RAPID PROTOTYPING OF IMAGE ANALYSIS ALGORITHMS ON AN ADAPTIVE FGPA ARCHITECTURE

Zahir Larabi, Linlin Zhang[1], Virginie Fresse[1] and Anne-Claire Legrand[2]

[1]Laboratoire Hubert Curien CNRS UMR 5516 (LHC),
[2]Laboratoire d'Informatique Graphique et d'Ingénierie de la Vision (LIGIV)
18 rue Benoit Lauras, 42000 Saint Etienne, France.
phone: + (0033) 477 91 57 93, fax: + (0033) 477 91 57 81,
email: {lin.zhang, virginie.fresse, anne.claire.legrand}@univ-st-etienne.fr

ABSTRACT

The aim of this work is to propose a fast and reliable design flow for the implementation of some image analysis algorithms on an adaptive architecture using an FPGA platform. This adaptive architecture is designed in a Globally Asynchronous Locally Synchronous (GALS) approach so that the hardware resources are stand-alone modules. Any modification only affects the target module, not the entire system. The design flow associated to this architecture includes IP libraries for all reused modules and a high-level development tool called Handle-C for the design of new modules. The image processing designer implements any image analysis algorithm in a reliable way without any hardware specialist.

1. INTRODUCTION

More and more image processing systems must be developed under hard real-time constraints and under harsh environments. FPGAs are increasingly used for such embedded real-time systems because they can achieve high-speed performances in a small footprint. From an FPGA synthesis point of view, the reconfigurable aspect ensures architecture adaptations by functional block modifications. These Systems On Chip (SoCs) are suitable for parameterised, dynamic or even for a class of algorithms. SoCs become more and more popular but the design is complex and time consuming. In most cases, image processing designers are high-level software practitioners. They rarely know one of the available Hardware Description Language (VHDL, Verilog) required for FPGA implementations. On the other hand, these algorithms are first developed using a high-level programming language (C, C++).

Our purpose of this work is to propose an adaptive architecture using an FPGA platform for image analysis applications. The proposed design flow is based on the linear effort property: changing a block to the architecture only depends on the block, not on the size of the reused architecture [1]. The architecture is based on reused modules stored into IP libraries and a high-level development tool used for new blocks.

This paper is organised into 4 further sections. Section 2 introduces the adaptive FPGA-based architecture. Section 3 describes the fast design flow proposed for this architecture. In section 4, an example is given with the implementation of multispectral imaging algorithm and the implementation results are presented in section 5, and Section 6 concludes the paper.

2. ADAPTIVE ARCHITECTURE

Image applications require acquisition operations, storage operations and processing operations. A control operation supervises the entire system. Moreover, the main characteristic of image analysis applications is an unbalanced data flow between input and output flows. The input data flow captures several images meaning that input data correspond to a high number of pixels. The output data flow represents a small number of data.

The presented adaptive architecture is based on all these characteristics.

2.1 Architecture description

The adaptive platform is built on a foundation of reusable Intellectual Property blocks designed to a pre-defined interface. The architecture model is based on separated input data flow and command flow. The reduced output data flow (the result flow) is mixed with command flow (Figure 1).

Using a Globally Asynchronous Locally Synchronous (GALS) approach [2,3], the structure is a set of modules. Logic that constitutes one module is synchronous and each module runs at its own frequency. Communications between modules are asynchronous and they use a handshake protocol design in a wrapper. The wrapper includes two independent asynchronous units. One unit receives frames from the previous module and the other unit sends frames to the following one at the same time.

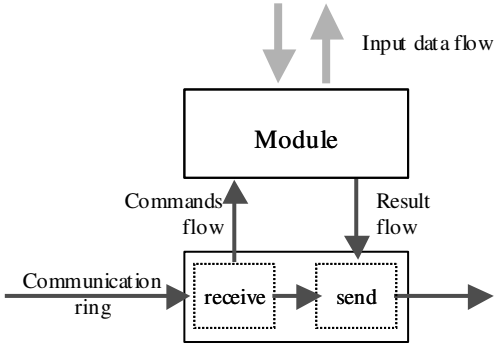

Figure 1 – Model of communication flows

The topology being explored is a hierarchical network built from a unidirectional communication ring. All modules are inserted around this ring. From this model and the communication ring, our adaptive architecture dedicated to image analysis algorithms is proposed in Figure 2.

