EURASIP Journal on Embedded Systems

Design and Architectures for Signal and Image Processing

Guest Editors: Markus Rupp, Dragomir Milojevic, and Guy Gogniat

Design and Architectures for Signal and Image Processing

EURASIP Journal on Embedded Systems

Design and Architectures for Signal and Image Processing

Guest Editors: Markus Rupp, Dragomir Milojevic, and Guy Gogniat

Contents

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 275975, 3 pages
doi:10.1155/2008/275975

Editorial
Design and Architectures for Signal and Image Processing

Markus Rupp (EURASIP Member),[1] Dragomir Milojevic,[2] and Guy Gogniat[3]

[1] *Institute of Communications and Radio-Frequency Engineering (INTHFT), Technical University of Vienna, 1040 Vienna, Austria*
[2] *BEAMS, Université Libre de Bruxelles, CP165/56, 1050 Bruxelles, Belgium*
[3] *Lab-STICC Laboratory, University of South Brittany, CNRS, UMR 3192, 56321 Lorient, France*

Correspondence should be addressed to Markus Rupp, mrupp@nt.tuwien.ac.at

Received 30 December 2008; Accepted 31 December 2008

The development of complex applications involving signal, image, and control processing is classically divided into three consecutive steps: a theoretical study of the algorithms, a study of the target architecture, and finally an implementation.

Today, such sequential design flow is reaching its limits for the following reasons.

(i) The complexity of today's systems designed with the emerging submicron technologies for integrated circuit manufacturing.

(ii) The intense pressure on the design cycle time in order to reach shorter time-to-market, and to reduce development and production costs.

(iii) The strict performance constraints that have to be reached in the end, typically low and/or guaranteed application execution time, integrated circuit area, overall system power dissipation.

An alternative approach to a traditional design flow, called algorithm-architecture matching, aims to leverage the design flow by a simultaneous study of both algorithmic and architectural issues, taking into account multiple design constraints, as well as algorithm and architecture optimizations, not only in the beginning but all the way throughout the design process.

Introducing such design methodology is also necessary when facing the new emerging applications such as high-performance, low-power, low-cost mobile communication systems and/or smart sensors-based systems.

This design methodology will have to face also future architectures based on multiple processor cores and dedicated coprocessors to achieve the required efficiency. NoC-based communications will become also mandatory for many applications to enable parallel interconnections and communication throughputs. Adaptive and reconfigurable architectures represent a new computation paradigm whose trend is clearly increasing.

This forms a driving force for the future evolution of embedded system design methodologies.

This special issue of the EURASIP Journal of Embedded Systems is intended to present innovative methods, tools, design methodologies, and frameworks for algorithm-architecture-matching approach in the design flow including system level design and hardware/software codesign, real-time operating system, system modelling and rapid prototyping, system synthesis, design verification, as well as performance analysis and estimation.

We received 24 submissions for this special issue of which we finally selected 11 for publication.

In the paper entitled "Flexible Hardware-Based Stereo Matching" Kristian Ambrosch et al. propose a novel technique for implementing a flexible block size, disparity range, and frame rate for hardware-based embedded adaptive stereo-vision systems. By reusing existing resources of a static architecture, rather than dynamic reconfiguration, the proposed technique allows both ASIC and FPGA implementations. Using the proposed architecture, the authors show the impact of the flexible stereo matching on the generated disparity maps for the sum of absolute differences (SADs), rank, and census transform algorithms. Finally, the authors quantify the resource usage and achievable performance when synthesized for an Altera Stratix II FPGA.

Back-projection (BP) is a costly computational step in tomography image reconstruction such as positron emission tomography (PET). To reduce the computation time, the paper entitled "High Speed 3D Tomography on CPU, GPU and FPGA" by Nicolas Gac et al. proposes pipelined,

prefetch, and parallelized architecture for PET BP (3PA-PET). The key feature of the proposed architecture is in the original memory access strategy, masking the high latency of the external memory by an efficient use of the intrinsic temporal and spatial locality of the BP algorithm. Proposed architecture is prototyped on a System on Programmable Chip (SoPC) to validate the system and to measure its performances. Time performances are then compared with a desktop PC, a workstation, and a graphic processor unit (GPU).

The paper entitled "A SIMD Programmable Vision Chip with High Speed Focal Plane Image Processing" by Dominique Ginhac et al. describes a high-speed analogue VLSI image acquisition and low-level image processing system based on dynamically reconfigurable SIMD processor array. The chip features a massively parallel architecture enabling the computation of programmable mask-based image processing for each pixel. A 64×64 pixel proof-of-concept chip, built in $35\,\mu$m standard CMOS process, with a pixel size of $35 \times 35\,\mu$m, is shown. A dedicated embedded platform including FPGA and ADCs has also been designed to evaluate the vision chip. The chip can capture up to 10 000 and process up to 5000 images per second.

The paper entitled "Design of a Real-time Face Detection Parallel Architecture Using High-Level Synthesis" by Nicolas Farrugia et al. describes an architecture specified using C language and synthesized using a high-level synthesis tool. Such approach allowed exploration of several implementation alternatives in order to find tradeoffs between processing speed and area of the PE. An instance of 25 PE running at 80 MHz is able to process 127 QVGA or 35 VGA images per second.

The paper entitled "Smart Camera Based on Embedded HW/SW Co-processor" by Romuald Mosqueron et al. describes an image acquisition and a processing system based on a new coprocessor architecture designed for CMOS sensor imaging. The system exploits the full potential CMOS selective access imaging technology because the coprocessor unit is integrated into the image acquisition loop. The acquisition and coprocessing architecture enables the dynamic selection of a wide variety of acquisition modes as well as the reconfiguration and implementation of high-performance image preprocessing algorithms. The experimental results show a large increase of the achievable performances. For instance, the new platform can successfully acquire and fully process up to 50 image-codes per second when applied to the detection and reading of bar codes in the case of a postal sorting application.

For low-volume applications like in professional electronics applications, FPGA are used in combination with DSP and GPP in order to reach required performances. Nevertheless, FPGA designs are static, which raises a flexibility issue with new complex applications. In this scope, dynamic partial reconfiguration (DPR) is used to bring a virtualization layer upon the static hardware of the FPGA. The contribution of the paper "An Evaluation of Dynamic Partial Reconfiguration for Signal and Image Processing in Professional Electronics Applications" by Philippe Manet et al. is to evaluate the interest and limitations when using DPR in real professional electronics applications, and to provide guidelines to improve its applicability. It makes an evaluation of DPR based on experiments made on a set of seven signal and image processing applications carried out in real conditions. It also identifies the missing elements and set of advantages for its use in professional electronic applications. Research directions are also proposed in order to improve its usage.

The paper entitled "Using High-Level RTOS Models for HW/SW Embedded Architecture Exploration: Case Study on Mobile Robotic Vision" by François Verdier et al. deals with the design of a System-on-Chip implementing the vision system of a mobile robot. Specific mechanisms necessary to build a high-level model of an embedded custom operating system able to manage real-time application are described. An executable RTOS model written in SystemC allowing an early simulation of the mobile robotic vision application is also detailed. Based on this model, a methodology is discussed and results are given on the exploration and validation of a distributed platform adapted to the vision system.

Signal processing algorithms become more and more performing and efficient as a result of new developments or at the release of new standards. Textual specifications have been substituted by reference software packages which have become the starting point of any design flow leading to the implementation of the algorithm. Therefore, designing an embedded application has become equivalent to port a generic SW on a, possibly heterogeneous, embedded platform. The paper entitled "A platform for the Development and the Validation of HW IP Components Starting from Reference Software Specifications" by Christophe Lucarz et al. describes a new platform aiming at supporting a step-by-step mapping of a reference software into SW and HW implementations. The platform provides a seamless interface between the software and hardware environments and in addition is supported by profiling capabilities able to analyze the transfers of data between HW and SW parts of the algorithm which help designers to optimize their design.

The paper entitled "A Priori Implementation Effort Estimation for HW Design Based on Independent-Path Analysis" by Rasmus Abildgren et al. presents a metric-based approach for estimating the hardware implementation effort (in terms of time) for an application in relation to the number of linear independent paths of its algorithms. By exploiting the relation between the number of edges and linear independent paths in an algorithm, the authors estimate implementation effort. This approach is implemented in an already existing design framework called "Design-Trotter" and offers a new type of tool to reduce the time-to-market.

The paper entitled "Multiple Word-length High-Level Synthesis" by Philippe Coussy et al. offers tradeoffs between the usage of uniform bit-length designs allowing for traditional automated design flows and nonuniform designs resulting in smaller circuits but to the extent of design complexity. The design flow, based on high-level synthesis (HLS) techniques, automatically generates a potentially pipeline RTL architecture described in VHDL. Both bit-accurate integer and fixed-point data types can be used

in the input specification. The generated architecture uses components (operator, register, etc.) that have different widths. The design constraints are the clock period and the throughput of the application. The proposed approach considers data word-length information in all the synthesis steps by using dedicated algorithms. While providing the same computing dynamic, the total area reduction ranges from 27% up to 80% compared to traditional (none or partial bit-width aware) high-level synthesis flows.

The paper entitled "Accuracy Constraint Determination in Fixed-Point System Design" by Daniel Menard et al. also deals with word length effects. Here, fixed-point system is modelled with an infinite precision version of the system and a single noise source located at the system output. Then, an iterative approach for optimizing the fixed-point specification under the application performance constraint is defined. Finally, the efficiency of this approach is demonstrated by experiments on an MP3 encoder.

Markus Rupp
Dragomir Milojevic
Guy Gogniat

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 386059, 12 pages
doi:10.1155/2008/386059

Research Article

Flexible Hardware-Based Stereo Matching

Kristian Ambrosch,[1] Wilfried Kubinger,[1] Martin Humenberger,[1] and Andreas Steininger[2]

[1] *Austrian Research Centers GmbH-ARC, 1220 Vienna, Austria*
[2] *Institute of Computer Engineering, Vienna University of Technology, 1040 Vienna, Austria*

Correspondence should be addressed to Kristian Ambrosch, kristian.ambrosch@arcs.ac.at

Received 28 February 2008; Revised 5 June 2008; Accepted 20 November 2008

Recommended by Dragomir Milojevic

To enable adaptive stereo vision for hardware-based embedded stereo vision systems, we propose a novel technique for implementing a flexible block size, disparity range, and frame rate. By reusing existing resources of a static architecture, rather than dynamic reconfiguration, our technique is compatible with application specific integrated circuit (ASIC) as well as field programmable gate array (FPGA) implementations. We present the corresponding block diagrams and their implementation in our hardware-based stereo matching architecture. Furthermore, we show the impact of flexible stereo matching on the generated disparity maps for the sum of absolute differences (SADs), rank, and census transform algorithms. Finally, we discuss the resource usage and achievable performance when synthesized for an Altera Stratix II FPGA.

1. INTRODUCTION

The deployment of autonomous systems mainly depends on their capability to navigate in unfamiliar environments as well as their object detection and classification abilities. The main requirement for the implementation of these two features is the availability of reliable three-dimensional information.

A technique able to cope with these requirements is stereo vision. Stereo vision uses two cameras side-by-side and extracts the displacement of the objects caused by the cameras' different viewpoints. The displacement, called disparity, is directly correlated to the distance of the objects, which can be calculated easily using triangulation. This way a three-dimensional depth map can be computed. Unlike other techniques, the depth map computed by stereo vision is typically centered on one of the cameras. Thus, object classification tasks can enhance their recognition performance by directly matching the depth map against the intensity image produced by the camera.

Another feature of stereo vision is that in contrast to other techniques, such as laser range finders, supersonic, or radar sensors, stereo vision comes along without active systems or movable mechanical parts.

Applications of stereo vision are not limited to common robot navigation [1] or distant measures in laproscopic surgeries [2]. They also include the use on autonomous vehicles like at the DARPA grand challenge [3] or the deployment in the NASA/JPL Mars Exploration Rover mission [4].

The core element of a stereo vision system, the stereo matching algorithm, has a high computational complexity. Fortunately, area-based algorithms, and box-filtering algorithms in particular, proved to be very suitable for solutions using hardware-based parallel processing [5], enabling the implementation of real-time embedded stereo vision systems with high frame rates.

Recent works reveal great advances in this field of research. Woodfill et al. proposed the DeepSea application specific integrated circuit (ASIC) [3], enabling the processing of 512×480 images at a disparity range of 52 pixels, a block size of 7×7, and a high frame rate of 200 fps. On the other hand, Lee et al. [6] presented a system processing 640×480 images with a disparity range of 64 pixels at 30 fps using field programmable gate arrays (FPGAs), having a considerably large block size of 32×32 pixels. Then again, the FPGA-based stereo vision system proposed by Perri et al. [7] operates on 512×512 images, having a block size of 3×3, a frame rate of 25.6 fps, and a large disparity range of 255 pixels.

Given the aforementioned applications, embedded stereo vision systems are likely to operate in an environment that is

subject to repeated changes. For instance, in the application of robot navigation for domestic robots, the environment of each room can be very different, demanding different features for the algorithms. A tidy kitchen with unique colored furniture usually has very sparse texture, and requires a highly increased block size for the algorithm. Otherwise, a typical teenager's room can be pretty untidy, including a high number of small objects spread over the room, which could be unrecognizably blurred when using the same block size as for the kitchen. Thus, as soon as the robot enters this room, a smaller block size would be advisable. As the robot has to interact with the furniture or handle single objects, it also has to recognise them at close distances. In these situations the stereo system should be able to recognise close objects rather than operate at a high frame rate. On the other hand, while the robot drives along the hallway, the application of the stereo system could often be reduced to collision avoidance. Here, neither the detection of close objects nor the large block size would be required, rather a high frame rate for covering distances in a speedy way. In any case, even if each of the aforementioned systems implements an algorithm with outstanding features, none of them is able to adapt its features to the situation. For instance, if the DeepSea's high frame of 200 fps is not required for the moment, there is no possibility for taking advantage of a reduced frame rate, like an increase in disparity range or block size.

This is caused by the fact that in contrast to software-based implementations, hardware-based solutions cannot easily change their behavior without deactivating parts of their resources. Jacobi et al. [8] proposed a method to dynamically reconfigure an FPGA for different block sizes using a sum of absolute differences (SADs) algorithm. Dynamically, reconfiguring an FPGA for different behavior is surely a very elegant method, but it drastically increases the complexity of the FPGA design and can only be performed on specific FPGAs. The time needed for the reconfiguration is related to the size of the reconfigured chip area. This time span must not be disregarded for the design of a real-time system that has to meet a hard deadline.

Therefore, we propose a novel technique to adapt block size, disparity range, and frame rate for hardware implementations of area-based stereo matching algorithms. This technique is suitable for FPGAs as well as ASICs, that is, it comes along without dynamic reconfiguration, enabling its implementation on real-time systems with short deadlines.

In Section 2 we give an overview of stereo matching algorithms. Section 3 presents our novel flexible hardware-based stereo matching technique and describes the implementation for adapting the block size as well as the disparity range. Then, we describe how a flexible frame rate follows. The experimental evaluation of our technique is given in Section 4. Here, we show a detailed presentation of our hardware-based stereo matching architecture implementing the proposed technique. Furthermore, we present the experimental results along with a discussion on them. Our final conclusions are revealed in Section 5.

2. STEREO MATCHING ALGORITHMS

2.1. Overview

Stereo matching algorithms are used to solve the correspondence problem of a pair of camera images. They extract the displacement of all objects and generate a disparity map. Therefore, stereo matching algorithms search for the correspondences in the stereo images using feature-, phase-, or area-based matching to calculate the disparities in the images. Feature-based matching searches for characteristics in the images, like edges or curves [9], and calculates the best matches according to their similarities. Phase-based algorithms band pass filter the images and extract their phase [10]. Area-based algorithms take blocks of pixels from both images and calculate their matching costs. This can be done in parallel for all analyzed pixels. When using a constant block size over the whole image, called box filtering [11], these algorithms are especially amenable to parallel and hardware-based solutions.

Color information can be used to improve the matching performance significantly [12]. However, the required hardware resources for processing color images with area-based algorithms on embedded real-time systems are still very high. To keep the focus on algorithms that are suitable for state-of-the-art hardware, we use gray-scale images only.

Based on the stereo taxonomy by Scharstein and Szeliski [11], most area-based algorithms start the matching procedure by applying a neighborhood transformation function T on the primary I_1 and secondary I_2 stereo image, with a defined block size s_t, to achieve better results. When using x and y for the neighborhood region's center pixel, the transformed images' pixel values $t_{1/2_{x,y}}$ are defined as

$$t_{1/2_{x,y}} = T(I_{1/2}, x, y, s_t). \tag{1}$$

Afterwards, the matching costs $c_{x,y,d}$ of the transformed images are calculated for each pixel x, y using the matching costs function C. This is performed for all disparity levels d that are within the disparity range

$$c_{x,y,d} = C(t_{1_{x,y}}, t_{2_{x+d,y}}). \tag{2}$$

Then, the matching costs are aggregated over a defined block size s_a for each disparity level. The aggregated matching costs $a_{x,y,d}$ are defined as

$$a_{x,y,d} = \sum_n \sum_m c_{x+m,y+n,d}, \tag{3}$$

where

$$n, m \in \left[-\frac{s_a - 1}{2}, \frac{s_a - 1}{2} \right]. \tag{4}$$

The matching procedure's total block size s_b is the sum of the transformation function and the aggregation's block sizes s_t and s_a. Finally, the most accurate matching costs have to be searched for. Their position defines the resulting disparity map's pixel value $d_{\mathrm{map}_{x,y}}$. In our work, this search is always defined as the search for the absolute minimum

matching costs value, also called the winner takes all (WTA) algorithm, which has a low complexity when implemented as a very large scale integration (VLSI) circuit [13]. Here, $d_{\max}$ is the maximum value of the disparity range and $d_{\min}$ is the start value for the search, being 0 if the image background is searched for as well

$$\begin{aligned} d_{\mathrm{map}_{x,y}} &= n \mid a_{x,y,n} \\ &= \min(a_{x,y,d_{\min}}, a_{x,y,d_{\min}+1}, \ldots, a_{x,y,d_{\max}}). \end{aligned} \tag{5}$$

For easing our descriptions we will assume $d_{\min} = 0$ in the following, without loss of generality.

When using area-based matching in poorly textured environments, the quality of the disparity map depends mainly on the total block size s_b. Otherwise, a large block size leads to a deformation of the image object edges [14] and high computational costs. Thus, the quality of the disparity maps benefits from algorithms that are able to adapt the block size to the texture coarseness in the images, for example, by increasing the transformation block size s_t or the aggregation's s_a.

For the description and the analysis of our flexible stereo matching technique, we focus on three stereo matching algorithms, that are suitable for hardware-based implementations. These are the SAD, the rank transform, and the census transform. For these algorithms we examined the achieved disparity map quality as well as the resource usage, when implementing the proposed technique using our hardware-based stereo matching architecture. Other common algorithms, like the sum of squared differences (SSDs) or the normalized cross correlation (NCC) are not analyzed, since they are well known for having too high resource usage when implemented on hardware [5].

2.2. Sum of absolute differences

SAD [15] is a simple and popular algorithm for FPGA- or ASIC-based stereo matching. One of the early implementations is [16], while there exist various recent implementations as well [8, 17–20].

SAD is an area-based algorithm that gets along without a transformation function, which reduces its resource usage when implemented in hardware. Its cost function C_{ad} is the absolute difference of the pixel values

$$C_{\mathrm{ad}}(t_{1_{x,y}}, t_{2_{x+d,y}}) = \left| t_{1_{x,y}} - t_{2_{x+d,y}} \right|. \tag{6}$$

2.3. Rank transform

The rank transform, originally proposed by Zabih and Woodfill [21], and more recently used by Banks and Bennamoun [22], is an algorithm that is very robust to local brightness variations [23]. It is very suitable for hardware implementations as shown in [24], where an FPGA implementation for the application of target tracking is used, which is quite similar to finding correspondences in stereo vision. However, it is rarely used for stereo matching. One of its implementations for stereo matching is shown in [5].

The transformation function of the rank transform T_{rank} is the number of pixels in a local neighborhood region N, having the dimensions of the block size s_t, with a smaller intensity value than the center pixel's value $I_{1/2}(x, y)$

$$\begin{aligned} &T_{\mathrm{rank}}(I_{1/2}, x, y, s_t) \\ &\quad = \left|\left| \{n, m \in N \mid I_{1/2}(n, m) < I_{1/2}(x, y)\} \right|\right|. \end{aligned} \tag{7}$$

The cost function is the absolute difference (6) as used in the SAD algorithm.

2.4. Census transform

The census transform [21], an algorithm with even higher robustness than the rank transform [25], is often used in hardware implementations because it offers a good tradeoff between resource usage and quality. Some of the earlier implementations on FPGAs are [26, 27]. More recently, the census transform has been used for ASIC implementations in [3, 28].

The transformation consists of a comparison function ξ, which is used to compare the center pixel's value i_1 with the pixel intensity values i_2 in the neighborhood region N,

$$\xi(i_1, i_2) = \begin{cases} 1 \mid i_1 > i_2, \\ 0 \mid i_1 \leq i_2. \end{cases} \tag{8}$$

Its result, 1 if the center pixel is larger, and otherwise 0, is then concatenated ($\bigotimes$) to a bit vector. Thus, the transformation function T_{census} is defined as

$$\begin{aligned} &T_{\mathrm{census}}(I_{1/2}, x, y, s_t) \\ &\quad = \bigotimes_{[n,m]\in N} \xi[I_{1/2}(x, y), I_{1/2}(n, m)]. \end{aligned} \tag{9}$$

The cost function C_{census} is defined as the hamming distance over the bit vectors

$$C_{\mathrm{census}}(t_{1_{x,y}}, t_{2_{x+d,y}}) = h\,\mathrm{dist}(t_{1_{x,y}}, t_{2_{x+d,y}}). \tag{10}$$

3. FLEXIBLE HARDWARE-BASED STEREO MATCHING

3.1. Flexible disparity range

To enable the disparity range's flexible calculation we split it into r partitions with d_r pixels and compute every single partition separately as depicted in Figure 1. That is, for each image line, r matching rounds are performed. The position of the partition's best match and its aggregated matching costs are stored in memory. Once all partitions are computed, the stored results are searched for the partition with the smallest matching value and its position value is selected as the resulting disparity. Thus, the disparity search is redefined as

$$\begin{aligned} d_{\mathrm{map}_{x,y}} &= n \mid a_{x,y,n} \\ &= \min[\min(a_{x,y,0}, \ldots, a_{x,y,d_r-1}), \ldots, \\ &\qquad \min(a_{x,y,(r-1)\cdot d_r}, \ldots, a_{x,y,r\cdot d_r-1})]. \end{aligned} \tag{11}$$

Using (11) for the disparity search, the maximum disparity $d_{\max}$ is defined by

$$d_{\max} = r \cdot d_r - 1. \tag{12}$$

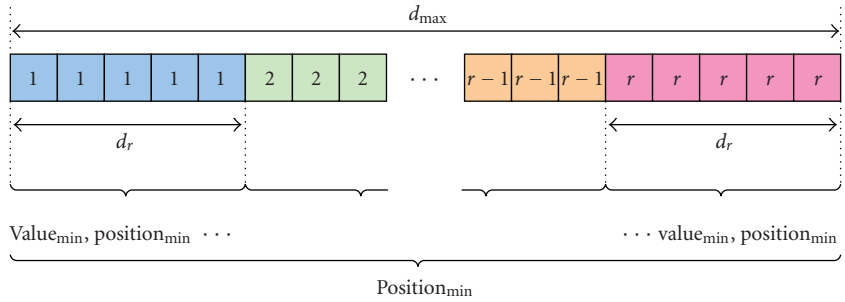


Figure 1: Disparity range split into r partitions.

Now the flexible disparity range is achieved by varying the number of calculation rounds r. The highest possible disparity range is only limited by the memory used to store the calculation rounds' interim values, that is, the position and costs of the best match for each pixel in the current round.

The key to a well-performing implementation is to pipeline the whole calculation. Pipelines enable a highly parallel execution but also require initialization times at the start of computation. The reason for computing one line per round, instead of one pixel per round, is to keep these pipeline initializations to a minimum. The price for this performance increase is a greater demand for memory, because the interim values, that is, the partitions' minimum values and positions, of the whole line need to be stored before the global minimum can be searched for.

Figure 2 presents the block diagram for the two-staged implementation of our round-based calculation technique.

The calculation of the partitions' interim values forms the first stage of the circuit. Here, the data of the stereo images are provided in the chip's internal memory. The stereo matching circuit reads the image data from the memory and calculates the matching costs for the disparity range partition of the current round only. Then the matching costs' minimum value is searched for. The selected minimum value and its position are stored in the internal memory. For better support of the memory architecture within FPGAs, which memory blocks have a fixed size and position on the chip surface, we use two separate blocks of memory for the matching costs' value and position. This way, merging single memory blocks with different positions on the chip can be avoided, or at least kept at a minimum. For each calculation round, the interim values are stored in a different pair of memory blocks. The current memory block is selected using a demultiplexer that has the calculation round counter as the input.

After all r calculation rounds have been performed, the final disparity value can be searched for by the extraction stage. Here, the pairs of block memory holding the interim values are accessed using a multiplexer that has the extraction round counter as the input. Once the last partition value has been analyzed, the smallest value's position is read and selected as the final disparity value.

For better resource utilization, we suggest duplicating the interim values' memory and assigning one set of memory to each stage, separating the two computations. By switching the assignment of the memory sets after each computed line, the disparity can be extracted for the previous line while the next interim values are calculated for the current line. This way, the two computations can be performed in parallel, resulting in a two-staged macropipeline. Figure 3 shows the pipelined circuit, where the assignment of the memory sets is performed by connecting it to the calculation stage using a de-multiplexer and a multiplexer for the extraction stage. The assignment of the sets is performed using the line counters' least significant bit (LSB) and negating it for the extraction stage.

3.2. Flexible block sizes

For the resource-efficient implementation of a flexible block size it is necessary to reuse the computation blocks, changing mostly the interconnects only, rather than switching between different computation blocks for each block size or deactivating large parts of them. To reuse a computation block for a larger block size, it is necessary that the computation can be split into two or more smaller functions similar to the predefined, using different input parameters. The results of these functions have to be merged afterwards. For a reuse of resources it must be guaranteed that the further computation is performed in the same way as for the smaller block size.

The transformation function (1) has the block size as a parameter and is therefore suitable for a deeper analysis. It seems possible to use two transformation functions and merge the results for the neighboring blocks, but the transformation function is not defined in general. Examining our three presented algorithms, for the rank transform an aggregation would apply as a merging function, while for the census transform the bitstream had to be merged. The SAD algorithm does not even have a transformation function. This leads to the conclusion that it is not possible to define a generic function for merging the transformation function's results and a technique that reuses the transformation function blocks would be restricted to specific algorithms.

The cost function (2) does not have the block size as a parameter and therefore does not directly correspond to it.

The aggregation function (3) is similar for all area-based algorithms [11]. Using a larger block size for the aggregation function leads directly to a higher number of matching cost summations. When the larger block size's number of summations is a multiple of a smaller one, the aggregation

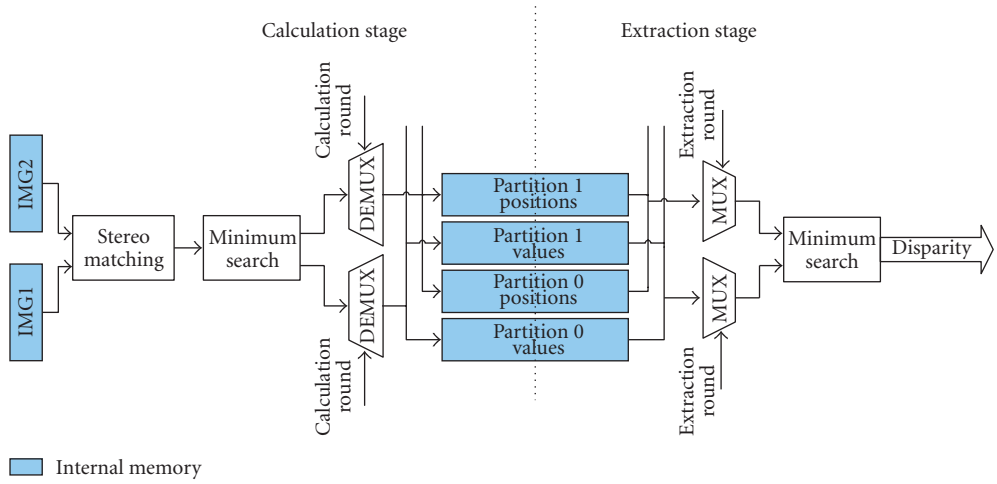


Figure 2: Round-based disparity calculation.

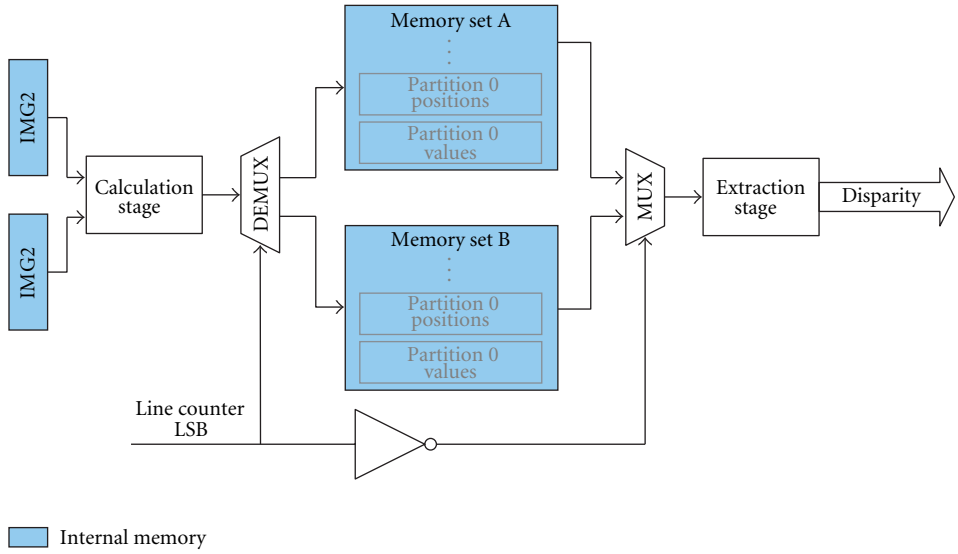


Figure 3: Pipelined disparity calculation.

blocks can be reused, having different input parameters. Equation (13) shows the new aggregation function for a horizontal enlargement of the aggregation block size from $s_{a_{\text{small}}}$ to $s_{a_{\text{large}}}$:

$$a_{\text{large}_{x,y,d}} = \sum_{n}\sum_{m_1} c_{x+m_1,y+n,d} + \cdots + \sum_{n}\sum_{m_k} c_{x+m_k,y+n,d}, \quad (13)$$

where

$$n \in \left[-\frac{s_a - 1}{2}, \frac{s_a - 1}{2} \right],$$
$$m_l \in \left[-\frac{s_{a_{\text{large}}} - 1}{2} + (l-1)s_{a_{\text{small}}}, -\frac{s_{a_{\text{large}}} - 1}{2} + l \cdot s_{a_{\text{small}}} - 1 \right], \quad (14)$$

k is the block size multiplication factor, and $a_{\text{large}_{x,y,d}}$ are the aggregated matching costs when using the enlarged block size.

As can be seen, the new aggregation function's sub-aggregations are similar to the original one in (3), with the exception that the boundaries are calculated differently. This results in the requirement for different inputs for the aggregation blocks.

The input parameters of the aggregation function are delivered by the cost function. Thus, switching the input parameters has to be performed before the cost function.

The results of the small aggregation units can be merged by $k - 1$ summations, when using the larger block size. Additionally, the input and output values have to be switched, except for the input of the primary image for the first cost function of each larger block, which always stays the same. This leads to $3 \cdot k - 1$ required additional k-input multiplexers.

Figure 4 shows the block diagram for the implementation, using $k = 2$. Here, the data of the transformed stereo

images are supplied in the chip's registers. These registers have to be shifted when switching to the next pixel's computation. For refilling the resulting emptied registers, the next pixels' transformation has to be performed preliminarily. The inputs of the cost functions and the outputs of the aggregation blocks are switched using multiplexers, which have the selected block size as the input.

When the smaller block size is selected, that is, the multiplexer's input 0 is selected, the even cost functions and aggregation blocks are connected, calculating the matching costs for the even disparity levels. Likewise, the odd cost functions and their aggregation blocks are connected for calculating the costs for the odd disparity levels.

As soon as the larger block size is selected, that is, the multiplexers' input 1 is selected, the even and the odd aggregation blocks, as well as the corresponding cost functions, are combined for calculating the same disparity level together. Now, the even part is used to calculate the left part of the larger block and the odd part calculates the right part. Both aggregation blocks' results are merged by a single summation and output as the even aggregation block's result. The odd aggregation block does not output a valid result in this situation. Therefore, its output is set to a higher value than the maximum achievable matching costs, for example, the maximum value plus one. This way, the WTA algorithm is forced to ignore these results, since the even aggregation block's valid results are by definition smaller. In any case, the output position of the WTA algorithm is now scaled by two and has to be adjusted by cutting off its LSB, dividing it by two.

The enhancement of the block size goes along with a reduction of the calculated disparity levels per calculation round. That is, the partition size d_r is divided by k. To ensure a constant disparity range, the number of calculation rounds has to be increased by the same factor. This leads to a lower frame rate of the stereo matching algorithm, when using the larger block size.

3.3. Flexible frame rate

The flexibility of the frame rate results from using a flexible disparity range and block size. When the demand for a higher frame rate exists (e.g., the robot operates at higher speed) the disparity range and the block size of the stereo matching algorithm can be reduced to ensure a shorter processing time.

Thus, the present frame rate of the system in fps is defined as

$$\text{fps} \cdot \frac{d_{\max} + 1}{d_r} \sim \text{const}, \tag{15}$$

where

$$d_r = d_{r_{\max}} \cdot \frac{1}{k_{\text{akt}}}, \tag{16}$$

k_{akt} is the currently selected block size multiplication factor, and $d_{r_{\max}}$ is the achievable disparity per round when $k_{\text{akt}} = 1$. The constant factor in (15) depends on the processing speed of the implemented design.

4. EXPERIMENTAL EVALUATION

4.1. Evaluation architecture

To evaluate the performance of our flexible stereo matching technique, we implemented a generic hardware-based stereo matching architecture that is suitable for different area-based matching algorithms.

Our architecture consists of three major pipeline stages as depicted in Figure 5, which are capable of working in parallel after a primary initialization and synchronization phase. These three pipeline stages are the input, the calculation, and the extraction stage. The interfaces between those stages are formed by memory blocks, each holding a full image line.

The application of our hardware-based stereo matching architecture includes robot navigation. This task requires the constant collection of data about the surrounding background to enable the localization of the robot's position. Thus, for the implementation of our architecture we assume that the disparity range has a fixed minimum value of $d_{\min} = 0$ and therefore the background of the images is always searched for as well.

Furthermore, for the last $d_{\max}$ pixels, there would be insufficient image data available in the secondary image to perform the matching for the whole disparity range. Since this would lead to border effects in the disparity map, we do not waste resources on the matching of this image region and simply stop the calculation at this point. Thus, the output disparity map's image width is reduced by $d_{\max}$ pixels. Also, the border regions of the image that would have border effects caused by the total block size s_b are not processed. These are the first and the last $(s_b - 1)/2$ image lines as well as the lines' first and last $(s_b - 1)/2$ pixels.

For a less complex implementation, we assume that the images used for the matching procedure are rectified [29] by an additional preprocessing stage. Thus, the search for correspondences is limited to the image lines.

The first stage of our architecture is the input stage. It reads the incoming image data from the input port and stores it in the internal memory. Here, we use one memory block for each single image line. The number of stored lines is defined by the total block size s_b plus one additional line to enable the implementation of a cyclic memory. This way, the input stage can access one memory block in write mode while the calculation stage reads the others, ensuring maximum independence of each stage.

The calculation stage performs the round-based matching as soon as the input stage has stored the first s_b image lines. It reads the image data necessary to perform the next pixel's matching from the memory. If the algorithm used consists of a transformation, the pixels are transformed instantly and stored at the end in the image buffer. This image buffer holds the transformed values for the following cost function. It has a height of s_a pixels, which is the aggregation block size, and a width of $d_r + s_a$ pixels. This image buffer is shifted by one pixel column after each pixel's computation, that is, at each clock cycle. Then our flexible block matching technique is applied as described in Section 3.2. The resulting aggregated matching costs of the current disparity partition

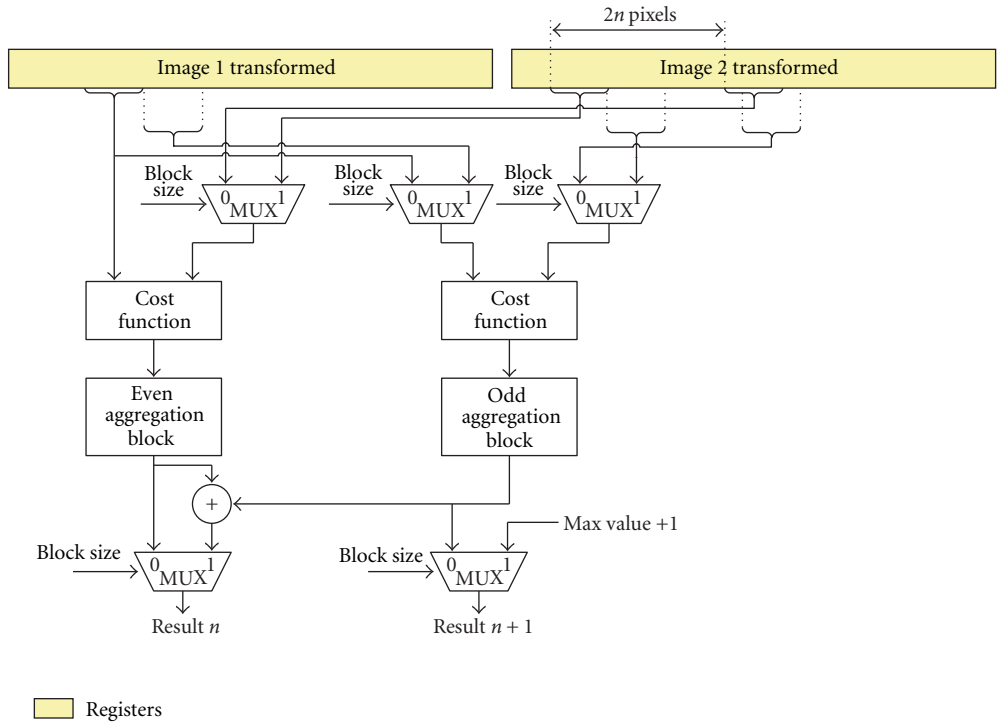


Figure 4: Flexible block size using $k = 2$.

