IEEE MELECON 2006, May 16-19, Benalmádena (Málaga), Spain

Novel Simulink Blockset for Image Processing Codesign

A Toledo, J. Suardíaz
Dept. Tecnología Electrónica.
U. Politécnica de Cartagena
Cartagena, Spain
{Ana.Toledo, Juan.Suardiaz}@upct.es

S. Cuenca, A. Grediaga
Dept. Tecnología Informática y Computación
Universidad de Alicante
Alicante, Spain
{sergio, gredi}@dtic.ua.es

Abstract—**In this paper we present a novel Simulink blockset which is conceived especially for image processing hybrid systems. The blockset is composed of several libraries with different abstraction levels (high-level, hardware and software) allowing the co-design and simulation of an entire hardware/software system. This background setting is complemented by a set of tools that automates the transition through the different descriptions of the system and helps to make the critical decisions in the partition phase. In addition, a co-design flow is proposed to take advantage of the blockset features. All these items shape a "co-design environment" that allows an easy and quick exploration of the design space to obtain optimised solutions which satisfy the system restrictions.**

I. INTRODUCTION

Image processing is a very extensive field which currently has a great number of applications. One of the main problems when implementing these systems arises from the necessity to fulfil certain requirements which become increasingly strict: processing speed, economic cost, reliability, development time, and so on… In many cases, fulfilling these requirements demands the integration of hardware and software components in the same application. Reconfigurable devices (mainly FPGAs) represent a promising alternative for implementing the hardware side of these hybrid systems due to their high performance and flexibility. However, designing these kinds of systems is difficult and is the subject of a relatively new field of research.

The hardware/software co-design method attempts to concurrently design both sides of the systems, reducing the development time and achieving the performance goals at the same time [1]. A significant part of the co-design problem consists of deciding on the software and hardware architecture for the system and the blocks to be implemented in the software running on programmable components and in specialised hardware. Moreover, both the partition of the system and the integration and synchronisation of the hardware and software are difficult tasks that compromise the viability of the final result

On the other hand, the building of specific hardware is not a simple task, and specialised skills are needed to obtain a correct design in a short time period. Because of this, FPGAs are not the first choice of designers of image processing systems, whose abilities are more focused on the design of "software algorithms".

Different approaches have been proposed for alleviating the work involved in co-designing HW/SW systems, but due to the complexity and magnitude of the process, only a few have been put to practice. There are also some approximations that focus specifically on the domain of image processing. The CHAMPION project [2] provides a collection of libraries of image processing components defined in VHDL and ANSI C. The model defined in C, which works the same as the VHDL model, is employed to generate the functional description of the system and permits simulation within the Khoros environment. It does not actually constitute a co-design environment because there is no software partition. Consequently, the target platform must be integrated exclusively for FPGAs. The CAMERON project [3] is based on the use of a high-level language targeted at SA-C image processing. This language incorporates an optimised compiler for FPGAs which translates the SA-C high-level code into two configurations: C code for a general purpose microprocessor and VHDL code for the co-processing hardware. In this case a division of hardware and software is established but a help mechanism is not provided for the partition phase, it does not allow the integration of pre-designed hardware blocks either.

In our approximation we have two main goals: first, to facilitate space design exploration, enabling a quick evaluation of different partitions in order to obtain the best approximation, and second, to accelerate the development of image processing algorithms in hardware by means of a blockset of optimised components. For base tools we used Mathworks Simulink [4] and Xilinx System Generator (XSG) [5]. Simulink is widely used among developers of

1-4244-0088-0/06/$20.00 ©2006 IEEE

image processing and computer vision systems, enabling a fast learning curve in our scenario. It also has a Video and Image Processing Blockset and a Signal Processing Blockset that can easily be integrated with our libraries. Furthermore, it has a code C generator and is capable of generating executables adapted for the special requirements of several microprocessors. XSG includes a code generator which, starting from the model, automatically builds a netlist in synthesisable VHDL code and a functional model for integration into the Simulink environment.

