Image Processing Application Development: From Rapid Prototyping to SW/HW Co-simulation and Automated Code Generation

Cristina Vicente[1], Ana Toledo[2], and Pedro Sánchez-Palma[1]

[1] Departamento de Tecnologías de la Información y Comunicaciones
E.T.S. Ingeniería de Telecomunicación - Universidad Politécnica de Cartagena
Campus Muralla del Mar S/N, 30.202 Cartagena, Spain
{Cristina.Vicente,Pedro.Sanchez}@upct.es
[2] Departamento de Tecnología Electrónica
E.T.S. Ingeniería Industrial - Universidad Politécnica de Cartagena
Campus Muralla del Mar S/N, 30.202 Cartagena, Spain
Ana.Toledo@upct.es

Abstract. Nowadays, the market-place offers quite powerful and low cost reconfigurable hardware devices and a wide range of software tools which find application in the image processing field. However, most of the image processing application designs and their latter deployment on specific hardware devices is still carried out quite costly by hand. This paper presents a new approach to image processing application development, which tackles the historic question of how filling the gap existing between rapid throwaway software designs and final software/hardware implementations. A new graphical component-based tool has been implemented which allows to comprehensively develop this kind of applications, from functional and architectural prototyping stages to software/hardware co-simulation and final code generation. Building this tool has been possible thanks to the synergy that arises from the integration of several of the pre-existent software and hardware image processing libraries and tools.

Keywords: image processing applications, component-based development, prototyping, co-simulation, automated code generation

1 Introduction

Today, Image Processing (IP) techniques find application in many different domains such as automated visual inspection of industrial products, medical imaging or biometric person authentication, among others [1][2]. The marketplace offers many IP-related products, ranging from platform-optimized software and hardware libraries to high-level prototyping and simulation tools. Nevertheless, none of these products actually covers the whole process of building IP applications. Actually, the historic question of how bridging the gap between design models and final system implementation remains still open, also when talking about these systems.

This paper presents a novel approach to IP application development which is aimed to cover the whole life cycle of this kind of products. In order to put this approach into practice, a new tool has been implemented which, following the growing trend toward component-based application development [3], integrates some of the previously existing IP-related products, instead of being built from scratch.

The rest of this paper is organized as follows. The common procedure followed to build IP applications is briefly reviewed in Section 2. In Section 3, a new IP Comprehensive Development (IP-CoDe) Tool is presented, which is intended to help building and evaluating both functional and architectural IP prototypes. The use of this tool to develop a complete study case is presented in section 4. Finally, some conclusions and future research lines are included in Section 5.

2 Building IP Applications

Building IP applications usually requires an initial rapid prototyping stage which helps selecting the algorithms that fulfill the system functional requirements. Commonly, this functional prototype is implemented by means of a high level programming language (C++, MATLAB, Java, etc), and generally incorporates the functionality provided by one of the multiple available IP libraries, e.g. Intel© Open Computer Vision (free Open Source library) [4], Intel© Integrated Performance Primitives [5], Matrox© Imaging Library [6], Mathworks© IP Toolbox [7], etc.

Once the functional prototype has been carefully tested, the application architecture must be defined in terms of a specific platform which might be composed of several processors, whether SW or HW, or both. Thus, the initial prototype is partitioned into functional units that can be mapped into the different processing elements. This architectural design stage produces a co-prototype which must be tested in order to ensure that cost and performance constraints are met for each particular application. Testing the selected co-prototype is usually accomplished by means of co-simulation techniques, which allow evaluating both software and hardware, and their interactions (synchronization, data transfer, etc).

Thus, building IP applications requires a great deal of IP algorithms and configurations to be explored. In fact, different functional prototypes can fulfill the initial IP requirements. For each prototype, different SW/HW partitions can be obtained and, for each partition, different mappings of its functional units into the various elements of the platform can be selected. Finally, different platforms can be considered candidates for a given application.

Each of these design tasks can be developed by means of different tools, but as stated in [8] "*a new generation of tool is required which helps bridging the gap between the exiting design tools. Such tool, should address the functional and architectural design stages, and reach both the software and hardware domains*".