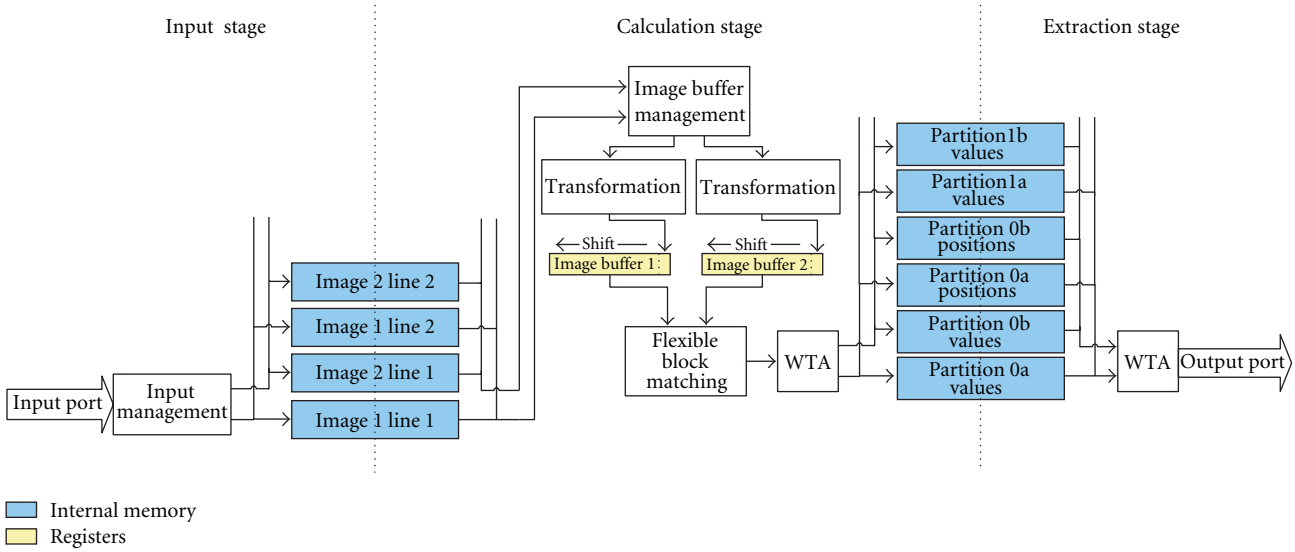


Figure 5: Evaluation architecture.

are the input for the following WTA algorithm. This WTA algorithm is implemented as a binary search tree as shown in Figure 6 and therefore fully pipelined. It outputs the minimum value and its position. Both are stored in the interface memory.

The pipelined implementation of the calculation stage results in a processing time of one clock cycle per pixel as soon as the pipeline is filled. Depending on the algorithm implemented and the partition size, this initialization time $t_{\text{init-calc}}$ varies between 50 and 70 clock cycles for each calculation round. Thus, the computation time of one image line is $r \cdot (\text{image\_width} - d_{\max} - s_b + 1 + t_{\text{init-calc}})$ clock cycles.

As described in Section 3.1, the interface between the calculation and the extraction stage is formed by two sets of memory blocks, which are switched between the stages. Hence, there exist two pairs of memory blocks for each round's interim values, namely a and b in Figure 5.

The extraction stage reads the results of all matching rounds and again selects the best match by using the WTA algorithm. Here, the WTA is implemented in series. Thus,

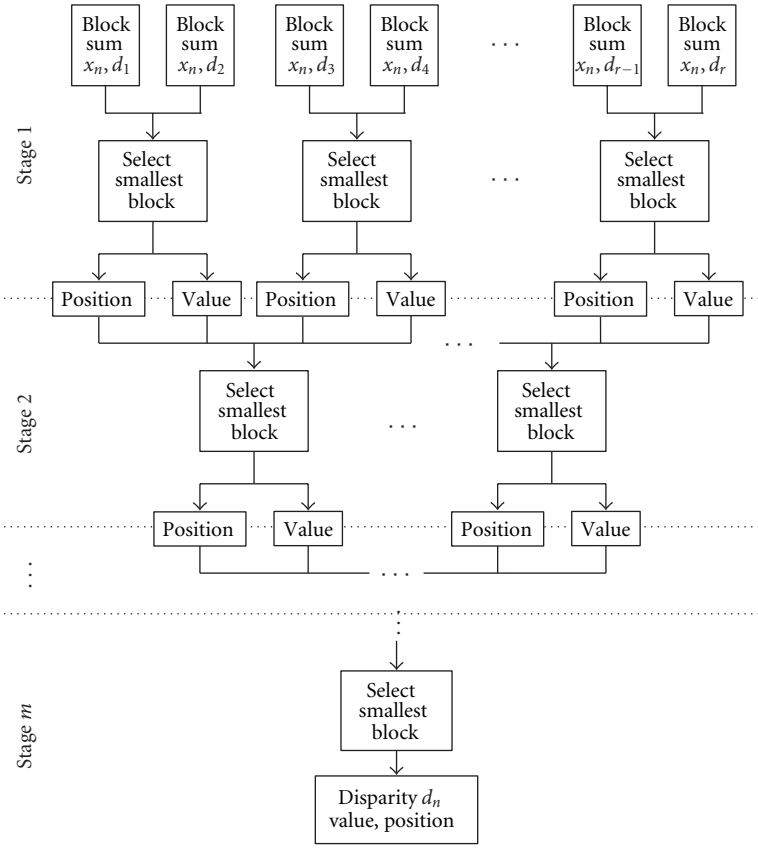
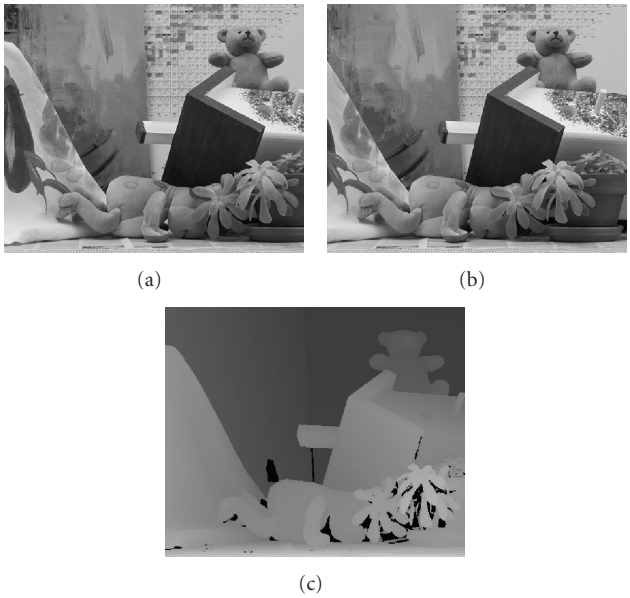


Figure 6: Tree-based WTA.



(a) (b)

(c)

Figure 7: Teddy images from the Middlebury dataset. (a), (b): camera images; (c): ground truth image.

the value and position of just one round is evaluated per clock cycle and the minimum value stored in the registers. When evaluating the pixel's last round, either the stored position or the current round's position is set on the output port as the resulting disparity. The serial implementation of the WTA algorithm is more resource aware than the parallel one of the calculation stage. The extraction stage needs to be initialized only once per line, in contrast to the calculation stage, which has to be initialized r times per line. Furthermore, the pipeline of the extraction stage is much shorter than that of the calculation stage, having an initialization time $t_{\text{init-extr}}$ of two to five clock cycles, depending on the chip's memory access delay. Thus, the extraction stage takes $t_{\text{init-extr}} + r \cdot (\text{image\_width} - d_{\max} - s_b + 1)$ clock cycles per image line, which is less than the calculation stage's computation time.

The total processing time of the architecture is given by the time to read the first s_b image lines, the processing time of the calculation stage for the following image_height $- s_b + 1$ image lines, and the extraction stage for the very last image line.

4.2. Test configuration

We evaluated our algorithms using the teddy images, illustrated in Figure 7, from the Middlebury stereo dataset [30], since this image set offers high- as well as low-textured surfaces. This image set has a resolution of 450 × 375, a maximum disparity of 60 and is converted to 8-bit gray scale. The ground truth of these images, which is defined as

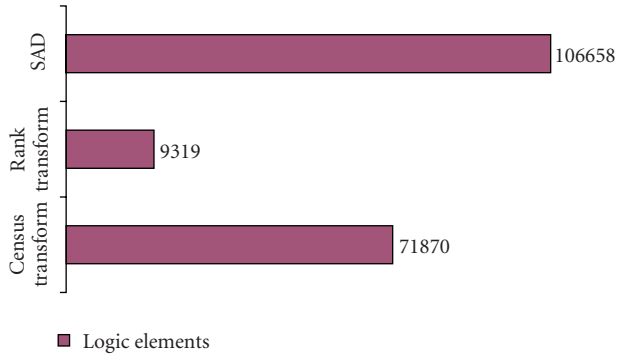


Figure 8: Required logic elements of the algorithms.

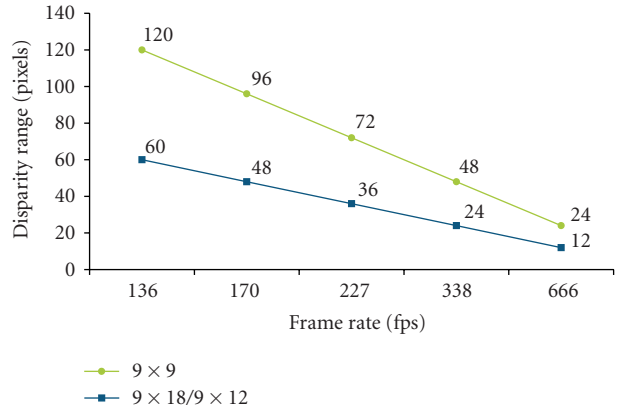


Figure 9: Frame rate vesus disparity range for block sizes $s_b = 9 \times 9$, 9×18 for SAD and 9×12 for rank and census transform, respectively.

the measured reference disparity map, is scaled by factor of 4. Thus, we also scaled our disparity maps this way to ensure a comparable result.

For the block size multiplication factor we used $k = 2$, since we expect that doubling the horizontal block size will be the most typical application. The disparity range was 120 to avoid distortions caused by a too small disparity range for both block sizes. We used a round size of 5, resulting in a partition size of 24 disparity levels per calculation round.

To demonstrate the need for a flexible disparity range, we also evaluated the image sets using a disparity range of 60. With the larger block size, this resulted in an effective disparity range of 30, which is half of the maximum disparity in the image sets.

The SAD algorithm was configured to $s_b = 9 \times 9$ for the smaller and $s_b = 9 \times 18$ for the larger block size. The rank and census transform also used $s_b = 9 \times 9$, distributed to $s_t = 7 \times 7$ for the transformation and $s_a = 3 \times 3$ for the aggregation, leading to $s_a = 3 \times 6$ and $s_b = 9 \times 12$ for the larger block size.

To evaluate the algorithms' quality, we performed a left/right consistency check and calculated the root mean square (RMS) over the deviations to the ground truth image as well as the found pixels and correct matches, which are within a maximum deviation of one pixel.

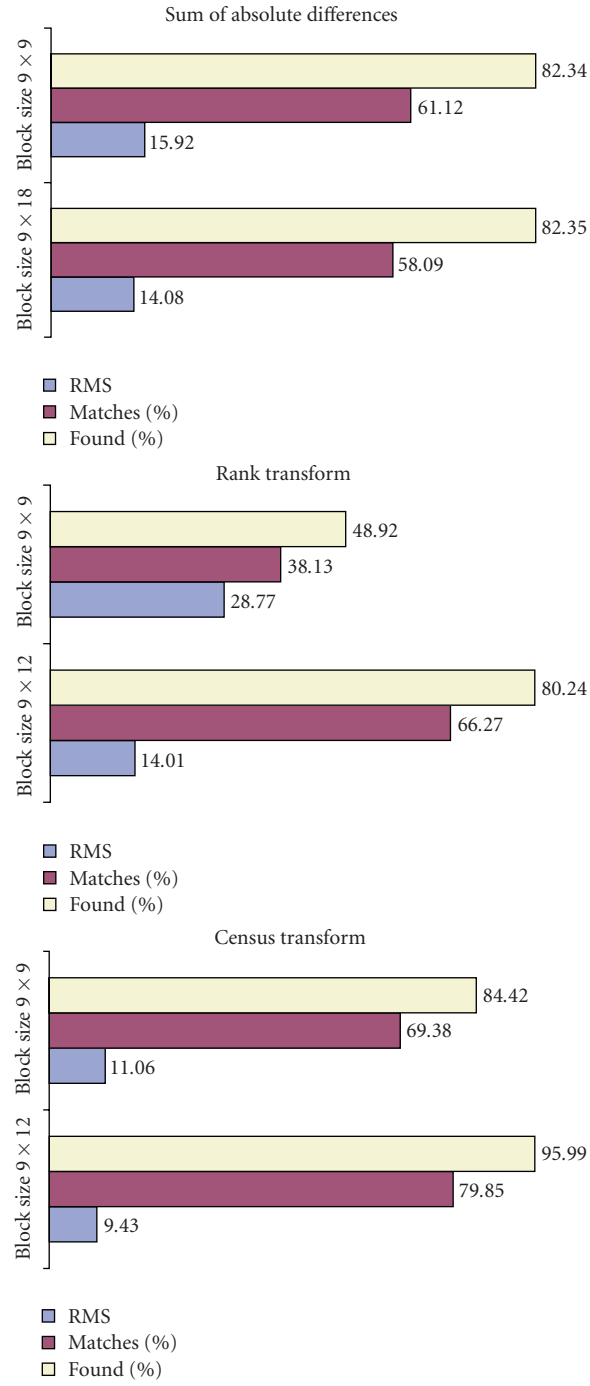


Figure 10: Evaluated algorithm quality for block sizes $s_b = 9 \times 9$, 9×18 for SAD, and 9×12 for rank and census transform, respectively.

4.3. Experimental results

We synthesized the implemented algorithms for an Altera EP2S130 with Altera Quartus II in order to analyze the algorithm's resource usage. All of them had a total memory consumption of 425 984 bits. Here, the image memory requires 81 920 bits for storing 10 lines for both images. Furthermore, storing the interim position requires 114 688

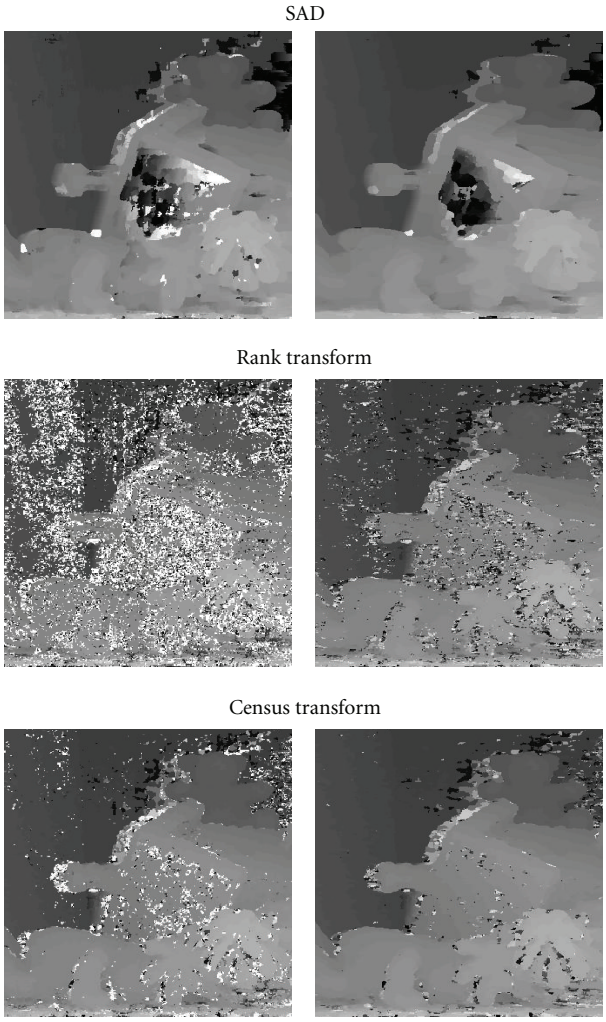


Figure 11: Disparity maps generated from the Middlebury dataset's teddy images. Left: block size $s_b = 9 \times 9$; right: block size 9×18 for SAD, and 9×12 for rank and census transform, respectively.

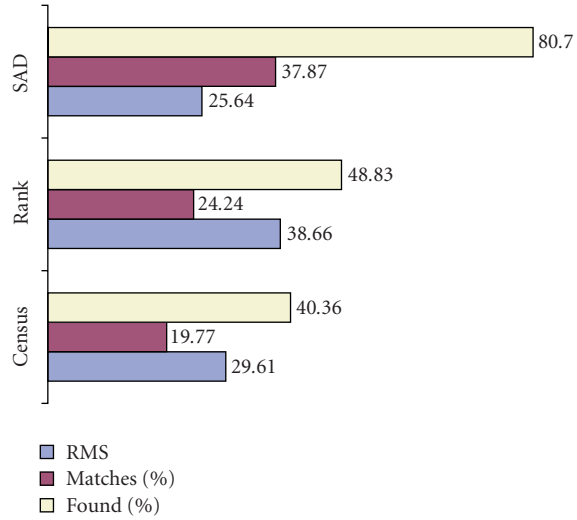


Figure 12: Evaluated algorithm quality with $d = 30$ and block size 9×18 for SAD and 9×12 for rank and census transform, respectively.

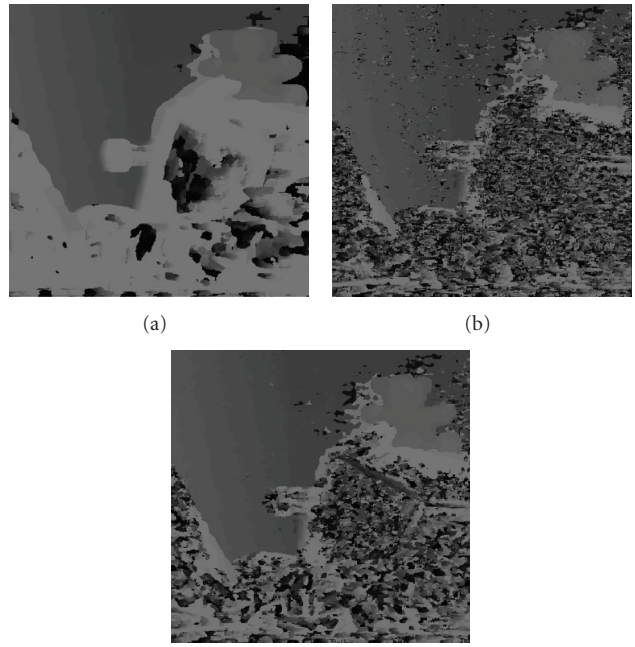


Figure 13: Disparity maps with $d = 30$ and block size 9×18 for SAD and 9×12 for rank and census transform, respectively. (a): SAD; (b): rank transform; (c): census transform.

memory bits and storing the 16-bit matching cost values requires 229 376 memory bits.

The algorithms' logic consumption in terms of logic elements is presented in Figure 8. The results show the highest values for the SAD algorithm due to its high number of aggregations, while the census transform requires slightly less logic. The rank transform has by far the smallest logic requirements, being 3.7 times smaller than that of the census transform. This is the reason, why the rank transform is of special interest for low-cost embedded stereo vision systems, even if it is well known that the census transform is the more robust algorithm [25].

Figure 9 reveals the disparity range versus the achieved frame rate for both block sizes using our flexible technique. As can be seen, the system is able to select the algorithm's features as required out of a disparity range between 12 and 120 pixels, a frame rate ranging from 136 fps to 666 fps, and a total block size of either 9×9 or 9×18 for SAD and 9×12 for the rank and census transform. As defined by (15), the frame rate times the number of calculation rounds $((d_{\max} + 1)/d_r)$ is nearly a constant. This shows that the static initialization times of our architecture, like filling the image memory or the final examination of the last image line, are negligible compared to the time required for the computation itself. Finally, it is illustrated that the disparity range can be easily doubled by selecting the smaller block size of $s_b = 9 \times 9$.

Our architecture produces 330×375 disparity maps with 120 disparity levels at 136 fps, which makes a total amount of about 2 billion disparity calculations per second. The DeepSea ASIC [31], producing 512×480 disparity maps with 52 disparity levels at 200 fps, has a total of 2.55 billion disparity calculations per second. Thus, our FPGA-based stereo architecture is just 22% slower than this ASIC-based solution, while having a 65% times larger total block size of 9×9 rather than 7×7.

Kuon and Rose [32] evaluated the achievable performance gain when moving from an FPGA to an ASIC. For FPGA designs making use of the internal block memory, they revealed a performance gain factor between 2.8 and 4.3, being 3.5 on average, when implementing the same design as an ASIC. This further outlines the potential of our flexible stereo matching technique, since it would outperform the DeepSea ASIC implementation by a minimum factor of 2.2 when implemented on the same platform, while offering the advantages of an adjustable stereo algorithm.

The time span to reconfigure the algorithm to a new feature set is one clock cycle. This reconfiguration is incorporated into the architecture's state change cycle that takes place before each frame is processed. Thus, no measurable reconfiguration time is required for adapting the algorithm's features.

The achieved RMS, found pixels, and correct matches of each algorithm on the teddy images are presented in Figure 10. The disparity maps are shown in Figure 11. Image areas, in which our architecture did not calculate the disparity in order to avoid border effects, are black and disregarded for the evaluation.

Both, the measurements and the depicted disparity maps show a great advantage in quality for the disparity maps from the rank and census transform when using the larger block size. This shows, that our technique also improves the quality of the output disparity maps, even if the block size of the transforms is constant and only the aggregation is enlarged. The SAD algorithm has an improved RMS at the larger block size, but the number of correct matches is slightly decreased. The disparity maps show that in this specific configuration, the house's roof in the teddy image contains noticeably more information for the larger block size, while the deformations caused by the block size in the rest of the image outweigh this advantage.

The measured quality for the insufficient disparity range is presented in Figure 12 and the corresponding disparity maps in Figure 13. It is noticeable that the reduced disparity range highly reduces the number of found and correct matches and increases the RMS. This further emphasizes the need for a correct adjustment of the disparity range.

5. CONCLUSIONS

The proposed flexible hardware-based stereo matching technique gives the ability to adapt the algorithm to varying application driven demands and therefore to ensure a highly accurate disparity map in real-time embedded stereo vision systems even under changing conditions. The flexible disparity range gives the opportunity to dynamically balance the detection of close objects against the frame rate. The opportunity to balance the disparity map's noise against the frame rate is assured by a flexible block size.

Our experimental evaluation of the presented technique shows a good response of the increased block size on low-textured surfaces for all algorithms, even if the transformation block size remains constant. In particular, the rank transform denotes good performance when a resource-aware implementation is demanded. Using our hardware-based stereo matching architecture, the census transform has the best overall quality. We pointed out that a correct disparity range is essential for a high-quality disparity map, which is easily achievable using our novel flexible technique. Due to the basic principle of our technique, only small amounts of additional memory, multiplexers, and summations are required, while remaining consistent with ASIC implementations by avoiding dynamic reconfiguration.

We compared our flexible stereo matching architecture with a well-known ASIC implementation, outlining the high performance of our architecture. Finally, we revealed the even higher potential of our flexible stereo matching technique when implemented on ASICs.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Sixth Framework Programme (FP6/2003-2006) under Grant agreement no. FP6-2006-IST-6-045350 (robots@home).

REFERENCES

[1] L. Mingxiang and J. Yunde, "Stereo vision system on programmable chip (SVSoC) for small robot navigation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '06)*, pp. 1359–1365, Beijing, China, October 2006.

[2] A. Naoulou, J.-L. Boizard, J. Y. Fourniols, and M. Devy, "A 3D real-time vision system based on passive stereo vision algorithms: application to laparoscopic surgical manipulations," in *Proceedings of the 2nd International Conference on Information and Communication Technologies (ICTTA '06)*, vol. 1, pp. 1068–1073, Damascus, Syria, April 2006.

[3] J. I. Woodfill, G. Gordon, and R. Buck, "The Tyzx DeepSea high speed stereo vision system," in *Proceedings of the Conference on Computer Vision and Pattern Recoginition Workshop (CVPR '04)*, vol. 3, p. 41, Washington, DC, USA, June-July 2004.

[4] L. Matthies, M. Maimone, A. Johnson, et al., "Computer vision on Mars," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 67–92, 2007.

[5] R. B. Porter and N. W. Bergmann, "A generic implementation framework for FPGA based stereo matching," in *Proceedings of IEEE Region 10 Annual International Conference on Speech and Image Technologies for Computing and Telecommunications (TENCON '97)*, vol. 2, pp. 461–464, Brisbane, Australia, December 1997.

[6] S. H. Lee, J. Yi, and J. S. Kim, "Real-time stereo vision on a reconfigurable system," in *Proceedings of the 5th International*

Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS '05), vol. 3553 of *Lecture Notes in Computer Science*, pp. 299–307, Samos, Greece, July 2005.

[7] S. Perri, D. Colonna, P. Zicari, and P. Corsonello, "SAD-based stereo matching circuit for FPGAs," in *Proceedings of the 13th IEEE International Conference on Electronics, Circuits, and Systems (ICECS '06)*, pp. 846–849, Nice, France, December 2006.

[8] R. P. Jacobi, R. B. Cardoso, and G. A. Borges, "VoC: a reconfigurable matrix for stereo vision processing," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS '06)*, p. 6, Rhodes Island, Greece, April 2006.

[9] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 993–1008, 2003.

[10] D. J. Fleet, "Disparity from local weighted phase-correlation," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (ICSMC '94)*, vol. 1, pp. 48–54, San Antonio, Tex, USA, October 1994.

[11] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1–3, pp. 7–42, 2002.

[12] A. Koschan, V. Rodehorst, and K. Spiller, "Color stereo vision using hierarchical block matching and active color illumination," in *Proceedings of the 13th International Conference on Pattern Recognition (ICPR '96)*, vol. 1, pp. 835–839, Vienna, Austria, August 1996.

[13] W. Maass, "On the computational power of winner-take-all," *Neural Computation*, vol. 12, no. 11, pp. 2519–2535, 2000.

[14] T. Kanade and M. Okutomi, "A stereo matching algorithm with an adaptive window: theory and experiment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 9, pp. 920–932, 1994.

[15] J. Banks, M. Bennamoun, and P. Corke, "Non-parametric techniques for fast and robust stereo matching," in *Proceedings of IEEE Region 10 Annual International Conference on Speech and Image Technologies for Computing and Telecommunications (TENCON '97)*, vol. 1, pp. 365–368, Brisbane, Australia, December 1997.

[16] A. E. Kayaalp and J. L. Eckman, "Near real-time stereo range detection using a pipeline architecture," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 6, pp. 1461–1469, 1990.

[17] C. Cuadrado, A. Zuloaga, J. L. Martín, J. Lázaro, and J. Jiménez, "Real-time stereo vision processing system in a FPGA," in *Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics (IECON '06)*, pp. 3455–3460, Paris, France, November 2006.

[18] Y. Jia, M. Li, L. An, and X. Zhang, "Autonomous navigation of a miniature mobile robot using real-time trinocular stereo machine," in *Proceedings of the IEEE International Conference on Robotics, Intelligent Systems and Signal Processing (RISSP '03)*, vol. 1, pp. 417–421, Changsha, China, October 2003.

[19] G. van der Wal, M. Hansen, and M. Piacentino, "The Acadia vision processor," in *Proceedings of the 5th IEEE International Workshop on Computer Architectures for Machine Perception (CAMP '00)*, pp. 31–40, Padova, Italy, September 2000.

[20] J. S. Yi, J. S. Kim, L. P. Li, J. Morris, G. Lee, and P. Leclercq, "Real-time three dimensional vision," in *Proceedings of the 9th Asia-Pacific Conference on Advances in Computer Systems Architecture (ACSAC '04)*, vol. 3189 of *Lecture Notes in Computer Science*, pp. 309–320, Beijing, China, September 2004.

[21] R. Zabih and J. I. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Proceedings of the 3rd European Conference on Computer Vision (ECCV '94)*, pp. 151–158, Stockholm, Sweden, May 1994.

[22] J. Banks and M. Bennamoun, "Reliability analysis of the rank transform for stereo matching," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 31, no. 6, pp. 870–880, 2001.

[23] H. Hirschmüller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '07)*, pp. 1–8, Minneapolis, Minn, USA, June 2007.

[24] J. D. Anderson, *Semi autonomous vehicle intelligence: real time target tracking for vision guided autonomous vehicles*, M.S. thesis, Brigham Young University, Provo, Utah, USA, 2007.

[25] B. Cyganek, "Comparison of nonparametric transformations and bit vector matching for stereo correlation," in *Proceedings of the 10th International Workshop on Combinatorial Image Analysis (IWCIA '04)*, vol. 3322 of *Lecture Notes in Computer Science*, pp. 534–547, Auckland, New Zealand, December 2004.

[26] P. Corke and P. Dunn, "Real-time stereopsis using FPGAs," in *Proceedings of IEEE Region 10 Annual International Conference on Speech and Image Technologies for Computing and Telecommunications (TENCON '97)*, vol. 1, pp. 235–238, Brisbane, Australia, December 1997.

[27] J. I. Woodfill and B. Von Herzen, "Real-time stereo vision on the PARTS reconfigurable computer," in *Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FPGA '97)*, pp. 201–210, Napa Valley, Calif, USA, April 1997.

[28] M. Kuhn, S. Moser, O. Isler, et al., "Efficient ASIC implementation of a real-time depth mapping stereo vision system," in *Proceedings of the of the 46th IEEE Midwest International Symposium on Circuits and Systems (MWSCAS '03)*, pp. 1478–1481, Cairo, Egypt, December 2003.

[29] Z. Zhang, "Determining the epipolar geometry and its uncertainty: a review," *International Journal of Computer Vision*, vol. 27, no. 2, pp. 161–195, 1998.

[30] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, vol. 1, pp. 195–202, Madison, Wis, USA, June 2003.

[31] J. I. Woodfill, G. Gordon, D. Jurasek, T. Brown, and R. Buck, "The Tyzx DeepSea G2 vision aystem, a taskable, embedded stereo camera," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR '06)*, p. 126, New York, NY, USA, June 2006.

[32] I. Kuon and J. Rose, "Measuring the gap between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 930250, 12 pages
doi:10.1155/2008/930250

Research Article

High Speed 3D Tomography on CPU, GPU, and FPGA

Nicolas GAC,[1,2] Stéphane Mancini,[1] Michel Desvignes,[1] and Dominique Houzet[1]

[1] *Grenoble-Images-Parole-Signal-Automatique Laboratoire (GIPSA-lab), Grenoble Institute of Technology (INPG), BP 46, 38402 Grenoble Cedex, France*
[2] *Equipes Traitement des Images et du Signal (ETIS), Centre National de la Recherche Scientifique (CNRS), Ecole Nationale Supérieure de l'Electronique et de ses Applications (ENSEA), Université de Cergy-Pontoise, 95000 Cergy-Pontoise Cedex, France*

Correspondence should be addressed to Nicolas GAC, nicolas.gac@lss.supelec.fr

Received 1 March 2008; Revised 24 June 2008; Accepted 12 November 2008

Recommended by Dragomir Milojevic

Back-projection (BP) is a costly computational step in tomography image reconstruction such as positron emission tomography (PET). To reduce the computation time, this paper presents a pipelined, prefetch, and parallelized architecture for PET BP (3PA-PET). The key feature of this architecture is its original memory access strategy, masking the high latency of the external memory. Indeed, the pattern of the memory references to the data acquired hinders the processing unit. The memory access bottleneck is overcome by an efficient use of the intrinsic temporal and spatial locality of the BP algorithm. A loop reordering allows an efficient use of general purpose processor's caches, for software implementation, as well as the 3D predictive and adaptive cache (3D-AP cache), when considering hardware implementations. Parallel hardware pipelines are also efficient thanks to a hierarchical 3D-AP cache: each pipeline performs a memory reference in about one clock cycle to reach a computational throughput close to 100%. The 3PA-PET architecture is prototyped on a system on programmable chip (SoPC) to validate the system and to measure its expected performances. Time performances are compared with a desktop PC, a workstation, and a graphic processor unit (GPU).

1. INTRODUCTION

Tomography consists of reconstructing an object from its projections via numerical methods [1]. This process is used in medical scanners, such as computed tomography (CT) or positron emission tomography (PET) scanners. PET is a nuclear imaging modality; its goal is to measure the spatial and temporal distribution of a radiotracer perfused in a patient's body. PET imaging is used in oncology, to detect, track, and visualize tumors. After data acquisition, the 3D image of the radiotracer is reconstructed offline from the measures (called sinograms) to diagnose pathologies. Oncology and other clinical applications need a high-quality reconstruction as fast as possible (few minutes at most) to reduce the device occupation and allow a patient repositioning. (A patient cannot experience a radiotracer twice in a short while and has to wait several months before a new examination, in case of bad camera positioning.) Also, dynamic PET is in need of even faster reconstruction.

Moreover, tomography is required in many other medical imaging techniques, such as 3D magnetic resonance imaging and 3D ultrasound imaging, or in other domains such as synthetic aperture radar (SAR), contactless control, and industrial X-ray applications. Therefore, the acceleration of the reconstruction algorithm is of great interest for various applications.

Due to the large amount of the acquired data and the complexity of the algorithms, reconstruction is a very time-consuming process. From a computing point of view, reconstruction methods can be classified into two main techniques: analytic (direct) reconstruction and iterative reconstruction. They both include a back-projection (BP) step that accounts for 50% to 70% of the processing time.

In 3D reconstruction, the computational complexity of the standard algorithm to reconstruct an N^3 dataset from N angles of projection is $O(N^4)$. In the previous decade, several algorithms have been proposed to reduce BP complexity. The lowest cost obtained is $O(N^3 \log N)$ but generally with a lower quality of reconstruction; also it does not take into account some required data management, which delay the process. Although CPUs have gained sufficient computing power for 2D reconstruction, with 3D reconstruction, the

increase of the amount of data for high-quality images leads to higher computing times. Iterative reconstruction algorithms may reach several hours of processing [2].

The algorithmic optimizations of reconstruction have reached some limits and it is becoming mandatory to reduce the computing time through architecture solutions. General purpose parallel computers benefit from recent competing technologies: the system on programmable chip (SoPC) and the general purpose graphical processing unit (GP-GPU).

This paper shows that a hardware implementation of the BP algorithm needs to overcome the memory bottleneck. This may be solved both by a loop reordering and the use of an efficient caching mechanism. Parallel hardware pipelines can be fed with a hierarchy of semigeneral purpose cache such as the 3D-AP cache [3]. The resulting architecture makes a better use of memory bandwidth than general purpose CPUs and GP-GPU.

The first parts of this paper present the use of the 3D BP algorithm and different solutions to accelerate it. Next, we present the memory bottleneck of a classical implementation of 3D BP to overcome. From this study, an efficient architecture is proposed: the pipelined, prefetch, and parallelized architecture for 3D PET BP (3PA-PET). The quality, complexity, and timing performance of the 3PA-PET architecture are also presented. Measures on its prototyping on a SoPC allow a comparison with the implementation of BP on CPU and GP-GPU.

2. 3D BP IN TOMOGRAPHY RECONSTRUCTION

In this section, we will first show that 3D PET BP and the 3D CT BP using, respectively, a parallel and a cone beam geometry are close algorithms. Then, we will present some related works on acceleration of these two BPs on several architectures.

2.1. BP algorithms

2.1.1. 3D parallel beam BP for PET

The detectors of a PET scanner are usually paving a cylinder and stacked in a set of rings of detectors [1]. The γ rays issued from the disintegration of a radiotracer particle are detected by a pair of sensors facing each other. The line which connects two sensors is called a line of response (LOR) and the coincidence events counted on one LOR are stored in a *bin*. All the bins are stored in a sinogram as illustrated in Figure 1. The reconstruction process attempts to estimate the image of the radiotracer distribution f that has produced the sinogram.

The sinogram p_{PET} is a 4D space along $(\Delta, \psi, u_{\parallel}, v_{\parallel})$. The coordinates $(\Delta, v_{\parallel})$ represent a couple of rings: Δ is the axial distance between the two rings (segment number) and $v_{\parallel}$ is the mean axial coordinate of the two rings (plane number). The coordinates $(\psi, u_{\parallel})$ represent one particular LOR between two rings: ψ is the azimuthal angle and $u_{\parallel}$ is the tangential coordinate (bin number).