II. BLOCKSET DESCRIPTION

Creating a suitable HW/SW co-design environment requires the hierarchical nature of the co-design flow to be reflected in the hierarchical structuring of the processing components. In our blockset the components have been grouped into four categories. This taxonomy has been defined with reference to the abstraction level (in the co-design flow), target platform (HW or SW), and functionality:

- *High-level components. These components are involved in the construction of a functional model. Their interface is therefore platform-independent, including only those parameters related to the functionality of the component shared by the SW and HW components.*
- *SW and HW components. These components appear after the partitioning, from the corresponding high-level blocks whose implementation was decided to be in software/hardware. As a result, their interfaces inherit the parameters of the parent high-level block as well as additional parameters related to software/hardware specifics.*
- *External interface components. This category comprises components that simulate the interrelation of the HW platform and external HW devices (such as memories, cameras or processors).*

To facilitate the co-design flow and ensure the inheritance of functionality while allowing finer control as the designer descends on the co-design flow, the interface of the components is divided into two classes: the inter-component interface and the intra-component interface. All parameters pertaining to the functionality of the components are grouped into the intra-component interface. The inter-component interface only contains mechanisms needed for component interconnection. This ensures that the components are connectable and interchangeable.

The *high-level* and *SW* components were created using blocks from several Simulink toolboxes and by incorporating blocks described in C using image-processing libraries such as the OpenCV library. A bottom-up approach has been followed for the HW components, using the basic blocks given in the System Generator.

A wide variety of image-processing algorithms have been implemented. It must be emphasised that the final implementation of these algorithms can be either in HW or in SW, and this implementation takes place by means of an automated process. Table I shows the entire list of available components.

TABLE I. BLOCKSET COMPONENTS

Conversions: *RGBtoYCbCr, YCbCrtoRGB, RGBtoI , Threshold, DoubleThreshold, Brightness_Contrast, Complement, ABS, Type_Convert.*
Filters: *GenericConv*, 3x3*Gaussian*, 3x3*Mean*, 2x1*Gradient*, *1x2Gradient*, *PrewittX*, *PrewittY*, *SobelX, SobelY*, *3x3LaplacianA*, *3x3LaplacianB*, *3x3Sharpen*, *5x5LoG*, *3x3Median, EdgeDetection.*
Morphology: *Erode_Dilate, Opening, Closing, Top-hat, Bottom_Hat, Skel, Thin, Morph.*
Arithmetic, Logic & Geometric: *Add, Subs, Scale*, *LogicalOp, Crop, Resize.*
Outputs: *Viewer, Viewer_RGB, ViewerYCbCr, ToWorkspace), Terminator.*
Inputs: *image, Image_RGB, Image_YcbCr , Video, Video_RGB, Video_YCbCr, USBVideo, USBVideo_RGB, USBVideo_YCbCr.*
Others*: Labelling, SemiLabelling, 0Moments, 01Moments, Peak_ValleyDetection.*

The components offer transparency of implementation details in the high-level design while maintaining the possibility for an advanced user to fine-tune the low-level models using parameters specific to the HW and SW components. For example, the arithmetic precision of the blocks in the data path is specified using Matlab expressions, thereby minimising the hardware used and avoiding the risk of overflow. This means that changing parameters automatically yields an appropriately customised implementation.

As a general rule, the HW components process the input pixels as they come in in raster scan order (from left to right and from top to bottom). This removes the need to have an entire image stored in order to begin processing. In turn, this reduces storage requirements and the number of memory accesses, which can otherwise cause a bottleneck. The components have been designed modularly. This enables the generation of simplified schemes, especially in HW, by eliminating shared structures in blocks that work in parallel.

For example, inside the generic convolution components, an algorithm and the corresponding tool for fully-automatic generation of optimized HW code with no limit on the mask size have been developed. The optimisation has been achieved by means of the parallel processing of the columns of pixels involved in each computation. Additionally, in the case of embedded ALUs were not available, the multipliers can been replaced by shift-registers and adders.

III. CODESIGN FLOW

Figure 1 summarises the co-design flow of the proposed methodology. First, a *functional model* needs to be described from the initial specifications. This model can be built using the *high-level libraries*. This is a critical phase in the design, because any possible mistakes made during the specification phase can lead to important delays in the design process or even to the need to create a totally new model. The functional verification of this high-level prototype can be performed quickly and easily using Simulink facilities.