3 The IP-CoDe Tool: An Integration Experience

Prototyping is a rapid and inexpensive way to validate system requirements. Usually, different prototypes are built in order to test different aspects of the application under development. As a matter of fact, functional models are built as software throwaway prototypes by means of specialized tools, different from those needed for architectural co-prototyping, where HW devices must be also taken into account. Integrating these tools under a unified environment would ease evolving functional prototypes to the corresponding co-prototypes, thus filling the existing gap between application design and implementation.

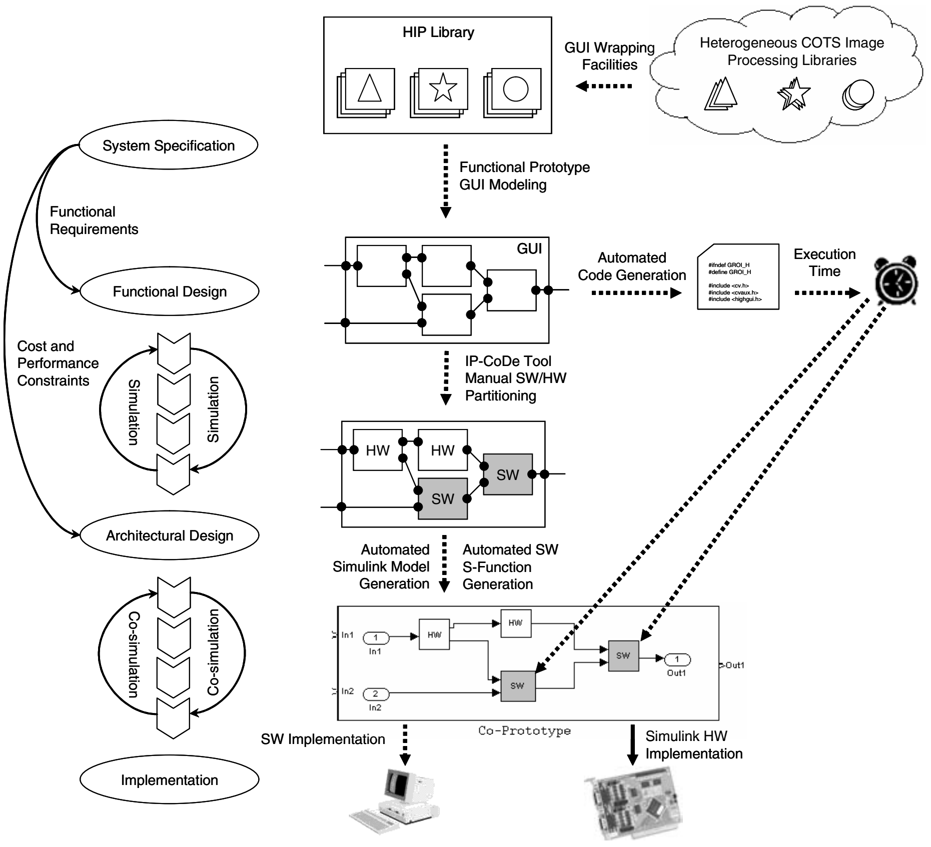
HIP Library
GUI Wrapping Facilities
Heterogeneous COTS Image Processing Libraries
System Specification
Functional Requirements
Functional Design
Functional Prototype GUI Modeling
GUI
Automated Code Generation
Execution Time
Cost and Performance Constraints
Simulation
Simulation
IP-CoDe Tool Manual SW/HW Partitioning
HW
HW
SW
SW
Architectural Design
Automated Simulink Model Generation
Automated SW S-Function Generation
Co-simulation
Co-simulation
In1
In2
Out1
Co-Prototype
SW Implementation
Simulink HW Implementation
Implementation

Fig. 1. Scheme of the IP application development life cycle using the IP-CoDe Tool.

In the following sections, our experience with IP product integration to build an IP Comprehensive Development Tool is detailed. This tool covers the whole IP application development life cycle, as shown in Fig. 1.

3.1 Functional Design

The first stage when building any application is to define its functional and non-functional requirements. IP application functional requirements typically deal with the selection of the algorithms that must be applied to input images in order to extract relevant visual features (color, shape, texture, etc), while non-functional requirements are commonly related to cost, synchronization and timing issues.