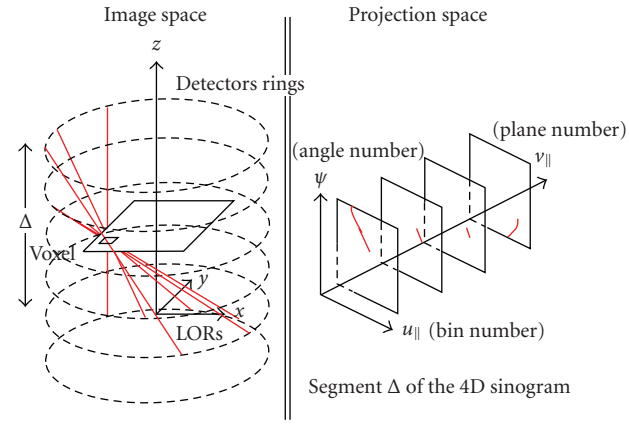


Figure 1: The acquired data are stored in a 4D sinogram, a sinogram bin corresponding to one particular LOR. To reconstruct one voxel, the data needed by the BP algorithm draw a 3D sinusoid in each segment Δ.

For each voxel (VOlume piXEL) of coordinate $\vec{r} = (x, y, z)$, the BP algorithm sums up all the sinogram bins corresponding to that voxel projection as follows:

$$f^{*}_{\text{PET}}(\vec{r}) = \iint p_{\text{PET}}(\Delta, \psi, u_{\parallel}(\psi, \vec{r}), v_{\parallel}(\psi, \Delta, \vec{r}))J_{\Delta} d\psi \, d\Delta, \tag{1}$$

J_{Δ} is a Jacobian and the parallel beam coordinates $(u_{\parallel}, v_{\parallel})$ are computed as

$$\begin{aligned} u_{\parallel}(\psi, \vec{r}) &= x \cos\psi + y \sin\psi + \text{offset}, \\ v_{\parallel}(\psi, \Delta, \vec{r}) &= \frac{\Delta}{2R_a} \cdot (x \sin\psi - y \cos\psi) + z + \text{offset}. \end{aligned} \tag{2}$$

Using $a_{ij}(\psi, \Delta)$ coefficients, we get

$$\begin{aligned} u_{\parallel}(\psi, \vec{r}) &= a_{00}x + a_{01}y + a_{03}, \\ v_{\parallel}(\psi, \Delta, \vec{r}) &= a_{10}x + a_{11}y + a_{12}z + a_{13}. \end{aligned} \tag{3}$$

2.1.2. Cone beam BP for CT

The cone beam BP used in CT imaging modalities uses a similar algorithm [4]. In CT, the data is the X-ray intensity reaching an X camera that rotates around the observed volume. It measures the attenuation due to the density of tissues. The density $f^{*}_{\text{CT}}(\vec{r})$ is computed from the measured data p_{CT} following:

$$f^{*}_{\text{CT}}(\vec{r}) = \int p_{\text{CT}}(\alpha, u_{\vee}(\alpha, \vec{r}), v_{\vee}(\alpha, \vec{r})) \cdot w(\alpha, \vec{r})^{2} \cdot d\alpha, \tag{4}$$

where α is the trajectory parameter of the camera. The cone beam coordinates $(u_{\vee}, v_{\vee})$ are computed as

$$\begin{aligned} u_{\vee}(\alpha, \vec{r}) &= (c_{00}x + c_{01}y + c_{02}z + c_{03}) \cdot w(\alpha, \vec{r}), \\ v_{\vee}(\alpha, \vec{r}) &= (c_{10}x + c_{11}y + c_{12}z + c_{13}) \cdot w(\alpha, \vec{r}), \end{aligned} \tag{5}$$

where c_{ij} depends on α (i.e., $c_{ij} = c_{ij}(\alpha)$) and

$$w(\alpha, \vec{r}) = \frac{1}{c_{20} \cdot x + c_{21} \cdot y + c_{22} \cdot z + c_{23}}. \quad (6)$$

2.1.3. Comparison of CT and PET

Although the CT BP is more complex due to the perspective transformation (6), these algorithms are quite similar. Indeed, the summation over α (trajectory parameter) for CT BP is equivalent to the summation over ψ and Δ for PET BP. Moreover, in these loops, both these BPs compute very similar projection coordinates $(u_{\parallel}, v_{\parallel})$ and $(u_{\vee}, v_{\vee})$. Nevertheless, the computation of the projection coordinates for CT BP needs a division by a distance weight $w(\alpha, \vec{r})$. Thus, the CT BP kernel has more arithmetic operations than PET BP has.

Supposing that one is able to design a pipeline that computes a sum update at each clock cycle, both for CT and PET BP, then the challenge is to fetch data along a complex path (a 3D sinusoid) in the acquired data (3D CT data or 4D PET sinogram). The method presented in this paper for solving the case of PET BP (parallel beam) could be transposed to solve the CT BP (cone beam).

2.2. Acceleration of reconstruction

Different computer architectures coupled with dedicated memory access strategies are used to accelerate the BP step of an analytic or iterative reconstruction, including general purpose processors [5–7], graphical processors [8–14], the cell processor [4, 15], or ASIC/FPGA architectures [16–20]. While most of these works have investigated cone beam BP, only a few of them have investigated 3D parallel beam BP [2, 5, 8, 9].

The parallelization of reconstruction algorithms on shared memory parallel general purpose computer [5] stays efficient only up to 4 processing units, because of conflictual accesses on the memory bus. Considering clusters of heterogeneous PCs [6, 7], efficiency of parallelization drops down quickly because of the costly communication between PCs. After 10 PCs, parallelization is not worthy. Yet on a distributed memory parallel computer, parallelization works very well. for 3D PET iterative reconstruction, Jones et al. [2] succeeded to get an acceleration factor of about 30 with 32 processors. Ni et al. [21] achieved an excellent acceleration factor of 300, when they parallelized the Katsevich algorithm, an exact cone beam BP with 300 CPUs.

Besides parallelization on several nodes of general purpose processors, more efficient engines such as the graphical processing unit (GPU) or the IBM Cell can be used. Current GPUs can be used either as a graphical pipeline, which is originally designed for [8–10], or as a multiprocessor chip thanks to the CUDA interface from Nvidia [10, 12–15]. For both options, the acceleration factor of GPU is high, about an order of magnitude for cone beam BP. Xu and Mueller [10] have observed that an implementation of the cone beam BP using the graphic pipeline is 3 times faster than the one made with the CUDA interface. Kachelrieß et al. [4] and Scherl et al. [15] present good result of acceleration of cone beam BP using the Cell processor. With its 1 + 8 cores, this architecture is an intermediate solution between general purpose parallel processors and GPU. The 8 vector engines have to be specifically programed. Nevertheless, Scherl et al. [11] have measured that a GPU with CUDA is 3 times faster than the Cell for the BP alone.

FPGA technology is an alternative to processors, allowing designers to make a customized architecture. Most often, it is used to prototype ASIC implementations. In this context, FPGA implementations of 2D parallel beam BP [16] and 3D cone beam BP [17–19] have been investigated. These architectures are made of several pipelines working in parallel. Moreover, like the image Pro by Siemens [18], several FPGA chips can be used in a single board to raise the computational power.

Two memory access strategies have been applied for all these architectures. In case the processor already has a memory cache, developers rely on it to optimize the external memory accesses. Otherwise, developers set up custom memory strategies in order to hide the memory access time. The most common approach is to use double buffering: the next required projection data is loaded from external memory, meanwhile the ongoing loaded projection data are back-projected. In this case, CPU and GPU memory strategies are based on an extensive use of the cache. For example Yang et al. [13] have observed that enabling a GPU cache is more competitive than software prefetching. On the other hand, the Cell and FPGA memory strategies have to be taken in charge by the software designers.

3. OVERCOMING THE MEMORY BOTTLENECK

In this section, we focus our study on finding out the best appropriate memory strategy to get the best fit between the 3D BP algorithm and a hardware architecture.

3.1. Memory access strategy

As the sinogram is kept in an SDRAM-like external memory, we need an efficient memory management to overcome its latency and allow a high level of parallelism. The main difficulty is to deal with the high strides of addresses due to the sinusoidal pattern of references in the 4D space. A cache would help to hide the high latency of the external memory despite these strides. Standard caches therefore are inefficient as they exploit temporal and address locality of references. Hence they are used at their best when the references follow a 1D linear pattern as a cache line is loaded when a miss occurs.

Indeed, as shown earlier, the reconstruction of a single voxel $f(\vec{x})$ needs to follow a 3D sinusoid in the 4D sinogram. Such a pattern is of poor address locality but has a high index locality. Moreover, because of the $v_{\parallel}$ dimension, the memory accesses for 3D BP have higher strides and are more distributed in the memory space than in the 2D BP case [22]. The challenge is to speed up these memory accesses in a 4D data structure.

for $n = 0$ to $n_{\max}$ **do**
  for Delta $= 0$ to $\text{Delta}_{\max}$ **do**
    for psi $= 0$ to $\text{psi}_{\max}$ **do**
      for $\vec{r} = \vec{r}_{\min}(n)$ to $\vec{r}_{\max}(n)$ **do**
$f(\vec{r}) += \text{bin}(\text{psi}, \text{Delta}, u_{\parallel}(\text{psi}, \vec{r}), v_{\parallel}(\text{psi}, \vec{x}))$

Algorithm 1: The loop reordering of the 3D BP improves spatial and temporal locality.

Therefore, a new cache mechanism is needed. Estimating which bins would be referenced would help the cache to download the needed bins during the computing process.

3.2. Improvement of spatial and temporal locality

A reconstruction with the voxel-driven bilinear interpolation (VBI) standard BP is made of three loops, as described in Algorithm 1: the first loop is on the voxels $\vec{r}$, the second on the segments (Delta), and the third on the angles (psi). Since voxels can be reconstructed independently, the loop on voxels can be split into two parts: one loop on blocks of voxels $(0, \ldots, n_{\max})$ and the other loop on the voxels of a block $n(\vec{r}_{\min}(n), \ldots, \vec{r}_{\max}(n))$.

A loop reordering increases the temporal and spatial locality of memory accesses. Indeed, for given psi and Delta values, the data $\text{bin}(\text{psi}, \vec{r})$ will be used several times for different voxels since the projection of a 3D block of voxels is a 2D plane in the 4D space of the sinogram.

Figure 2 shows that the proposed loop reordering allows to cache a part of the sinogram. The BP of a block of voxels makes the references follow a coherent 3D sinusoid in the sinogram along the time.

3.3. Mean bin reuse rate (MBRR)

To give a theoretical estimation of the best achievable cache efficiency, the mean bin reuse rate (MBRR) is defined as the ratio between the number of bins accessed in cache memory by the processing units and the number of bins loaded in cache memory from the external memory. The ideal MBRR can be computed analytically. It depends on the shape of the block of reconstructed voxels. Figure 3 presents this optimal MBRR computed for each segment versus the size of the block.

4. A 3P ARCHITECTURE FOR PET

In this section, we present the pipelined, prefetched, and parallelized architecture for PET (3PA-PET). The 3PA-PET architecture is made of a high-performance pipeline connected to a 3D adaptive and predictive cache (3D-AP cache). It allows to perform an update of a voxel value up to 1 operation per clock cycle (100% pipeline utilization), even for high latency memories.

4.1. Pipelined architecture

The pipeline in Figure 4 implements the different steps of the VBI standard BP: the computation of $u_{\parallel}(\text{psi}, \vec{r})$ and $v_{\parallel}(\text{psi}, \vec{r})$, the bilinear interpolation of the bin, and finally the accumulation of the voxel value. The forward flow control is done by packets passing through each stage of the pipeline.

The 4 bins needed for the bilinear interpolation are fetched through the memory bridge. This bridge controls the 3D-AP cache and can freeze (or not) the pipeline depending on the requested data availability. A backward flow control synchronizes the pipeline and the 3D-AP cache.

4.2. Prefetch architecture

The 3D-AP cache [3] masks the latency of the external memory so that the pipeline is no more systematically stalled. The memory bridge gets four bins from the cache at each clock cycle.

The 3D-AP cache is a semigeneric cache memory mechanism that prefetches references following a continuous path into a 3D memory space. It was originally designed as a cache for a computer vision lip tracking application [3] but it targets a large class of multidimensional processing algorithms. In the 3PA-PET architecture, the 3D-AP cache is tuned to follow the references needed to reconstruct a block of voxels, as shown in Figure 5. The pipeline issues spatial coordinates of the requested bin, here $(\text{psi}, u_{\parallel}, v_{\parallel})$, to the 3D-AP cache. A part of the sinogram, namely, the *cached zone*, is copied in an embedded memory. A tracking mechanism tries to maintain the center of the cached zone in the mean coordinate of the referenced data.

The 3D-AP cache estimates dynamically which data is likely to be requested in the future. This is done by a statistical analysis on each axis of the previous references. Moreover, the 3D-AP cache masks the data transfer between the external memory and the internal cache memory. In the mean time, the cache grabs new data from the external memory, the data shared by the old and the new cached zone stay available for the processing unit.

The 3D-AP cache parameters have to be set beforehand by the user. In this study, we set for each dimension the value of the following five parameters.

(i) *Cut off and sampling frequencies:* the mean coordinate is computed by a first-order low-pass IIR filter configured by these two frequencies.

(ii) *Cached zone size:* this zone is notified to the memory bridge to be available in cache. In this study, this size is a static parameter.

(iii) *Guard zone size* when the mean coordinate is out of this zone, the cache zone is updated.

(iv) *Cache speed:* it has to be set according to the speed of the data accesses performed by the application on each spatial dimension.

The cache is customized to allow four concurrent accesses to the bins needed to perform a bilinear interpolation. Figure 6 gives a simplified view of the 3D-AP cache to illustrate the

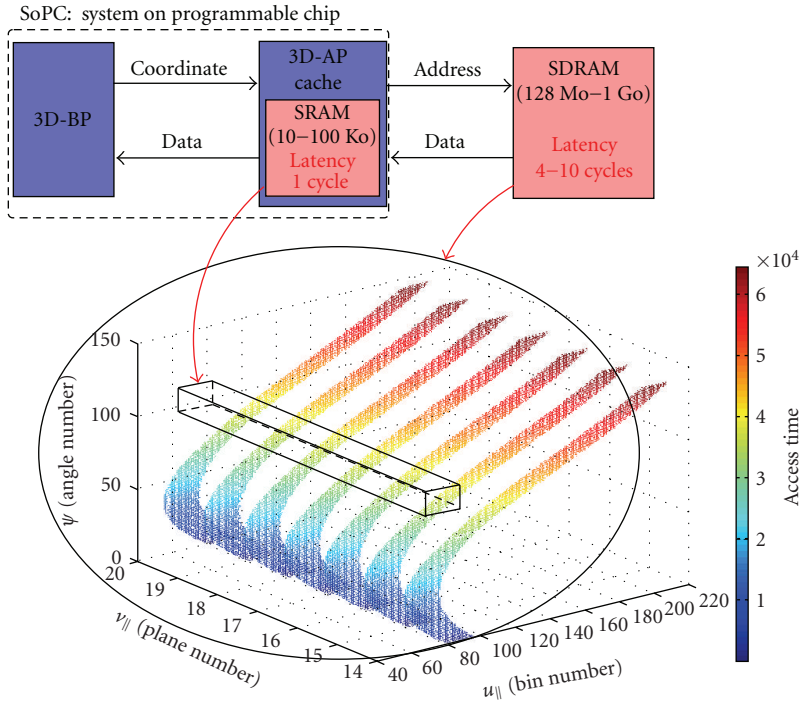


Figure 2: The memory access strategy is based on a fast and small cache memory inside the SoPC. The cache predicts the needs of the 3D BP unit and therefore succeeds to mask the high latency (4–10 cycles at 200 Mhz) of the slower and bigger external SDRAM memory.

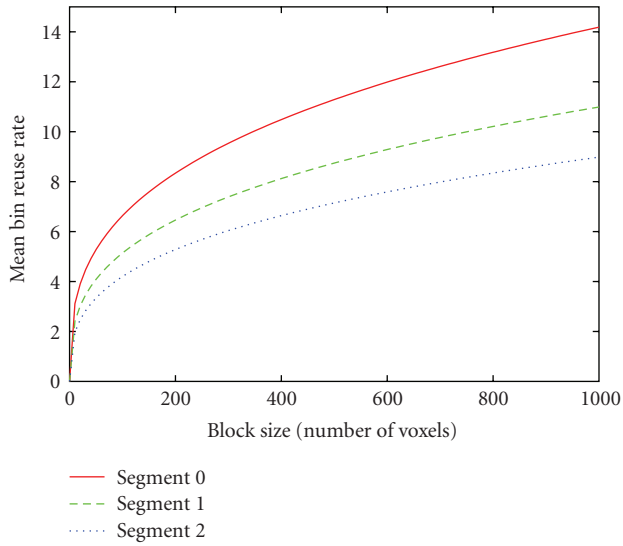


Figure 3: Mean bin reuse rate (MBRR) estimated for a 3D BP without bilinear interpolation versus the size of reconstructed blocks of voxels for each segment of a Siemens HR+ sinogram (span 9 with 96 angles of projection).

involved memory architecture. The cache control unit grabs data from the external memory and splits the incoming data words to the different embedded memories. The cache control unit also manages the cache misses that could occur for some requested bins.

4.3. *Parallelized architecture*

To increase the computing power, several pipelines are parallelized. A hierarchical cache reduces the memory bus occupation, when BP units work in parallel. In this hierarchical design, one leaf cache is associated to one 3D BP unit while a root cache is feeding each of these leaf caches.

The spatial locality of the references of the pipelines is enabled by reconstructing a set of neighbor blocs. A pipeline reconstructs one bloc of voxels and all the pipelines share a loop over psi. Some of the sinogram data are shared by the pipelines in the same way they are shared to reconstruct one bloc of data. The bins needed during the reconstruction of a set of blocs draw a 3D sinusoid. The cache concept presented previously with one unit applies here in the same manner. Each leaf cache stores a 3D sinusoid needed to reconstruct a bloc. A higher level cache stores the union of these sinusoids as presented in Figure 7.

5. 3PA-PET PERFORMANCES

5.1. *Accuracy of reconstruction*

The implemented VBI standard BP is a fixed-point version of the original algorithm. Moreover, the sinogram data is converted in *float* to *short integer* (16 bits). The accuracy of reconstruction of 3PA-PET is measured between a reference reconstruction software and a software bit-true model of 3PA-PET.

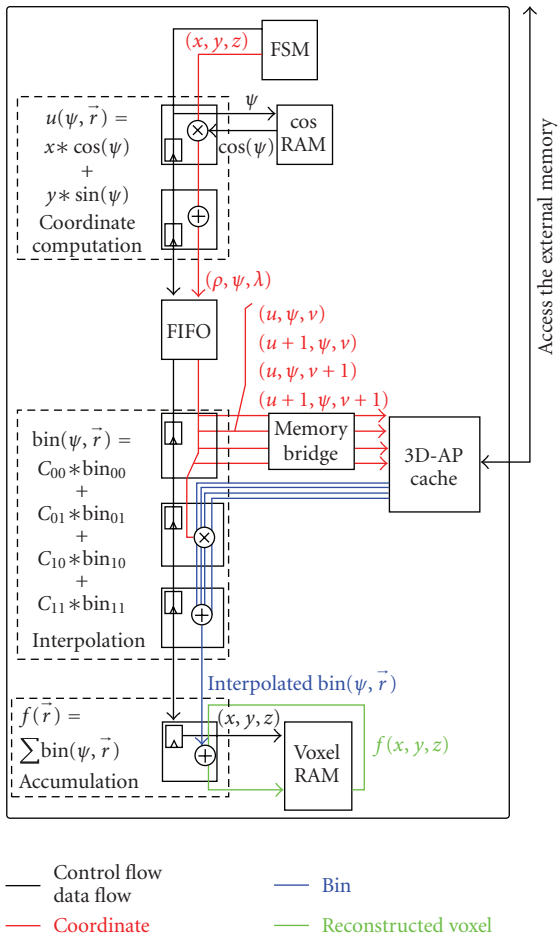


Figure 4: Pipeline of 3PA-PET.

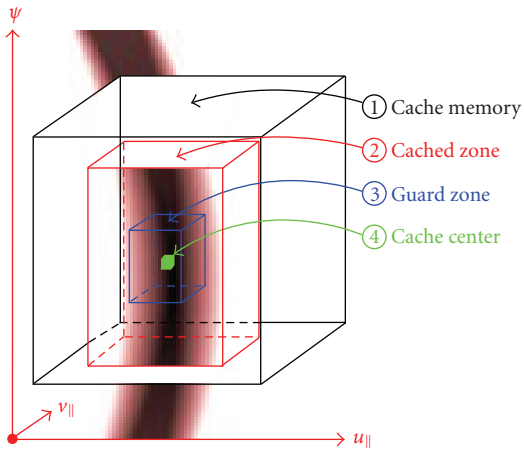


Figure 5: 3A-AP cache zones.

The reference dataset used is a sinogram of a 3D Shepp-Logan volume of 128 × 128 × 63 voxels. This phantom is a standard volume used in tomography to measure the accuracy of reconstruction. The sinogram is obtained from the STIR open source tool kit [23]. The volumes reconstructed by STIR and by 3PA-PET are shown on Figures 8 and 9.

Table 1: Accuracy of reconstruction and compared reconstructions for the Shepp-Logan phantom.

Compared volumes	Data	MAPE	PSNR
Accuracy of reconstruction			
STIR/original	Float	3.89%	10.5 dB
VBI-Flt/original	Float	3.88%	10.5 dB
VBI-Fix/original	Float	3.88%	10.5 dB
VBI-Flt/original	Int16	3.97%	10.5 dB
VBI-Fix/original	Int16	3.97%	10.5 dB
Compared reconstructions			
STIR/VBI-Flt	Float	0.35%	21.5 dB
VBI-Fix/VBI-Flt	Float	0.13%	26.2 dB
VBI-Fix/VBI-Flt	Int16	0.13%	23.0 dB
VBI-Fix/VBI-Flt	Int16/flt	1.1%	19.0 dB

The accuracy of reconstruction of the 3PA-PET BP is measured with two metrics: the mean absolute percentage error (MAPE) and the peak signal-to-noise ratio (PSNR). Both compare a volume f_1 with a volume of reference f_{ref}. The PSNR corresponds to the ratio between the maximum of f_{ref} (dynamic range) and the mean squared error (MSE) of f_1 compared to f_{ref},

$$\text{PSNR} = 20 \cdot \log_{10} \frac{\max(f_{\text{ref}})}{\sqrt{\text{MSE}(f_{\text{ref}}, f_1)}}. \quad (7)$$

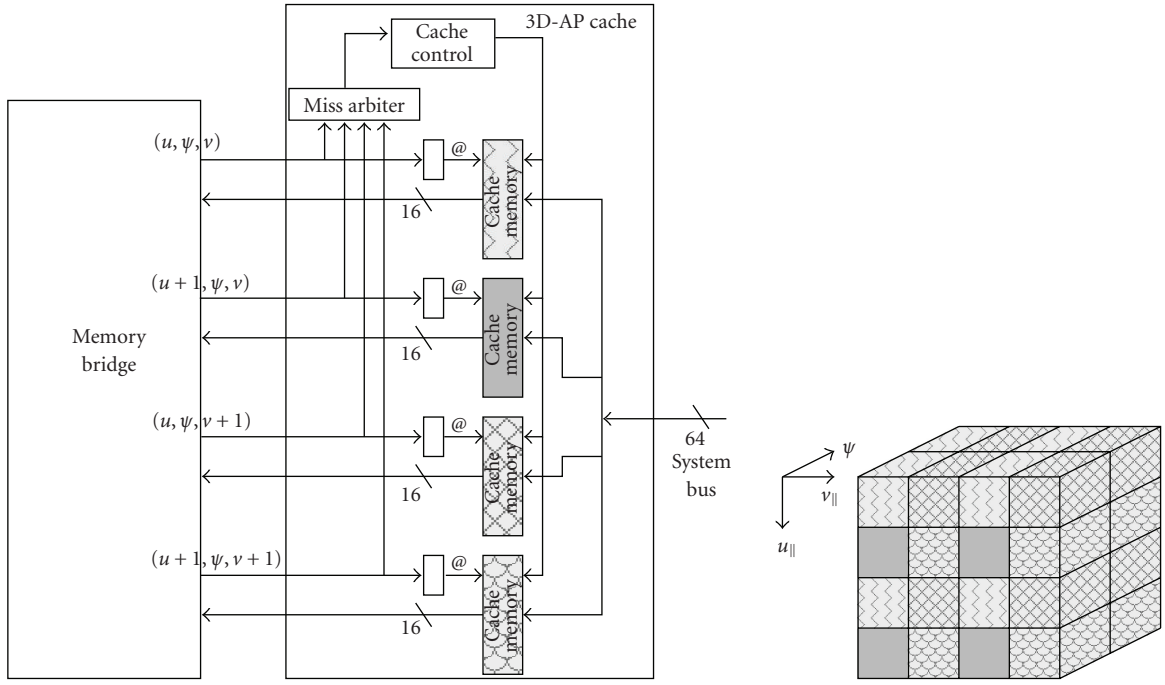
In Table 1, we compared the reference volume and the volumes reconstructed with STIR, with VBI floating-point arithmetic (VBI-flt) or with VBI fixed-point arithmetic (VBI-fix). All of the reconstruction methods have an intrinsic error around 3.9% with a PSNR of 10.5 dB when compared with the original volume. The floating-point and the fixed-point implementations have an MAPE of 0.13% and a PSNR of 23 dB. With different data types (*short int* versus *float*), the MAPE is about 1.1% and the PSNR of 19 dB. Thus we can conclude that the 3PA-PET implementation of the VBI BP is an accurate reconstruction system.

5.2. 3PA-PET complexity

The hardware resources used by the 3PA-PET architecture are presented in Table 2. The main BP FSM and the root cache control are shared between all of the units of the 3PA-PET architecture. Therefore, the cost of an additional pipeline is only 800 slices. The sizes of a leaf and the root caches are, respectively, 2 KB and 18 KB. Hence, 9 BP units fit in a Xilinx Virtex 2 Pro VP30 chip and 16 units in a virtex 4 FX100.

5.3. Efficiency of reconstruction

In order to assess the efficiency of the 3PA-PET architecture, we have measured the BP time on an Avnet development board connected to a PC host through a PCI interface



(a) Customized concurrent 3D-AP cache architecture

(b) Mapping of bins to memory buffers

Figure 6: Memory architecture for bilinear interpolation.

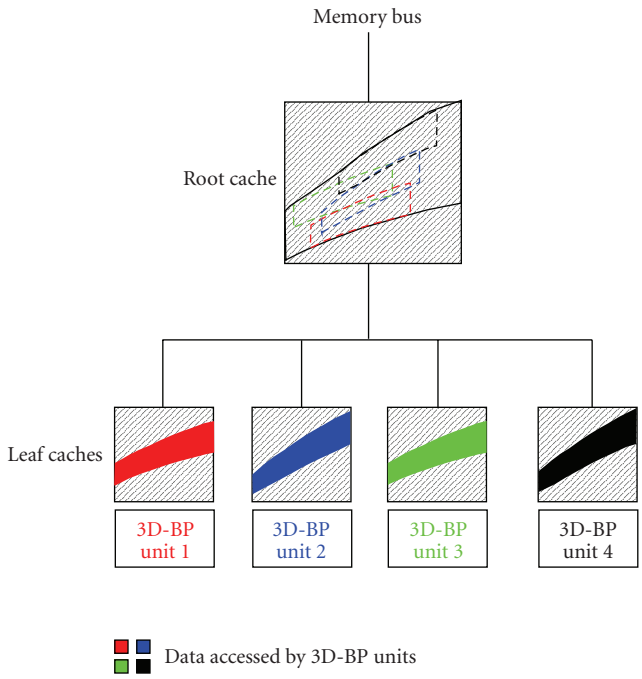


Figure 7: Each leaf cache is fed by the root cache.

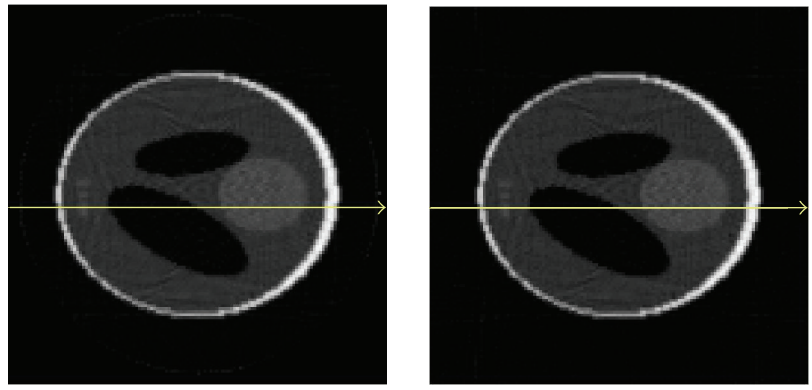
Table 2: Hardware resources used by the 3PA-PET on a Xilinx Virtex 2 Pro VP30.

	1 unit	4 units	9 units
3D BP			
CLB slices	573 (4.2%)	1817 (13.3%)	3924 (28.6%)
Multipliers	12 (9%)	48 (35%)	108 (79%)
3D-AP cache			
CLB slices	672 (4.9%)	2830 (20.6%)	4804 (35.1%)
RAMs	2 kB (0.6%)	24 kB (7.8%)	36 kB (11.7%)
3D BP + 3D-AP cache			
CLB slices	1245 (9.1%)	4637 (32.9%)	8728 (63.7%)

(see Figure 10). The board contains an external SDRAM memory and a Xilinx system on programmable chip (SoPC). In order to investigate the 3PA-PET behavior with respect to the memory features, we have plugged it with a fake memory bus which could be set with different values of memory latency (l_{mem}) and memory bandwidth (BW_{mem}). The memory simulator estimates the time to access N_{line} lines of S_{line} bytes following the relationship

$$t_{\text{mem}} = N_{\text{line}} \cdot \left(l_{\text{mem}} + \frac{S_{\text{line}} - 1}{BW_{\text{mem}}} \right). \tag{8}$$

The times of reconstruction presented in this section are in clock cycles and scaled to one operation. An operation corresponds to one update of a voxel. The number of voxel's



STIR (a) 3PA-PET (b)

Figure 8: A slice of the 3D Shepp-Logan phantom reconstructed by STIR and 3PA-PET BP.

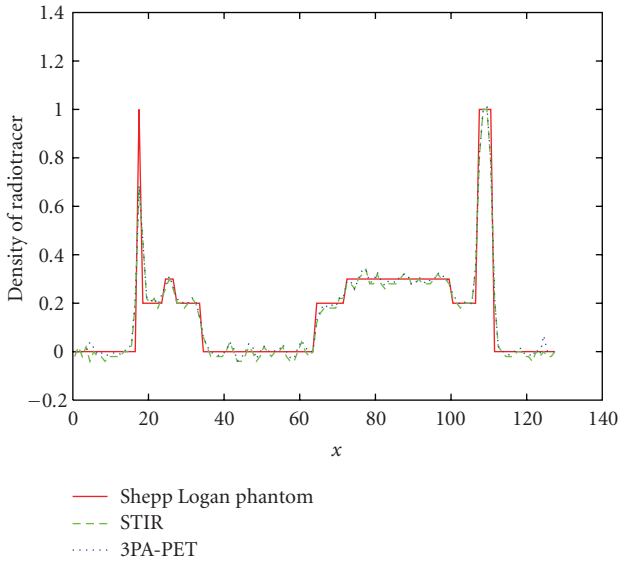


Figure 9: Profile of the 3D Shepp-Logan phantom slices corresponding to the lines on Figure 8.

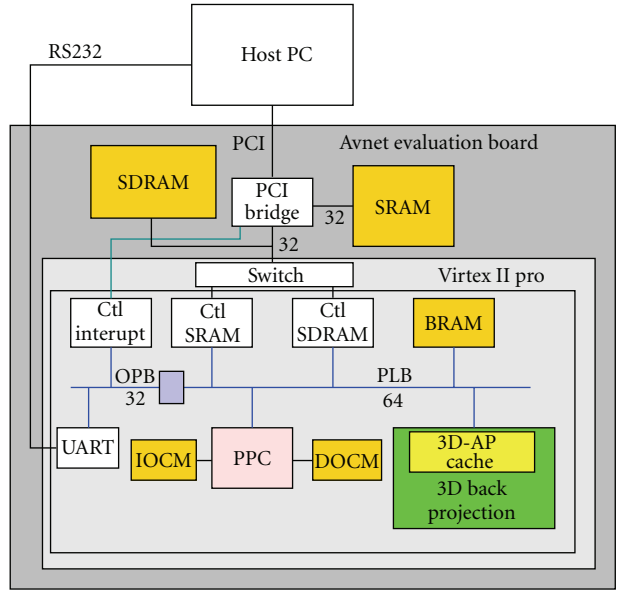


Figure 10: Evaluation system.

updates is equal to the number of voxels multiplied by the number of segments times the number of angles.

The results presented in Figure 11 are achieved with one BP unit for the segment +2 which represents the worse case because the memory accesses draw the most incurvated 3D sinusoid. 3PA-PET is robust to high latencies and low bandwidth: the pipeline computes a voxel update in about 1 clock cycle, even for a memory latency of 30 cycles. This shows that the 3D-AP cache succeeds to take advantage of the high spatial and temporal locality of the BP algorithm presented in Section 3. The 3D-AP cache follows the 3D memory path drawn during the BP process rather well. The cache miss rate stays low (about 0.05% with $l_{\text{mem}} = 5$ cycles and $BW_{\text{mem}} = 8$ bytes/cycle) which means that the 3D-AP cache prediction is satisfactory and manages to hide the external memory latency.

As illustrated in Figure 12, the parallel 3PA-PET performances are not plenty satisfactory. Indeed, the efficiency of parallelization decreases with the number of BP units. For instance, with a memory latency of 5 and for a complete BP, 4 units allow an acceleration of 3.2 (1.25 cycle/op per pipeline) and 8 units allow an acceleration of 4.7 (1.7 cycle/op per pipeline). Because the more units are working in parallel, the more busy the memory bus is. However, the hierarchical cache allows to make parallelization a little bit more efficient thanks to the exploitation of the spatial and temporal locality existing between the data retrieved by each BP unit. Moreover, the measured MBRR for 8 units between the leaf loads and the leaf requests stays close to the MBRR measured for a single unit. This MBRR is about 8 for 8 units and 9 for 1 unit.

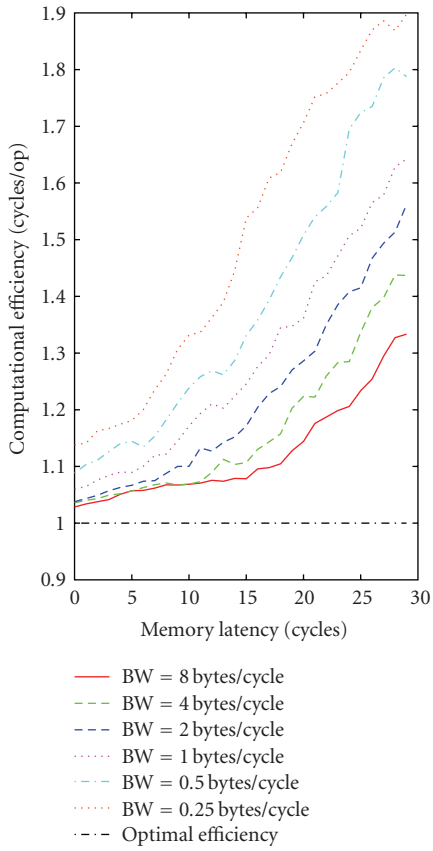


Figure 11: Cycles per operation for one unit of BP with respect to the latency and bandwidth (BW) of the external memory.

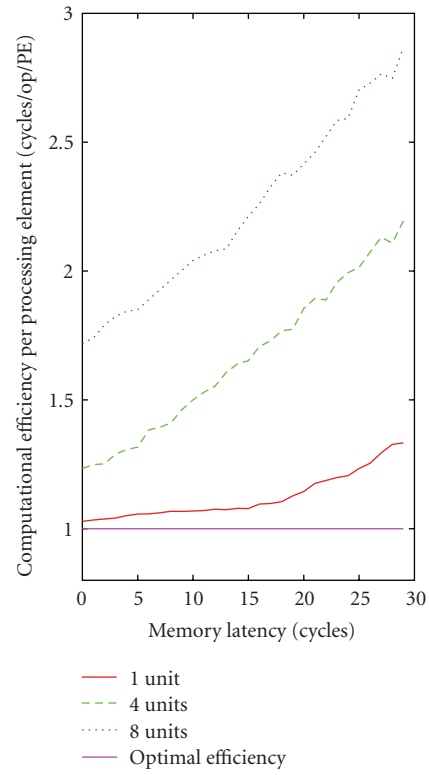


Figure 12: Cycles per operation per processing units for 1, 4, and 8 units of BP with respect to the latency and bandwidth of the external memory.

6. COMPARISON WITH GENERAL PURPOSE AND GRAPHICS PROCESSORS

In Table 3, the 3PA-PET execution times are compared with STIR and the ones from software VBI BP on a desktop PC, a workstation and a GPU.

6.1. CPU implementation

Different software versions of BP, nonoptimized (v1), and optimized (v2 and v3) have been tested and compared to the STIR one on a Pentium 4 and on a bi-Xeon dual core. Two techniques of optimization have been applied with an extensive use of the cache memory and a reduction of the arithmetical operations.

First, an acceleration factor of 3 is obtained due to the reconstruction through blocks of voxels. This software loop reordering increases the use of the L1 cache (16 Ko). Indeed, the time of reconstruction with and without introduction of data locality, is, respectively, 54.7 seconds (v1) and 17.4 seconds (v2).

