryption and an Internet camera server including Motion-JPEG compression. The appliance's hardware can be reconfigured at run-time by the user through a browser interface, thus allowing to switch between several image manipulation "plug-in" functions. The appliance's software is based on the μClinux operating system [3]. In the next two sections we describe the functionality and system architecture in more detail and, next, explain the design process followed to realize this system.

3.1. System functionality and architecture

Figure 1 (a) shows the context and the functionality of Cam-E-leon. The actual webcam is shown in the middle of the figure. A number of image manipulation plug-ins, selectable at run-time by the user via a Web browswer and downloadable over the network from a reconfiguration server, demonstrate the concept of networked reconfiguration. Specifically for this webcam we developed BRPP (Boot-up Reconfigurable Platform Protocol) to allow the camera platform to discover and retrieve available (re)configurations over the network. At boot time the reconfigurable platform localizes a reconfiguration server, a machine that stores and serves a number of HW/SW configurations to its clients. The reconfigurable appliance negotiates its characteristics and required services with the reconfiguration server. The server responds by providing a 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.

Figure 1 (b) and (c) show Cam-E-leon's architecture and implementation, respectively. The appliance is implemented using three boards. The actual camera, a 1280x1024 pixel CMOS color image sensor developed by IMEC's spin-off Fillfactory [4], is mounted on a separate board together with some I/O and is clocked at 10 MHz. The system software handles the network protocol layers (excluding the 3DES encryption which is accelerated in hardware) as well as the (re)configuration and control of the FPGAs. All the system software including the μClinux OS runs on an ETRAX100 processor mounted on a board obtained from Axis Communications [5] and 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 network. A third, custom developed, board contains two Virtex 800 FPGAs [1] together with 2 MB SRAM memory each for data storage. This board can operate between 20 and 50 MHz. The FPGAs are used where a software implementation cannot meet the performance requirements and/or where flexibility is still required. This is the case for the image acquisition functionality (camera interface, color reconstruction, image manipulation plug-ins, JPEG compression) and for the 3DES encryption used in the VPN IPSEC layer). For more details about Cam-E-leon, we refer the interested reader to [1].

3.2. Design flow

The design of a complex hardware-software system necessitates a high-level reference model from which every component can be refined towards its final implementation. In our case, we start from a full C/C++ software implementation of the system making use as much as possible of Linux and open-source software modules. The design process starts from this reference model and uses our C++ based OCAPI-xl [7] design flow to gradually refine these C++ descriptions to a level where HDL code for the hardware parts and C code for the hardware-dependent software parts can be generated automatically. Parts that will be implemented in software and do not directly access hardware parts, can of course be compiled directly from the reference code.

C++ based design methodologies are amongst the latest attempts to deal with the increased complexity of SoC designs by introducing object-oriented programming techniques into the design process [8]. The essential idea of all C++ based methodologies is to introduce a set of semantic primitives required for the design of hardware (and software), complemented with the necessary simulation and code-generation engines, under the form of an extensible library of classes. The semantic primitives and the underlying computational