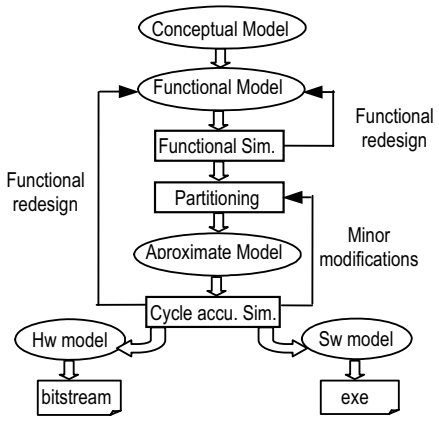

Figure 1. Proposed co-design flow

Next, a partitioning phase is required during which the user can select the parts to be implemented using hardware elements and those to be implemented using software. The *approximated model* includes a transparent depiction of all interfaces required for the communication of hardware- and software-partitioned components. A new simulation can be performed to check that certain specific collateral effects, such as data type conversion or any similar conversion process, do not affect the final result. A hardware-in-the-loop simulation and a fast simulation based on synchronisation indicators or flags were considered when developing this simulation kernel. Important design data must be obtained at this point in order to evaluate the partition. This data should include the estimated latency and the area occupancy derived from simulated hardware components and the estimated execution times derived from simulated software components.

Two different types of model must be created after validating this partition. The first type consists of synthesisable hardware models that can be downloaded onto an FPGA device, and the second type comprises executable software models. All these new models can be simulated again, at low-level, using in this case tools specifically associated with each type of component. Hardware elements can be simulated using a connection between the environment and the ModelSim hardware simulation tool, and software models can be simulated and implemented using a link to the Real Time Workshop tool.

To complement the proposed methodology, several tools have been developed to ease the transition between the co-design stages. The *Hierarchical-Descent Tools* (HDT) allow the automatic generation of the approximated *model* from the *functional model*, including the transparent depiction of all interfaces (*virtual interfaces*) required for communication between hardware- and software-partitioned components. The HDTs also generate the *Hardware* and *Software* models from the previous ones, replacing the virtual interfaces with real ones and adding synthesisable interfaces for the external HW components (cameras, memories, and so on…). Other tools that have been developed include several partitioning aids based on estimations of the execution time of the software and the area occupation and latency of the hardware.

IV. A PRACTICAL CASE STUDY

In order to confirm the versatility of both the blockset and the design proposal methodology, an application was developed that can be used to detect defects in the quality control process of the manufacturing of preserved orange segments. The objective of the proposed system is to reject broken orange segments or segments that are too small to be packed with a particular quality grade. A specification prerequisite of an inspection rate of 25 frames per second was also a factor.

A modular processing architecture was designed using the developed co-design blockset. An initial phase involves modelling the most appropriate data path using high-level processing components and blocks described in C and Matlab. Following the inspection algorithm flow, an image acquisition device initially generates a grey-scale levelled image that has to be processed. Then a Gaussian filter is performed to remove possible noise. The threshold process can now be used to obtain a binary image. This operation takes advantage of the fact that all orange segments appear as dark objects against the luminous background of the conveyor belt, since the backlit illumination method has been taken into account.

The resulting binary image is processed using an opening operation to remove small objects that might remain after the threshold process. At this point the data path is divided into two threads. The filtered binary image is introduced into a block that carries out the moments computation (orders 0 and 1). At the same time, edge detection is performed on the binary image to generate the perimeter of the remaining objects. This result is fed into a block designed to perform an area calculation which is associated with a standard measure of the perimeter length expressed in pixels. Both results feed into a final block in which a neuronal network carries out the classification of the segment according to the standard quality criteria, taking the area and the compactness measurements into account.

After constructing the *functional model* it was necessary to adjust certain parameters in order to achieve the optimum processing algorithm. The proposed library offers the possibility of parameterising components using variables accessible from the Matlab workspace and allows the adjustments to be made without needing to alter the structure or composition of any of the functional models.

During this tuning phase the segmentation threshold and the size and form of the structural elements for the opening operation were adjusted. This was carried out with several functional simulations using different images taken from the actual process, and the offline training of the neuronal classifier. Training and validation data were extracted from

the functional model itself, inserting the special components that download this data into the workspace.