Secondly, the reduction of the arithmetic operations to compute the projection coordinates allows an acceleration by a factor 7 (software v3/software v2 in Table 3). Indeed, the time performance is improved by a factor 2 due to an incremental computation of the coordinates as done by Kachelrieß et al. [4] and again by a factor 3.5 when the inner loop is over z. The optimized code (VBI-flt(v3)) is presented in Algorithm 2.

for $n = 0$ to $n_{\max}$ **do**
  for delta $= 0$ to $\text{delta}_{\max}$ **do**
    for psi $= 0$ to $\text{psi}_{\max}$ **do**
      $u_{\parallel} = xn_0 \cdot \cos\psi + yn_0 \cdot \sin\psi$
      for $xn = xn_0$ to $xn_{\max}$ **do**
        $u_{\parallel} = u_{\parallel} + \cos\psi$
        for $yn = yn_0$ to $yn_{\max}$ **do**
          $u_{\parallel} = u_{\parallel} + \sin\psi$
          for $zn = zn_0$ to $zn_{\max}$ **do**
$f(xn, yn, zn) += \text{bin}(\text{delta}, \text{psi}, u_{\parallel}, v_{\parallel})$

Algorithm 2: Reduction of operations to compute $u_{\parallel}$ (same techniques used for $v_{\parallel}$).

Finally, this code has been parallelized using the *pthread* C-library to use the four cores of a bi-Xeon dual core workstation. One thread is associated to the reconstruction of one block.

6.2. GPU implementation

Current GPUs are cost effective solutions for the implementation of 3D tomography reconstruction because of their

Table 3: Compared time performance for the 3D PET BP of a $128 \times 128 \times 63$ volume from a Siemens HR+ sinogram (5 segments, span 9, 96 angles of projection). Throughput of reconstruction (cycles per voxel update) is presented for the global architecture and per processing element (PE).

3D-BP algorithm	PE (threads)	Time	Cycles/Op	
			/PE	total
Desktop PC: Pentium 4 (core frequency = 3.2 Ghz, $BW_{mem} = 64$ GB/s)				
STIR[1]	1	11.13 s	70.4	70.4
VBI-flt(v1)	1	54.7 s	355	355
VBI-flt(v2)	1	17.4 s	113	113
VBI-flt(v3)	1	2.5 s	16	16
Workstation: bi-Xeon dual core (core frequency = 3 Ghz, $BW_{mem} = 10.6$ GB/s)				
STIR[1]	1 (1)	5.74 s	34.5	34.5
VBI-flt(v3)	1 (1)	1.17 s	7.1	7.1
VBI-flt(v3)	2 (2)	583 ms	7.06	3.53
VBI-flt(v3)	4 (4)	294 ms	7.12	1.78
GPU: GTS8800 (shader frequency = 1.2 Ghz, $BW_{mem} = 64$ GB/s)				
VBI-flt(v4)	96 (192)	99 ms	25.9	0.27
VBI-flt(v5)	96 (192)	50 ms	13.0	0.14
FPGA[2]: virtex 4 (frequency = 200 Mhz, $BW_{mem} = 0.8$ GB/s, $l_{mem} = 25$ nanoseconds)				
VBI-fix	1	2.5 s	1	1
VBI-fix	4	774 ms	1.25	0.31
VBI-fix	8	526 ms	1.7	0.21
ASIC[3]: one memory bank (frequency = 1.2 Ghz, $BW_{mem} = 4.8$ GB/s, $l_{mem} = 25$ nanoseconds)				
VBI-fix	1	499 ms	1.21	1.21
VBI-fix	4	214 ms	2.07	0.517
VBI-fix	8	135 ms	2.62	0.328
ASIC[3]: five memory banks (frequency = 1.2 Ghz, $BW_{mem} = 24$ GB/s, $l_{mem} = 25$ nanoseconds)				
VBI-fix	40	27 ms	2.62	0.065

[1] Time normalized to a $128*128*63$ volume. (STIR reconstructs $64^2\pi*63$ cylindrical volumes).
[2] 35 Mhz results scaled to 200 Mhz ($l_{mem} = 5$ cycles).
[3] 35 Mhz results scaled to 1,2 Ghz ($l_{mem} = 30$ cycles).

high level of parallelism. Moreover, the Nvidia GPUs are efficiently and easily programed with the CUDA environment.

The Nvidia Geforce 8880 family has 2 to 16 vector processors (12 in our case), each one having 8 stream processors. It is programmable using standard C language with a few extensions without any knowledge about graphics pipeline. A nonincremental code is parallelized to run efficiently on these 12×8 multithreaded stream processors. One thread is associated to one voxel reconstruction. Threads are grouped in blocks (16×16 in our case) which are scheduled at runtime, one block per vector processor. Each couple of vector processors are associated with an 8 KB L1 2D texture read-only cache memory with 1D and 2D hard-wired interpolation. Moreover, the GPU offers a high memory bandwidth ($BW_{mem} = 64$ GB/s) and uses floating-point computation. This makes it possible to efficiently parallelize the BP loops, as blocks of voxels correspond to 2D blocks of threads having access to the read-only sinogram organized in 2D arrays through the 2D cache memory. The voxels are also organized in 2D arrays, each divided in 64 16×16 blocks associated to a grid of 64 16×16 blocks of threads. Thus each thread is responsible of 63 voxels considering a $63 \times 128 \times 128$ volume.

Two versions of thread code have been implemented. In the VBI-flt(v4) thread code, the loop over ψ is the inner loop, while in VBI-flt(v5) thread code, the loop over z is the inner loop as it is done for the VBI-flt(v3) CPU code. This allows a reduction of the number of projection coordinate computation. A speedup factor of 2 is obtained with this code optimization (see Table 3).

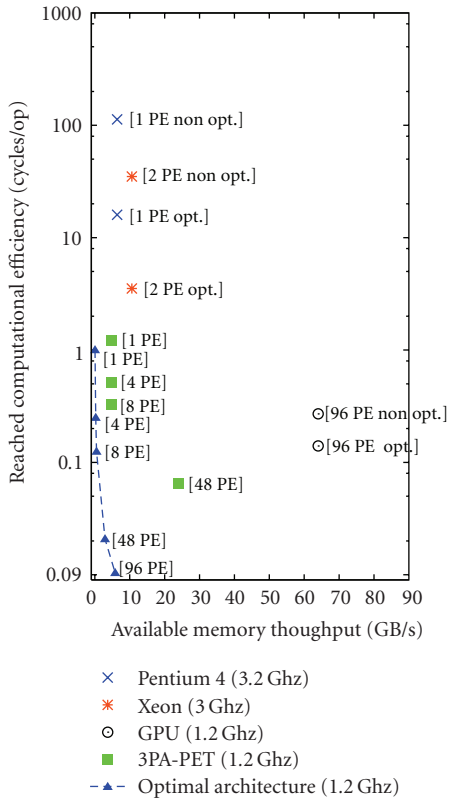


Figure 13: Memory throughput exploitation by CPU (optimized and nonoptimized code), GPU (optimized and nonoptimized code), and 3PA-PET implementations. 3PA-PET and optimal architecture results are obtained with $l_{\text{mem}} = 25$ nanoseconds.

6.3. Discussion

The reconstruction times and efficiencies, global and per processing element (PE), are presented in Table 3 for CPU, GPU, and our 3PA-PET. To fairly compare our architecture with other technologies, the time measured on a Virtex 2 Pro has been scaled to a virtex 4. Indeed, this technology is the same generation as the CPU and GPU used in this study. We have scaled the 35 MHz results to the GPU frequency (1.2 GHz) as well. This higher frequency could be reached through the design of a customized integrated circuit like an ASIC. Moreover, as Nvidia GTS 8800 GPU has five memory banks, we also present a prospective ASIC architecture which would have also five memory banks coupled with 5 processing blocks of 8 BP units each.

For the 200 MHz and the 1.2 GHz 3PA-PET, a memory latency (l_{mem}) of 25 nanoseconds has been used for the simulated memory bus. It corresponds, respectively, to a latency of 5 and 30 clock cycles.

On one hand, the GPU is the fastest hardware solution with a final reconstruction time of 50 milliseconds. The ratio of computation over memory access is high enough for the GPU to allow the automatic overlapping of memory accesses with computations by the thread scheduling mechanism. Thus, due to its greatest computational power (96 PEs and hard-wired interpolation), Nvidia 8800 GTS graphic processor is 10 times faster than 3PA-PET mapped on a virtex 4, 10 times faster than a Xeon dual core, and 50 times faster than a Pentium 4.

On the other hand, 3PA-PET is the most efficient architecture with a computational efficiency per processing element (PE) of about 2 cycles per operation for a processing block made of 8 units coupled with one memory bank. Because of the fewer available computational and memory resources, the FPGA technology does not allow to have an efficient 3PA-PET system compared to GPU. Nevertheless, considering that a market would exist to justify it, an ASIC with five memory banks and five units of 3PA-PET (8 pipelines each) running at 1.2 Ghz would be twice faster than the Nvidia GPU, 20 times than a Xeon dual core, and 100 times than a Pentium 4. Furthermore, a better tuning of the 3D-AP cache would allow to increase the available parallelism and again increase the speedup. Also, an ASIC implementation would be likely to have a lower consumption than a GPU (Nvidia 8800 GTS needs 130 Watts).

All the studied architectures succeed to benefit from the spatial and temporal localities without any developer effort to set a double buffering memory strategy. This is only possible because of their own memory cache (1D cache for CPU, 2D texture cache for GPU, and the semigeneral purpose 3D-AP cache for 3PA-PET). Nevertheless, 3PA-PET is the one that best exploits the memory throughput, as illustrated in Figure 13. In this figure, all the 3D BP implementations are placed according to their computational efficiency (cycles/op) and to their available memory throughput (GB/s). The "optimal architecture" in this figure corresponds to a hardware architecture that would have an optimal balance between its computational and memory throughputs. Each PE of this optimal architecture computes one operation per cycle and its prefetching memory strategy only loads the necessary data in cache and delivers it in time to the processing units. Of course, the more PEs it has, the greater the memory throughput has to be. As one can observe, 3PA-PET is the architecture with a cache-based memory strategy that is the closest to the optimal one. This makes 3PA-PET the architecture with the best potential of acceleration.

7. CONCLUSION

This paper presents several ways to speed up the BP algorithm on different target architectures: general purpose CPU, GPU, and FPGA/ASIC. These solutions exploit the temporal and 3D spatial locality that can be found in the BP algorithm. A suitable loop reordering shows to be efficient despite the high nonlinearity of the algorithm. The 3PA-PET (prefetched and parallelized architecture for PET) architecture is the one that makes the best use of this locality and allows a high level of parallelization with a high computational throughput.

Thanks to the 3D-AP cache together with a loop reordering, 3PA-PET architecture proves to be an efficient parallel architecture that overcomes the memory bottleneck. Indeed, as it has been measured on an SoPC prototype, the pipelines are seldom stalled and the high latency and low bandwidth of memories can be overcome. Moreover,

the comparison between the 3PA-PET architecture with a general purpose processor and a GPU highlights 3PA-PET efficiency. On one hand, the GPU has the best reconstruction time on a wall clock (followed by 3PA-PET and the CPU), on the other hand 3PA-PET makes the best use of the pipeline and clock cycles. An ASIC implementation with the same technological resources than of GPU would be of lower power consumption and would be faster: it would be twice faster than today's GPUs and 20 times faster than CPUs.

To conclude, the method of loop reordering and the use of an appropriate cache could be extended to other algorithms. The architecture principles presented in this article could be applied for the cone beam BP needed in CT reconstruction.

REFERENCES

[1] P. E. Kinahan, M. Defrise, and R. Clackdoyle, "Analytic image reconstruction methods," in *Emission Tomography: The Fundamentals of PET and SPECT*, pp. 421–442, Elsevier Academic Press, San Diego, Calif, USA, 2004.

[2] J. P. Jones, A. Rahmim, M. Sibomana, et al., "Data processing methods for a high throughput brain imaging PET research center," in *Proceedings of the IEEE Nuclear Science Symposium (NSS '06)*, vol. 4, pp. 2224–2228, San Diego, Calif, USA, October-November 2006.

[3] S. Mancini and N. Eveno, "An IIR based 2D adaptive and predictive cache for image processing," in *Proceedings of the 19th Conference on Design of Circuits and Integrated Systems (DCIS '04)*, p. 85, Bordeaux, France, November 2004.

[4] M. Kachelrieß, M. Knaup, and O. Bockenbach, "Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware," *Medical Physics*, vol. 34, no. 4, pp. 1474–1486, 2007.

[5] M. Schellmann, T. Kösters, and S. Gorlatch, "Parallelization and runtime prediction of the Listmode OSEM algorithm for 3D PET reconstruction," in *Proceedings of the IEEE Nuclear Science Symposium (NSS '06)*, vol. 4, pp. 2190–2195, San Diego, Calif, USA, October-November 2006.

[6] D. W. Shattuck, J. Rapela, E. Asma, A. Chatzioannou, J. Qi, and R. M. Leahy, "Internet2-based 3D PET image recontruction using a PC cluster," *Physics in Medicine and Biology*, vol. 47, no. 15, pp. 2785–2795, 2002.

[7] T. He, J. Ni, and G. Wang, "A heterogeneous windows cluster system for medical image reconstruction," in *Proceedings of the 1st International Multi-Symposiums on Computer and Computational Sciences (IMSCCS '06)*, vol. 1, pp. 410–415, Zhejiang, China, June 2006.

[8] K. Chidlow and T. Möller, "Rapid emission tomography reconstruction," in *Proceedings of the 3rd Eurographics/IEEE TVCG Intenational Workshop on Volume Graphics (VG '03)*, vol. 45, pp. 15–26, Tokyo, Japan, July 2003.

[9] G. Pratx, G. Chinn, F. Habte, P. Olcott, and C. Levin, "Fully 3-D list-mode OSEM accelerated by graphics processing units," in *Proceedings of the IEEE Nuclear Science Symposium (NSS '06)*, vol. 4, pp. 2196–2202, San Diego, Calif, USA, October-November 2006.

[10] F. Xu and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Physics in Medicine and Biology*, vol. 52, no. 12, pp. 3405–3419, 2007.

[11] H. Scherl, B. Keck, M. Kowarschik, and J. Hornegger, "Fast GPU-based CT reconstruction using the Common Unified Device Architecture (CUDA)," in *Proceedings of the IEEE Nuclear Science Symposium and Medical Imaging Conference (NSS-MIC '07)*, vol. 6, pp. 4464–4466, Honolulu, Hawaii, USA, October-November 2007.

[12] T. Schiwietz, S. Bose, J. Maltz, and R. Westermann, "A fast and high-quality cone beam reconstruction pipeline using the GPU," in *Medical Imaging 2007: Physics of Medical Imaging*, vol. 6510 of *Proceedings of SPIE*, San Diego, Calif, USA, February 2007.

[13] H. Yang, M. Li, K. Koizumi, and H. Kudo, "Accelerating backprojections via CUDA architecture," in *Proceedings of the 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 52–55, Lindau, Germany, July 2007.

[14] D. Riabkov, X. Xue, D. Tubbs, and A. Cheryauka, "Accelerated cone-beam backprojection using GPU-CPU hardware," in *Proceedings of the 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 68–71, Lindau, Germany, July 2007.

[15] H. Scherl, S. Hoppe, F. Dennerlein, et al., "On-the-fly-reconstruction in exact cone-beam CT using the cell broadband engine architecture," in *Proceedings of the 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 29–32, Lindau, Germany, July 2007.

[16] M. Leeser, S. Coric, E. Miller, H. Yu, and M. Trepanier, "Parallel-beam backprojection: an FPGA implementation optimized for medical imaging," *The Journal of VLSI Signal Processing*, vol. 39, no. 3, pp. 295–311, 2005.

[17] X. Li, T. He, S. Wang, G. Wang, and J. Ni, "P2P-enhanced distributed computing in EM medical image reconstruction," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '04)*, vol. 2, pp. 822–828, Las Vegas, Nev, USA, June 2004.

[18] B. Heigl and M. Kowarschik, "High-speed reconstruction for C-arm computed tomography," in *Proceedings of the 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 25–28, Lindau, Germany, July 2007.

[19] I. Goddard and M. Trepanier, "High-speed cone-beam reconstruction: an embedded systems approach," in *Medical Imaging 2002: Visualization, Image-Guided Procedures, and Display*, vol. 4681 of *Proceedings of SPIE*, pp. 483–491, San Diego, Calif, USA, February 2002.

[20] Terarecon, http://www.terarecon.com.

[21] J. Ni, J. Deng, H. Yu, T. He, and G. Wang, "Analysis of performance evaluation of parallel Katsevich algorithm for 3-D CT image reconstruction," in *Proceedings of the 1st International Multi-Symposiums on Computer and Computational Sciences (IMSCCS '06)*, vol. 1, pp. 258–265, Zhejiang, China, June 2006.

[22] N. Gac, S. Mancini, and M. Desvignes, "Hardware/software 2D-3D backprojection on a SoPC platform," in *Proceedings of the ACM Symposium on Applied Computing (SAC '06)*, vol. 1, pp. 222–228, Dijon, France, April 2006.

[23] K. Thielemans, S. Mustafovic, and C. Tsoumpas, "STIR: software for tomographic image reconstruction release 2," in *Proceedings of the IEEE Nuclear Science Symposium (NSS '06)*, vol. 4, pp. 2174–2176, San Diego, Calif, USA, October-November 2006.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 961315, 13 pages
doi:10.1155/2008/961315

Research Article

An SIMD Programmable Vision Chip with High-Speed Focal Plane Image Processing

Dominique Ginhac, Jérôme Dubois, Michel Paindavoine, and Barthélémy Heyrman

Laboratoire d'Electronique Informatique et Image (LE2I), UMR CNRS 5158, Health-STIC Federative Research Institute (IFR100), Burgundy University, 21078 Dijon, France

Correspondence should be addressed to Dominique Ginhac, dginhac@u-bourgogne.fr

Received 1 March 2008; Revised 13 June 2008; Accepted 12 November 2008

Recommended by Dragomir Milojevic

A high-speed analog VLSI image acquisition and low-level image processing system are presented. The architecture of the chip is based on a dynamically reconfigurable SIMD processor array. The chip features a massively parallel architecture enabling the computation of programmable mask-based image processing in each pixel. Extraction of spatial gradients and convolutions such as Sobel operators are implemented on the circuit. Each pixel includes a photodiode, an amplifier, two storage capacitors, and an analog arithmetic unit based on a four-quadrant multiplier architecture. A 64×64 pixel proof-of-concept chip was fabricated in a 0.35 μm standard CMOS process, with a pixel size of 35 μm $\times$ 35 μm. A dedicated embedded platform including FPGA and ADCs has also been designed to evaluate the vision chip. The chip can capture raw images up to 10 000 frames per second and runs low-level image processing at a framerate of 2 000 to 5 000 frames per second.

1. INTRODUCTION

Today, digital cameras are rapidly becoming ubiquitous, due to reduced costs and increasing demands of multimedia applications. Improvements in the growing digital imaging world continue to be made with two main image sensor technologies: charge-coupled devices (CCDs) and CMOS sensors. Historically, CCDs have been the dominant image-sensor technology. However, the continuous advances in CMOS technology for processors and DRAMs have made CMOS sensor arrays a viable alternative to the popular CCD sensors. This led to the adoption of CMOS image sensors in several high-volume products, such as webcams, mobile phones, PDAs, for example. Furthermore, new recent technologies provide the ability to integrate complete CMOS imaging systems at focal plane, with analog-to-digital conversion, memory and processing [1–5]. By exploiting these advantages, innovative CMOS sensors have been developed and have demonstrated fabrication cost reduction, low power consumption, and size reduction of the camera [6–8].

The main advantage of CMOS image sensors is the flexibility to integrate processing down to the pixel level. As CMOS image sensors technologies scale to 0.18 μm processes and under, processing units can be realized at chip level (*system-on-chip* approach), at column level by dedicating processing elements to one or more columns, or ar pixel-level by integrating a specific unit in each pixel or local of neighboring pixels. Most of the researches deal with chip and column level [9–12]. Indeed, pixel-level processing is generally dismissed because pixel sizes are often too large to be of practical use. However, as CMOS scales, integrating a processing element at each pixel or group of neighboring pixels becomes more feasible since the area occupied by the pixel transistors decreases, leading to an acceptable small pixel size. A fundamental tradeoff must be made between three dependent and correlated variables: pixel size, processing element area, and fill-factor. This implies various points of view:

(1) for a fixed fill-factor and a given processing element area, the pixel size is reduced with technology improvements, as a consequence, reducing pixel size increases spatial resolution for a fixed sensor die size;

(2) for a fixed pixel size and a given processing element area, the photodiode area and the fill-factor increase as technology scales since the area occupied by the

pixel transistors in each processing element decreases. It results in better sensibility, higher dynamic range and signal-to-noise ratio;

(3) for a fixed pixel size and a given fill-factor, the processing element can integrate more functionalities since the transistors require less area as technology scales. Consequently, the image processing capabilities of the sensor increase.

In summary, each new technology process offers (1) to integrate more processing functions in a given silicon area, or (2) to integrate the same functionalities in a smaller silicon area. This can benefit the quality of imaging in terms of resolution, noise, for example, by integrating specific processing functions such as correlated double sampling [13], antiblooming [14], high dynamic range [15], and even all basic camera functions (color processing functions, color correction, white balance adjustment, gamma correction) onto the same camera-on-chip [16]. Furthermore, employing a processing element per pixel offers the ability to exploit the high speed imaging capabilities of the CMOS technology by achieving massively parallel computations [17–20].

In this paper, we discuss hardware implementation issues of a high-speed CMOS imaging system embedding low-level image processing. For this purpose, we designed, fabricated, and tested a proof-of-concept 64 × 64 pixel CMOS analog sensor with per-pixel programmable processing element in a standard 0.35 μm double-poly quadruple-metal CMOS technology.

The rest of the paper is organized as follows. Section 2 is dedicated to the description of the high speed algorithms embedded at pixel-level. Section 3 is a general description of the characteristics of the sensor. These characteristics are well detailed in Section 4, which talks about the design of the circuit, with a full description of the main components such as the photodiode structure, the embedded analog memories, and the arithmetic unit are successively described. In Section 5, we describe the test hardware platform and the chip characterization results, including an analysis of the fixed pattern noise. Finally, some experimental results of high-speed image acquisition with pixel-level processing are provided in Section 6 of this paper.

2. HIGH SPEED FOCAL PLANE IMAGE-PROCESSING CAPABILITIES

In an increasingly digital world, the most part of imaging systems has become almost entirely digital, using only an analog-to-digital (ADC) between the sensor and the processing operators. However, low-level image processing usually involves basic operations using local masks. These local operations are spatially dependent on other pixels around the processed pixel, since the same type of operations is applied to a very large dataset, these low-level tasks are computationally intensive and require a high bandwidth between the image memory and the digital processor. In this case, an analog or a mixed-approach can offer superior performance leading to a smaller, faster, and lower power solution than a digital processor [21]. Low-level image processing tasks are inherently pixel-parallel in nature. Integrating a processing element within each pixel based on a single instruction multiple data (SIMD) architecture is a natural candidate to cope with the processing constraints [18]. This approach is quite interesting for several aspects. First, SIMD image-processing capabilities at focal plane have not been fully exploited because the silicon area available for the processing elements is very limited. Nevertheless, this enables massively parallel computations allowing high framerates up to thousands of images per second. The parallel evaluation of the pixels by the SIMD operators leads to processing times, independent of the resolution of the sensor. In a classical system, in which low-level image processing is externally implemented after digitization, processing times are proportional to the resolution leading to lower framerates as resolution increases. Several papers have demonstrated the potentially outstanding performance of CMOS image sensors [22–24]. Krymski et al. [22] describe a high speed (500 frames/s) large format 1024 × 1024 active pixel sensor (APS) with 1024 ADCs. Stevanovic et al. [23] describe a 256 × 256 APS which achieves more than 1000 frames/s with variable integration times. Kleinfelder et al. [24] describe a 352 × 288 digital pixel sensor (DPS) with per pixel bit parallel ADC achieving 10 000 frames/s or 1 Giga-pixels/s.

Secondly, the high speed imaging capability of CMOS image sensors can benefit the implementation of new complex applications at standard rates and improve the performance of existing video applications such as motion vector estimation [25–27], multiple capture with dynamic range [28–30], motion capture [31], and pattern recognition [32]. Indeed, standard digital systems are unable to operate at high framerates, because of the high output data rate requirements for the sensor, the memory, and the processing elements. Integrating the memory and processing with the sensor on the same chip removes the classical input output bottleneck between the sensor and the external processors in charge of processing the pixel values. Indeed, the bandwidth of the communication between the sensor and the external processors is known as a crucial aspect, especially with high resolution sensors. In such cases, the sensor output data flow can be very high, and needs a lot of hardware resources to convert, process and transmit a lot of information. So, integrating image processing at the sensor level can alleviate the high data rate problem because the pixel values are preprocessed on-chip by the SIMD operators before sending them to the external world via the communication channel. This will result in data reduction, which allows sending the data at lower data-rates, and reduces the effect of the computational-load bottleneck.

Thirdly, one of the main drawbacks to design specific circuits integrating sensing and processing on the same chip is that these vision chips are often built as special-purpose devices, performing specific and dedicated tasks, and not reusable in another context [33]. So, it can be widely beneficial to integrate a versatile device, whose functionality can be easily modified. Moreover, except the basic operations such as convolutions with small masks, the majority of computer vision algorithms require the sequential execution of different successive low-level image processing on the

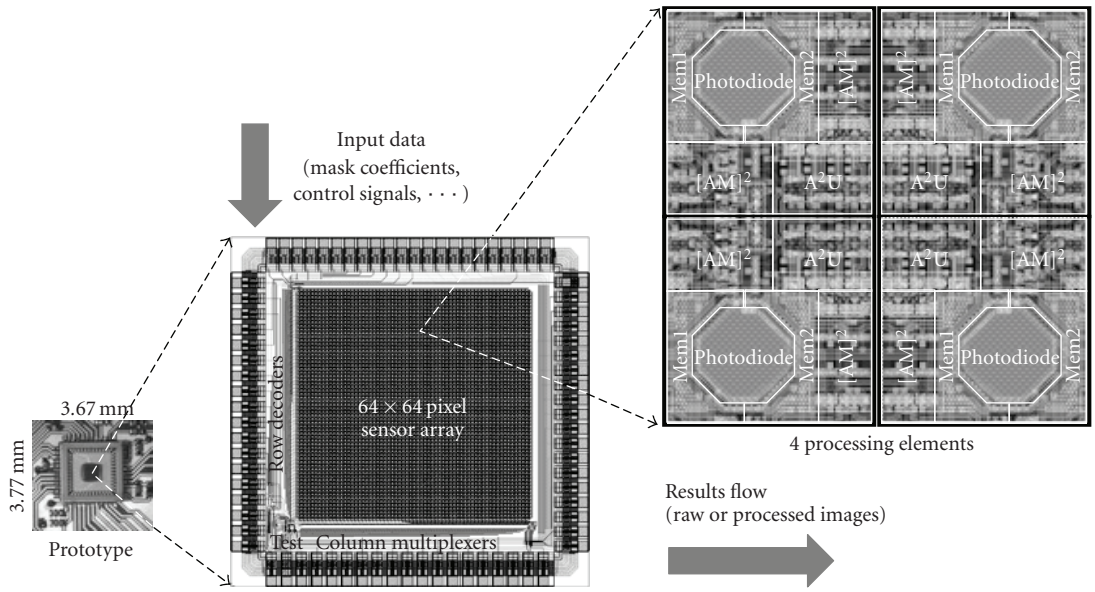


Figure 1: Overview of the image sensor with a processor-per-pixel array.

same data. So, each processing element must be built around a programmable execution unit, communication channels, and local memories dedicated to intermediate results. Because of the very limited silicon area, the processing units are necessarily very simple, providing the best compromise between various factors such as versatility, complexity, parallelism, processing speeds and resolution.

To sum up, the flexibility to integrate processing down to the pixel level allows us to rearchitect the entire imaging system to achieve much higher performances [34]. The key idea is (1) to capture images at a very high framerate, (2) to process the data on each pixel with an SIMD programmable architecture exploiting the high on-chip bandwidth between the sensor, the memory and the elementary processors, and (3) to provide results at the best framerate depending on the complexity of the image processing. In this paper, we present our approach to the design of a massively parallel, SIMD vision chip based implementing low level image processing based on local masks. Our analog processing operators are fully programmable devices by dynamic reconfiguration, and can be viewed as a software-programmable image processor dedicated to low-level image processing. The main objectives of our design are (1) to evaluate the potential for high-speed snapshot imaging and, in particular, to reach a 10 000 frames/s rate, (2) to demonstrate a versatile and reconfigurable processing unit at pixel level, and (3) to provide an original platform for experimenting with low-level image processing algorithms that exploit high-speed imaging.

3. DESCRIPTION OF THE ARCHITECTURE

The proof-of-concept chip presented in this paper is depicted in Figure 1. The core includes a two-dimensional array of 64 × 64 identical processing element (PE). It follows the single instruction multiple data (SIMD) computing paradigm. Each of the PE is able to convolve the pixel value issued from the photodiode by applying a set of mask coefficients to the image pixel values located in a small neighborhood. The key idea is that a global control unit can dynamically reconfigure the convolution kernel masks and then implements the most part of low-level image processing algorithms. This confers the functionality of programmable processing devices to the PEs embedded in the circuit. Each individual PE includes the following elements.

(i) A photodiode dedicated to the optical acquisition of the visual information and the light-to-voltage transduction.

(ii) Two *analog memory, amplifier and multiplexer* structures called $[AM]^2$, which serve as intelligent pixel memories and are able to dissociate the acquisition of the current frame in the first memory and the processing of the previous frames in the second memory.

(iii) An *analog arithmetic unit* named A^2U based on four analog multipliers, which performs the linear combination of the four adjacent pixels using a 2×2 convolution kernel.

In brief, each PE includes 38 transistors integrating all the analog circuitry dedicated to the image processing algorithms. The global size of the PE is $35\,\mu m \times 35\,\mu m$ ($1225\,\mu m^2$). The active area of the photodiode is $300\,\mu m^2$, giving a fill-factor of 25%. The chip has been realized in a standard $0.35\,\mu m$ double-poly quadruple-metal CMOS technology and contains about 160 000 transistors on a 3.67 mm × 3.77 mm die (13.83 mm^2). The chip also contains test structures on the bottom left of the chip. These structures are used for detailed characterization of the photodiodes and processing units.

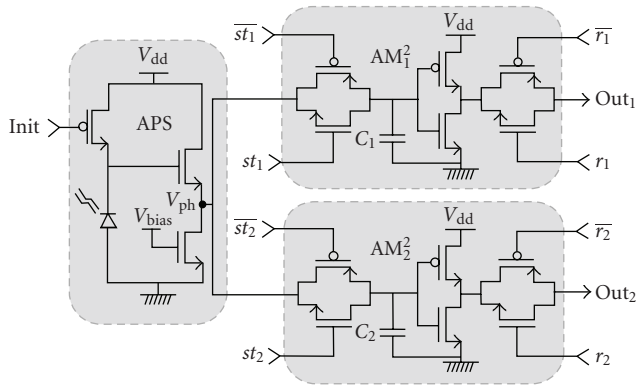


Figure 2: The $[AM]^2$ structure.

4. CIRCUIT DESIGN

4.1. Pixel structure

Each pixel in the CMOS image sensor array consists of a photodiode and a processing unit dedicated to low-level image processing based on neighborhoods. In our chip, the type of photodiodes is one of the simplest photo element in CMOS image sensor technology. It consists of N-type photodiodes based on an n^+-type diffusion in a p-type silicon substrate. The depletion region is formed in the neighborhood of the photodiode cathode. Optically generated photocarriers diffuse to neighboring junctions [35]. In order to achieve good performances, the photodiodes should be designed and optimized carefully, in order to minimize critical parameters such as the dark current and the spectral response [36]. The shape of photodiode layout, the structure of the photodiode, and the layout have significant influences on the performance of the whole imager [37, 38]. The active area of the photodiode absorbs the illumination energy and turns that energy into charge carriers. This active area must be large as possible in order to absorb a maximum of photons. In the mean time, the control circuitry required for the readout of the collected charges and the interelement isolation area must be as small as possible in order to obtain the best fill factor. We have theoretically analyzed, designed and benchmarked different photodiodes shapes [39], and finally, an octagonal shape based on 45° structures was chosen (see Figure 1). More details about the photodiodes design can be found in the aforementioned paper. Here, only the basic concept behind the photodiodes design has been briefly overviewed.

The second part of the pixel is the analog processing unit. Existing works on analog pixel-level image processing can be classified into two main categories. The first one is intrapixel, in which processing is performed on the individual pixels in order to improve image quality, such as the classical active pixel sensor or APS [9, 40]. The second category is interpixel, where the processing is dedicated to groups of pixels in order to perform some early vision processing and not merely to capture images. Our work clearly takes place in this category because our main objective is the implementation of various in situ image processing using local neighborhoods. Based on this design concept, this forces a rethinking of the spatial distribution of the processing resources, so that each computational unit can easily use a programmable neighborhood of pixels. For this purpose, the pixels are mirrored about the horizontal and the vertical axes in order to share the different *analog arithmetic units* (A^2Us). As example, a block of 2×2 pixels is depicted in Figure 1. The main feature of its innovative distribution is to optimize the compactness of the metal interconnections with pixels, to contribute to a better fill factor, and to provide generality of high-speed processing based on neighborhood of pixels.

4.2. Analog memory, amplifier and multiplexer: $[AM]^2$

In order to increase the algorithmic possibilities of the architecture, the key point is the separation of the acquisition of the light inside the photodiode and the readout of the stored value at pixel-level [41]. Thus, the storage element should keep the output voltage of the previous frames whereas the sensor integrates photocurrent for a new frame. So, we have designed and implemented dedicated pixels including a light sensitive structure and two specific circuits called *analog memory, amplifier, and multiplexer* ($[AM]^2$), as shown in Figure 2.

The system has five successive operation modes: reset, integration, storage, amplification, and readout. All these phases are externally controlled by global signals common to the full array of pixels. They all occur in parallel over the sensor (*snapshot* mode) in order to avoid any distortion due to a row-by-row reset. In each pixel, the photosensor is an N-type photodiode associated with a PMOS transistor reset. This switch resets the integrating node to the fixed voltage V_{dd}. The pixel array is held in the reset mode until the *init* signal raises, turning the PMOS transistor off. Then, the photodiode discharges for a fixed period, according to the incidental luminous flow. The first NMOS transistor acts as a transconductance, producing the voltage V_{ph}, directly proportional to the incident light intensity. The integrated voltage is polarized around $V_{dd}/2$ by the second NMOS transistor. The calibration of the structure is ensured by the positive reference bias voltage ($V_{bias} = 1.35\,V$).

Following the acquisition stage, two identical subcircuits $[AM]_i^2$ (with $i = 1, 2$) take place to realize the storage phase of V_{ph}. Each $[AM]^2$ includes three pairs of NMOS and PMOS transistors and a capacitor which acts as an analog memory. The subcircuit $[AM]_i^2$ is selected when the st_i signal is turned on. Then, the associated analog switch is open allowing the integration of the photogenerated current in the corresponding C_i capacitor. Consequently, the capacitors are able to store the pixel value during the frame capture from one of the two switches. The capacitors are implemented with double-polysilicium. The size of the capacitors is as large as possible in order to respect the fill-factor and the pixel-size requirements. The capacitors values are about 40 fF. They are able to store the pixel value for 20 milliseconds with an error lower than 4%. Behind the storage subcircuit, a basic CMOS inverter is integrated. This inverter serves as a linear high-gain amplifier since the pixel signal is polarized

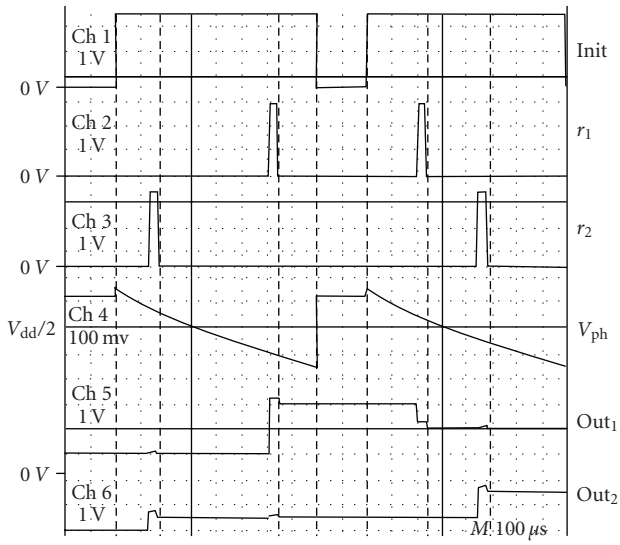


Figure 3: High-speed sequence capture with basic image processing.

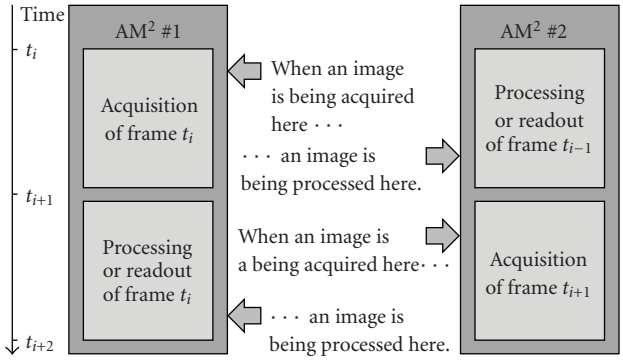


Figure 4: Parallelism between capture sequence and readout sequence.

around $V_{dd}/2$. Finally, the last phase consists in the readout of the stored values in the capacitors C_i. The integrated voltage across the capacitor C_i can be readout on the output out_i through one of the two switches, controlled by the r_i signals.