As stated in section 2, the functional modeling stage is usually accomplished by means of one of the multiple IP libraries available. Although these libraries are functionally overlapped to some extent, they cover different aspects and consequently, they can be considered complementary. Thus, being able to simultaneously employ a mixture of them for building functional prototypes would be both useful and enriching. However, each IP library uses its own defined data structures, function calling conventions and error handling mechanisms, making it difficult to join them together.

In order to integrate the functionality provided by several of the existing IP libraries and toolboxes [4-7], the IP-CoDe Tool provides a wrapping mechanism which allows building homogeneous and inter-connectable IP components from heterogeneous IP functions. Wrappers allow mapping data representations, adding functionality to (or masking unneeded functionality of) components, and provide a higher level of abstraction to the components [9][10].

The IP-CoDe Tool provides a template for homogeneous IP component generation. In order to fill in this template, the user must provide (1) the signature of the function being wrapped, i.e. the number and type of its parameters, (2) the external interface of the component, i.e. the number and type of its connectors, and (3) the function that links each component connector to one of the function parameters. Once this template has been filled in, the IP-CoDe Tool automatically generates the corresponding wrapper, and thus a new component which is added to a repository of homogeneous and inter-connectable IP components for its latter use.

The IP-Code Tool allows building functional prototypes in a very rapid and intuitive way due to its Graphical User Interface (GUI), which makes it possible to "drag and drop" and interconnect any number of components selected from the repository. Any functional model depicted using this tool may be wrapped as well, in order to build a new higher-level functional component.

When the depicted functional prototype seems to be complete, the user can automatically obtain the corresponding code, which can be compiled and linked in order to produce a running prototype that allow testing the functional behavior using different input data (see Fig. 1).

3.2 Architectural Design

Building an architectural co-prototype implies selecting a specific platform on which to deploy the functional prototype. Thus, at this stage a SW/HW partitioning must be decided and non-functional requirements must be tested. As mentioned in section 2, these tests require a co-simulation tool.

Among the existing simulation tools, Simulink [11] is one of the most popular, mainly owing to its straight forward connection to MATLAB and to its graphical easy-to-use interface. Actually, Simulink can be used as a co-simulation tool, as both SW and HW blocks can be incorporated as a part of the system under simulation. SW blocks can be obtained from the many existing Simulink Toolboxes, and can also encapsulate MATLAB, C/C++, FORTRAN and Ada functions (S-functions). HW blocks can be obtained from the various HW device-specific Toolboxes (e.g. System Generator [12] for the Xillinx FPGA[1]). These are some of the reasons why Simulink has been selected as the co-simulation tool for the IP-CoDe Tool.

After a SW/HW partitioning of the functional prototype has been decided (manually so far), the IP-CoDe Tool automatically deploys the corresponding Simulink co-prototype (mdl file). Each SW block in this co-prototype is then automatically filled in with an S-function automatically built by the IP-CoDe Tool from the corresponding component in the functional prototype. The estimated execution time of each SW block is then calculated, as this information is required for timing and synchronization purposes during the co-simulation (see Fig. 1).

In order to fill in the HW blocks, a library of IP high-level HW components has been created from a set of low-level functions included in the System Generator Toolbox for Simulink [11]. Some other HW blocks have also been included in this library directly from VHDL[2]-cores using a wrapper mechanism to allow their interconnection with the former ones. It is worth noting that all these HW blocks can be directly simulated by Simulink[3] without needing to buy any specific HW device. However, Simulink can also be used for generating and transferring the corresponding VHDL code for each HW block to a target FPGA in order to speed up the co-simulation.

Once the co-prototype is finished, co-simulation allows checking the non-functional requirement fulfillment. For instance, the execution time information provided by the co-simulation allows checking whether the synchronization and timing requirements are met. In the same way, other low-level HW requirements (e.g. maximum working frequency or FPGA area occupancy), can also be retrieved and checked.

3.3 Implementation

After carefully testing and fine-tuning the co-prototype, the final code of the IP application can be automatically obtained. Actually, the code associated to each software block is obtained during the functional prototyping stage, and the VHDL code corresponding to the HW blocks is straightly obtained by the System Generator Toolbox.