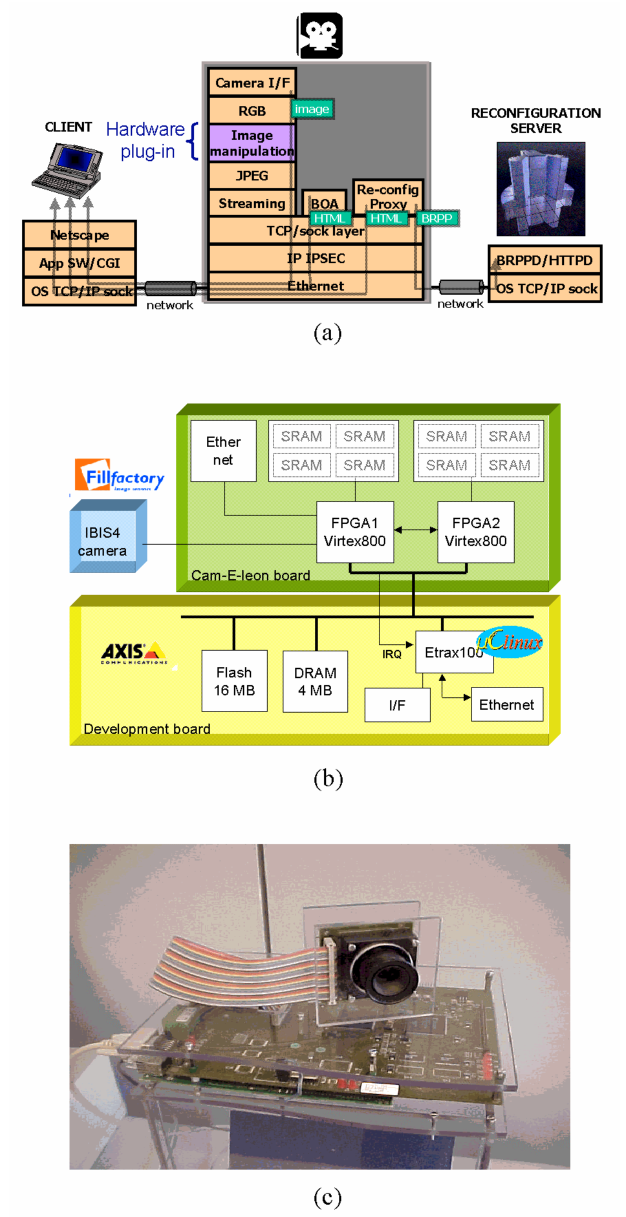

Figure 1. The Cam-E-leon system

model can vary from one methodology to the other. OCAPI-xl can be considered a good example of a C++ based design methodology, specifically intended for the design of heterogeneous HW/SW systems. To support parallel execution, OCAPI-xl provides the notion of a process as the basic level of parallelism and hierarchy. Communication between processes is defined using three basic primitives: messages, semaphores, and shared variables. To increase flexibility, a direct interface to include C++ code is implemented using the so-called Foreign Language Interface (FLI) which allows to run any snippet of C/C++ code inside an OCAPI-xl simulation. The OCAPI-xl code compiles into executable code and supports code-generation into other languages: VHDL/Verilog for hardware and C for software. In the next paragraphs we illustrate the OCAPI-xl design process using one of the larger hardware blocks: the JPEG encoder.

Figure 2 shows the different phases in the design process of the JPEG encoder. We start from an openly available JPEG encoder model included in a video conferencing application [9]. In the first step, the parallel threads inside the encoder are identified and the corresponding C code is partitioned into OCAPI-xl processes as indicated in Figure 2.b. In this way we obtain the following processes: the *color converter* transforms the color information from RGB to YUV encoding; the *line buffer* re-groups the camera input into 8x8 blocks; the *2D-DCT* calculates the two-dimensional Discrete Cosine Transform; the *quantizer* quantizes the DCT output and simultaneously performs the zigzag reordering; finally, the *Huffman* performs the run-length and Huffman encoding.

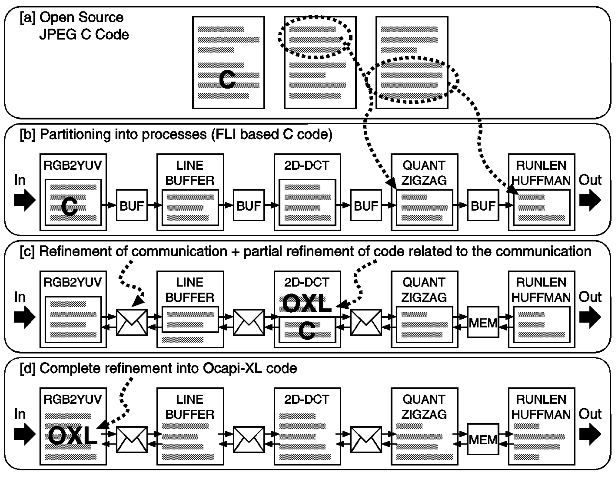
[a] Open Source JPEG C Code
C
[b] Partitioning into processes (FLI based C code)
In
RGB2YUV
C
BUF
LINE BUFFER
BUF
2D-DCT
BUF
QUANT ZIGZAG
BUF
RUNLEN HUFFMAN
Out
[c] Refinement of communication + partial refinement of code related to the communication
In
RGB2YUV
LINE BUFFER
2D-DCT
OXL
C
QUANT ZIGZAG
MEM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

Figure 2. JPEG encoder design flow

In the second step, the communication refinement takes place. This includes introduction of the appropriate communication primitives, i.e. messages and memory buffers between the processes, as well as writing of the OCAPI-xl communication code inside of each process, as shown in Figure 2.c. However, the core functionality of each process is still the same plain C code from the reference code using the FLI mechanism of OCAPI-xl.