Figure 3 describes the experimental results of successive acquisitions in an individual pixel. The acquisitions occur when one of the two signals st_1 or st_2 goes high. The two combinations of acquisitions are presented in this example. After the first reset, st_2 is followed by st_1 whereas the inverted sequence of acquisitions is realized after the second reset. The signal V_{ph} gives the voltage corresponding to the incidental illumination on the pixel and the the two outputs (out_1 and out_2 give the voltage stored in each of the capacitors when the associated readout signal raises.

One of the main advantages of the two $[AM]^2$ structures is that the capture sequence can be made in the first memory in parallel with a readout sequence and/or processing sequence of the previous image stored in the second memory, as shown in Figure 4.

Such a strategy has several advantages.

(1) The framerate can be increased (up to 2×) without reducing the exposure time.

(2) The image acquisition is time-decorrelated from image processing, implying that the architecture performance is always the highest, and the processing framerate is maximum.

(3) A new image is always available without spending any integration time.

4.3. Analog arithmetic unit: A^2U

When designing a pixel-level processing unit, you should consider adopting efficient strategies to minimize the silicon area occupied by the processor. To that end, we designed an analog arithmetic unit (A^2U) which is able to perform convolution of the pixels with a 2×2 dynamic kernel. This unit is based on four-quadrant analog multipliers [42, 43] named M1, M2, M3, and M4, as illustrated in Figure 5. Each multiplier requires 5 transistors. So, the transistor count of the complete unit is only 22 transistors. The last two transistors not depicted on the Figure 5 serve as an output switch, driven by the column signal. It features relative small area, simplicity, and high speed. These characteristics make it an interesting choice in low-level image processing embedded at focal plane. Each multiplier M_i (with $i = 1, \ldots, 4$) takes two analog signals V_{i1} and V_{i2} and produces an output V_{iS} which is their product. The outputs of multipliers are all interconnected with a diode-connected transistor employed as load. Consequently, the global operation result at the V_S point is a linear combination of the four products V_{iS}. Image processing operations such as spatial convolution can be easily performed by connecting the inputs V_{i1} to the kernel coefficients and the inputs V_{i2} to the corresponding pixel values.

In order to keep the analysis simple, it is assumed in this section that the contribution of parasitic capacitances is negligible. Considering the MOS transistors operating in subthreshold region, the output node V_{iS} of a multiplier can be expressed as a function of the two inputs V_{i1} and V_{i2} as follows:

$$k_r(V_{ThN} - V_{i1})(V_{i1} - V_{i2} - V_{ThN}) = (V_{i1} - V_{iS} - V_{ThN})(V_{i2} - V_{iS} - V_{ThN} - V_{ThP}) \quad (1)$$

with k_r represents the transconductance factor, V_{ThN} and V_{ThP} are the threshold voltages for the NMOS and PMOS transistors. Around the operating point ($V_{dd}/2$), the variations of the output node mainly depend on the product $V_{i1}V_{i2}$. So, the equation 1 can be simplified and finally, the output node V_{iS} can be expressed as a simple first-order of the two input voltages V_{i1} and V_{i2}.

$$V_{iS} = MV_{i1}V_{i2} \quad \text{with} \quad M = \frac{k_r - 1}{2V_{ThN} + V_{ThP}} \approx 8.07/\text{V}. \quad (2)$$

The important value of the coefficient M gives to the structure a good robustness by limiting the impact of the second-order intermodulation products. The first consequence is

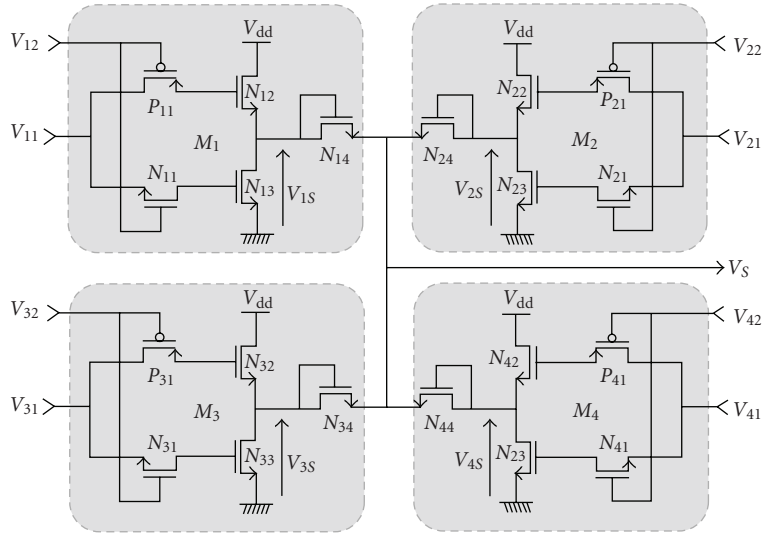


FIGURE 5: The A^{2}U structure.

a better linearity of our multiplier design integrating only 5 transistors.

The theoretical analysis has been validated by an experimental study realized on the test structures embedded on the chip. Figure 6 shows the experimental measures obtained with two cosine signals as inputs of the multiplier structure.

$$\begin{aligned} V_{i1} &= A\cos(2\pi f_1) \quad \text{with} \quad f_1 = 2.5\,\text{kHz}, \\ V_{i2} &= B\cos(2\pi f_2) \quad \text{with} \quad f_2 = 20\,\text{kHz}. \end{aligned} \tag{3}$$

In an ideal case, the voltage V_S at the output of the multiplier can be expressed as

$$V_S = \frac{AB}{2}[\cos(2\pi(f_2 - f_1)) + \cos(2\pi(f_2 + f_1))]. \tag{4}$$

The frequency spectrum, represented in Figure 6(b) contains two main frequencies (17.5 kHz and 22.5 kHz) around the carrier frequency. The residues which appear in the spectrum are known as intermodulation products. Intermodulation is caused by nonlinear behavior of the structure (around 10 kHz and 30 kHz) and the insulation defects of input pads (at 40 kHz). Nevertheless, the amplitude of these intermodulation products is significantly lower than the two main frequencies. Indeed, at 40 kHz, the level of intermodulation is 9 dB under the level of the main frequencies. Therefore, the contribution of the insulation defect is eight times smaller than the main signals. Furthermore, the experimental measures highlighted that the linearity of the multiplier is maximum for input signals with amplitude range between 0.6 V and 2.6 V. This corresponds to the typical range of values coming from the pixel since the [AM]^2 structures provide values around $V_{dd}/2 = 1.65$ V.

5. CHIP CHARACTERIZATION

An experimental 64 × 64 pixel image sensor has been developed in a 0.35 μm, 3.3 V, standard CMOS process with poly-poly capacitors. Its functional testing and its characterization were performed using a specific hardware platform. The hardware part of the imaging system contains a one million Gates Spartan-3 FPGA board with 32MB SDRAM embedded. This FPGA board is the XSA-3S1000 from XESS Corporation. An interface acquisition circuit includes three ADCs from analog device (AD9048), high-speed LM6171 amplifiers and others elements such as the motor lens. Figure 7 shows the schematic and some pictures of the experimental platform.

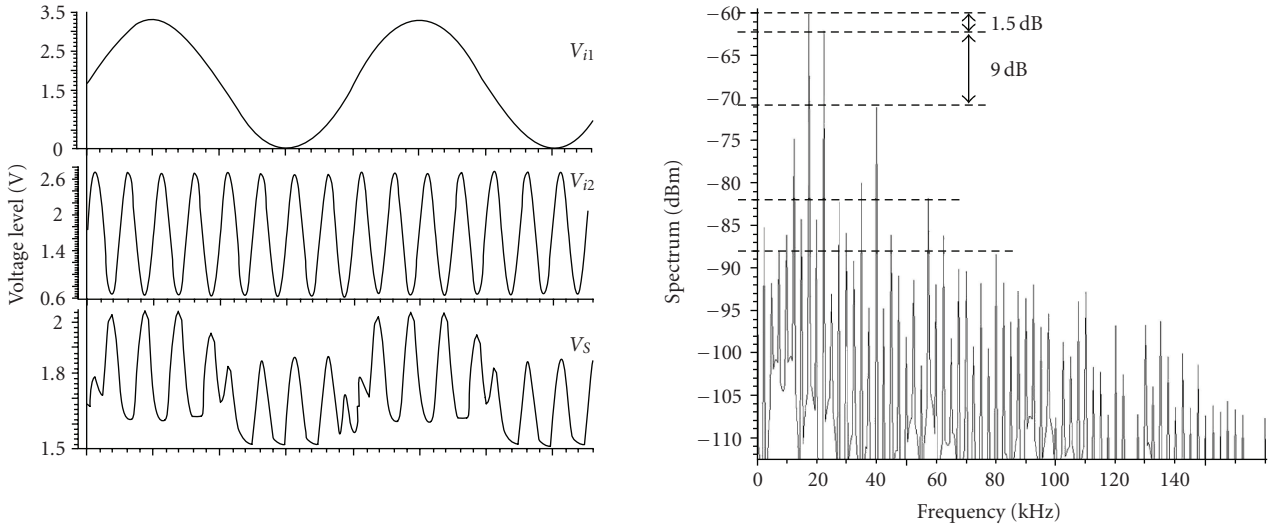
5.1. Electrical characterization

The sensor was quantitatively tested for conversion gain, sensitivity, fixed pattern noise, thermal reset noise, output levels disparities, voltage gain of the amplifier stage, linear flux, and dynamic range. Table 1 summarizes the main chip properties and the characterization results.

To determine these values, the sensor included specific test pixels in which some internal node voltages can be directly read. The test equipment hardware is based on a light generator with wavelength of 400 nm to 1100 nm. The sensor conversion gain was evaluated to 54 μV/e$^-$ RMS with a sensitivity of 0.15 V/lux.s, thanks to the octagonal shape of the photodiode and the fill-factor of 25%. At 10 000 frames/s, measured nonlinearity is 0.12% over a 2 V range. These performances are similar to the sensor described in [24]. According to the experimental results, the voltage gain of the amplifier stage of the two [AM]^2 is $Av = 12$ and the disparities on the output levels are about 4.3%.

5.2. Fixed pattern noise

Image sensors always suffer from technology related nonidealities that can limit the performances of the vision system. Among them, fixed pattern noise (FPN) is the variation in output pixel values, under uniform illumination, due to



(a) Multiplication of cosine signals

(b) Frequency spectrum of the result of multiplication

Figure 6: Experimental results of the multiplication of two cosine signals by the four-quadrant multiplier.

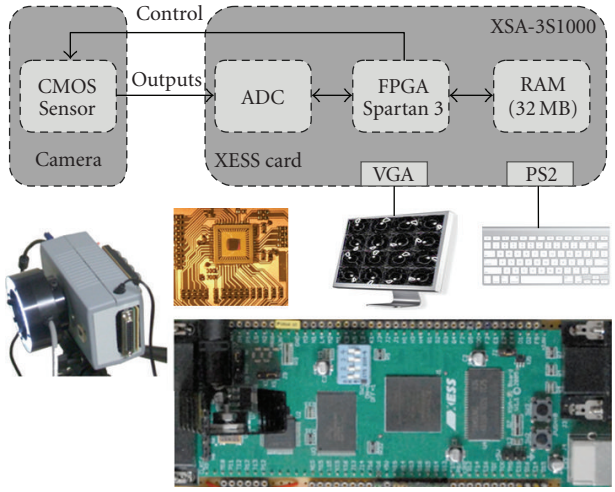


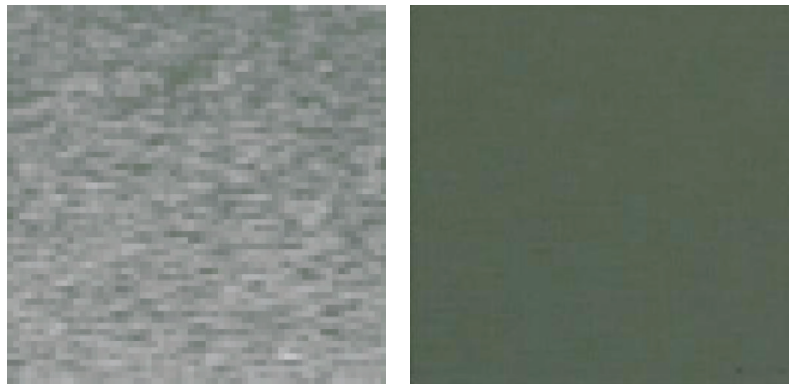
Figure 7: Block diagram and pictures of the hardware platform including FPGA board and CMOS sensor.

Table 1: Chip characteristics and measurements.

Technology	0.35 μm 2-poly 4-metal CMOS
Array size	64 × 64
Chip area	13.8 mm^2
Pixel size	35 μm × 35 μm
Number of transistors	160 000
Number of transistors/pixel	38
Sensor fill factor	25%
Dynamic power consumption	110 mW
Conversion gain	54 μV/e$^-$ RMS
Sensitivity	0.15 V/lux.s
Fixed pattern noise retina (FPN), dark	225 μV RMS
Thermal reset noise	68 μV RMS
Output levels disparities	4.3%
Voltage gain of the amplifier stage	12
Linear flux	98.5%
Dynamic range	68 dB

device and interconnect mismatches across the image sensor. Two main types of FPN occur in CMOS sensors. First, offset FPN which takes place into the pixel is due to fluctuations in the threshold voltage of the transistors. Second, the most important source of FPN is introduced by the column amplifiers used in standard APS systems. In our approach, the layout is symmetrically built in order to reduce the offset FPN among each block of four pixels and to ensure uniform spatial sampling, as already depicted in the layout of a 2 × 2 pixel block in Figure 1.

Furthermore, our chip does not include any column amplifier since the amplification of the pixel values takes place into the pixel by means of an inverter. So, the gain FPN is very limited and only depends on the mismatch of the two transistors. FPN can be reduced by correlated double sampling (CDS). To implement CDS, each pixel output needs to be read twice, once after reset and a second time at the end of integration. The correct pixel signal is obtained by substracting the two values. A CDS can be easily implemented in our chip. For this purpose, the first analog memory stores the pixel value just after the reset signal and the second memory stores the value at the end of integration. Then, at the end of the image acquisition, the two values can be transferred to the FPGA, responsible for producing the difference. In Figure 8 the two images show fixed pattern noise with and without CDS using a 1 millisecond integration time. On the left image, the FPN is mainly due to the random variations in the offset voltages of the pixel-level analog structures. The



(a) (b)

Figure 8: Images of fixed pattern noise (a) without CDS and (b) with CDS for an integration time of 1 millisecond.

experimental benchmarks of our chip reveal an FPN value of 225 μV RMS. The right picture shows the same image after analog CDS, performed as described above. The final FPN has been reduced by a factor of 34 to 6.6 μV. In the rest of the results, CDS has not been implemented since FPN has low values. Only, an entire dark image is substracted from the output images on the FPGA. Focus has been made on the development of low-level image processing using the two analog memories and the associated processing unit.

6. HIGH-SPEED IMAGE PROCESSING APPLICATIONS

In this section, we provide experimental results of image processing implemented on our high-speed vision chip. First, we demonstrate the possibility of acquisition of raw images at different framerates, up to 10 000 frames/s. Secondly, we present an implementation of edge detection, based on the well-known Sobel operator.

6.1. Sample images

The prototype chip was used for acquisition of raw images. First, sample raw images of stationary scenes were captured at different framerates, as shown in Figure 9. In the three views, no image processing is performed on the video stream, except for amplification of the photodiodes signal. From left to right, we can see a human face obtained at 1 000 frames/s, a static electric fan at 5 000 frames/s, and an electronic chip at 10 000 frames/s.

Figure 10 represents different frames of a moving object, namely, a milk drop splashing sequence. In order to capture the details of such a rapidly moving scene, the sensor operates at 2 500 frames/s. and stores a sequence of 50 images. The frames 1, 5, 10, 15, 20, 25, 30, and 40 are shown in the figure.

6.2. Sobel operator

The characteristics of our sensor, especially the analog processing unit, make it extremely useful for low-level image processing based on convolution masks. In this paragraph, we report an edge detector, the Sobel detector. The Sobel operator estimates the gradient of a 2D image. This algorithm is used for edge detection in the preprocessing stage of computer vision systems. The classical algorithm uses a pair of 3×3 convolution kernels (5), one to detect changes along the vertical axis (h_1) and another one to detect horizontal contrast (h_2). For this purpose, the algorithm performs a convolution between the image and the sliding convolution mask over the image. It manipulates 9 pixels for each value to produce. The value corresponds to an approximation of the gradient centered on the processed image area:

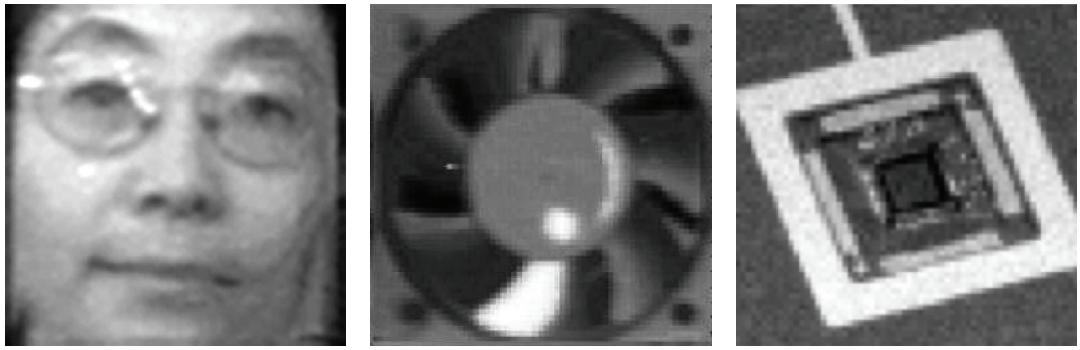
$$h_1 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad h_2 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}. \tag{5}$$

The structure of our architecture is well adapted to the evaluation of the Sobel algorithm. It leads to the result directly centered on the photosensor and directed along the natural axes of the image. The gradient is computed in each pixel of the image by performing successive linear combinations of the 4 adjacent pixels. For this purpose, each 3×3 kernel mask is decomposed into two 2×2 masks that successively operate on the whole image. For the kernel h_1, the corresponding 2×2 masks are:

$$m_1 = \begin{pmatrix} -1 & 0 \\ -1 & 0 \end{pmatrix}, \qquad m_2 = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{6}$$

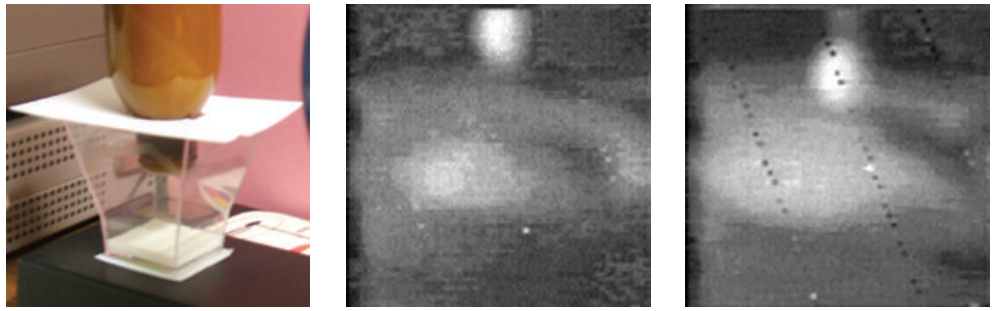
Figure 11 represents the 3×3 mask centered on the pixel ph_5. Each octagonal photodiode ph_i ($i = 1, \ldots, 9$) is associated with a processing element PE_i, represented with a circle on the figure. Each PE_i is positioned on the bottom right of its photodiode, as in the real layout of the circuit (see Figure 1). The first mask m_1 contributes to evaluate the following series of operations for the four PE_is:

$$\begin{aligned} V_{11} &= -(V_{\mathrm{ph}_1} + V_{\mathrm{ph}_4}), \\ V_{12} &= -(V_{\mathrm{ph}_2} + V_{\mathrm{ph}_5}), \\ V_{14} &= -(V_{\mathrm{ph}_4} + V_{\mathrm{ph}_7}), \\ V_{15} &= -(V_{\mathrm{ph}_5} + V_{\mathrm{ph}_8}), \end{aligned} \tag{7}$$

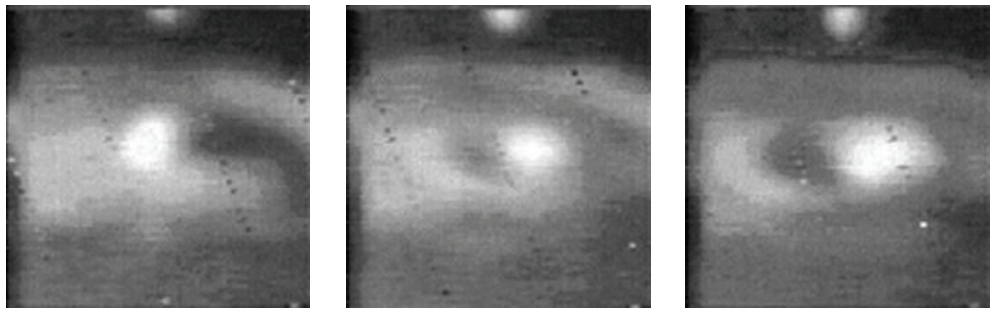


(a) (b) (c)

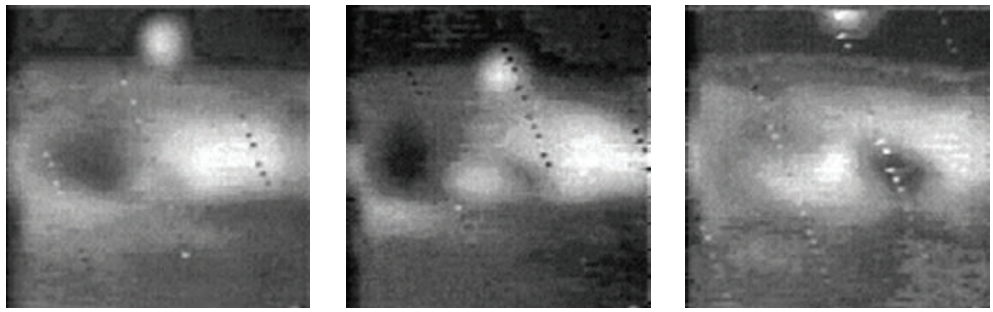
Figure 9: Various raw images acquisition at 1 000, 5 000 and 10 000 frames/s.



Experimental system Frame 1 Frame 5



Frame 10 Frame 15 Frame 20



Frame 25 Frame 30 Frame 40

Figure 10: A 2 500 frames/s video sequence of a milk drop splashing.

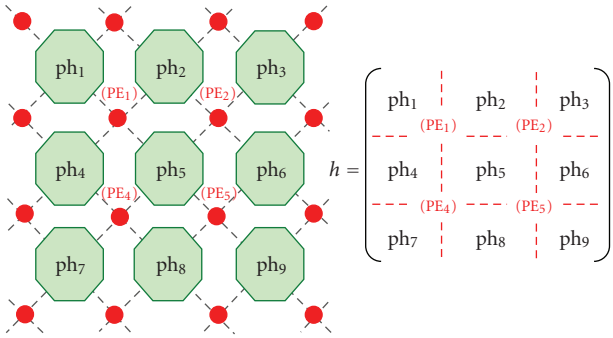


FIGURE 11: 3 × 3 kernel used by the 4 processing elements.

and the second mask m_2 computes:

$$\begin{aligned} V_{21} &= +(V_{\mathrm{ph}_2} + V_{\mathrm{ph}_5}), \\ V_{22} &= +(V_{\mathrm{ph}_3} + V_{\mathrm{ph}_6}), \\ V_{24} &= +(V_{\mathrm{ph}_5} + V_{\mathrm{ph}_8}), \\ V_{25} &= +(V_{\mathrm{ph}_6} + V_{\mathrm{ph}_9}) \end{aligned} \tag{8}$$

with V_{ij} corresponding to the result provided by the processing element PE_j ($j = 1, 2, \ldots, 9$) with the mask m_i ($i = 1, 2$), and V_{ph_k} ($k = 1, 2, \ldots, 9$), the voltages representing the incidental illumination on each photodiode ph_k. Then, the evaluation of the gradient at the center of the mask can be computed by summing the different values on the external FPGA. Note that $V_{12} = -V_{21}$ and $V_{15} = -V_{24}$. So, the final sum can be simplified and written as $V_{h1} = V_{11} + V_{22} + V_{25} + V_{14}$. If we define a retina cycle as the time spent for the configuration of the coefficients kernel and the preprocessing of the image, the evaluation of the gradient on the vertical direction only spends a frame acquisition and two retina cycles. By generalization, the estimation of the complete gradient along the two axes spend 4 cycles because it involves 4 dynamic configurations.

In short, the dynamic assignment of coefficient values from the external processor gives the system some interesting dynamic properties. The system can be easily reconfigured by changing the internal coefficients for the masks between two successive computations. First, this allows the possibility to dynamically change the image processing algorithms embedded in the sensor. Secondly, this enables the evaluation of some complex pixel-level algorithms, implying different successive convolutions. The images can be captured at higher framerates than the standard framerate, processed by exploiting the the analog memories and the reconfigurable processing elements and output at a lower framerate depending of the number of the dynamic reconfigurations. Moreover, the analog arithmetic units implementing these pixel-level convolutions drastically decrease the number of single operations such as additions and multiplications executed by an external processor (an FPGA in our case) as shown in Figure 7. Indeed, in the case of our experimental 64 × 64 pixel sensor, the peak performance is equivalent to 4 parallel signed multiplications by pixel at 10 000 frames/s, that is, more than 160 million multiplications per second.

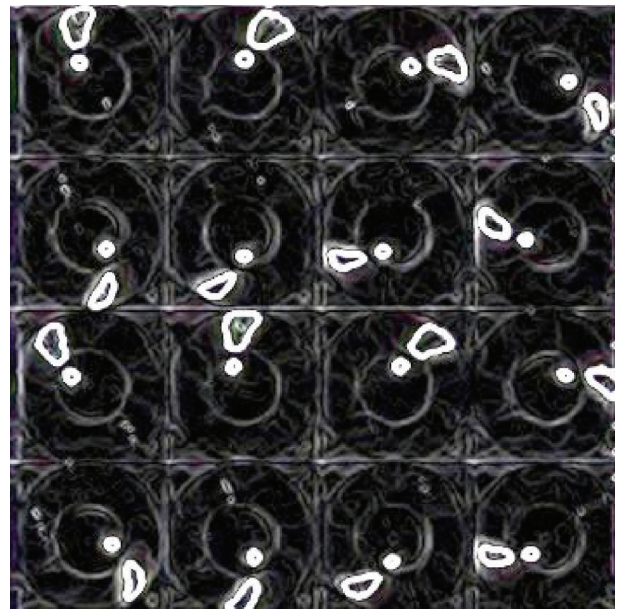


FIGURE 12: Sequence of 16 images with Sobel operator.

With a VGA resolution (640 × 480), the performance level would increase to a factor of 75, leading to about 12 billion multiplications per second. Processing this data flow by external processors will imply important hardware resources in order to cope with the temporal constraints.

As an illustration of the Sobel algorithm, Figure 12 is an example sequence of 16 images of a moving object, namely, an electric fan. Two white specific markers are placed on the fan, that is a small circle near the rotor and a painted blade. The speed rotation of the fan is 3750 rpm. In order to capture such a rapidly moving object, a short integration time (100 microseconds) was used for the frames acquisition. The Sobel algorithm allows to distinguish clearly the two white markers even with a high framerate.

6.3. *Strategies used for general spatial filter*

In the preceding sections, we focused on 2 × 2 and 3 × 3 convolution masks. In the case of a 2 × 2 mask, the coefficients are fixed once before the beginning of the acquisition frame. In the case of a 3 × 3 mask, two possibilities can occur. First, the 3 × 3 mask presents some symmetrical properties (such as the Sobel kernel) and then the coefficients values can be fixed as in a 2 × 2 mask. Second, if the mask is not symmetric, it is necessary to dynamically reconfigure the coefficients during the acquisition frame. For masks which size is greater than 3 × 3 and more generally in the case of an $N \times N$ mask, a dynamic reconfiguration of coefficients is necessary during the acquisition frame in order to evaluate the successive values of the linear combinations of pixels.

7. COMPARISON WITH OTHER SIMD VISION CHIPS

Table 2 shows an overview of some representative SIMD vision chips based on different alternatives for implementing

Table 2: Comparison with other SIMD sensors.

Chip	This work	SCAMP-3 [44, 45]	MIMD IP chip [46]	ACE16k [18, 47]	VCS-IV [48, 49]	PVLSAR2.2 [50]
Technology	0.35 μm	0.35 μm	1.2 μm	0.35 μm	0.35 μm	0.8 μm
Resolution	6 × 64	128 × 128	80 × 78	128 × 128	64 × 64	128 × 128
Pixel pitch	35 μm × 35 μm	49.35 μm × 49.35 μm	45.6 μm × 45 μm	75.5 μm × 75.3 μm	67.4 μm × 67.4 μm	60 μm × 60 μm
Fill factor	25%	5.6%	33%	?	10%	30%
Transistors/PE	38 tr.	128 tr.	9 tr.	198 tr.	84 tr.	50 tr.
PE type	Analog	Analog	Analog	Analog	Digital	Digital
Framerate	10 000 fps	1000 fps	9 600 fps	1000 fps	1000 fps	1000 fps
Image processing	Mask-based image processing	Low-level image processing	Spatial convolutions	Spatial convolutions	Low-level image processing	Low to mid-level image processing

vision processing at focal plane. The first column corresponds to the chip described in this paper. On the second column, we can find the main characteristics of SCAMP-3, a general purpose processor array implementing a variety of low-level image processing tasks at a high framerate. The third column describes a multiple instruction multiple data image processing chip (MIMD IP Chip) performing spatial convolutions using kernels from 3 × 3 to 11 × 11. The fourth column presents a vision chip integrating an imager and an array of mixed-signal SIMD processing elements that can process gray scale images with cellular neural network universal machines (CNN-UMs) mode of operation. The fifth vision chip in the table is a programmable SIMD vision chip that can perform various early visual processing such as edge detection, smoothing or filtering by chaining processing elements and reconfiguring the hardware dynamically. Finally, the last one is a programmable artificial retina in which each pixel contains a tiny digital processing element capable of gray-level image processing from low- to mid-level vision (motion detection, segmentation, pattern recognition).

Compared to this state-of-the-art of high-speed CMOS image sensors, one easily sees that the chip reported in this paper leads to some major improvements. With a pixel size of 35 μm by 35 μm, the pixel pitch is almost 1.5 more compact than the smallest pixel described in [44, 45]. This contributes either to a better resolution of the sensor for a given chip area or a lower cost for a given resolution. At the same time, as compared with the other processing elements, our solution relies on a compact analog arithmetic unit based on a four-quadrant multiplier architecture using only 38 minimal size transistors. This leads to a fill factor of about 25%. Consequently, the active area of our photodiode is bigger compared to the major part of the other chips, providing a best sensibility to the sensor at high framerates of thousands of images per second.

From the performance point of view, all the chips on the table implement low-level image processing by programming or dynamically configuring the processing elements. Our solution is able to capture image, run user-defined 3 × 3 convolution masks, and provide the results on the sensor output bus in less than 200 microseconds, giving a framerate of 5000 images per second. These temporal performances are compatible and even slightly higher than the other sensors since each of them is only able to provide low-level image processing results at 1000 frames per second, except the MIMD IP Chip.

From the programability point of view, our chip can suffer from less versatility compared to other programmable sensors. Indeed, the implementation of complex low-level image processing requires successive reconfigurations of the internal masks coefficients of the processing elements. This task may be more difficult in comparison with algorithms easily written in a machine-level language for the programmable sensors, such as the SCAMP-3 sensor.

8. CONCLUSION AND PERSPECTIVES

An experimental pixel sensor implemented in a standard digital CMOS 0.35 μm process has been described in this paper. Each 35 μm × 35 μm pixel contains 38 transistors implementing a circuit with photocurrent integration, two analog memory, amplifier, and multiplexer ($[AM]^2$), and an analog arithmetic unit (A^2U).

Experimental chip results reveal that raw image acquisition at 10 000 frames per second can be easily achieved using the parallel A^2U implemented at pixel-level. With basic image processing, the maximal framerate slows down to about 5 000 fps. The potential for dynamic reconfiguration of the sensor was also demonstrated in the case of the Sobel operator.

The next step in our research will be the design of a similar circuit in a modern 130 nm CMOS technology. The main objective will be to design a pixel of less than 10 μm × 10 μm with a fill factor of 20%. A second possibility would be the design of a sensor with emergent technologies using amorphous silicon in which 3D pixels can be built. With a photosensitive layer placed just behind the optical part, the pixel fill-factor can reach 100% since the processing elements can be packed in the empty space below the photodiode.

Thus, with the increasing scaling of the transistors in such technologies, we could consider the implementation of more sophisticated image processing operators dedicated to face localization and recognition. Previous work of our team [51] has demonstrated the needs of dedicated CMOS sensors embedding low-level image processing such as features extraction. Moreover, actual works [52] focus on a recent face detector named convolutional face finder (CFF) [53]. CFF

is based on a multilayer convolutional neural architecture. The CFF consists of six successive neural layers. The first four layers extract characteristic features, and the last two perform the classification. Our objective would be to implement at pixel-level the first layers based on convolutions by different masks from 2×2 to 5×5.

In order to evaluate this future chip in some realistic conditions, we would like to design a CIF sensor (352×288 pixels), which leads to a 3.2 mm $\times$ 2.4 mm in a 130 nm technology. The exploitation of high FPS capability with this sensor could be achieved by two complementary ways. The first one is to integrate a dedicated input/output module, which is able to cope with a Giga pixel per second bandwidth. Such modules have been already designed in other high-speed CMOS sensors. As an example, we can cite the digital pixel sensor [24] with his 64 bit (8-pixel) wide bus operating at 167 MHz, able to output 1.33 GB/s. The second way is to build a large sensor by assembling 64×64 pixel modules with a dedicated output bus for each of them. For example, with a 384×256 pixel sensor, this solution only requires $6 \times 4 = 24$ dedicated outputs. In the same time, we will focus on the development of a fast analog to digital converter (ADC). The integration of this ADC on future chips will allow us to provide new and sophisticated vision systems on chip (ViSOC) dedicated to digital embedded image processing at thousands of frames per second.

REFERENCES

[1] E. R. Fossum, "Active pixel sensors: are CCDs dinosaurs?" in *Charge-Coupled Devices and Solid State Optical Sensors III*, vol. 1900 of *Proceedings of SPIE*, pp. 2–14, San Jose, Calif, USA, February 1993.

[2] E. R. Fossum, "CMOS image sensors: electronic camera-on-a-chip," *IEEE Transactions on Electron Devices*, vol. 44, no. 10, pp. 1689–1698, 1997.

[3] A. El Gamal, D. X. D. Yang, and B. A. Fowler, "Pixel-level processing: why, what, and how?" in *Sensors, Cameras, and Applications for Digital Photography*, vol. 3650 of *Proceedings of SPIE*, pp. 2–13, San Jose, Calif, USA, January 1999.

[4] P. Seitz, "Solid-state image sensing," in *Handbook of Computer Vision and Applications*, vol. 1, pp. 165–222, Academic Press, New York, NY, USA, 2000.

[5] D. Litwiller, "CCD vs. CMOS: facts and fiction," *Photonics Spectra*, vol. 35, no. 1, pp. 154–158, 2001.

[6] C. H. Aw and B. A. Wooley, "A 128×128-pixel standard-CMOS image sensor with electronic shutter," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 12, pp. 1922–1930, 1996.

[7] M. J. Loinaz, K. J. Singh, A. J. Blanksby, D. A. Inglis, K. Azadet, and B. D. Ackland, "A 200-mW, 3.3-V, CMOS color camera IC producing 352×288 24-b video at 30 frames/s," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 12, pp. 2092–2103, 1998.

[8] S. G. Smith, J. E. D. Hurwitz, M. J. Torrie, et al., "A single-chip 306×244-pixel CMOS NTSC video camera," in *Proceedings of the 45th IEEE International Solid-State Circuits Conference (ISSCC '98)*, pp. 170–171, San Francisco, Calif, USA, February 1998.

[9] O. Yadid-Pecht and A. Belenky, "In-pixel autoexposure CMOS APS," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 8, pp. 1425–1428, 2003.

[10] P. M. Acosta-Serafini, I. Masaki, and C. G. Sodini, "A 1/3″ VGA linear wide dynamic range CMOS image sensor implementing a predictive multiple sampling algorithm with overlapping integration intervals," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 9, pp. 1487–1496, 2004.

[11] L. J. Kozlowski, G. Rossi, L. Blanquart, et al., "Pixel noise suppression via SoC management of tapered reset in a 1920×1080 CMOS image sensor," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 12, pp. 2766–2776, 2005.

[12] M. Sakakibara, S. Kawahito, D. Handoko, et al., "A high-sensitivity CMOS image sensor with gain-adaptive column amplifiers," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 5, pp. 1147–1156, 2005.

[13] R. H. Nixon, S. E. Kemeny, C. O. Staller, and E. R. Fossum, "128×128 CMOS photodiode-type active pixel sensor with on-chip timing, control, and signal chain electronics," in *Charge-Coupled Devices and Solid State Optical Sensors V*, M. M. Blouke, Ed., vol. 2415 of *Proceedings of SPIE*, pp. 117–123, San Jose, Calif, USA, February 1995.

