Image Processing systems in FPGA: Components-Connectors Methodology

Miguel Angel García
Directora: Patricia Borensztejn
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
miguel.garcia@gmail.com - patricia@dc.uba.ar

***Abstract*—In this paper a methodology for designing and implementing image processing systems using hardware/software co-design in FPGA is presented alongside a case study: stereo visual odometry solution in FPGA. The methodology focus on devising the system components and their interactions through abstract connectors early in the design process, then use that information to ease the following stages: Implementation and testing in a general purpose processor, Port to embedded platform and test, Speed Optimization, using an iterative optimization strategy to reach real time performance, and finally Experimentation. Components encapsulation produces solutions easy to be integrated into other systems and components implemented in software or hardware easy to be re used as libraries or IPCores.**

I. Introduction

The implementation of image processing systems in FPGA allows designers to save power and reduce area, a key to efficient autonomous embedded systems, replacing software solutions running in high speed processors with others that make use of hardware-software co-design to achieve real-time execution speeds.

Hardware-software co-design adds another level of complexity to the system designing process and, hence, it becomes more difficult for designers to model, build and maintain systems using only implementation languages. Many work flows have been proposed to overcome those limitations and reduce design time, two examples are a flow centered in Simulink tools [1] for modeling the solution and automatically generate HDL code and a methodology proposed by Pedre et al. [2] to build the solution in a general purpose processor and then port it to an embedded platform and optimize it there.

In this paper we propose a methodology for hardware-software co-design of image processing systems in FPGA focused on partitioning the system into its principle runtime units, called components (not to be confused with static code entities, like functions or classes, or physical components, like CPU or RAM), with a balance between functional cohesion and coupling, and clearly defining their interactions through abstract connectors, like shared data; events; procedure call, to describe. Abstract connectors can later be mapped to different implementations in hardware or software, for example: shared data could be implemented as shared variables in external ram or as a dedicated address space in a block ram inside the FPGA; events could be implemented using a software event dispatching library or in hardware using wires; a procedure call could be implemented as a function call in software or with a wire that starts a Finite State Machine (FSM) implemented in hardware.

With focus in a components-connectors view of the system, a view that shows how components cooperate though connectors, many further complex tasks can be simplified:

- Design: The interactions between components can be described early in the design process independently of the hardware/software partition using abstract connectors.
- Prototyping: many implementations for some component could by tried without modifying the rest of the system, maintaining its interface.
- Software implementation and testing: unit test components, replace external interface components with others that simulate external stimulus.
- Optimization: raw profiling techniques can be used at the system level and then the slowest component profiled in detail; Implementing a component in hardware creates an IPCore with semantic meaning and coherent interface, hence reusable in other projects.

The following section details the proposed methodology, including its four stages: A) Design, B) Implementation and testing in a general purpose processor, C) Port to embedded platform and test, D) Speed Optimization and E) Experimentation, and how we plan to use it in a case study: stereo visual odometry solution in FPGA, to estimate a vehicle position from a sequence of images taken by two cameras mounted on itself, I'm developing as part of my undergraduate thesis.

II. Methodology and its Application to a Case Study

The proposed methodology is inspired in the work of Pedre et al. [2] with a components-connectors architecture approach and an iterative optimization process. In this section we are going to review each stage of the design and implementation process and how we plan to apply it in the case study: stereo visual odometry solution in FPGA.

Prior to starting the design, a complete specification of the system, including functional requirements and quality attributes, like processing speed and power consumption, is needed. For the case study we wrote a functional specification detailing the interface between the system and the world and then described critical parts of its behavior using MATLAB,

thanks to its high level functions for image processing. The main quality attributes ordered by priority are:

- Speed: process at a rate of at least 10 FPS.
- Flexibility: be able to use different methods for image acquiring.
- Ease of testing: test the system in a simulated environment prior to mount it on a real vehicle.
- Ease of reuse: integration of the implementation in other systems should be easy and well documented.
- Power consumption: We want to minimize energy requirements to ease the integration of the solution in autonomous systems.
- Area: If possible we want to let room in the FPGA for the implementation of other functions.

After the specification is ready the design and implementation will be done in the following stages:

A. Design

This stage is the key of the methodology, in it the *system architecture* will be modeled with a components-connector view partitioning the system into its runtime components and describing their interactions through connectors.

We identified the following components for the stereo visual odometry solution:

- Stereo image acquiring
- Image features detector
- Features matcher
- 3D Points triangulation
- Translation and rotation calculator for a set of 3D points between two instants

Quality attributes are the main drivers of the partition and connectors selection, hence taking a role on the design from the beginning.