Finally, the C code of the core functionality is gradually rewritten into OCAPI-xl code, resulting in an executable specification out of which HDL code can be generated. After synthesis of the generated HDL code, the JPEG block occupies approximately 38 % of the Virtex800 FPGA running at 33 MHz.

The obvious advantage of the presented methodology is the possibility to approach a design in a completely incremental fashion. At each stage, the complete simulation test benches from previous refinements are available and any new code introduced can be checked against any of these test benches.

Table 1. Simulation times for JPEG versions

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

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

4. JBits

JBits software is a set of Java classes which provide an Application Programming Interface (API) to access the Xilinx FPGA bitstream [10] . The interface operates on either bitstreams generated by Xilinx design tools, or on bitstreams read back from actual hardware. This permits all configurable resources like look-up tables routing and flip-flops in the FPGA to individually configured under software control. The API can be used to construct complete circuits and to modify existing circuits. In addition, the object-oriented support in the Java programming language has permitted a library of parameterizable cores to be implemented. The interface to the hardware is provided by a standard HW interface, XHWIF. Part of the XHWIF is a TCP/IP based remote network access support. This enables remote hardware configuration and debugging capabilities. Using the JBits interface, software can be written which produces circuits and provides support for dynamic reconfiguration.

Although direct JBits calls occur at a very low level, higher level tools have been built on the foundation of configuration calls using the object oriented model afforded by the Java language. These are the JRoute API [11] and the Run-Time Parameterizable cores [12].

The JRoute API is a run-time reconfigurable router that provides individual routing resource selection and auto-routing capabilities for JBits-based designs. At the lowest level, the user can specify a list of all individual routing resources that define a point-to-point connection. An auto-router provides the highest level of abstraction but the least control.

The JBits Run-Time Parameterizable core specification (RTP) provides a means for abstracting away the low-level JBits configuration calls, thereby creating an abstraction level similar to the traditional hardware design languages. Examples of these cores are adders, shifter, counters or combinations of these.

5. Conclusions

This paper demonstrates a user scenario of platform FPGAs in the context of web based appliances. The capability of run time reconfiguration, using the JBits technology allows for a software like user experience in terms of flexibility, combined with a high performant and low cost implementation target.

References

[1] Virtex, http://www.xilinx.com/publications/xcellonline/

[2] MicroBlaze, http://www.xilinx.com/ipcenter/

[3] μClinux, http://www.uclinux.org/

[4] Fillfactory, http://www.fillfactory.com/

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

[6] 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.

[7] G. Vanmeerbeeck et al., "Hardware/Software partitioning of embedded systems in OCAPI-xl", in *Proceedings of the 9th International Symposium on Hardware/Software Co-design (CODES-2001)*, Copenhagen, Denmark, pp. 30-35, April 2001.

[8] D. Verkest, J. Kunkel, and F. Schirrmeister, "System level design using C++", in *Proceedings of Design, Automation and Test in Europe Conference (DATE-2000)*, Paris, France, pp. 74-81, March 2000.

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

[10] S. Guccione, D. Levi and P. Sundarajan , "JBits : Java based interface for reconfigurable computing", Proceedings of 2nd Annual Military and Aerospace Applications for Programmable Devices and Technologies.

[11] E. Keller, "JRoute, a Run-Time Routing API for FPGA", Proceedings of the Reconfigurable Architecture Workshop, 2000, pp 874-881.

[12] S. Guccione, D. Levi, "Run-Time Parameterizable Cores", Proceedings of the 9th International Workshop on field-Programmable Logic and Applications, 1999, pp215-222