[14] S.-G. Wuu, H.-C. Chien, D.-N. Yaung, et al., "A high performance active pixel sensor with 0.18 μm CMOS color imager technology," in *Proceedings of the IEEE International Electron Devices Meeting (IEDM '01)*, pp. 555–558, Washington, DC, USA, December 2001.

[15] S. Decker, R. D. McGrath, K. Brehmer, and C. G. Sodini, "A 256×256 CMOS imaging array with wide dynamic range pixels and column-parallel digital output," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 12, pp. 2081–2091, 1998.

[16] K. Yoon, C. Kim, B. Lee, and D. Lee, "Single-chip CMOS image sensor for mobile applications," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 12, pp. 1839–1845, 2002.

[17] A. I. Krymski and Tu. Nianrong, "A 9-V/Lux-s 5000-frames/s 512×512 CMOS sensor," *IEEE Transactions on Electron Devices*, vol. 50, no. 1, pp. 136–143, 2003.

[18] G. L. Cembrano, A. Rodríguez-Vázquez, R. C. Galán, F. Jiménez-Garrido, S. Espejo, and R. Domínguez-Castro, "A 1000 FPS at 128×128 vision processor with 8-bit digitized I/O," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 7, pp. 1044–1055, 2004.

[19] L. Lindgren, J. Melander, R. Johansson, and B. Möller, "A multiresolution 100-GOPS 4-Gpixels/s programmable smart vision sensor for multisense imaging," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 6, pp. 1350–1359, 2005.

[20] Y. Sugiyama, M. Takumi, H. Toyoda, et al., "A high-speed CMOS image sensor with profile data acquiring function," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 12, pp. 2816–2823, 2005.

[21] D. A. Martin, H.-S. Lee, and I. Masaki, "A mixed-signal array processor with early vision applications," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 3, pp. 497–502, 1998.

[22] A. Krymski, D. Van Blerkom, A. Andersson, N. Bock, B. Mansoorian, and E. R. Fossum, "A high speed, 500 frames/s, 1024×1024 CMOS active pixel sensor," in *Proceedings of the Symposium on VLSI Circuits*, pp. 137–138, Kyoto, Japan, June 1999.

[23] N. Stevanovic, M. Hillebrand, B. J. Hosticka, and A. Teuner, "A CMOS image sensor for high-speed imaging," in *Proceedings of the 47th IEEE International Solid-State Circuits Conference (ISSCC '00)*, pp. 104–105, 449, San Francisco, Calif, USA, February 2000.

[24] S. Kleinfelder, S. Lim, X. Liu, and A. El Gamal, "A 10000 frames/s CMOS digital pixel sensor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 12, pp. 2049–2059, 2001.

[25] D. Handoko, S. Kawahito, Y. Takokoro, M. Kumahara, and A. Matsuzawa, "A CMOS image sensor for focal-plane low-power motion vector estimation," in *Proceedings of the Symposium on VLSI Circuits*, pp. 28–29, Honolulu, Hawaii, USA, June 2000.

[26] S. Lim and A. El Gamal, "Integration of image capture and processing: beyond single chip digital camera," in *Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II*, vol. 4306 of *Proceedings of SPIE*, pp. 219–226, San Jose, Calif, USA, January 2001.

[27] X. Liu and A. El Gamal, "Photocurrent estimation from multiple non-destructive samples in a CMOS image sensor," in *Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II*, vol. 4306 of *Proceedings of SPIE*, pp. 450–458, San Jose, Calif, USA, January 2001.

[28] D. X. D. Yang, A. El Gamal, B. Fowler, and H. Tian, "A 640×512 CMOS image sensor with ultrawide dynamic range floating-point pixel-level ADC," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 12, pp. 1821–1834, 1999.

[29] O. Yadid-Pecht and E. R. Fossum, "CMOS APS with autoscaling and customized wide dynamic range," in *Proceedings of the IEEE Workshop on Charge-Coupled Devices and Advanced Image Sensors*, vol. 3650, pp. 48–51, Karuizawa, Japan, June 1999.

[30] D. Stoppa, A. Simoni, L. Gonzo, M. Gottardi, and G.-F. Dalla Betta, "Novel CMOS image sensor with a 132-dB dynamic range," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 12, pp. 1846–1852, 2002.

[31] X. Liu and A. El Gamal, "Simultaneous image formation and motion blur restoration via multiple capture," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '01)*, vol. 3, pp. 1841–1844, Salt Lake, Utah, USA, May 2001.

[32] C.-Y. Wu and C.-T. Chiang, "A low-photocurrent CMOS retinal focal-plane sensor with a pseudo-BJT smoothing network and an adaptive current Schmitt trigger for scanner applications," *IEEE Sensors Journal*, vol. 4, no. 4, pp. 510–518, 2004.

[33] P. Dudek and P. J. Hicks, "An analogue SIMD focal-plane processor array," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '01)*, vol. 4, pp. 490–493, Sydney, Australia, May 2001.

[34] A. El Gamal and H. Eltoukhy, "CMOS image sensors," *IEEE Circuits and Devices Magazine*, vol. 21, no. 3, pp. 6–20, 2005.

[35] J. Lee and R. Hornsey, "CMOS photodiodes with substrate openings for higher conversion gain in active pixel sensors," in *Proceedings of the IEEE Workshop on CCDs and Advanced Image Sensors*, Crystal Bay, Nev, USA, June 2001.

[36] C.-Y. Wu, Y.-C. Shih, J.-F. Lan, C.-C. Hsieh, C.-C. Huang, and J.-H. Lu Jr., "Design, optimization, and performance analysis of new photodiode structures for CMOS active-pixel-sensor (APS) imager applications," *IEEE Sensors Journal*, vol. 4, no. 1, pp. 135–144, 2004.

[37] I. Shcherback, A. Belenky, and O. Yadid-Pecht, "Empirical dark current modeling for complementary metal oxide semiconductor active pixel sensor," *Optical Engineering*, vol. 41, no. 6, pp. 1216–1219, 2002.

[38] I. Shcherback and O. Yadid-Pecht, "Photoresponse analysis and pixel shape optimization for CMOS active pixel sensors," *IEEE Transactions on Electron Devices*, vol. 50, no. 1, pp. 12–18, 2003.

[39] J. Dubois, D. Ginhac, M. Paindavoine, and B. Heyrman, "A 10 000 fps CMOS sensor with massively parallel image processing," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 706–717, 2008.

[40] O. Yadid-Pecht, B. Pain, C. Staller, C. Clark, and E. R. Fossum, "CMOS active pixel sensor star tracker with regional electronic shutter," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 2, pp. 285–288, 1997.

[41] G. Chapinal, S. A. Bota, M. Moreno, J. Palacin, and A. Herms, "A 128 × 128 CMOS image sensor with analog memory for synchronous image capture," *IEEE Sensors Journal*, vol. 2, no. 2, pp. 120–127, 2002.

[42] C. R. Ryan, "Applications of a four-quadrant multiplier," *IEEE Journal of Solid-State Circuits*, vol. 5, no. 1, pp. 45–48, 1970.

[43] S. Liu and Y. Hwang, "CMOS Squarer and Four-Quadrant Multiplier," *IEEE Transactions on Circuits and Systems I*, vol. 42, no. 2, pp. 119–122, 1995.

[44] P. Dudek, "Implementation of SIMD vision chip with 128 × 128 array of analogue processing elements," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 6, pp. 5806–5809, Kobe, Japan, May 2005.

[45] P. Dudek and S. J. Carey, "General-purpose 128 × 128 SIMD processor array with integrated image sensor," *Electronics Letters*, vol. 42, no. 12, pp. 678–679, 2006.

[46] R. Etienne-Cummings, Z. K. Kalayjian, and D. Cai, "A programmable focal-plane MIMD image processor chip," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 1, pp. 64–73, 2001.

[47] A. Rodríguez-Vázquez, G. Liñán-Cembrano, L. Carranza, et al., "ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs," *IEEE Transactions on Circuits and Systems I*, vol. 51, no. 5, pp. 851–863, 2004.

[48] T. Komuro, I. Ishii, M. Ishikawa, and A. Yoshida, "A digital vision chip specialized for high-speed target tracking," *IEEE Transactions on Electron Devices*, vol. 50, no. 1, pp. 191–199, 2003.

[49] T. Komuro, S. Kagami, and M. Ishikawa, "A dynamically reconfigurable SIMD processor for a vision chip," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 1, pp. 265–268, 2004.

[50] F. Paillet, D. Mercier, and T. Bernard, "Second generation programmable artificial retina," in *Proceedings of the 12th Annual IEEE International ASIC/SOC Conference*, pp. 304–309, Washington, DC, USA, September 1999.

[51] F. Yang and M. Paindavoine, "Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1162–1175, 2003.

[52] N. Farrugia, F. Mamalet, S. Roux, F. Yang, and M. Paindavoine, "A parallel face detection system implemented on FPGA," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '07)*, pp. 3704–3707, New Orleans, La, USA, May 2007.

[53] C. Garcia and M. Delakis, "Convolutional face finder: a neural architecture for fast and robust face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1408–1423, 2004.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 938256, 9 pages
doi:10.1155/2008/938256

Research Article

Design of a Real-Time Face Detection Parallel Architecture Using High-Level Synthesis

Nicolas Farrugia,[1] Franck Mamalet,[1] Sébastien Roux,[1] Fan Yang,[2] and Michel Paindavoine[2]

[1] *MAchine to machine technologies Tangible Interactions expertiSe on devices Laboratory (MATIS), Orange Labs, 28 Chemin du Vieux Chène, 38243 Meylan, France*
[2] *Laboratory of Electronics Informatics Image (LE2i), Health-STIC Federative Research Institute (IFR100), Burgundy University-Engineer Science Center, 21078 Dijon, France*

Correspondence should be addressed to Franck Mamalet, franck.mamalet@antispamorange-ftgroup.com

Received 10 March 2008; Revised 20 June 2008; Accepted 12 November 2008

Recommended by Dragomir Milojevic

We describe a High-Level Synthesis implementation of a parallel architecture for face detection. The chosen face detection method is the well-known Convolutional Face Finder (CFF) algorithm, which consists of a pipeline of convolution operations. We rely on dataflow modelling of the algorithm and we use a high-level synthesis tool in order to specify the local dataflows of our Processing Element (PE), by describing in C language inter-PE communication, fine scheduling of the successive convolutions, and memory distribution and bandwidth. Using this approach, we explore several implementation alternatives in order to find a compromise between processing speed and area of the PE. We then build a parallel architecture composed of a PE ring and a FIFO memory, which constitutes a generic architecture capable of processing images of different sizes. A ring of 25 PEs running at 80 MHz is able to process 127 QVGA images per second or 35 VGA images per second.

1. INTRODUCTION

Face detection and analysis in images defining a parallel and video streams is an important research field and has many applications in security access control, image indexing, and person identification. New applications on power-constrained devices are foreseen, like video coding in mobile videoconference and intelligent user interfaces.

Many algorithms for face detection have been proposed in the past twenty years [1]. The chosen face detection method is the Convolutional Face Finder (CFF), introduced by Garcia and Delakis in [2]. It leads to the best performance on standard face databases. The CFF is an image-based neural network approach that allows robust detection in real world images of multiple semifrontal faces of variable size and appearance, rotated up to ±20 degrees in image plane and turned up to ±60 degrees. In [3], the authors have shown that the CFF algorithm can be implemented efficiently on embedded software platforms, while keeping a good detection rate and a low false alarm rate. Many optimisations were done, leading to significant gains in terms of processing speed and memory requirements, thus enabling face detection on a mobile phone with five QCIF (176×144) frames per second and only 220 KBytes of memory [3]. However, face detection is most often the first step of a face analysis process and will require a faster system.

There have been a few attempts at hardware systems for face detection [4, 5] but the authors report lower detection rates and higher false alarm rates than with the CFF [2], and frame rates up to 50 QVGA (320×240) frames per second. So far, no hardware implementation of the CFF algorithm has been reported.

We, therefore, aim to design the first fast and robust face detection optimised hardware by defining a parallel architecture for the CFF algorithm. In [6], we have described an optimised algorithm architecture matching methodology, consisting of dataflow modelling of the algorithm, parallelism extraction, and complexity analysis. Using this information, we have performed a coarse-grain design space exploration, which enabled us to specify an efficient parallel architecture, consisting of a ring of PE where each PE processes the whole face detection algorithm on a small block

of data and communicates a small amount of data to one of its neighbours. In this paper, we present the implementation of such a PE capable of handling the successive convolutions of the CFF algorithm.

In order to ease and accelerate the implementation of complex algorithms, new methodologies have been studied and developed in the past twenty years. High-Level Synthesis (HLS) methods produce designs by specifying them using a high-level language like C. Many approaches and tools for HLS are now available [7–11] and are starting to be mature enough to be used for complex designs. In this paper, we propose a guided HLS approach using the UGH tool [11]. Using this approach, we are able to explore and implement several PE alternatives.

The remainder of the paper is organised as follows. In Section 2, we present the CFF algorithm and previous works which have introduced a dataflow model of the algorithm and a theoretical architecture of PEs efficiently implementing this algorithm. In Section 3, we present our high-level synthesis based exploration which enables us to evaluate several implementation alternatives. We give processing times and synthesis results for four PE alternatives and select one optimal PE. In Section 4, we present a parallel architecture composed of the previous PE, which enables us to process images of different sizes. We, finally, give the performances of this architecture, and we compare our face detection systems with those in the literature. We will then conclude this paper and give our perspectives in Section 5.

2. THE CFF ALGORITHM-OVERVIEW AND PREVIOUS WORKS

2.1. Overview of the CFF algorithm

The Convolutional Face Finder was presented in [2] and relies on convolutional neural networks introduced by LeCun and Bengio [12].

In this paper, we will only consider the core of the face localization process as depicted in Figure 1. The convolutional neural network used to implement the face detector has been previously optimised in [3], and consists of a set of two different kinds of layers.

(i) CSi layers are called convolutional layers and contain a certain number of planes. Each element in a plane receives input from a small neighbourhood (biological local receptive field) in the planes of the previous layer. Each plane can be considered as a feature map that has a fixed feature detector, which corresponds to a pure convolution with a mask applied over the planes in the previous layer. A bias is added to the results of each convolutional mask. Multiple planes are used in each layer so that multiple features can be detected. Once a feature has been detected, its exact location is less important. Hence, each convolutional CSi layer is typically done with horizontal and vertical steps of two pixels which correspond to perform local averaging and subsampling operations. Then, the results are passed through a hyperbolic tangent function, used as an activation function. As a result, the CSi layer also performs a reduction by two of the dimensionality of the input.

(ii) Ni layers are called classification layers and are applied after feature extraction and input dimensionality reduction of CSi layers. These layers correspond to a multilayer perceptron.

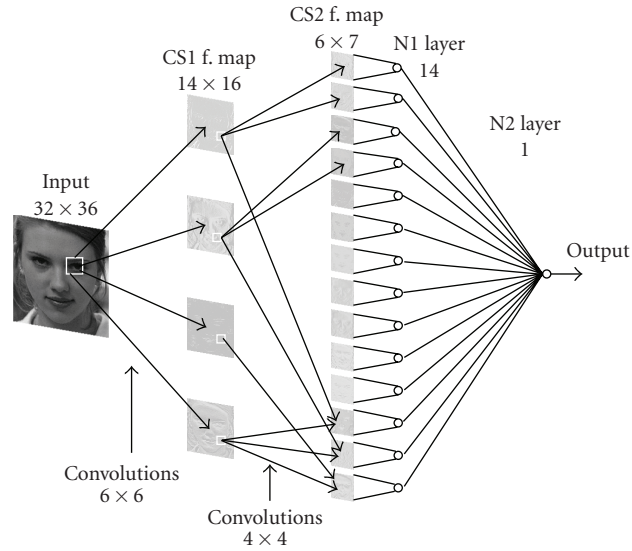


Figure 1: Convolutional Face Finder pipeline.

All parameters (convolution kernels, biases, neuron weights) have been learnt automatically using a modified version of the backpropagation algorithm with momentum [2]. In the remainder of this paper, we will only consider the face localization process when training has been completed.

The detail of CFF layers is given in Figure 1.

(i) CS1 layer performs convolutions with masks of dimension 6×6. It contains four feature maps and therefore performs four convolutions on the input image.

(ii) CS2 layer performs 4×4 convolutions and has fourteen feature maps. Each of the four-subsampled feature maps of CS1 is convolved by two different 4×4 masks, providing the first eight feature maps of CS2. The other six feature maps of CS2 are obtained by fusing the results of two 4×4 convolutions on each possible pair of CS1 feature maps.

(iii) N1 layer contains 14 sigmoid neurons. Each neuron is fully connected to exactly one feature map of the CS2 layer whose size is 6×7.

(iv) N2 layer consists of a unique neuron fully connected to all the neurons of the N1 layer. The output of this neuron is used to classify the input image as face (positive answer) or non-face (negative answer).

2.2. Previous works—dataflow model description

In [6], we have established a design space exploration methodology based on dataflow modelling and parallelism

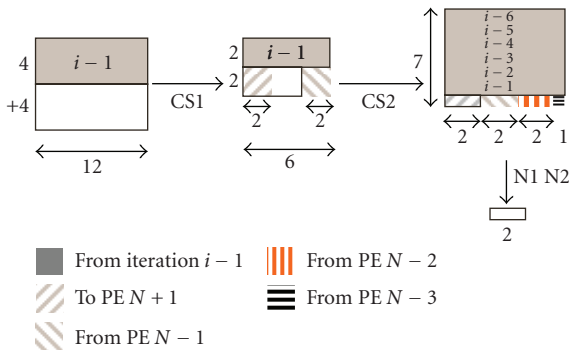


FIGURE 2: Dataflow model for one PE of the CFF algorithm for producing two data of N1 layer.

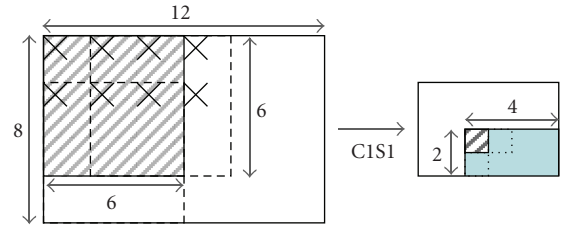


FIGURE 3: Local dataflow of CS1 and kernel coefficient reusing.

extraction of the CFF algorithm. This exploration has shown that the maximum parallelisation efficiency is obtained by massive data parallelism exploitation with a ring of PE.

In Figure 2, we detail the corresponding dataflow model for a given path in the CFF algorithm, which involves four successive PEs (N, $N-1$, $N-2$, and $N-3$) and seven successive temporal iterations for the calculation of two N1 layer output data.

Each PE processes the whole algorithm on a block of 8 lines of 12 pixels and transfers a small amount of data (overlapping parts) to one of its neighbours. The several slashed parts in Figure 2 represent the data to be transferred between successive PEs.

To start with, CS1 6×6 convolutions are applied on a 12×8 block of data with horizontal and vertical steps of two pixels, thus producing 2×4 output data. Then, PE N has to send a 2×2 output data to PE $N+1$ and receives a 2×2 block from PE $N-1$. Using data from previous iteration, CS2 is applied on a 4×6 data block which produces two data to be used as input of the N1 layer. These data are sent to the nearest neighbour and each PE gathers five data from three other PEs. With the six previous iterations, two N1 output data and two face detections can be computed.

3. IMPLEMENTATION OF A PROCESSING ELEMENT

In this section, we detail a complete implementation of a PE able to process the whole CFF algorithm. The design of this PE is guided by the results of the DSE/AAA methodology presented in [6]. The coarse-grain PE model previously used in this exploration does not detail neither the internal dataflow needed to process the convolutions nor the way the coefficients for the convolutions are handled.

In this paragraph, we will describe the local dataflow necessary to implement each CFF layer on the same PE. We will see that the requirements of the CFF algorithm in terms of dataflow management make it difficult to do an optimised design in manual VHDL coding.

We will then present a high-level synthesis based implementation which enables us to quickly evaluate several implementations of this complex dataflow. We then give simulation and synthesis results of this implementation.

3.1. Convolutions and local dataflow

In the design of a PE, we need to consider the local dataflow of the computation in terms of input data and also in terms of convolution kernel coefficients. Each PE processes CS1, CS2, N1, and N2 layers on an input block of 12×8 pixels, as described in Figure 2. To illustrate the complexity of the local dataflow, we will focus on the CS1 layer whose details are given in Figure 3.

The slashed part on the left side (resp., on the right side) represents the first 6×6 input window (resp., the result of CS1 on this input window). Dotted borders represent the next horizontal and vertical input windows (resp., the next results) considering horizontal and vertical steps of two input pixels. There are eight input windows to compute, depicted in Figure 3 by crosses on their top-left corner. These windows comprise many overlapping data: 17% of the input image block is involved in the computation of six different windows.

Thus, the first coefficient of each kernel is applied to these data, and each coefficient is used eight times. In addition, each window is processed with four different CS1 kernels. This complex local dataflow can also be represented sequentially as a 5-deep "for" loop nest written in C language, as depicted in Figure 4. The inner operation in the loop nest is a Multiplication-Accumulation (MAC) between a kernel coefficient, an input data, and a previous accumulation. Furthermore, as presented in Section 2.1, the CS2 and N1 layers also have very complex local dataflow.

Our main goal is to design a PE which processes the whole algorithm on a window of size 12×8 and communicates with its neighbours as described in the dataflow definition in the Section 2.1. Concerning the local dataflow of the algorithm, there are three main issues to cope with.

(1) PE memory organisation and bandwidth: each feature map has to be stored and read back in a parallel way to enable local task parallelism (e.g., two CS2 feature maps computed in parallel).

(2) Fine scheduling and address generation: a choice of scheduling has to be made to efficiently exploit data locality and convolution regularity, and address sequences have to be computed for each memory.

(3) Inter-PE communication and temporal iterations: overlapping parts between PEs have to be transferred and data between successive iterations have to be stored and read back.

Each issue can be addressed with several implementation solutions. For example, issues one and three involve a choice of the type of memory (FIFO or shared RAM, dual port RAM, register banks, etc.) and its size, as well as the definition of the links between PEs. In addition, there are many fine scheduling choices to exploit different kinds of parallelism, and for each scheduling, complex address generation has to be computed for each memory.

The complexity and interdependency of these issues make it hard to do a manual RTL description for each possible implementation solution. Even if theoretical study using polyhedral models and loop transformation techniques [13] could be done in order to define an optimised fine scheduling of one loop nest, such a study on the whole algorithm is beyond the scope of this paper.

In the next paragraph, we present a High-Level Synthesis (HLS) flow which enables us to explore several fine schedulings, memory organisations and bandwidths, and inter-PE communications, in high-level C language, and to automatically generate a synthesizable PE including datapath, control and address generation. Our objective is to quickly obtain an efficient and functional PE prototype which will be used to implement the parallel dataflow described in Section 2.2.

```
for (i4 = 0; i4<4; i4 ++)          //filter number
  for (i0 = 0; i0<2; i0 ++)        // output window row number
    for (i1 = 0; i1<4; i1 ++)      // output window column number
      for (i2 = 0; i2<6; i2 ++)    // coefficient row number
        for (i3 = 0; i3<6; i3 ++)  // coefficient column number

{result[i1][i4][i0]+ = data[2 × i0 + i2][2 × i1 + i3]× coeff[i4][i2][i3] ;}
```

Figure 4: Sequential description of CS1: Loop nest.

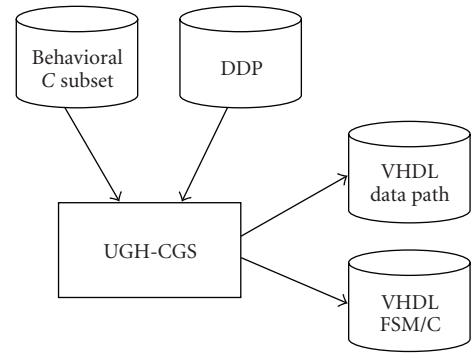


Figure 5: UGH coarse-grain scheduling design flow.

3.2. PE design using a high-level synthesis tool

(1) *High-level synthesis*

We have chosen to use an HLS approach in order to accelerate exploration, implementation, and validation of the PE. Many approaches for HLS have been proposed. There are commercial tools [7–9] which provide complete frameworks for hardware synthesis. Such tools are designed to accelerate and ease the development in a hardware project. The formalisms, formats, or platforms used are thus proprietary and often very specific and locked. Therefore, these tools may not be well suited for research purposes. On the contrary, some academic tools are open-source [10, 11], and provide the user with more flexibility and visibility on the tool behaviour. In Section 3.3, we present our implementation using UGH (User-Guided HLS) tool, an open-source HLS tool integrated in a larger framework for SoC design called dysident, and was developed by LIP6 laboratory. This tool is adapted to our study for two main reasons. First, it allows an architecture to be targeted by specifying the available resources before synthesis. This allows us to rely on the previous results of our manual PE implementation [6]. Second, UGH allows us to describe directly in C language all data input/output transfers, address calculation and dataflow management (e.g., ping pong strategies, modulo addressing, multiple data sources, etc.), which are required for our application [11].

(2) *UGH design flow*

A complete description of the UGH design flow can be found in [11]. In this paper, we will focus on the first part of UGH design flow (Figure 5) which performs a Coarse-Grain Scheduling (CGS) with a behavioural C description of the algorithm and a Draft Data Path (DDP). The C description uses a subset of the C language whose main limitation is the forbidden use of pointers. Special data types are introduced to handle variable bit sizes (e.g., unsigned integer coded on 4 bits). The DDP describes the available resources for the target design (e.g., ALUs, RAMs, multipliers, bitwise logic, etc.) and UGH-CGS binds and schedules the algorithm on the available resources, and generates a VHDL Finite State Machine (FSM) and a VHDL structural datapath which are both synthesisable.

Two particular features of UGH have been much used in our design.

(i) *Automatic parallelisation, datapath, and address generation*: UGH-CGS is able to detect parallelism at the instruction level and schedules it on the available resources which are specified in the DDP file. UGH-CGS is then able to automatically generate the corresponding datapaths and all necessary links (multiplexers and corresponding connections) between the resources, and also all control signals such as memory addresses

(ii) *Flexible communication model*: inputs and outputs can be specified in a direct and flexible way, by simply writing specific "ugh_read" and "ugh_write" operations at any time of the source C code, and by defining the corresponding input or output port in the DDP.

Given that the C description exhibits some parallelism at the instruction level, the first feature makes it possible to quickly test and compare several implementation alternatives

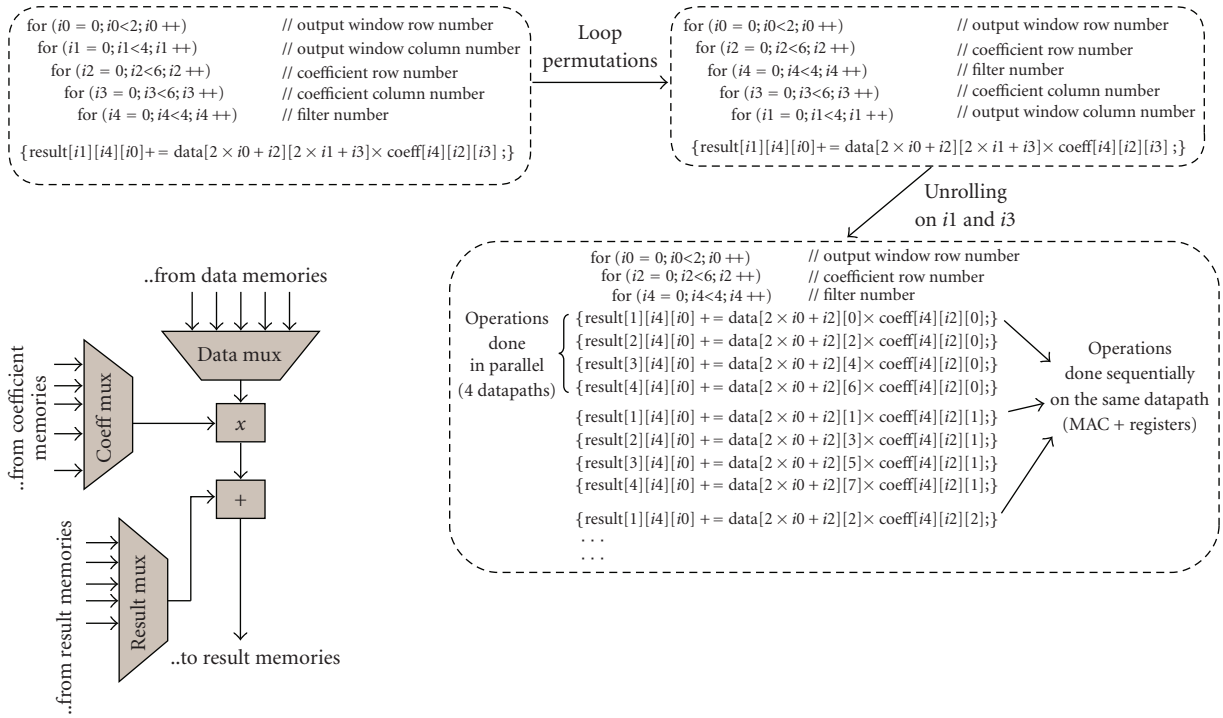


Figure 6: Example of loop transformations done on CS1, interpretation, and an associated datapath.

(e.g., degree of parallelism). The second feature enables us to describe our PE with the same dataflow as in our previous dataflow modelling, by specifying the data acquisition and the successive transfer between PEs as described earlier.

(3) *Data paths and low level parallelisation*

As explained in the previous paragraph, in order to enable UGH-CGS to parallelise computation on the available resources (DDP), a manual loop transformation and unrolling has to be done. For instance, in Figure 6 we present a loop nest permutation and unrolling that enables the exhibition of an inner loop with four independent MACs using the same kernel coefficient. Then, specifying in the DDP file that four multipliers and four adders are available, UGH-CGS is able to automatically generate four appropriate datapaths and all necessary control signals. An example of such a datapath is shown at the bottom-left of Figure 6.

Such transformations have been done for each CFF layer. Table 1 details for each CFF layer an example of loop nest permutation enabling UGH to parallelise the algorithm on four or eight datapaths.

(4) *Memory issues*

UGH memory models are only single or dual port memory. In order to be able to parallelise the computation on several datapaths, multiple memory banks have to be introduced, thus increasing the memory bandwidth (e.g., to read four distinct columns of input data for CS1, we use four memories). This can be easily managed with UGH by dispatching data on different C arrays. UGH generates one memory bank per C array and handles address generation for each memory. In this way, we control the memory access parallelism, by specifying exactly the number and size of the memory needed for our system.

Furthermore, address calculation has to be performed carefully in order to handle seven successive iterations, which can be simply done in C by computing the necessary addresses using increments and modulos (e.g., in CS1, four new lines are acquired and four previous lines are reused in the current iteration).

3.3. PE design case study

(1) *Algorithm description*

Using techniques described above, we describe the full CFF algorithm which consists of CS1, CS2, N1, and N2 layers. Each layer is described using a partially unrolled loop nest with MAC operations. Using an appropriate DDP, UGH successfully generates the FSM and the datapaths implementing the entire algorithm. We also define the appropriate I/O ports in the PE in order to handle input data acquisition, coefficient loading, CS1 and CS2 data transfers, and face detection results output.

(2) *Design exploration and performance analysis*

We use the flexibility offered by UGH to explore several architecture alternatives for the PE. This can be done by

Table 1: Low-level parallelisation of the CFF algorithm.

Layer (convolution size)	Number of windows (data parallelism)	Number of filters (task parallelism)	Total complexity (number of MAC)	Parallelisation scheme on 4 datapaths	Parallelisation scheme on 8 datapaths
CS1 (6×6)	8	4	1152	4 windows	4 windows and 2 filters
CS2 (4×4)	2	20	664	2 windows and 2 filters	2 windows and 4 filters
N1 (6×7)	2	14	1176	2 windows and 2 filters	2 windows and 4 filters
N2 (1×1)	2	14	28	2 windows and 2 filters	2 windows and 4 filters

Table 2: Cycle-time details for each PE version.

Layer	PE_A	PE_B	PE_C	PE_D
	Number of cycles/efficiency			
CS1	1344/0.86	740/0.78	437/0.66	261/0.55
CS2	738/0.87	412/0.81	251/0.66	252/0.33
N1 + N2	1346/0.89	749/0.80	457/0.66	466/0.33

modifying the available resources given in the DDP, and by doing some transformations in the C source code. In order to compute the convolutions, we describe datapaths in the DDP, with MAC blocks performing multiply accumulate operations. Each datapath contains a MAC block and one to four register files. Each register file is composed of four 16 bits registers. We have explored four datapath alternatives to build the following PE versions.

(i) PE_A: 1 datapath only with 1 MAC block, 4 register files.

(ii) PE_B: 2 datapaths, each with 1 MAC block, 2 register files.

(iii) PE_C: 4 datapaths, each with 1 MAC block, 1 register file.

(iv) PE_D: 8 datapaths, each with 1 MAC block, 1 register file.

Figure 7 depicts the architecture of PE_C. Dotted lines and boxes represent everything that is automatically handled by UGH: links, FSM, addresses, and so on. The remaining grey parts are provided in the DDP. TH blocks represent activation functions which are applied after each convolution.

We then obtain 4 PE versions containing 1, 2, 4, and 8 MAC blocks, respectively. We have validated the VHDL codes generated by UGH using the same test sets as in the behavioural initial C description of the PE and in a VHDL test bench. The processing cycle times for each version are summed up in Table 2, where each layer of the CFF algorithm is detailed. We also point out in this table the efficiency of parallelisation, by computing the theoretical minimum number of cycles T_{th} achievable for each CFF layer (the number of MAC operations given in Table 1 divided by the number of MAC blocks); the efficiency is then equal to $\text{Eff} = T_{\text{th}}/T_{\text{par}}$. We can see in Table 2 that the average efficiency of PE_A is about 0.9. As PE_A contains only 1 MAC, it allows no possible parallelisation. The efficiency of PE_A can be interpreted as follows. 90% of the time is used for computing the MACs and the remaining 10% corresponds to controlling overhead due to the handling of loops and addresses. PE_B and PE_C have average efficiencies of 0.8 and 0.7, respectively, which corresponds to average acceleration factors of 1.6 and 2.8. This efficiency remains high considering the complexity of the algorithm. However, we can point out that for PE_D, there is a significant loss in efficiency which can be interpreted as follows. The CS1 layer is successfully parallelised on 8 MACs but for CS2 and N1 N2, memory bandwidth is not large enough to enable an efficient use of 8 MACs simultaneously. For instance, an efficient parallelisation of N1N2 on 8 MACs would require a simultaneous access to four distinct coefficients and eight distinct data. This could be addressed by using data and coefficient buffer memories to provide more local parallelism and reusing of data.

Synthesis results for each version are given in Table 3 for a Virtex-4 SX35 FPGA. No major modification has been made on the VHDL code generated by UGH. Implementation memory banks are balanced between flip flops, distributed RAM (RAM implemented in logic blocks [14]), and block RAM (embedded static RAMs), depending on the memory bank size and the number of ports needed. DSP48 blocks are used to implement MAC blocks in the datapaths. The maximum frequency of each version of the PE is 80 MHz and is mainly due to unpipelined utilisation of DSP48 blocks and the large number of memories which imply much multiplexing logic. From Table 3, we see that synthesis results are close from one version to another. This is due to the fact that the size of the PEs is mostly because of the size of the control unit (finite state machine) and of the multiplexers, rather than the datapaths which are only MAC blocks and registers. Hence, we can conclude that PE_C is a good compromise as it enables a high acceleration factor (2.7) and requires about the same resources as PE_A and PE_B. Therefore, we choose PE_C as the best implementation for the PE (in the rest of the paper, we will refer to PE instead of PE_C) because it achieves a good tradeoff between size and efficiency.

Table 3: Synthesis results for each PE version on Virtex-4 SX35 FPGA.

	PE_A	PE_B	PE_C	PE_D
	Device occupancy/capacity			
Number of slices	2258/15360	2259/15360	2466/15360	2866/15360
Number of flip flops	363/30720	388/30720	345/30720	457/30720
Number of block rams	15/192	19/192	19/192	19/192
Number of DSP48	5/192	6/192	8/192	12/192
Maximum frequency	80 MHz	80 MHz	80 MHz	80 MHz

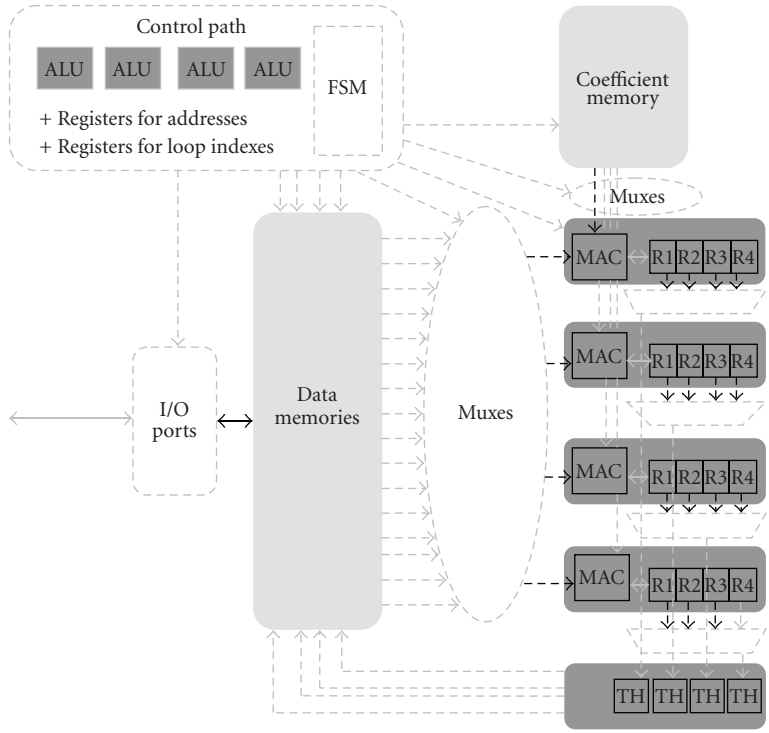


Figure 7: Architecture of a PE with 4 datapaths (PE_C).