In the case study we will decouple the image acquiring component from the rest of the system using a shared data connector to be able to use different, not yet well defined, image acquiring methods. This way consumers of the images don't need to know details about how the images are captured. This will also ease testing by making it easy to replace the use of data from external cameras with a set of predefined images.

We expect also to use a pipeline architecture in the other components to exploit parallelization and reach the target speed.

After defining the architecture, each component will be designed using standard tools, like UML class and sequence diagrams.

B. Implementation and Testing in a general purpose processor

The designed solution will be coded in C++ and tested in a general purpose processor. We choose C++ because its low runtime support needs and compilers available for the vast majority of CPU architectures makes the software solution developed in this stage serves not only as a reference model but also, possible, as part of the final software that will run in the embedded processor.

Each component will be tested isolated before final integration in an attempt to reduce error propagation and minimize debugging time in hard to catch bugs. Then the complete solution will be tested to detect integration errors.

In the case study we plan to use mock ups to allow the testing of components that depends on not yet developed units, for example image acquiring could be implemented temporary with OpenCV. This approach could also be used for rapid evaluation of different implementations for certain parts, like image features detection or descriptors extraction.

Public data sets, like The KITTI Vision Benchmark Suite [3] [4] [5], will be used to simplify the preparation of input data for the functional tests.

Finally we will use Valgrind Memcheck and Helgrind to check correct memory and threads usage.

C. Port to embedded platform and test

The whole software will be migrated to the embedded processor. The details about how this is done heavily depends on the tools used. For the case study we are using a Digilent Nexys 4 development board with an Artix-7 FPGA from Xilinx, thus we will use Xilinx Platform Studio (XPS).

First we will use Xilinx Embedded Development Kit (EDK) to build the embedded platform formed by a Microblaze processor, an interrupt controller, a controller for the 16MB external RAM available in the development board and a UART for debugging purposes. Address space allocation will be done at this step.

Then the software platform will be configured, using Xilinx Software Development Kit (SDK), making choices related to the configuration of operating system and drivers. We plan to use xilkernel as the OS, it is a lightweight kernel with a POSIX API and is already integrated in XPS.

After that the software solution from the previous stage will be migrated to the embedded platform using Xilinx SDK.

Then the whole solution will be tested in the embedded platform. Taking advantage of the components encapsulation, image acquiring components will be replaced with others that enable us to use previously acquired images, loading them from a computer. Data output from the system will be sent to the computer via UART. This way gathered data can later be analyzed using all the tools available for PC, like graph plotting software.

D. Speed Optimization

The final stage is to optimize the solution in the embedded platform to achieve the desired execution speed. To do this we will use an iterative approach with a cycle of:

- Profile
- Analyze
- Optimize

In the case study we will use Xilinx SDK profiling features based on GNU gprof, this requires that a hardware timer be added to the platform in EDK and that the solution software be compiled with profiling options enabled. The solution with the profiling functionality will be run in the embedded platform processing a sequence of stereo images to collect information. The images will be fed with the same mechanism using on the testing stage.

The gathered data regarding number of calls and time spent in each instruction will be analyzed to detect bottlenecks (single or limited number of components or resources that limits the performance of the entire system) in the execution tree.

The detected bottlenecks will be probably optimized porting the involved components to hardware. Porting all or part of a software component to hardware requires the algorithm to be implemented as a FSM in HDL, encapsulating it in an IPCore, connecting it to the system through a bus, like AXI4, and building a custom software driver to use it. As it can be seen it is a complex task and, thus, we will only do it when there isn't another way to optimize a bottleneck and we will use Xilinx EDK to create the IPCore and the Jiri Gaisler [6] two process method to port the algorithm to HDL.

In the case study we expect the image features detection and extraction to be the slowest components and will probably take advantage of the easy parallelization of the algorithms to implement them in hardware using multiple IPCores.

After an optimization is done the optimized component will be unit tested and the embedded system functionally tested to reduce the risk of introducing bugs while optimizing, then the cycle will be repeated until the target execution speed is reached, at that point the built solution will be ready for use.

E. Experimentation

The solution will be tested in the ExaBot [7], a low cost research and education robot, developed by the team of the Robotics and Embedded Systems Laboratory (Facultad de Ciencias Exactas y Naturales, UBA).

III. Case Study

In this section we will introduce the stereo visual odometry solution for autonomous vehicles localization.