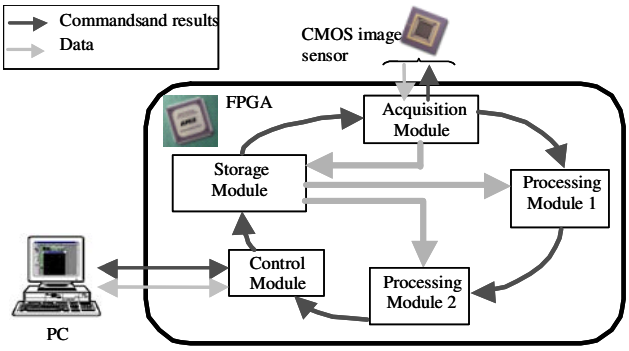

Figure 2 –The proposed adaptive architecture for image analysis algorithms

2.2 Module description

The modular principle can be shown at different levels (Figure 3): one type of operation is implemented by means of a **module** (acquisition, storage, processing…). Each module includes **units** that carry out a function (decoding, control, correlation, data interface…), and these units are shaped into basic **blocks** (memory, comparator…). Special units such as the decode unit and all wrapper units are equal to all modules.

The number and the type of modules depend on the application. As image analysis algorithms require several types of operations, this structure contains several types of modules:

- The acquisition module produces incoming data/images. A CMOS image sensor is used for our prototype. This CMOS image sensor receives configuration information from the acquisition module and the captured images are sent to the acquisition module by the sensor.
- The storage module stores incoming data from the acquisition module. According to the size of data to store, memory banks can be FPGA-embedded memories or external memory devices.
- The processing modules contain the logic required to process images/data. A time-consuming operation can be distributed onto several identical processing modules.

All these modules are supervised by a control module:

- The control module sends commands and empty frames to every/each module through the communication ring. Each frame consists of 4 bytes. As several frames are continuously sent in the ring, empty frames are used by any module to send results back to the control module.

The number of modules is theoretically unlimited for each type of module except the control module. The control of the system is not distributed on all modules but fully centralized on the single control module, which performs decisions and scheduling operations

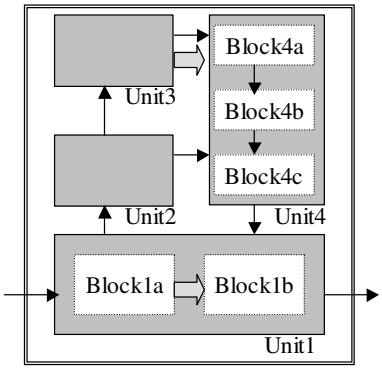

Figure 3 – Structure of the inserted modules

2.3 Modification analysis

Whatever the algorithm previously implemented, modifications or architecture adaptations are required for a new design. Modifications can either be hardware or software. Four levels of modifications are identified:

- **External devices:** for data and image grabber, acquisition devices are interchangeable. This architecture can accept cameras, CCD sensors, and data from a storage device…Since features and format of data depend on the device, and the acquisition module must be adapted.
- **Algorithm:** processing module can accept any processing operations that meet the targeted characteristics previously described (i.e. unbalanced data flow and parallelism).
- **Parameters adaptation:** from a given image analysis algorithm, some parameters can vary: size of full-analysed images, size or location of studied windows, shape of some tools…
- **Parallel operations (scheduling):** for a given algorithm the number of processing modules can vary to improve the parallelism. So the scheduling orchestrated by the control module changes.

According to the type of modifications, only some units/blocks inside modules need to be changed. Modifying one module inside the architecture does not affect other modules, as they are independent. Modules that depend on one or several modifications must be analysed. As a consequence of this reusability, all modules are numbered and classified into two categories:

- Modules that remain unchanged are static modules. Functional blocks can be immediately reused without any modification.
- Modules that are algorithm-dependent or architecture-dependent are dynamic modules. In this case, some functional blocks must be changed.

Table 1 – Static and dynamic module in the adaptive FPGA architecture according to the required modification

	Control	Processing	Acquisit°	Storage
Ext. device	Static	Static	Dynamic	Dynamic
Algorithm	Dynamic	Dynamic	Static	Dynamic
Parameters	Static	Dynamic	Static	Dynamic
Scheduling	Dynamic	Static	Static	Static

3. DESIGN FLOW

A fast and reliable design flow for this adaptive architecture is proposed in figure 4. The input description is the C or C++ algorithm described by the image processing designer. From previous implementation, the image-processing designer identifies the dynamic and static blocks. Static blocks are VHDL IP stored in a ***predefined IP block*** library. Some information about the number of resources and the running frequency are also given for each static block.