4. MULTI-PE PARALLEL SYSTEM

In this section, we present a parallel architecture based on the previously designed PE. In Section 4.1, we describe this architecture and its application for processing a whole image of any size. In Section 4.2, we give its performances, in terms of frame processed per second, for different image sizes, and finally, we compare these performances to other face detection systems.

4.1. Generic parallel architecture

In previous work [6], we have shown that a good tradeoff between efficiency and number N_{PE} of PE is obtained when the input image is divided into $P = N_{PE}$ blocks of 8 rows of 12 pixels. Each block is processed by one PE, and each PE is connected to two other PEs, thus building a ring architecture. Considering data overlapping, this allows the processing of an image of width $12 + (N_{PE} - 1) * 8$ and of any height greater or equal to 36 (the minimum number of rows necessary to compute a face detection). We rely on this result to establish a generic architecture using a ring of PEs and a FIFO memory (Figure 8).

We divide the input image into vertical strips of width $12 + (N_{PE} - 1) * 8$ and process each strip by dividing it into blocks of 12×8 as described above. The FIFO is connected to the first and the last PEs of the ring and is used to provide the overlapping data from the previous strip and to write the overlapping data for the next one (grey blocks in the top of Figure 8). We have successfully simulated a 4 PE ring with FIFO. Synthesis estimates show that a ring of 25 PE fits in a Virtex-5 LX 330 device, the largest FPGA available.

4.2. Performances

In this architecture, each PE works synchronously. At each vertical iteration, four new input lines are loaded in the PEs and if we consider that this load is done in pipeline with the

Table 4: Comparison of hardware face detection implementation.

Implementation	Frame size	Frame rate	Clock freq. (MHz)
CFF embedded software [3]	176 × 144	10	624
Kianzad et al. [15]	320 × 240	12	125
McCready [16]	320 × 240	30	12,5
Theocharides et al. [4]	320 × 240	52	500
Nguyen et al. [5]	320 × 240	42	12,5
Hori et al. [17]	320 × 240	30	100
CFF 4 PE ring	320 × 240	**30**	**80**
CFF 25 PE ring	320 × 240	**127**	**80**
CFF 25 PE ring	640 × 480	**35**	**80**

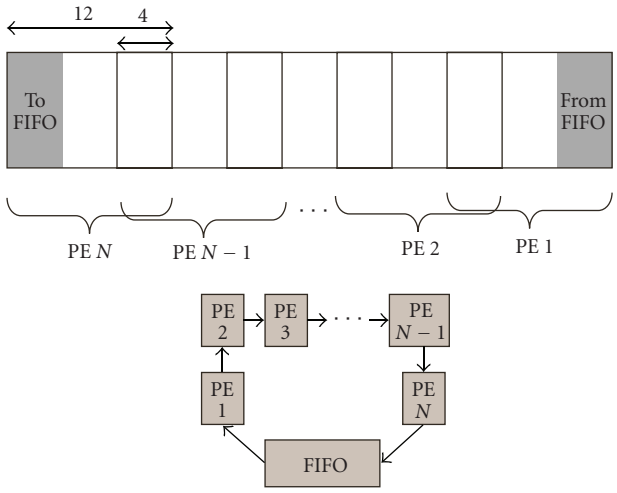


Figure 8: Generic ring and a FIFO architecture.

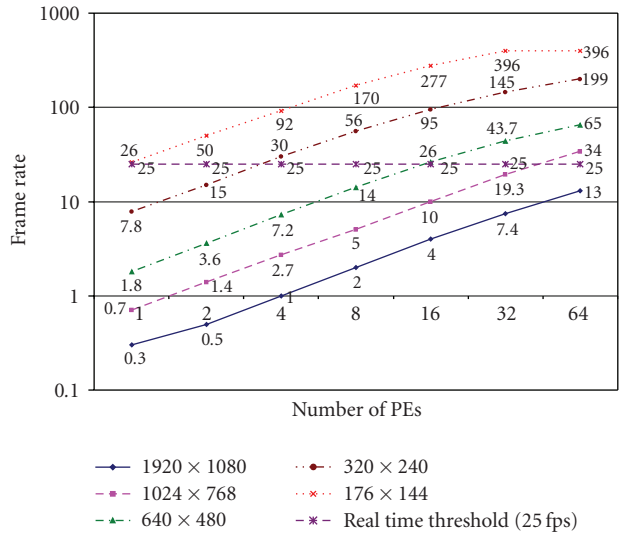


Figure 9: Performances of PE ring architecture at 80 MHz.

algorithm computation, we can give the processing time of the algorithm on a complete image of size width L and height M using the following formulae.

(i) T_{CFF}: time to process the CFF algorithm on one 8×12 block.

(ii) Number of vertical iterations: $N_{\text{it}} = (M - 4)/4$.

(iii) Number of vertical strips per PE: $N_{\text{str}} = (L - 4)/(8 \times N_{\text{PE}})$.

(iv) Processing time of a strip: $T_{\text{str}} = N_{\text{it}} * T_{\text{CFF}}$.

(v) Total processing time: $T_{\text{tot}} = N_{\text{str}} * T_{\text{str}}$.

In order to detect faces of different sizes, an image pyramid is built to apply the CFF algorithm to each image (see [3]). This pyramid is constituted of a set of subimages obtained from the initial image by applying a reduction factor of 1.2 to each dimension. We can obtain an estimation of the total processing time of the entire pyramid by adding together the processing times of each subimage.

In Figure 9, we sum up the performances (in frames per second) of this parallel architecture on several image sizes and for 1 to 64 PEs (to improve image readability both axes have logarithmic scales). We also represent the real-time threshold of 25 fps on this figure. An architecture consisting of four PEs fits in a Virtex-4 SX35 FPGA and performs real-time face detection at 30 QVGA images per second. When considering a larger system with 25 PEs, we can process 127 QVGA images per second, which leaves enough time to consider other face analysis tasks in real-time following face detection.

Table 4 presents a comparison between the main hardware implementations of face detection found in literature. All these implementations are done on FPGA hardware and therefore run at relatively low clock frequencies (except [4] which is an ASIC implementation). We can see that our system compares well with the others in terms of frame rate, depending on the number of PEs implemented. The main drawback of the other implementations is usually a significant loss in overall detection rates or even a lack of detection performance measure. most works do not give results of detection accuracy of their hardware implementation. Contrary to this, our system has the same detection performances as the original CFF software implementation [3], therefore it is very robust in terms of detection accuracy and has a very low false alarm rate [2].

Area comparison is not straightforward since the FPGA targets of the considered implementations are different. In [16], the authors report an area of 89856 Logic Elements and approximately 442 Kbits of memory on a system which contains eighteen 10k100 Altera FPGAs. In [5], the authors report an area of 15050 Logic Elements and 268 Kb of memory on a Altera Stratix FPGA. Our 4-PE ring on Virtex-4 uses approximately 8802 Xilinx Slices and about 1272 Kb of embedded memory. We cannot make a direct comparison between these implementations but a rough estimate can be given by considering that one Altera Logic Element is the equivalent of one to two Xilinx Virtex-4 Slices. Hence, our system has a slightly higher hardware cost in terms of logic and uses much more embedded memory. This is due to the large number of coefficients and temporary results which have to be stored in the FPGA. However, as said earlier, our system is capable of a very robust face detection with complex backgrounds.

5. CONCLUSIONS—PERSPECTIVES

We have implemented a parallel architecture for face detection composed of Processing Elements based on the CFF algorithm. Using a high-level synthesis approach, we were able to explore several PE architecture alternatives. We selected a PE with four datapaths exploiting efficiently local parallelism of the algorithm. This PE has been successfully simulated and synthesised on a Virtex-4 SX35 FPGA, and occupies approximately 16% of the device, and is capable of running at a maximum frequency of 80 MHz. We have then presented a ring of PE with a FIFO memory, which constitutes a generic parallel and scalable architecture able to process images of variable sizes. Such an architecture with four PEs can process up to 30 QVGA images per second. An architecture with 25 PE achieves real-time face detection of VGA images.

We are currently working on optimising this PE in order to be able to implement more PE in our targeted FPGAs, and to increase its maximum clock frequency. In future work, we will investigate the generalisation of such modelling and architectures for other algorithms based on convolutional neural networks. We will also investigate the opportunity of using source-to-source transformations in order to enhance the user-guided approach proposed by UGH, by providing a semiautomated tool which transforms the C code before inputting UGH; such techniques can be used to ease memory partitioning and loop unrolling.

ACKNOWLEDGMENTS

The authors would like to thank Clément Quinson for his help in starting up with UGH, as well as Frédéric Pétrot and Ivan Augé of the UGH team for their help with specific points on the UGH tool, and their encouragement on this work.

REFERENCES

[1] M.-H. Yang, D. J. Kriegman, and N. Ahuja, "Detecting faces in images: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[2] C. Garcia and M. Delakis, "Convolutional Face Finder: a neural architecture for fast and robust face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 11, pp. 1408–1423, 2004.

[3] S. Roux, F. Mamalet, and C. Garcia, "Embedded Convolutional Face Finder," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 285–288, Toronto, Canada, July 2006.

[4] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, "Embedded hardware face detection," in *Proceedings of the 17th IEEE International Conference on VLSI Design (VLSID '04)*, pp. 133–138, Mumbai, India, January 2004.

[5] D. Nguyen, D. Halupka, P. Aarabi, and A. Sheikholeslami, "Real-time face detection and lip feature extraction using field-programmable gate arrays," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 4, pp. 902–912, 2006.

[6] N. Farrugia, F. Mamalet, S. Roux, F. Yang, and M. Paindavoine, "A parallel face detection system implemented on FPGA," in *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '07)*, pp. 3704–3707, New Orleans, La, USA, May 2007.

[7] V. Kathail, S. Aditya, R. Schreiber, B. Ramakrishna Rau, D. C. Cronquist, and M. Sivaraman, "PICO: automatically designing custom computers," *Computer*, vol. 35, no. 9, pp. 39–47, 2002.

[8] S. McCloud, "Using a catapult c-based flow to speed implementation and increase flexibility," Tech. Rep., Mentor Graphics, Wilsonville, Ore, USA, 2003.

[9] "Dk design suite datasheet," Agility Design System Inc., http://agilityds.com/literature/dk_datasheet_01000_hq_screen.pdf.

[10] P. Coussy, G. Corre, P. Bomel, E. Senn, and E. Martin, "High-level synthesis under I/O timing and memory constraints," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '05)*, vol. 1, pp. 680–683, Kobe, Japan, May 2005.

[11] I. Augé, F. Pétrot, F. Donnet, and P. Gomez, "Platform-based design from parallel C specifications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 24, no. 12, pp. 1811–1826, 2005.

[12] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time-series," in *The Handbook of Brain Theory and Neural Networks*, pp. 255–258, MIT Press, Cambridge, Mass, USA, 1998.

[13] S. Girbal, N. Vasilache, C. Bastoul, et al., "Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies," *International Journal of Parallel Programming*, vol. 34, no. 3, pp. 261–317, 2006.

[14] "Virtex-4 user guide, chap. 5: Configurable logic blocks (clbs)," June 2008, http://www.xilinx.com/.

[15] V. Kianzad, S. Saha, J. Schlessman, et al., "An architectural level design methodology for embedded face detection," in *Proceedings of the 3rd International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS '05)*, pp. 136–141, Jersey City, NJ, USA, September 2005.

[16] R. McCready, "Real-time face detection on a configurable hardware system," in *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications (FPL '00)*, pp. 157–162, Villach, Austria, August 2000.

[17] Y. Hori, K. Shimizu, Y. Nakamura, and T. Kuroda, "A real-time multi face detection technique using positive-negative lines-of-face template," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR '04)*, vol. 1, pp. 765–768, Cambridge, UK, August 2004.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 597872, 13 pages
doi:10.1155/2008/597872

Research Article

Smart Camera Based on Embedded HW/SW Coprocessor

Romuald Mosqueron,[1] Julien Dubois,[2] Marco Mattavelli,[1] and David Mauvilet[1]

[1] *GR-LSM, Faculté STI, Ecole Polytechnique Fédérale de Lausanne (EPFL), CH 1015 Lausanne, Switzerland*
[2] *Laboratoire LE2I, Faculté des Sciences Mirande, Université de Bourgogne, 21000 Dijon, France*

Correspondence should be addressed to Romuald Mosqueron, romuald.mosqueron@epfl.ch

Received 1 March 2008; Revised 29 July 2008; Accepted 8 September 2008

Recommended by Guy Gogniat

This paper describes an image acquisition and a processing system based on a new coprocessor architecture designed for CMOS sensor imaging. The system exploits the full potential CMOS selective access imaging technology because the coprocessor unit is integrated into the image acquisition loop. The acquisition and coprocessing architecture are compatible with the majority of CMOS sensors. It enables the dynamic selection of a wide variety of acquisition modes as well as the reconfiguration and implementation of high-performance image preprocessing algorithms (calibration, filtering, denoising, binarization, pattern recognition). Furthermore, the processing and data transfer, from the CMOS sensor to the processor, can be operated simultaneously to increase achievable performances. The coprocessor architecture has been designed so as to obtain a unit that can be configured on the fly, in terms of type and number of chained processing stages (up to 8 successive predefined preprocessing stages), during the image acquisition process that can be defined by the user according to each specific application requirement. Examples of acquisition and processing performances are reported and compared to classical image acquisition systems based on standard modular PC platforms. The experimental results show a considerable increase of the achievable performances.

1. INTRODUCTION

Nowadays, *smart* cameras are more and more applied for their specific performances and their processing capabilities in different application fields. We can distinguish three typical classes of *smart* cameras.

(i) *Artificial retinas:* in which dedicated processing is directly integrated aside the pixel. The processing capabilities are usually fixed or limited to a few simple and local functions [1–3].

(ii) *Standard cameras directly connected to a computer via a standard interfaces:* all the processing is performed into the computer CPU [4, 5].

(iii) *Cameras including embedded processing units:* the processing is performed into the camera and only the processing results or some image features are transferred outside the camera [6, 7].

For the class of the artificial retinas cameras, the image processing capabilities, or better the pixel imaging capabilities, are usually fixed locally to a small pixel neighborhood and remain very limited in scope. No application specific processing can be added at the image acquisition stage. However, such kind of sensors can achieve very high frequency acquisition rates that are necessary for some class of applications. In the case of a standard camera interfaced with a computer, the data transfer between the camera and the computer is limited by the connection interface. When dealing with applications requiring high frame-rate or very high resolution cameras, usually the problem is the amount of data that needs to be transferred to the CPU. This may largely exceed the available standard interface bandwidth. For such class of applications, different camera architectures that include embedded processing units have been developed.

This paper proposes a novel smart camera architecture based on a specific coprocessor designed for on-line industrial process control. This paper describes a coprocessor unit design providing an interface for the full control of the sensor acquisition process driven from the main application CPU. This key feature enables the acquisition to be controlled on the fly in function of the algorithm's scheduling. The main processor and the coprocessor are, respectively, in

charge of the high-level tasks, the acquisition and processing decision imposed by the application, and the lower-level tasks, characterized by high level of processing regularity and parallelism. The processing can be adapted easily, thanks to the structure, to the application's requirements.

The paper is organized as follows. Section 2 presents the context and the objectives of the new embedded system. Section 3 presents the coprocessor platform. The coprocessor architecture is presented in Section 4 and its features are discussed in detail. The performance of the coprocessor architecture are reported in Section 5 and compared to a classical image acquisition and processing scheme. Results of a complete postal sorting application are presented in Section 6 showing the potential parallelism of this platform. Finally, Section 7 concludes the paper presenting the perspectives of further work.

2. CONTEXT AND COPROCESSOR INTO PROCESSING/ACQUISTION LOOP

For image-processing applications requiring very high-performances, an adaptive image acquisition stage is very often the key feature to satisfy the real-time constraints. Although we can nowadays observe the wide availability of low-cost high-speed high-resolution sensors, the high pixel rate to be transferred to the central processing unit from the image sensor is often the main system bottleneck in terms of performance. This fact indeed pushes the theoretically achievable system performance to higher and higher levels so as to be able to cover new demanding applications. However, higher pixel rates require new architectural approaches so as to reduce the costs of the interfacing and processing stages that are now the real bottleneck of such systems. Whenever such high pixel rate can be reduced according to the analysis of its intrinsic semantic content (i.e., image portions can be discarded not being relevant for the application), the response time of common system architectures is too slow to timely adapt the acquisition stage to the relevant image sequence content. Indeed, the transfer time from the sensor to the CPU unit is often too large to enable the system to react, and finally results in being too expensive in terms of equipment and interfaces.

The coprocessing approach has been investigated in the last few years by several authors. Some works presented in literature are based on hardware coprocessing designs specifically dedicated to a single application [8–10]. The performance improvements reported in literature are quite relevant, when comparing architectures with or without coprocessor, those results present speed-up factors up to few hundreds. Other authors have proposed generic systems whose property is the possibility to implement different algorithms on a coprocessing-based architecture [11]. In the class of "generic" coprocessor units, only a few authors have mentioned the possibility to control the image acquisition stage simultaneously with the processing stage. Gorgon proposed a coprocessor unit to control the acquisition stage of charge coupled devices (CCDs) sensor [12]. Jung et al. presented a preprocessing unit to control CMOS sensor [13], but the achieved functionality operates only on the specific image corrections used to compensate physical limitation of the CMOS sensor. The key point is to control the acquisition on the fly in function of the different algorithm tasks.

The integration of a coprocessing element into the image acquisition loop of a CMOS sensor has very interesting features. Standard CCD-based image systems are synchronous and require that the full image is downloaded before proceeding to a new acquisition. CMOS sensors are much more flexible because not only they are intrinsically asynchronous, but they are also capable of performing image acquisitions on limited section of the sensor up to the acquisition of single pixels. For several applications such flexibility can be successfully exploited so as to reduce the data transfer to the central CPU thus considerably reducing the necessary data bandwidth. As a consequence, the overall processing requirement of the application has just to process a limited portion of the original image. The key to achieve such results is to be able to provide to the main application the necessary information to adapt the acquisition stage without the need to transfer the full image to the central CPU. In other words, CMOS imaging can achieve the following:

(i) a selective image acquisition stage depending on the image content itself and on the requirements of the application,

(ii) a relevant reduction of the data volume to be transmitted to the processing unit once the selective acquisition stage has been activated.

The condition for which such features can be achieved is that a "coprocessing" element is inserted in the image acquisition loop driven by the "high-level" application. Different approaches can be considered; the implementation of an embedded ASIC or a configurable processor is one of them. The processing capabilities of these components, as the STV0676 (ST Microelectronics: STV0676 datasheet; available at http://www.datasheetcatalog.org/datasheet/stmicroelectronics/9068.pdf) or the CMOS coprocessor introduced by [14–16], are examples. However, the level of flexibility of such architecture is quite limited since only a few processing parameters can be configured according to the application constraints. All the components are fixed with their own processing. A recently designed chip with a coprocessor is the named OMAP DM-510 (Texas Instrument: OMAP DM-510 page; available at http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=12802&contentId=41258). Such component is very interesting, but only addresses low-power applications, typically mobile phones. Moreover, all the chip components have dedicated interfaces, thus the control of specific CMOS is very complex or even impossible, for the number of control signals to handle. For instance, the high-speed CMOS sensor MT9M413 (Micron-Aptina: MT9M413 datasheet; available at http://www.aptina.com/products/images_ensors/mt9m413c36stc/#overview) required almost 150 pins to be implemented (included data). The "artificial retina" approach has similar features in terms of processing possibilities [17]. These architectures are frequently developed

to process a neighborhood of the pixels [1–3], however the process is usually reduced to small size neighborhoods (usually smaller than 16×16). The acquisition can be controlled, but is dependent on the architecture and the targeted applications. So as to obtain the features presented above, a heterogenous architecture is proposed: a processor associated to an FPGA. The acquisition is a task of the FPGA, therefore any CMOS sensor can be controlled. Consequently, any CMOS sensor with specific acquisition modes (as, e.g., scan line) can be interfaced. The flexibility of the heterogeneous structure offers a large panel of solutions for the implementation of image processing for on-line industrial control.

In such architecture the "coprocessing" unit besides the control of the acquisition stage becomes naturally in charge of the standard low-level repetitive tasks such as filtering, denoising, and binarization. In fact, the full control of the acquisition stage enables the right control of the preprocessing tasks usually performed at the level of the central CPU or high-level application.

For instance, the "instructions" for a selective image acquisition stage, that is, an acquisition stage for which only a (small) portion of the image that presents certain features needs to be "acquired" and transmitted to the central CPU for further high-level processing, are handled by the "coprocessor" accessing directly the CMOS sensor itself in an asynchronous manner. At this point also the processing associated to the specific feature "found" in the image can be efficiently implemented at the "coprocessor" level. Then, only the "selected" image portion already preprocessed and/or prefiltered is transferred to the central CPU unit. The coprocessing task schedule can be selected on the fly depending on the acquisition commands and is adapted to the acquisition form that is region/pixel-based. By this approach, the necessary data bandwidth can be drastically reduced eliminating in most of the cases the major system limitation. An example of achievable performance for some classical preprocessing stage is provided in Section 5. The main processor, freed from image acquisition and preprocessing tasks, can then be used for further processing and/or high-level algorithms defined by the specific application.

The challenging aspects of the coprocessor design are mainly related to the variable acquisition mode (i.e., input image format and layout). Obviously, the bandwidth associated to a window processing can be optimized; moreover, the nature, the complexity, and the number of possible processing stages can be adapted at each acquisition mode. Many different acquisition modes are then available. In all modes, a window can be selected in the full-range image, the size and the integration time are defined, and a subsampling (on Y and X) can also be specified. In the simple multiexposition mode, the same window is acquired several times or periodically and the delay between two acquisitions can be defined. Also in the tracking multiexposition mode, the window can be translated. Such modes allow to create a "subimage" image by row or column accumulation when the sensor is used as line sensor even with lines varying their position during the acquisition itself.

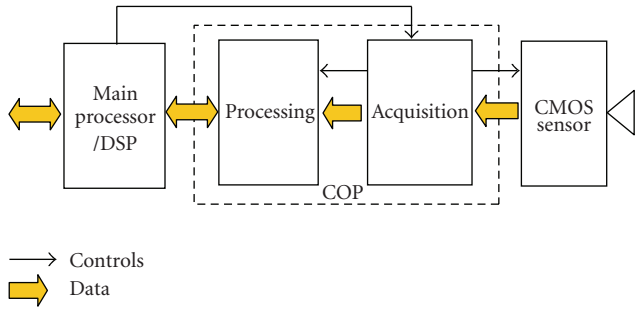


Figure 1: Block diagram of the coprocessor-based architecture.

The first interesting result achieved by implementing this architecture is that relevant speed-up factors are obtainable for reconfigurable processing modules, thus providing enough flexibility in terms of choices of processing and in terms of acquisition modes defined on the fly by the application itself (selection and preprocessing of any kind of area of interest). The second interesting result is that such on-the-fly adaptation of the acquisition mode yields a further bandwidth reduction for the transfer of the image data to the central CPU. This feature represents for some application a further speed-up in the overall system performance in terms of reduction of processing or increase of the achievable acquisition/processing frame rate.

3. ARCHITECTURE OF THE COPROCESSOR CAMERA

The overall system can be described as an autonomous intelligent camera with powerful embedded processing when compared with modular systems associated with a computer.

The system has been thought for monitoring applications such as road monitoring [18] or intrusion detection or any other similar application. Quality control and control of industrial processes, where very high frame-rate on specific image sections are required, is another application field of the system. For such kind of processes, only the "relevant" portions of the images are necessary to be transmitted to the host CPU for further processing. In some cases, only the result of the preprocessing, or of the processing (i.e., the detected feature), is needed to be transmitted outside the system to a local host PC or via Internet.

The system is composed of an embedded frame-grabber equipped, at different levels, of processing capabilities for the image acquired by the sensor and it is illustrated in Figure 1. This figure illustrates the main architectural components of the camera with embedded coprocessing stage. In this architecture, the coprocessor part (COP) is divided in two parts: a part dedicated to the acquisition and a second part dedicated to a preprocessing stage.

The association of the processing and acquisition stages aims at reducing the pixel rate for applications where "irrelevant" image portions are detected by the coprocessor. The processing is then complemented by the processor for higher-level tasks at a possibly lower pixel-rate. The partition

of the tasks is made by exploiting the specificity of each element, to use it as efficiently as possible, thus reducing the pixel-rate when possible and the processing time so as to increase the overall throughput. This architectural approach to the processing of sequences is particularly adapted to and performing, but not limited to, tracking applications, pattern recognition applications, and compression applications. Video compression is generally used in camera systems so as to reduce the bandwidth of the data transfer and to be able to use a standard communication channel without addition of acquisition boards such as the camera-link for instance. However, with high performance sensors, there is immediately the problem of the connection that becomes now the system "bottleneck", and prevents from transferring the images rate provided by the sensor. The system described in this paper also supports the implementation of a compression stage (thanks to multimedia processor's functionalities) that makes it possible to approach to the sensor limit capabilities.

The system is composed of a compact stack of 4 boards, enabling to easily interface various types of sensor/cameras and thus answering to various resolution and acquisition speed requirements in the most modular and economic way. The four boards described hereafter are the following:

(i) the motherboard containing the main processor,

(ii) the communication board,

(iii) the board including the coprocessor,

(iv) the camera interface board.

3.1. Motherboard

The motherboard contains a Nexperia PNX1500 (Philips: PNX 1500 datasheet; available at http://www.nxp.com/acrobat_download/literature/9397/75010486.pdf) processor at 300 MHz and includes functions for the sound and image processing. This processor has been selected for the powerful VLIW core and for the variety of supported integrated interfaces such as Ethernet controller, DDRam controller, and PCI. Moreover, it includes a 32-bit TriMedia 3260 CPU core with 5-issue slot engine and 31 pipelined functional units. It operates up to 10000 Mops and 1300 Mips and can execute up to 5 operations per clock cycle. In addition, Nexperia integrates a graphic 2D engine able to display up to a resolution of 1024×768 to 60 Hz. A large library can be used to program several standards (MPEG, H263, etc.) and several interfaces. The Nexperia power consumption depends on the processing charge, typically it is around 1.5 W. Around this processor, we can find communication interfaces such as Ethernet and ISDN, as well as acquisition and rendering of video images and analogical sound. The motherboard contains a 64 Mbytes DDR Ram (333 MHz) to store data. In addition, there are 32 Mbytes of flash memory used to store the programs or different information. Several interface components can communicate via a PCI bus, like Ethernet and VGA. Thus, motherboard contains an internal bus which is a PCI and all the platforms communicate via this bus.

3.2. Communication board

The second board is based on an FPGA Spartan XL (Spartan XL (Xilinx: Spartan XL datasheet; available at http://direct.xilinx.com/bvdocs/publications/ds060.pdf)) to manage the PCI arbiter, the communication interfaces such as USB2.0 and Firewire that can be driven to connect the camera with digital standard interfaces. This board provides four functionalities. It extends the communication of the motherboard with standards USB 2.0, IEEE1394, and Ethernet 10/100. The ethernet connection is important, because systems and a computer can communicate by this interface with a simple IP number. It also provides a centralized power supply for the system. On this board, a converter N/A is used to display on a standard VGA monitor. All the components cited before have a PCI input to avoid different link with the processor.

3.3. Acquisition and processing board

This board is the COP part in charge of the acquisition and the preprocessing stages of the video signal coming from the CMOS sensors or cameras. The preprocessing part is independent from the acquisition part. Its architecture is illustrated in Figure 2. The main functions are partitioned into two FPGAs. The first FPGA is a Virtex2Pro VP4-fg456-5 (Xilinx: Virtex 2 Pro datasheet; available at http://direct.xilinx.com/bvdocs/publications/ds083.pdf), their functions are to acquire images and communicate the configuration and orders to the camera. This FPGA drives the camera and can be adapted to several sensors or cameras; it is just an implementation with the right driver. Therefore, it implements a wide variety of acquisition modes (random region acquisition, variable image size, variable acquisition modes line/region based, multiexposition image). The second FPGA is a Virtex2Pro vp20-fg676-5[3], and high-performance image preprocessing (calibration, filtering, denoising, binarization, pattern recognition) is its principal function. These FPGAs include, respectively, 1 IBM Power PC, 28 18 kb BRAMs, 3008 slices and 2, 88, 9280 for the large one. Both FPGAs communicate through two high-speed serial channels, specific to Xilinx, called RocketIO (Xilinx: RocketIO User Guide; available at http://www.xilinx.com/bvdocs/publications/ug024.pdf). Moreover, each FPGA communicate with the processor via the PCI bus. Xilinx FPGAs have been chosen, because of their high-speed serial communication, PowerPC are available, and they are flexible and reprogrammable. Interface drivers are already developed and optimized by Xilinx.

This board contains, in addition of FPGAs:

(i) 2 SDRAM until 128 Mbytes associated with the acquisition FPGA used to store several images,

(ii) 2 ZBT until 8 Mbytes associated with the coprocessing FPGA used for processing tasks,

(iii) 4 optocoupled inputs,

(iv) 4 optocoupled output,

(v) 2 encoders.

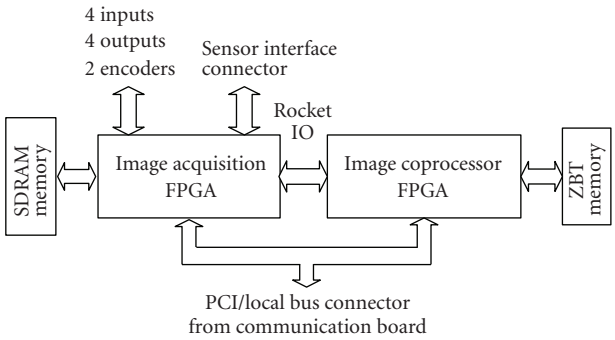


Figure 2: Block diagram of acquisition and processing board architecture.

Power consumption of this board also depends on the processing performed. The power dissipation ranges from 3 up to 12 W depending on FPGA and memories utilization. For example, in the implemented application, explained Section 6, power dissipation is below 7 W.

3.4. Camera interface board

The fourth board is a simple interface board between the FPGA board and the camera. This board depends on the camera interface. It can be adapted and changed. Thus, the system can be used with several CMOS image sensors, for the results described in this paper, a sensor IBIS4-1300 (Cypress: IBIS4-1300 datasheet; available at http://www.datasheetcatalog.org/datasheet2/2/044ipph9289eg3l6qa3eadzds1fy.pdf), with a resolution of 1280 × 1024 pixels and a 40 MHz pixel frequency has been used for the experimental results. Its interface is a Camera Link.

The main boards communicate through bus PCI v2.2 allowing to transfer a large number of data (up to 133 Mbytes/s) to the host processor (in accordance with Nexperia bus interface). The entire architecture is illustrated in Figure 3, with processing components and available interfaces.

However, the main idea of the system architecture is indeed to reduce as much as possible the data rate after the coprocessor unit by transmitting only the processed image sections or by controlling the acquisition.

This architectural solution provides exceptional processing potential and offers wide communication possibilities. The processing part is built with pipelined or parallel HW processing modules to obtain high performance. Furthermore, a processing and data transfer, from CMOS sensor to processor, can be operated in parallel so as to increase performances.

The main additional advantages of this system, besides the capability of controlling the acquisition loop and the achievable processing performances compared to a traditional modular PC system, can be summarized as follows:

(i) a low dissipated power,

(ii) compact dimensions,

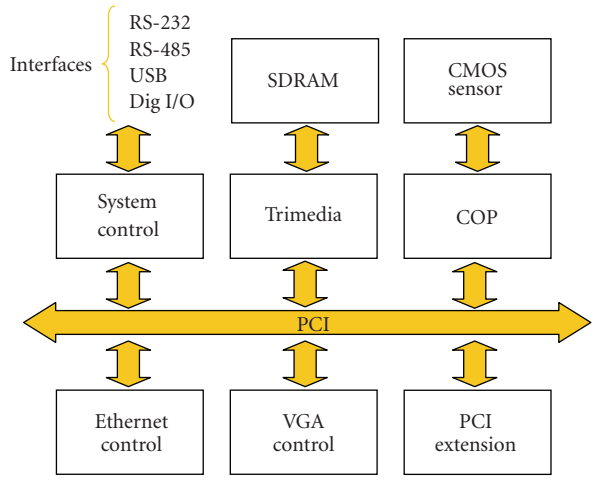


Figure 3: Block diagram of the system architecture.

(iii) a greater robustness (mean time between failures) because it does not integrate mobile components (ventilators, hard disks),

(iv) a greater commercial lifespan because components in the computers world are very volatile and cannot be replaced with components having the same characteristics. Sometimes, after only a few years, partial redesign of the system is required to critical applications.

Power supply of the system is operated at 12 V and can reach up to 36 W. In the test case applications, it works at about 20 W, including the power supply of the CMOS camera. Dimensions of this system are 150 × 150 × 60 mm. In future versions, dimensions will be reduced and the internal communication bus could be directly replaced by the PC104 or another similar performing bus.

4. COPROCESSOR DESIGN

The essential problem of the coprocessor architecture is the tradeoff between processing efficiency and flexibility required to exploit the CMOS potential features. Two different parts essentially constitute the COP architecture (Figure 4): the processing and the acquisition parts. The functional blocks constituting the processing part are a processor interface (PCI interface), a command controller, a processing controller, a processing unit, and an SRAM. Thus, the COP architecture is essentially constituted by the following functional blocks (Figure 4):

(1) a processor interface (bus interface),

(2) a bus bridge, a command controller,

(3) a processing controller,

(4) a processing structure,

(5) a CMOS sensor interface.

The command controller receives the acquisition commands, the processing commands from the main application. The

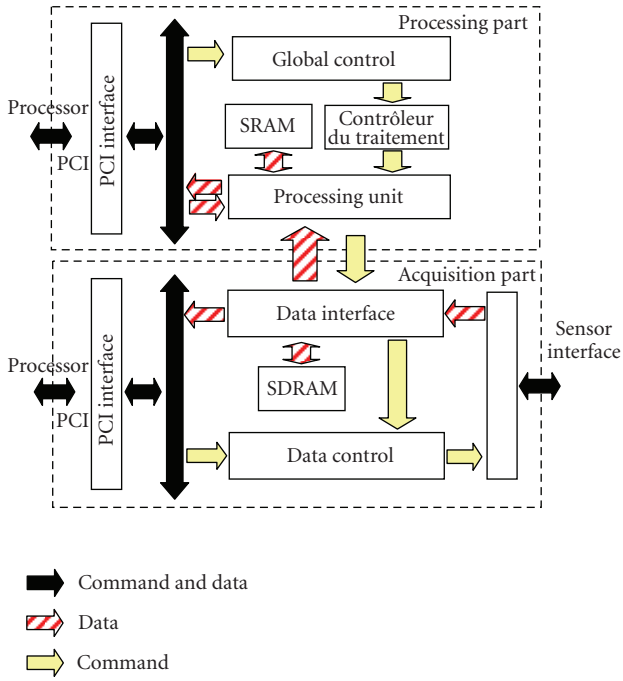


Figure 4: Block diagram of the COP architecture.

task scheduling is controlled by the processing controller and is executed by the processing structure unit configured according to the received commands. The data and image portions, provided by the main CPU and used by the coprocessor for the actual processing tasks, are transferred to the processing structure via the bus bridge and via the processing controller. This feature enables to implement a true coprocessing stage and not a simple preprocessing.

The link between sensor and acquisition part is specific for each image sensor, consequently it should be modified after any sensor change. The connection between acquisition and processing is standard, therefore independent of the sensor. Acquisition commands are constituted of parameters defined to cover a large number of acquisition modes to enable to interface a large sensor sort (linear CCD, CMOS matrix). Eventually, the connection with coprocessor and processor are linked with standard PCI. Hence, the coprocessor is independent of the processor and could be used as embedded IP with any PCI system. The coprocessor architecture enables a full data rate to be obtained on PCI bus.

The possibility to adapt the number and nature of the processing and to operate on variable size/shape images is provided by the flexibility of the processing structure unit.

In essence, it is constituted by five different components (Figure 5): CONTROL_MEM is in charge of the main memory, CONTROL_PRO is in charge of the processing control, the processing modules, the system control, and the FIFO is in charge of the temporary storage. Such architecture implements several options for the data flow control (Figure 5). The input data, provided by CMOS sensor and by the processor, are referred to in Figure 5, respectively, with the numbers 1 and 3. There is no FIFO in 1 since there is a memory in the CMOS interface. The broadcasting nets referred to as 2, 4, and 5 allow to copy the data and transfer them on each output branch. The copy is specified for each net by the command word. The nets referred to as 2 permit to transfer the input image without processing. The nets 4/5 permit to transfer the result image between two processings, simultaneously with data loading/result reading. The processing structure unit can be configured to adapt its processing in function of the acquisition mode and in function of the high-level application via software. The current acquisition data has to be stored into an internal memory to allow the preprocessing stage. Several types of preprocessing require a pixel neighborhood for each pixel process. A common way to operate is to use a video line to store few image rows. Unfortunately, such solution is not possible because the subimage size is not fixed. In the architectural solution presented here, an internal cache memory is associated at each processing. Consequently, the processing flow might not be synchronized with the output data flow of the memory MEM. Such solution enables to decrease the number of accesses to MEM. The size and features of the cache are defined to match the selected processing.