Localization is a key for autonomous navigation, required to accomplish certain goals and avoid hazards, and it's typically performed using a combination of wheel odometry (from rotary encoders), inertial sensing (gyroscopes and accelerometers) and external systems (GPS). All this approaches has limitations: wheel odometry is unreliable in rough terrain (wheels tend to slip and sink), inertial sensors are prone to drift and some common operation environments are GPS denied. Visual odometry estimates vehicle position and orientation from a sequence of camera images, offering a complement to other odometry methods: it is insensitive to soil mechanics, has lower drift rates than mosts IMUs, doesn't need any external system and produces a 6 degrees of freedom motion estimate. Maimone et al. [8] give an interesting review on how visual stereo odometry is used on NASA's MER missions to overcome limitations in odometry calculations in the rovers Spirit and Opportunity.

The process of stereo odometry follows a certain scheme, as described by Sünderhauf et al. [9]:

1) Acquire a pair of images from the left and right camera of our stereo rig.
2) Find interest points (you may also call them features or landmarks if you prefer) in the images.
3) Calculate the 3D coordinates of those interest points.
4) Match the interest points between images taken from different viewpoints.
5) Use the matches to calculate the motion between viewpoints

IV. Work Development

We have started to work in the development of an stereo visual odometry solution on March 2014 as part of my undergraduate thesis.

After two months of reading about computer vision and FPGA design, reviewing prior implementations and investigating tools, we are working in the specification and plan to have it ready by July. Then we will move to the design stage and expect to start the implementation and testing in a general purpose processor in August. By mid September we plan to be porting the solution to the embedded platform and starting optimization on October.

The implementation will be done using a Digilent Nexys 4 development board with an Artix-7 FPGA from Xilinx, hence the Xilinx Platform Studio will be used for the development of hardware and software for the embedded system.

We will try the solution in real robots of the Robotics and Embedded Systems Laboratory part of the Computer Department of the Facultad de Ciencias Exactas y Naturales, UBA.

We wish to release the implementation under an open source license to promote its use in others people works.

V. Expected Results

We expect that applying the proposed methodology in the case study helps to reduce debugging effort thanks to the encapsulation of components and the use of unit and functional testing.

We also foresee that thanks to the components-connectors approach the built solution will be re usable as a whole or in parts thanks. Particularly components optimized in hardware would led to re usable IPCores.

We expect that, thanks to the FPGA technology, the stereo visual odometry solution will satisfy the size and power consumption constraints of most autonomous vehicles.

We believe that using the proposed optimization strategy the detected critical components and hence those that must be

optimized will be the image features detector and the features descriptors extractor. We also anticipate that optimizing those components in software using multi threading, exploiting the algorithm parallelization, will led to a bottleneck in the access to memory and hence the components will need to be implemented in hardware to overcome this issue.

ACKNOWLEDGMENT

This work was made possible thanks to UBACYT 2014-2017 "System on Chip (SoC) in FPGA (Field Programmable Gate Array)" in combination with the Xilinx University Program responsible of the software licenses and a reduced price for the FPGA development boards.

REFERENCES

[1] K. Kintali and Y. Gu, "Model-based design with simulink, hdl coder, and xilinx system generator for dsp," *Mathworks*, 2012.

[2] S. Pedre, E. Todorovich, P. Borensztejn, and T. Krajnik, "A co-design methodology for processor-centric embedded systems with hardware acceleration using fpga," *VIII IEEE Southern Programmable Logic Conference: SPL 2012*, pp. 7–14, March 2012. [Online]. Available: http://publicaciones.dc.uba.ar/Publications/2012/TBKP12

[3] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[5] J. Fritsch, T. Kuehnl, and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," in *International Conference on Intelligent Transportation Systems (ITSC)*, 2013.

[6] J. Gaisler, ""a structured vhdl design method," in fault-tolerant microprocessors for space applications," *Gaisler Research, pp. 41–50*.

[7] J. C. Pablo De Cristóforis, Sol Pedre and A. Stoliar, "Exabot: a mini robot for research, education and popularization of science," in *VI Latin American Summer School in Computational Intelligence and Robotics - EVIC2009, Santiago, Chile*, 2009.

[8] Y. C. Mark Maimone and L. Matthies, "Two years of visual odometry on the mars exploration rovers," *Jet Propulsion Laboratory, California Institute of Technology*, 2007.

[9] N. Sünderhauf and P. Protzel, "Stereo odometry - a review of approaches," *Chemnitz University of Technology*, 2007.