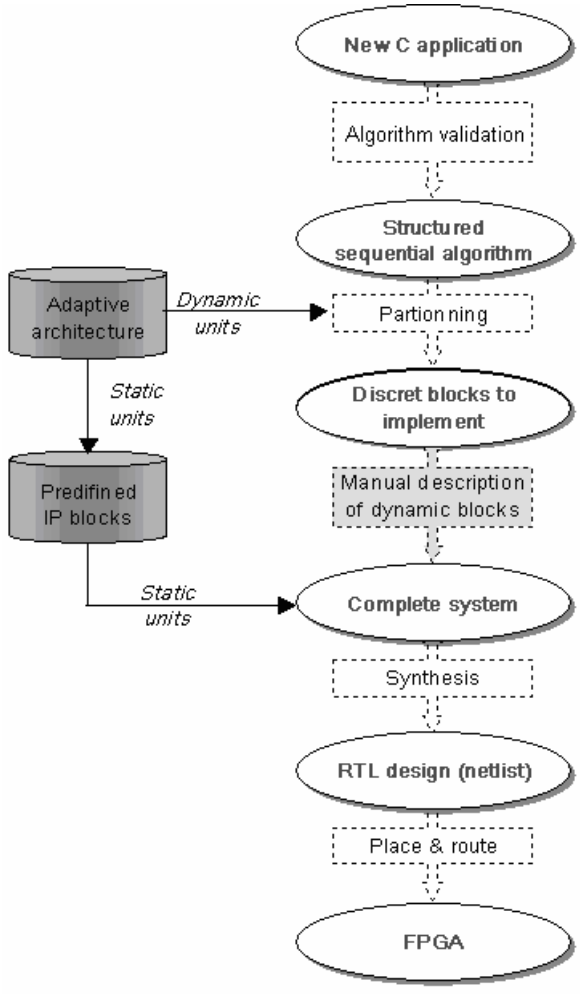

Figure 4 – Design flow associated to the adaptive architecture

Dynamic blocks are designed by means of the DK Design Suite Tool used in the design flow [4]. DK Design Suite uses a C-based language called Handel-C, a subset of ISO-(ANSI-C) with the necessary constructs added for hardware design. Handel-C allows the image processing designer to describe the behaviour of the intended dynamic blocks in the same sense as a software programmer describes the intended behaviour of a processor executing his programs.

The manual translation from C-code to Handel-C is greatly simplified because of the similar syntax and importantly the similar level of abstraction. Several levels of translation are proposed by Celoxica, as shown in figure 5:

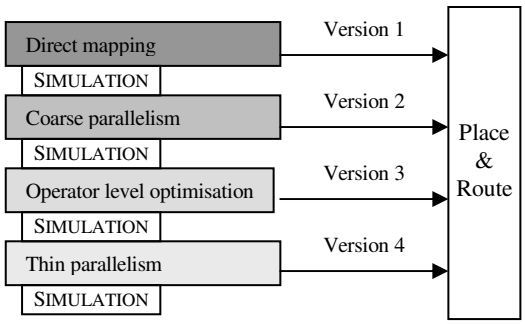

Figure 5 – Design flow for the dynamic units' development with the DK Design Suite Tool

The first version is **a direct mapping** from C code to Handel-C. This task is a "word for word" translation. A **Coarse parallelism** consists of locating tasks that can be simultaneously executed. A thinner version is an **"Operator level" optimisation.** It consists of using, as best one can, specifications of Handel-C language. In particular, some high-level C operators can be replaced by a simple shift in a gate level. The last version is the **fine-grained parallelism,** meticulous analysis of instructions or set of instructions to optimise the execution and propagation times and to detect potential parallelisable operations.

The first three versions can be done by the image processing designer himself. But for higher performances, the fine-grained parallelism modifications require hardware abilities.

The Handel-C translation for the dynamic blocks is the only manual stage in the design flow, represented in grey color in the Fig. 4. All the others are automatically generated thanks to appropriate Place and Route tools. Once the C-code is translated into a Handel-C description, all following stages are automatically achieved.

4. IMPLEMENTATION

A prototype platform with a Stratix II 2S60 FPGA device [5] associated with an IBIS 4 CMOS sensor is used. The first implementation is required for static module characterisation. This architecture consists of four modules, (one module per type of operation) and one CMOS sensor. The previous implemented algorithm had the following characteristics:

- Image size: 320×256 pixels

- Frequencies: $F_{acquisition}$=50 MHz, $F_{storage}$=100 MHz, $F_{control}$=150 MHz, $F_{processing}$=50 MHz.