4 A Practical Study Case: Detecting Contours in Skin Regions

In order to test the IP-CoDe Tool, a complete IP application has been developed which allows detecting contours in human skin regions contained in color images.

Firstly, a functional prototype of the study case application was graphically built using the depicting and interconnection facilities provided by the IP-Code Tool GUI.

[1] FPGA stands for Field Programmable Gate Array.

[2] VHDL stands for VHSIC (Very High-Speed Integrated Circuit) Hardware Design Language.

[3] System Generator blocks are supplied with a functionally equivalent software Simulink block that can be used for simulation. On the other hand, HW blocks directly obtained from VHDL-cores can also be simulated in Simulink using an external tool, e.g. ModelSim [13].

When the prototype seemed to be finished it was compiled and the corresponding executable version was automatically generated. However, after testing this prototype using several input images, some errors were detected and thus, the prototype had to be modified, fine-tuned and recompiled and again.

Once the functional prototype had been tested and the execution time associated to each component had been measured, the corresponding architectural co-prototype was built by manually selecting which components should be implemented in HW and which ones in SW, thus allowing the Simulink model (mdl file) to be automatically generated and co-simulated.

It is worth noting that, despite the changes and adjustments introduced during the functional prototyping, completing this stage took just a few minutes. On the other hand, it also should be noticed that despite the small size of the images employed, the co-simulation took about fifteen minutes to complete. In the case of full-sized images, the co-simulation time should be measured in hours. This leads to the conclusion that changes introduced at the architectural level have a much greater impact in the deploying time than those performed at the functional level.

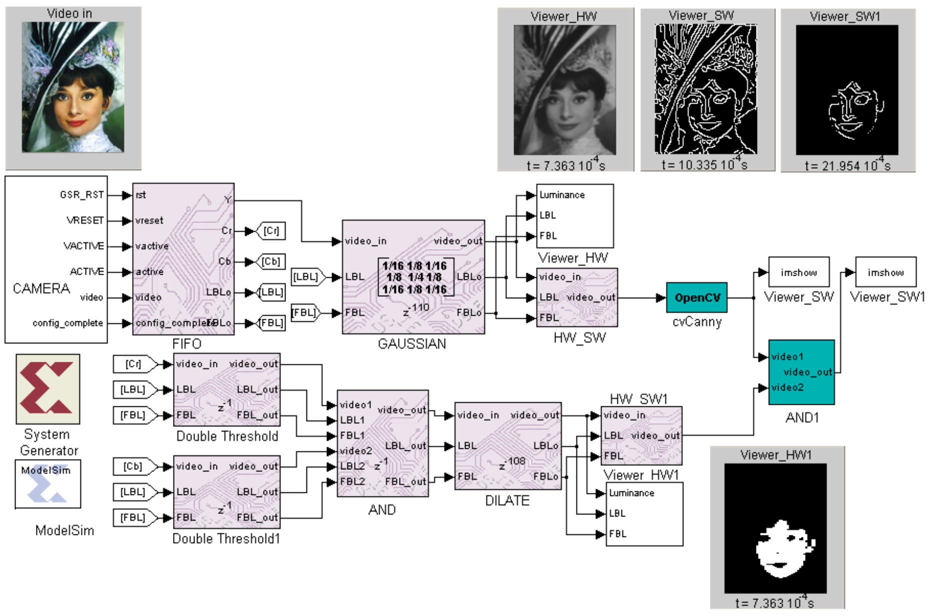

Fig. 2. Simulink co-simulation screenshot. SW blocks are shown in a plain-color while patterned ones denote HW components. Images resulting from each SW/HW processing step are shown together with the corresponding temporization.

The final co-prototype and the co-simulation results are shown in Fig. 2 where different kinds of blocks are shown: patterned blocks represent HW components while plain-color ones correspond to SW elements. HW blocks were obtained from two different sources: System Generator Simulink Toolbox (Xillinx), and a wrapped

VHDL-core obtained from Nallatech [14]. Similarly, SW blocks were obtained from Matlab C/C++ and Intel OpenCV functions. White blocks represent components needed for the simulation (visualization probes or external elements, e.g. a camera model) but that will not have any code associated in the final implementation.