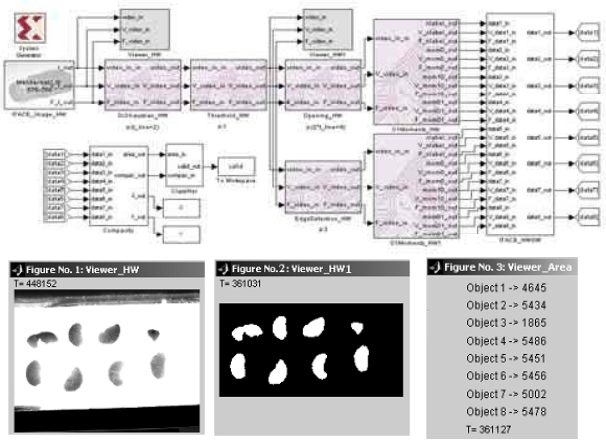

Figure 2. Co-simulation of Aproximated model after partition phase

Once the system had been validated at functional level, a temporal requirement validation was performed. The processing time was estimated, first assuming total system implementation using a software processor (PC). In order to calculate this time specification, the software target was selected for each block of the functional model (default option) and the approximate *model* was automatically generated using the HDTs. The result was an estimated value of 180ms/frm, which is not fast enough for the predefined requirements.

Several partitions were evaluated before finding the best solution. Every partition was generated manually by clicking on the blocks and selecting the final target (HW or SW). The acquisition system was directly connected to the hardware components and a double buffer framework was chosen for the implementation of the interface between software and hardware components. The tools that were developed automatically generated the model and gave us the required estimates. The final partition is shown in fig. 2. As can be seen from the figure, the hierarchical descent tool carried out the replacement of the high-level blocks with the corresponding hardware and software blocks, automatically creating a virtual interface between the partitions (IFACE_HW_SW) and a virtual camera (IFACE_ Imagen_HW) that generates the video signals corresponding to a 778x576-sized grey-levelled image. The results of co-simulation are shown in the same figure. The estimates gave us a latency of 37.05ms for the hardware side (assuming a conservative 50MHz frequency value), and 1.1ms for the software. Because of the concurrent processing of both sides (as a result of the double buffer) the entire process time of one frame is equal to the HW latency, with a rate near to 27frm/s.

After this validation the HDTs generated the *hardware* and *software* models. A "hardware-in-the-loop" schema was adopted to verify the accuracy of these models, avoiding the time-consuming VHDL simulation. As the configurable platform we chose the PCI board Nallatech Ballynuey3 populated with a Xilinx VirtexII FPGA device and the add-on board Nallatech BallyVision which includes the camera interface circuit. Table I summarises the actual results from the implementation tool (Xilinx ISE7.1)

TABLE II. HARDWARE IMPLEMENTATION RESULTS

Virtex2-XC2V 3000 Device	
Slices	30% (15% Flip-Flops)
LUTs	23%
BRAM	45%
fmax	65.1MHz

V. CONCLUSIONS

A novel Simulink blockset especially conceived for image processing domain has been presented. The Blockset facilitates the co-design of hardware/software systems by means of a high-level library and several automatic tools that perform the transition between models with different abstraction levels. Every blockset component has two interchangeable versions with different implementations (hw and sw). The blockset also includes virtual components and hw/sw interfaces for improving the co-simulation of the whole systems, including external devices like video cameras.

As a case study, the development of an industrial system has been presented. The blockset allows a quick exploration of the design space to get the system requirements.

REFERENCES

[1] Giovanni de Micheli, R.K. Gupta, Hardware/Software co-design, Proceedings of IEEE 85 (3) (1997) 349–365. [4] S. Edwards, et al., Design of embedded systems: formal models,validation and synthesis, Proceedings of IEEE 85 (3) (1997) 366–390.

[2] S. Natarajan, et al. Automatic Mapping of Khoros-Based Applications to Adaptative Computing systems. Nashville, TN: Univ Tenesse Press, 1999.

[3] Draper, J. Ross Beveridge, A. P. Willem Böhm, C. Ross, M. Chawathe. Accelerated Image Processing on FPGAs. IEEE transactions on image processing,vol 12. Dec 2003.

[4] The Math Works Inc., http://www.mathworks.com

[5] System Generator: Reference guide, http://www.xilinx.com/