We assume that an algorithm with these four modules was implemented with the following scheduling:

- Two single exposure image frames are recorded within a short time interval t and t+Δt
- Images are sent to the storage module.
- Sub-images are sent to the processing module
- Results are sent to the control module.

To illustrate our adaptive architecture with a fast and easy example, the new algorithm has the same scheduling as the previous one and the size of images remains identical. We concentrate the study around the processing module as the only dynamic module. Therefore, all modules except the processing module are static modules. For more complex algorithms, the design flow will be the same, only the number and type of modules differ.

4.1 Algorithm: multispectral imaging

Multispectral imaging has emerged as a technology that can guarantee high quality images for many uses in contexts. Offering independence from the illuminate and observer effects makes traditional RGB imaging taken in different contexts inconsistent. This type of imaging is particularly critical for high-end color reproduction such as artwork reproduction, multi-ink printing and hyperspectral satellite observation. Common spectral image processing is the evaluation of spectral sensor responses reconstruction in color system calibration process [6], the spectral image databases indexation [7] and the hyperspectral image identification. For comparing and evaluating spectral application results, metrics on spectra are used to evaluate the closeness of spectral matches.

The aim of the spectra image processing under study is to compare two spectral images. The first step of algorithm is to apply a segmentation stage using spectral and spatial dimensions. The second step is to match the different areas between the two segmented images. Metrics are then computed between the spatial mean spectra of each image areas. Results are combined to conclude on spectral image matching. The previously implemented scheduling is identical to the scheduling required for this algorithm. The only modification lies in the processing operation which is implemented on the processing module. The processing operation to be implemented on the processing module is:

$$F(i,j) = \sum_x \sum_y s_1(x,y) \text{ XNOR } s_2(x-i, y-j)$$

where s1 and s2 respectively present the pixel values of the interrogation windows from image 1 and 2.

This image processing is time consuming and user-definable according to the application.

4.2 Global architecture analysis

Modules have already been implemented and proposed as IP blocks such as the acquisition, control and storage modules. These stored modules in libraries and resource information are given in Table 2.

The only task for the image-processing designer is the development of the processing module.

Table 2 – Predefined resources and frequency for all static modules

	Logic cells	Registers	Mem bits	Frequency (MHz)
Storage	280	422	524 288	100
Acquisition	264	225	0	50
Control	278	265	32	150

4.3 Processing module analysis

For the processing module, only the processing unit itself need to be modified, all the others remain unchanged. White units correspond to the static units and the grey ones to the dynamic units in the presented structure in Figure 6.

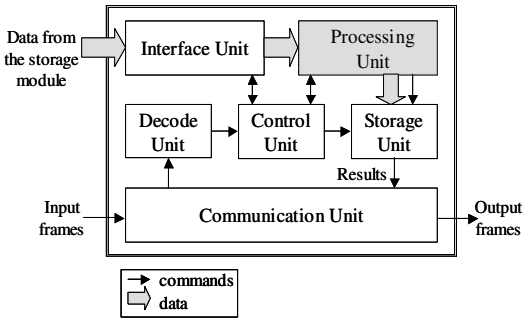

Figure 6 – Structure of the dynamic processing module. Only the processing unit is a dynamic unit

Static units have already been developed and stored in libraries as well. Information about these static blocks is given in Table 3.

Table 3 – Predefined resources for static units inside the processing module

	Logic cells	Registers	Mem bits
Comm.	33	34	0
Decode	12	24	0
Control	42	49	0
Storage	48	63	0
Interface	5	4	0

The used frequency is 50 MHz for the processing module.

5. RESULTS AND INTERPRETATION

The external interface of dynamic units remains identical in all the algorithms. For example, the interface of the processing unit implemented algorithm. This interface is proposed to the image-processing designer and it is presented in Figure 7.

All external signals for this block are inserted in a Handel-C file that can be directly reused by the image-processing designer. Using these signals, the image-processing designer develops the correlation function in a C-code and translates it into a Handel-C version [8].

From the fully manual implementation, all IP blocks except the processing unit are reusable. The original C-algorithm required for the processing unit is translated from C to Handel-C for an automatic implementation [9]. To test the efficiency of the proposed design flow, two ways of implementation have been performed. The first implementation is a full manual implementation. The complete architecture

is described in VHDL. The second implementation is based on the proposed design flow in Figure 7.