Fig. 2 shows the images resulting from every SW/HW processing step. These images show the instant when they have been generated, thus allowing estimating the temporization of the final implementation. Fig. 3 shows the results obtained by the co-prototype using a different input image. This example proves that the designed application is robust to different skin colors.

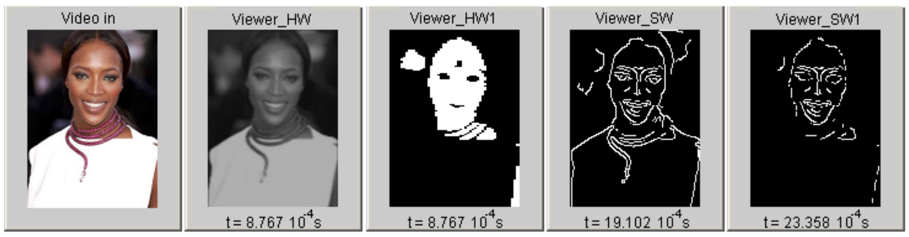

Fig. 3. Results obtained using a different input image. From left to right: original image, smoothed intensity component, skin mask obtained by applying a threshold to the chroma components, Canny contours [15], and logical AND applied to the two previous images.

5 Conclusions and Future Research

This paper presents a new approach to IP application development that covers from functional and architectural prototyping stages to SW/HW co-simulation and final code generation. Building such a comprehensive tool has been possible thanks to the synergy that arises from the integration of several preexistent IP-related products. A complete IP application for contour detection in human skin regions has been wholly developed using the IP-CoDe Tool as a study case.

At present, the IP-CoDe Tool only allows building feed-forward functional prototypes. Extending this functionality to allow the presence of loops will widen the range of applications that could be created. It would also be interesting to find new tools, which being integrated with the existing ones could help automating the SW/HW partitioning to some extent, finding bottlenecks, or detecting which parts of the generated code are more susceptible of being parallelized.

Acknowledgements

This work has been partially supported by the Spanish CITYT Project COSIVA (TIC 2000-1765-C03-02), the European Community Project EFTCOR (DPI2002-11583-E), and the mobility program for researchers of the Technical University of Cartagena (PMPDI-UPCT-2004). We would also like to thank Dr. M. Pinzolas-Prado for his help and support, and A. J. Martínez-Lamberto and J. A. Martínez-Navarro for their collaboration in the IP-CoDe Tool implementation.

References

1. Bovik, A.: Handbook of Image and Video Processing, Academic Press (2000) 749-869.
2. Vicente-Chicote, C., Fernández-Andrés, C., Sánchez-Palma, P.: Automated Visual Inspection Systems Development from a Generic Architectural Pattern Description (in Spanish), NOVATICA, Vol. 171, (2004) 63-65.
3. Bass, L., et al..: Volume II: Technical Concepts of Component-Based Software Engineering, SEI Technical Report CMU/SEI-2000-TR-008. May 2000.
4. Intel® OpenCV. Available: http://www.intel.com/research/mrl/research/opencv
5. Intel® IPP. Available: http://www.intel.com/software/ products/ipp/
6. Matrox© MIL version 7.5. Available: http://www.matrox.com/ imaging/products/mil
7. The Mathworks© Image Processing Toolbox 5.
   Available: http://www.mathworks.com/ products/image/
8. Perrier, V.: A look inside Electronic System Level (ESL) design, CMP United Business Media, EEDesign.com, Article Id. 18402916, March 26 (2004).
9. Dean, J.C., Vigder, M.R.: System Implementation Using Commercial Off-The-Shelf Software, National Research Council Canada (NCR), Report 40173 (1997).
10. Troya, J. M., Vallecillo, A.: Controllers: Reusable Wrappers to Adapt Software Components. Information & Software Technology, Vol. 43(3), (2001) 189-202.
11. Simulink® 6. Available: http://www.mathworks.com/products/simulink/
12. System Generator. Available: www.xilinx.com/products/design_resources/design_tool
13. ModelSim. Available: www.model.com
14. Nallatec sample IP VHDL-core. Available: www.Nallatech.com
15. Canny, J.F.: A Computational Approach to Edge Detection. IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol. 8 No. 6 (1986) 679-698.