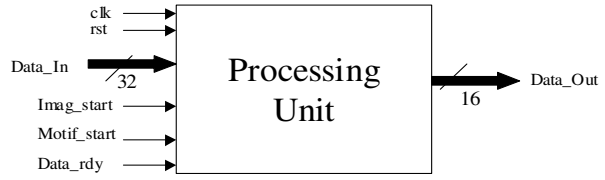

Figure 7 – Structure of the processing unit

The image-processing designer writes the C-function and translates it in a Handel-C language. Two Handel-C versions are proposed. The first version is a direct mapping from C-code to Handel-C and the second version is a coarse parallelism.

Comparative results in term of using resource and the execution time are in Table 4 and Table 5.

Table 4 – Resource results for the dynamic processing unit of the processing module

	LUTs	Flip Flop (FFs)	Mem bits
Version 3	2106	1501	0
Version 4	631	219	1280
VHDL	253	270	1280

Table 5 – Timing results for dynamic processing unit inside the processing module

	Max frequency (MHz)	N° of clock period
Version 3	85	41 923
Version 4	90	30 675
VHDL	135	4 887

.

Once written, the processing unit is included in the complete design (with all VHDL modules) and the complete architecture is automatically generated.

Both versions are efficient because they use the available resources without exceeding the available number. In the same way, the maximum frequency fits with the processing module's required maximum frequency.

On the other hand, the approximate development time remains fast with the Handel-C. Modifications from C-code to Handel-C are difficult to quantify because it can be done in few hours for an experienced person. Nevertheless, the adaptive architecture can reach high performance.

6. CONCLUSION AND PERSPECTIVES

We proposed an adaptive architecture suitable for image analysis applications. The GALS approach is used to provide an architecture whose structure is a set of stand-alone blocks. With this architecture, a fast design flow whose most of stages are automatic is proposed. The only manual stage is the C to Handel-C translation by means of the DK Design Suite [10]. This language is based on ANSI-C so that the image-processing designer can implement new block without any hardware requirement. All previously described blocks are stored in library for an immediate reuse. According to the type of modifications, dynamic modules of the architecture are identified. The designer uses the interfaces to develop the new block. Manual translation only concern the dynamic part, the architecture remains mainly unchanged. The generated IP core does not give optimised results as VHDL description but give sufficient result for most applications suitable to our architecture in a short development time.

REFERENCES

[1] A. Jantsch, *Networks on Chip*, Kluwer Academic Pulishers, Boston, 2003

[2] S. W. Moore, G. S. Taylor, P. A. Cunningham, R. D. Mullins and P. Robinson, "Self-calibrating clocks for globally asynchronous locally synchronous systems" in *Proc. International Conf. Computer Design,* IEEE CS Press, Los Alamitos, Califonia, September 16-20,2000, pp. 73-78.

[3] V. S. P. Rapaka and D. Marculescu, "A mixed-clock issue queque design for globally asynchronous, locally synchronous processor cores" in *Proc. ISLPED2003*, Seoul, Korea, Auguest 25-27,2003, pp. 372-377.

[4] Celoxica "DK Design Suite". www.celoxica.com

[5] Altera Corp. "Altera Stratix 2S60 NIOS II Development board", Datasheet, 2005. http://www.altera.com

[6] E. P. Murphy, "A Testing Procedure to Characterize Color and Spatial Quality of Digital Cameras Used to Image Cultural Heritage", PhD Thesis of Center for Imaging Science, Rochester Institute of Technology, Rochester, NY, 2005.

[7] O. Kohonen, T. Jaaskelainen, M. Hautakasari, J. Parkkinen and K. Miyazawa: "Organizing spectral image database using self-organizing maps" *The Journal of imaging science and technology*, vol. 49 , no 4 , pp. 431 - 441, 2005.

[8] M. J. Peearson, C. Melhuish, A. G. Pipe, M. Nibouche, L. Gilhesphy, K. Gurney and B. Mitchinson "Design and FPGA implementation of an embedded real-time biologically plausible spiking neural network" in *Proc.FPL200*, Tampere, Finland, Auguest 24-26.2005, pp.582-585.

[9] C. Bobda, B. Blodget, M. Huebner, A. Niyonkuru, A. Ahmadinia and M. Majer, "Designing partial and dynamically reconfigurable applications on Xilinx Virtex-II FPGAs using HandelC", Technical Report 03-2004, University of Erlangen-Nuremberg, Erlangen, Germany, November 2004.

[10] A. E. Sjogren and C. J. Myers, "Interfacing Synchronous and Asynchronous Modules Within a High-Speed Pipeline" in *Proc.ISSS98*, Hsinchu, Taiwan, December 2-4,1998, pp.573-583.