The processing modules are sharing the same input and output busses that are connected to the bidirectional main memory bus. So in order to store the results in the same memory, the input data enables to cascade the processing or to apply the same processing several times [19]. The tasks implemented in HW can be scheduled by the user by simple ordering of the tasks execution without any ordering constraint. To illustrate the intrinsic processing potential of the coprocessing, the performance of different implemented processings are reported in Section 6. A full application using SW/HW processing is presented in Section 5, to illustrate the capacity of the full system in terms of processing and data flow parallelism.

5. EXAMPLE OF ACHIEVABLE PERFORMANCE

The image processing presented in this section are commonly used by on-line industrial control systems. All these processing algorithms are quite *regular* and require a neighborhood of pixels and common operators (accumulation, multiplication, square, sorting, and logical). Therefore, they present a high potential parallelism that can be exploited during execution. Thus a hardware implementation is directly inserted in the coprocessor architecture. The image acquisition FPGA contains the camera driver, a PCI core and a rocket IO core. These FPGA resources are used for about 90%. No processing is implemented on this component. All the processings of the coprocessor are implemented in the other FPGA. Three different processing types have been implemented in the coprocessor:

(i) a median filter on different basic kernels (1×3, 1×5, 3×3),

(ii) a local adaptive binarization (Niblack algorithm) with a neighborhood of 8×8 or 16×16 pixels [20],

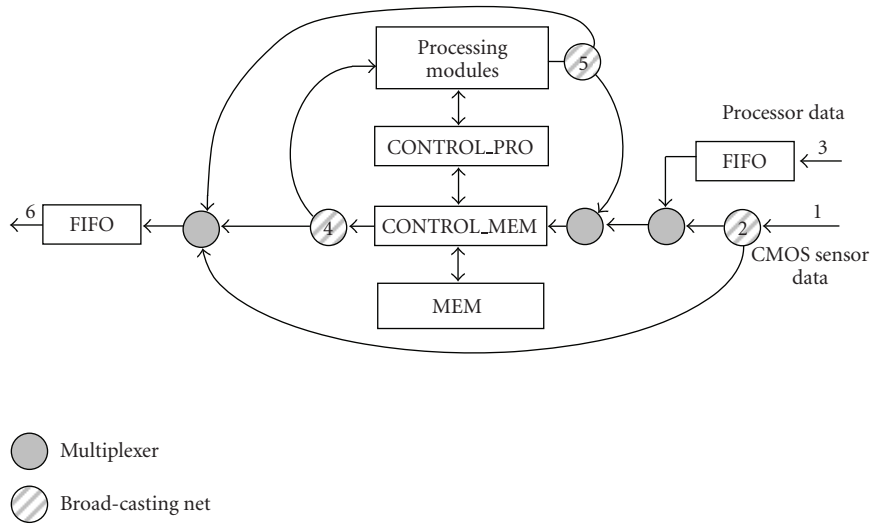


Figure 5: Diagram of the COP functional.

Table 1: Time processing of median filter 3×3.

	3×3	1×3, 1×5
Number of slices	313	265
Number of block RAM	9	
Number of mult 16×16	0	
Frequency (MHz)	100	
Image size	Time processing (ms)	
512×512	5.22	2.61
256×256	1.30	0.65
128×128	0.32	0.16

(iii) a binary pattern recognition based on block matching with 32×32 and 64×64 pattern size.

All the implementation of the three processing includes a cache memory to provide data without latency to the designed architecture. The performances and the required hardware-resources obtained by the coprocessor architecture are reported in Table 1 for the median filter, Table 2 for the local adaptive binarization, and Tables 3 and 4 for the pattern recognition.

The median filter is a simple sliding-window spatial filter that replaces the center value in the window with the median of all the pixel values in the window. The median filter is normally used to reduce noise in an image. In the median filter implementation, two kinds of filter are implemented: a one dimensional (1×3, 1×5) and a two dimensional (3×3). The required resources for each kind are 265 and 313 slices for processing, the reunification use 893 slices which include all the processing tasks and the handling of all transfers and local buffers. Used in the preprocessing stage, the local adaptive binarization not only provides a very performing algorithm in presence of illumination or object variations, but also allows to reduce the bandwidth to the central CPU like an artificial retina sensor can perform. For example, a 1024×1024 full-range image requires 1 Mbytes to be stored but the binarized image only 1 Mbits. If an area can be selected in the full-range image, for example a 256×256, the result image size would reduce to 64 Kbits. This process allows gaining a factor 64 on the original bandwidth. A threshold is processed for each pixel considering a fixed size neighborhood (8×8 or 16×16 can be selected). The following algorithm defines the local adaptive binarization:

$$\begin{aligned} &\text{Threshold} = \text{Mean(NE)} - [0.1875 \times \text{Std(NE)}] \\ &\text{If } \text{Std}^2\text{(NE)} < \text{STDREF then } B = 0 \\ &\text{Else if } P < \text{Threshold then } B = 1 \text{ else } B = 0, \end{aligned} \tag{1}$$

where NE is the considered neighborhood, STDREF is the standard deviation of reference (determined experimentally), P is the pixel grey level and B is the binary value. Figure 6 describes the pipelined implementation of this algorithm (as mean, standard deviation, and variance). A cache memory (not represented in Figure 6) provides 4 pixels to the accelerator at each cycle. The signal Sel8o16 enables the size of neighborhood to be selected.

For the third processing, the binary shapes search, different tests have been made to compare with different image size and search sizes. The goal of this processing is to recognize one or more shapes in a binary image. The search window and the binary shape are both binary. The different shapes are searched in the defined window. The detection is based on the comparisons using the logical operator XNOR, which replaces the traditional matching criterion based on the sum of absolute difference (SAD). The binary correlation algorithm is defined as follows:

$$f(i,j) = \sum_{x=0}^{M-1} \sum_{y=0}^{M-1} [s_1(x,y) \text{XNOR} s_2(x-i, y-j)] \tag{2}$$

i and j between 0 and $N - M$. with S1 the binary shape ($M \times M$ size), S2 the binary search window ($N \times N$ size).

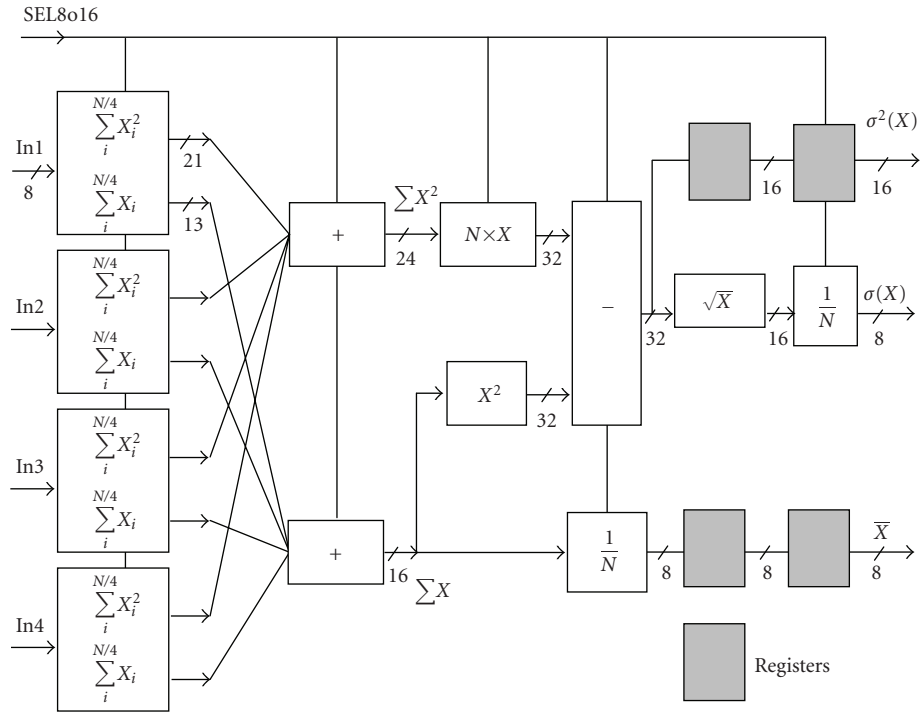


Figure 6: Block diagram of the coprocessor component implementing local adaptive binarization process.

Table 2: Local adaptative binarization.

Number of points		2	4	8
Number of slices		477	965	1605
Number of mult 16×16		5	9	17
Frequency (MHz)		25	25	25
Size		Time processing (ms)		
Image	Block			
512×512	M16 × 16	158.09	79.04	39.52
	M8 × 8	40.80	20.40	10.20
256×512	M16 × 16	76.66	38.33	19.16
	M8 × 8	20.12	10.06	5.03
256×256	M16 × 16	37.75	18.59	9.87
	M8 × 8	9.92	4.96	2.48
128×128	M16 × 16	8.17	4.09	2.04
	M8 × 8	2.34	1.17	0.59
64×64	M16 × 16	1.54	0.77	0.38
	M8 × 8	0.52	0.26	0.13
Cycle time		40	20	10

This binary approach is commonly used in the optical character recognition field [21]. The binary shapes usually represent characters with different orientations. The entire processing is implemented and require 3021 slices and 9 blocks RAM of 18 kb each. In the table, only the required slices for a single processing are reported.

A comparison has been done between the performance obtained by the coprocessor architecture (COP) and a PC, Bi-Xeon 1.7 GHz, 256 Mo Ram, Rambus 800 MHz (2×400 MHz). The performance results reported in Table 5 do not consider camera frame-grabber transfer time for the PC-based platform. The comparison shows that, besides the achieved speed-up factor up to a factor of 5 that would certainly result in higher considering the frame-grabber transfer time, the central CPU in the coprocessor approach is fully available for further processing. Moreover, when

Table 3: Binary shapes research with a 50 MHz frequency.

Shape size	64×64		32×32		16×16	
Block size	64×128		32×64		16×32	
Mem. blocks	6		3		2	
Slices/	2328	1300	1250	700	750	400
detect. block	127		70		40	
detection block used	16	8	16	8	16	8
Image Size	Processing time (ms)					
512×512	16.05	9.2	4.93	32.1	18.4	9.86
256×256	2.96	2.015	1.155	5.92	4.03	2.31
256×512	6.9	4.32	2.39	13.8	8.64	4.78
128×128	0.335	0.375	NC	0.67	0.75	NC

Table 4: Time to search a shape in an image (ms).

Number of shapes	5	4	3	2	1
256×512	68.52	59.88	51.24	42.6	33.96
256×256	32.67	28.64	24.61	20.58	16.55
128×128	6.73	5.99	5.24	4.49	3.74

Table 5: Processing comparisons.

Processing	PC (Mpixel/s)	COP (Mpixel/s)
Median 1×3	41	100
Median 1×5	28	100
Median 3×3	27	50
Niblack 8×8	5	25
Niblack 16×16	4	14

a bandwidth reduction is possible by means of adaptive acquisition, the coprocessor approach provides much higher speed-up gains.

6. APPLICATION EXAMPLE: READING A BAR CODE FOR POSTAL SORTING

6.1. Application description

The postal sorting is a real-world example showing the processing possibilities and the achieved level of parallelism of the system [22]. The goal of this application is to read bar codes on the letters, to enable the automatic sorting at the different stages of the logistic postal letter handling. If the bar codes cannot be read, the letter is rejected and need to be processed manually. This application has been developed with the objective of replacing an exiting platform which integrates a camera associated with a PC. The new embedded solution has been developed to increase as much as possible the processing performances and to obtain a portable and more flexible system. Indeed due to the fact that bar codes printed on letters may be of bad quality or superposed to other visual information the possibility of implementing more complex processing increase the rate of correct detections/decodings achievable. Ideally, to correctly read the largest percentage of bar codes, each processing stage

Figure 7: Example of an image which contains a bar code.

should require as much as possible processing power so as to guarantee that the bar code area is correctly localized (framed in Figure 7). In reality the processing resources are limited and, results are easier to extract and process a small part of the image that with a high probability includes the bar code, instead of dealing with the entire image of the letter which includes extra information that can potentially create errors for the code bar detection and decoding. In the postal sorting test application example, a letter is grabbed with the CMOS camera, and the speed of the transporter is around 4 meters per second. In the coprocessor platform, the processing stages used are

(i) transposition,

(ii) high pass filtering,

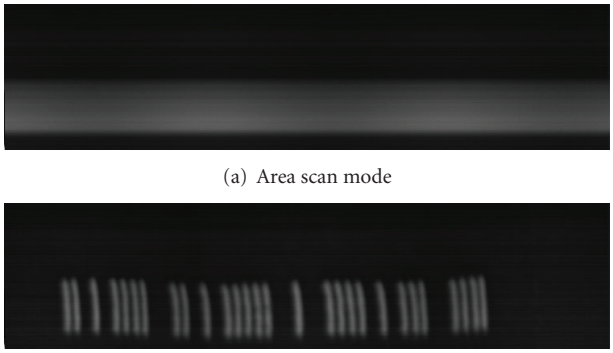
(iii) dilatation plus subsampling,

(iv) blobbing.

The final blobbing task is performed in the main processor. Details of these processing stages are provided in the following section. The final task is to read the bar code and send it to the postal sorting machine.

6.2. Details of the processing

So as to grab a letter, the CMOS camera is configured in the line scan mode since the high speed of the transporter (4 meters per second) would result into an image deformed as reported in Figure 8(a). The camera grabs the same line during a predefined number of lines or continuously and the acquisition FPGA rebuilds an image; this mode is shown in Figure 8(b). In this picture, the difference between the two modes is shown, and particularly the effect on the bar code. In area scan mode, the bar code would be completely unreadable. The first processing applied is a transposition. The transposition is used to rotate the rebuilt image to 90 degrees. A transposition is necessary because the other



(a) Area scan mode

(b) Line scan mode

Figure 8: Visualization of the bar code with the two different acquisition modes of the camera.

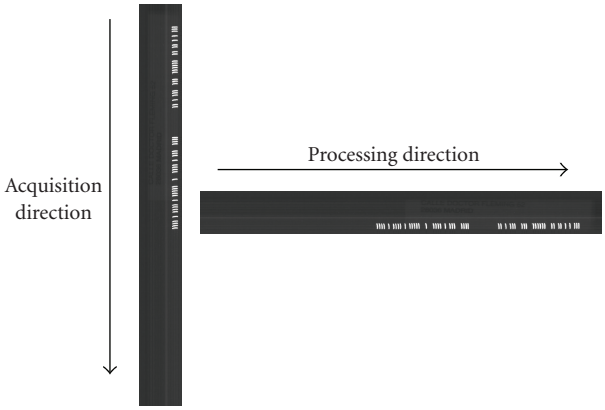


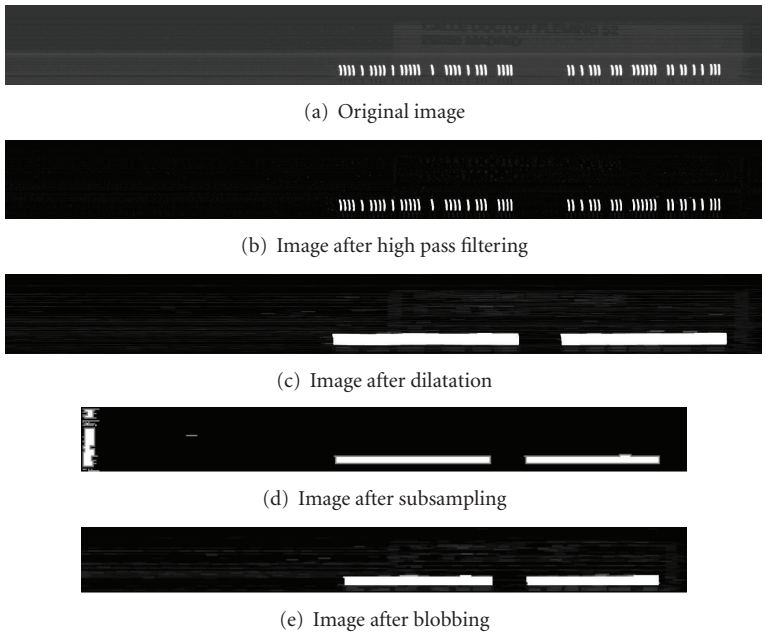
Figure 9: Transposition of an image to change in the appropriate processing sense.

processing stages are only compatible with a horizontal scanning (Figure 9). Four 32 bits words of four lines are stored in the memory, transposed, and stored again at a correct place in the SRAM memory. Transposition operation uses 4 registers to store temporally the result. The resources used by the FPGA are 186 slices (only about 2%). The first real image processing is a high pass filter. The high pass filter is used to delete the background and to raise the white bar code as shown in Figure 10(b) compared to the original image in Figure 10(a). As shown previously, the data bus is a 32 bits bus, and transfers 4 pixels at the same time. This task processes 4 pixels at the same time (i.e., 4 filters are active simultaneously). The high pass filter is a convolution between the image and the window which includes the coefficients. Results are directly stored in the SRAM. Obviously the coefficients of the filters are reprogrammable to be adapted to any other application. Resources used for this processing are 1125 slices (12%) and 44 (4×11) multipliers (50%).

The second processing stage is a dilation. The dilation is used to complete the region which integrates the bar codes, thus it will be easier to detect this region as shown in Figure 10(c). This latter processing is performed to replace the central pixel by the maximum value of the 32 neighbor pixels.

The subsampling is the third processing. In fact, only one of 4 pixels is transferred, moreover one line over four (Figure 10(d)). The goal is to divide the size of the image by 16 and consequently the original pixel bandwidth. The dilation and the subsampling are executed in the same tasks to reduce memory access. As described in the two last processing, results are stored into the SRAM.

These three last processing stages are performed in one dimension (line dimension) to obtain the best result and to reduce the processing time with the access of the second dimension. The implemented processings (including the SRAM address controller) request all together 2623 slices (26%) and 44 multipliers (46%). Each processing is spilt in several dependent tasks in the pipelined architecture. Full application with all driver interface use 3880 slices (41%), 4 BRAMs (4%), and 44 multipliers (50%). 3 BRAMs are used like FIFOs to save the transfer of data in the interface, due to different clock domains. (*processing* ⇒ *PCI*, *RocketIO* ⇒ *processing* and *PCI* ⇒ *processing*). The fourth BRAM is used to store all the configuration sent via the PCI bus (configuration of each processing and of the application). Between each processing (transposition, high pass filter, and dilation + subsampling), a storage of results is made. Results after the high pass filter need to be stored and kept. In this image, only the areas will be sent after the coordinates calculation. As seen in the previous section, only the blobbing is made by the SW processor and all the other processing stages are performed by the FPGA (coprocessor). The blobbing is the last processing stage of the code bar detection. After the dilation, several white (or grey) areas are labeled. In Figure 10(d), two large areas are detected (the number of areas depends on the number of sections in which the bar code is partitioned), that correspond to the area including the bar codes, but other areas may be detected which are probably not part of the code. The goal is to determine the coordinates of the two zones that contain the code. The processor receives the result image of the dilation from the SRAM of the coprocessor and stores it into the DDR RAM associated with the processor. So as to detect a region (blob), the image is described row by row and when a pixel value exceeding the threshold is founded, the object is squared and associated with a label. Once the image is fully analyzed and labeled, the two largest areas, that are chosen probably including the bar code and the coordinates of these two objects, are extracted. Figure 11 shows a part of the blobbing image where white areas are detected (squared in grey). The transfer of these coordinates is made to the coprocessor and the coprocessor transfers only the selected regions to the processor. The regions are taken on the filtered image which is stored temporally in the SRAM (Figure 12). When bar codes are transferred to the processor, the decoding can be activated. To decode the bar code, an FFT is made following several tests to read correctly percentage rates approaching 100% of bar code detected. As shown in Figure 12, the decoded bar code value in the picture example is "11110101111010111110010011110101110011111" and after all the processing the system correctly read the code.



(a) Original image

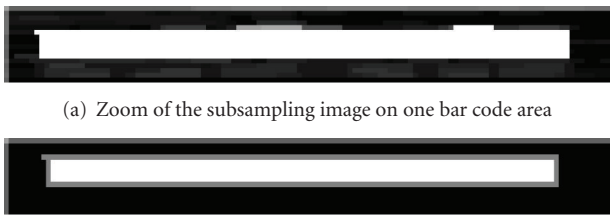
(b) Image after high pass filtering

(c) Image after dilatation

(d) Image after subsampling

(e) Image after blobbing

Figure 10: Resulting images after each processing.



(a) Zoom of the subsampling image on one bar code area

(b) Zoom on the same bar code area after the blobbing

Figure 11: Principle of the blobbing processing.

Figure 12: Zone transferred after processing to read the bar code.

(a) Original image not readable at sight

(b) Image readable after processing

Figure 13: Preprocessing for improvement of "unreadable" bar code.

In Figure 13, the efficacy of the system is illustrated also with a bad bar code image which is not even readable at sight and without an appropriate processing. However, the system can correctly read it. Here, the bar code cannot be read exactly, but the system reads the correct bar code which is "11100110110101011111100101110101011100111 1". It proves the efficiency of the system.

So as to speed up processing tasks and decoding, the different stages can be performed in parallel and not sequentially. The application is divided in three main stages: acquisition (task 1), processing (task 2), and reading (task 3). Task 2 is the association of all the processing tasks executed by the COP and the blobbing including all the data transfer. These 3 tasks are executed in parallel to gain time and increase the number of letters processed. In Figure 14, a sequential sequence (task 1 following 2 and 3) is shown and during the acquisition the processor and the coprocessor do not work. The same remarks are valid when the coprocessor or the processor works simultaneously.

The specificity of the platform is that the 3 principal actions can work in parallel (i.e., tasks 1, 2, and 3 simultaneously). When an image is grabbed at time T, then COP processes the image T-1 and the processor reads the bar code of the image T-2. Only the transfer between the FPGAs prevents an acquisition or a processing into the coprocessor. Moreover, the processor is implemented using a full DMA mode and can work continuously. It receives data and at the same time it decodes the bar code. By using this configuration, the processing time is the same from acquisition to the output, but the number of processed letters is increased. Results are shown in the following section, and a comparison between a sequential mode, a parallel mode, and the PC performances is provided.

6.3. Results and comparisons

In this section, the results of the processing are provided. A comparison between the classical PC-based system and the

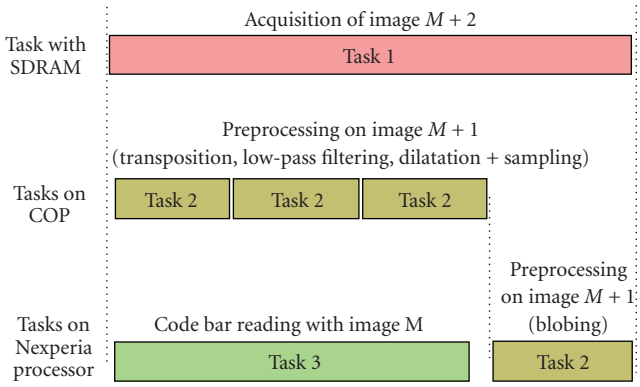


Figure 14: Scheduling of the 3 different tasks in parallel.

Table 6: Processing time.

Processing	Time (ms)
Acquisition	15.4
Transfer between the 2 FPGAs	4.6
Transposition	1.54
High pass filtering	1.54
Dilatation plus subsampling	1.54
Transfer in the processor of the subsampling image	0.15
Blobbing	4
Transfer in the processor of the bar code image	0.11
Reading the bar code image	12
Total	40.88

Table 7: Platform versus PC (approximative time).

	Sequential	Parallel	PC (ROI)
Time processing (ms)	40	40	40
Number of letter	15	30	15
Theoretical speed (m/s)	4	8	4

coprocessor platform is made. The PC platform is equipped with the camera (BCi4; Vector international/CCAM Technologies, available at http://www.vector-international.be/) associated with the compatible frame grabber. The coprocessor platform is obviously equipped with the same camera. The PC has a processor 3.2 GHz and 1 Go of RAM.

In Table 6, the necessary time for each processing needed to decode a bar code is shown. The tests were made with an image of 180 pixels width and 1712 rows captured. It is about a standard acquisition for a letter. Transfer is considered as a processing in the table. The transfer time from the coprocessor to the processor by the PCI is considerably reduced. The transfer of an entire image by the PCI takes 2.3 microseconds, but to transfer a subsampled image, it is 16 times lower (0.15 microsecond) and for the bar code it is 20 times lower (0.11 microsecond). This saving is a very important factor to speed up the overall processing performance.

(a) Coprocessor platform plus camera

(b) PC and coprocessor platform

Figure 15: Portability: coprocessor platform versus PC.

Table 7 presents the comparison between

(i) the coprocessor platform and a sequential reading,

(ii) the coprocessor platform and a parallel reading,

(iii) the current PC platform.

In the sequential and parallel mode, the processing time is approximately the same as a PC platform, but employing the parallelism, the number of the processed letters can be increased (i.e., number of images). In the case of the PC, the size of the processed image is reduced to a small ROI (around 512×70), against 1712×180 with the coprocessor platform. If the size is reduced to include correctly the bar code and not the image of the letter, the number of letters read can be increased up to 50. Moreover, the coprocessor platform is more efficient in hostile environment, small in size, and equivalent in terms of the percentage of bar codes correctly read. The portability of the two systems is illustrated in Figure 15. The size is reduced and results in being more appropriate for the integration in an industrial process.

7. CONCLUSION

Despite the increasing speed of PC processors and bus frequencies, the implementation of embedded coprocessor architectures expressly conceived for image sensors and inserted in the acquisition loop presents several advantages. Very high processing speed and reduced image data bandwidth are achievable. The architecture also provides

high degree of flexibility in the preprocessing stage for the different acquisition modes specific of CMOS imaging. Moreover, new processing tasks can be easily added in function of the application and the platform supports a wide panel of data interfaces. A complete test case application has been successfully implemented on this platform, with significant performance improvements when compared to a classical PC-based platform.

REFERENCES

[1] L. F. L. Y. Voon, G. Cathebras, B. Bellach, B. Lamalle, and P. Gorria, "Silicon retina for real-time pattern recognition," in *Sensors and Camera Systems for Scientific, Industrial, and Digital Photography Applications II*, vol. 4306 of *Proceedings of SPIE*, pp. 168–177, San Jose, Calif, USA, January 2001.

[2] R. D. Burns, C. Thomas, P. Thomas, and R. Hornsey, "Pixel-parallel CMOS active pixel sensor for fast object location," in *Ultrahigh- and High-Speed Photography, Photonics, and Videography*, vol. 5210 of *Proceedings of SPIE*, pp. 84–94, San Diego, Calif, USA, August 2003.

[3] J. Dubois, D. Ginhac, M. Paindavoine, and B. Heyrman, "A 10 000 fps CMOS sensor with massively parallel image processing," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 3, pp. 706–717, 2008.

[4] Y. Shi, R. Taib, and S. Lichman, "GestureCam: a smart camera for gesture recognition and gesture-controlled web navigation," in *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision (ICARCV '06)*, pp. 1–6, Singapore, December 2006.

[5] R. Y. D. Xu, "A computer vision based whiteboard capture system," in *Proceedings of IEEE Workshop on Applications of Computer Vision (WACV '08)*, pp. 1–6, Copper Mountain, Colo, USA, January 2008.

[6] R. Mosqueron, J. Dubois, and M. Paindavoine, "High-speed smart camera with high resolution," *EURASIP Journal on Embedded Systems*, vol. 2007, Article ID 24163, 16 pages, 2007.

[7] M. McErlean, "An FPGA implementation of hierarchical motion estimation for embedded object tracking," in *Proceedings of the 6th IEEE International Symposium on Signal Processing and Information Technology (ISSPIT '06)*, pp. 242–247, Vancouver, BC, Canada, August 2006.

[8] B. Bosi, G. Bois, and Y. Savaria, "Reconfigurable pipelined 2-D convolvers for fast digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 3, pp. 299–308, 1999.

[9] C. W. Murphy and D. M. Harvey, "Reconfigurable hardware implementation of BinDCT," *Electronics Letters*, vol. 38, no. 18, pp. 1012–1013, 2002.

[10] N. W. Bergmann and Y. Y. Chung, "Video compression with custom computers," *IEEE Transactions on Consumer Electronics*, vol. 43, no. 3, pp. 925–933, 1997.

[11] C. Hinkelbein, A. Kugel, R. Maenner, et al., "Pattern recognition algorithms on FPGAs and CPUs for the ATLAS LVL2 trigger," *IEEE Transactions on Nuclear Science*, vol. 47, no. 2, pp. 362–366, 2000.

[12] M. Gorgon and J. Pryzybylo, "FPGA based controller for heterogenous image processing system," in *Proceedings of Euromicro Symposium on Digital Systems Design (Euro-DSD '01)*, pp. 453–457, Warsaw, Poland, September 2001.

[13] Y. H. Jung, J. S. Kim, B. S. Hur, and M. G. Kang, "Design of real-time image enhancement preprocessor for CMOS image sensor," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 1, pp. 68–75, 2000.

[14] K. Tiri, D. Hwang, A. Hodjat, et al., "A side-channel leakage free coprocessor IC in 0.18 μm CMOS for embedded AES-based cryptographic and biometric processing," in *Proceedings of the 42nd Design Automation Conference (DAC '05)*, pp. 222–227, Anaheim, Calif, USA, June 2005.

[15] S. G. Smith, J. E. D. Hurwitz, M. J. Torrie, et al., "A single-chip CMOS 306 $\times$ 244-pixel NTSC video camera and a descendant coprocessor device," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 12, pp. 2104–2110, 1998.

[16] X.-R. Yu, Z.-B. Dai, and X.-H. Yang, "A parallel co-processor architecture for block cipher processing," in *Proceedings of the 7th International Conference on ASIC (ASICON '07)*, pp. 842–845, Guilin, China, October 2007.

[17] F. Paillet, D. Mercier, and T. M. Bernard, "Second generation programmable artificial retina," in *Proceedings of the 12th Annual IEEE International ASIC/SOC Conference*, pp. 304–309, Washington, DC, USA, September 1999.

[18] M. Bramberger, M. Quaritsch, T. Winkler, B. Rinner, and H. Schwabach, "Integrating multi-camera tracking into a dynamic task allocation system for smart cameras," in *Proceedings of IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS '05)*, pp. 474–479, Como, Italy, September 2005.

[19] J. Dubais and M. Mattavelli, "Embedded co-processor architecture for CMOS based image acquisition," in *Proceedings of IEEE International Conference on Image Processing (ICIP '03)*, vol. 2, pp. 591–594, Barcelona, Spain, September 2003.

[20] O. D. Trier and A. K. Jain, "Goal-directed evaluation of binarization methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 12, pp. 1191–1201, 1995.

[21] S. V. Rice, F. R. Jenkins, and T. A. Nartker, "The fifth annual test of OCR accuracy," Tech. Rep. TR-96-01, Information Science Research Institute, Las Vegas, Nev, USA, 1996.

[22] R. Mosqueron, J. Dubois, and M. Mattavelli, "Smart camera with embedded co-processor: a postal sorting application," in *Optical and Digital Image Processing*, vol. 7000 of *Proceedings of SPIE*, pp. 1–12, Strasbourg, France, April 2008.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 367860, 11 pages
doi:10.1155/2008/367860

Research Article

An Evaluation of Dynamic Partial Reconfiguration for Signal and Image Processing in Professional Electronics Applications

Philippe Manet,[1] Daniel Maufroid,[2] Leonardo Tosi,[3] Gregory Gailliard,[2] Olivier Mulertt,[4] Marco Di Ciano,[5] Jean-Didier Legat,[1] Denis Aulagnier,[6] Christian Gamrat,[7] Raffaele Liberati,[8] Vincenzo La Barba,[9] Pol Cuvelier,[10] Bertrand Rousseau,[1] and Paul Gelineau[2]

[1] *Université catholique de Louvain, Place du Levant 3, 1348 Louvain-la-Neuve, Belgium*
[2] *Thales Communications, Boulevard de Valmy 160, 92704 Colombes, France*
[3] *CESVIT MICROELETTRONICA, Via F. Frediani, 59100 Prato, Italy*
[4] *MBDA, Avenue Réaumur 1, 92358 Le Plessis Robinson, France*
[5] *Tecnopolis CSATA, Str P. Casamassima km 3, 70010 Valenzano Bari, Italy*
[6] *Aerospace Division, Thales, Avenue de la 1ere DFL 10, 29283 Brest, France*
[7] *CEA LIST, CEN Saclay, 91191 Gif Sur Yvette, France*
[8] *ELETTRONICA, Via Tiburtina Valeria km 13, 700, 00131 Rome, Italy*
[9] *Thales Italia, Via E. Mattei 20, 66013 Chieti Scalo, Italy*
[10] *Thales Communications, Rue des Frères Taymans 28, 1480 Tubize, Belgium*

Correspondence should be addressed to Philippe Manet, philippe.manet@uclouvain.be

Received 29 February 2008; Revised 11 July 2008; Accepted 3 November 2008

Recommended by Guy Gogniat

Signal and image processing applications require a lot of computing resources. For low-volume applications like in professional electronics applications, FPGA are used in combination with DSP and GPP in order to reach the performances required by the product roadmaps. Nevertheless, FPGA designs are static, which raises a flexibility issue with new complex or software defined applications like software-defined radio (SDR). In this scope, dynamic partial reconfiguration (DPR) is used to bring a virtualization layer upon the static hardware of FPGA. During the last decade, DPR has been widely studied in academia. Nevertheless, there are very few real applications using it, and therefore, there is a lack of feedback providing relevant issues to address in order to improve its applicability. This paper evaluates the interest and limitations when using DPR in professional electronics applications and provides guidelines to improve its applicability. It makes a fair evaluation based on experiments made on a set of signal and image processing applications. It identifies the missing elements of the design flow to use DPR in professional electronics applications. Finally, it introduces a fast reconfiguration manager providing an 84-time improvement compared to the vendor solution.

1. INTRODUCTION

Signal and image processing applications require a lot of computing resources. Until recently, many of them were implemented using digital signal processor (DSP) that provides flexibility and computing power at a low cost. Nevertheless, since the beginning of the decade, the frequency of the processor cores does not increase anymore while their power consumption increases dramatically with performances [1, 2]. This power wall leads to unmanageable thermal and autonomy issues in embedded systems with an emphasis for battery-powered applications. Despite those limitations, the application roadmaps are still requiring a substantial increase of computing resources. In order to cope with those limitations, consumer electronics applications use the system-on-chip (SoC) approach where optimized accelerators supply DSP and GPP cores to meet the system constraints within a limited power envelope [3]. This approach has the main drawback to be very expensive and leads to complex systems that are difficult to validate.

Therefore, they are used for very high volumes applications like mobile phone handsets [4].

Professional electronics applications are characterized by low volumes and a high design count compared to consumer ones preventing the use of an expensive SoC approach. They rather use DSP or GPP combined with FPGA in order to increase the computation capabilities and meet requirements imposed by the roadmaps. Nevertheless, FPGA designs are static and lack flexibility compared to a pure DSP approach. However, roadmaps' evolution is shifting to multistandards or even software defined applications, requiring a virtualization layer to adapt or change their behavior dynamically. It is for example the case for the software-defined radio (SDR) application [5]. In order to cope with this flexibility issue, DPR of FPGA can be used [6, 7]. Indeed, it brings virtualization upon static hardware making it possible to handle hardware functional blocks like software components. Thank to this feature, SDR is announced by Xilinx to be the main target applications for DPR [8].

During the last decade, DPR has been widely studied in academia [9–11]. There are many works related to the component programming issue as well as case studies for possible applications [12–17]. Nevertheless, all those experiments are carried out for research purpose, and only a few of them take into account its impact for real applications. Indeed, despite all the researches done, it exists today very few real applications using DPR. Therefore, there is a lack of feedback on its use in real products providing relevant research issues to address in order to improve its applicability.

The contribution of this paper is to evaluate the interest and limitations when using DPR in real professional electronics applications, and to provide guidelines to improve its applicability. First, it makes a fair evaluation based on experiments made on a set of seven signal and image processing applications carried out in real conditions. Second, based on a precise analysis of the current flow for real usage, it identifies the missing elements for its use in professional electronics applications with highlights on the issues raised by SDR. Third, it identifies a set of advantages for using DPR in professional electronics applications. Fourth, it provides research directions in order to improve its usage. Fifth, it introduces a fast reconfiguration manager used by the experiments providing an 84-time improvement compared to the vendor solution. To our knowledge, it is the first reconfiguration manager working at this speed on the Virtex 4. Indeed in [18], the full speed has been tested only in Virtex II for an 8-bit ICAP.

This paper is based on the researches carried out as part of the RECOPS project that aims to study the use of DPR in military applications [19]. The evaluation is made on FPGA from the Virtex family from Xilinx Inc., Calif, USA. As they are the biggest matrixes available supporting DPR, they were selected as the FPGA platform for the project.

In the remainder of this paper, Section 2 explains the specificities of professional electronics applications. Section 3 exposes the DPR possibilities in the latest Xilinx Virtex components with its current design flow usable in real applications. Sections 4 and 5 give the main interests for using DPR in professional electronics applications and the issues for their implementation using DPR. Section 6 presents the experiments carried out. Section 7 details the main results obtained, and Section 8 discusses remaining issues raised by this technology based on the experiments, followed by the conclusion in Section 9.

2. PROFESSIONAL ELECTRONICS APPLICATIONS

Nowadays, the electronics market is mainly directed by consumer applications. They are characterized by a very short time to market, high volumes, autonomous and battery-powered applications, and have very few validations or qualification constraints.

However, professional electronics applications are quite different. They are sold in low volumes, and they take longer time for development. They are often maintained, or even updated, during their lifetime that can reach several decades. They need to address precise requirements and validation processes. Moreover, most of them are strongly coupled with other components inside complex systems. It is for example the case for the electronic subsystems in a plane, or for an airport radar part of an air traffic management system. They are also quite diverse. Indeed, a radar application, for example, performs in real time a very large FFT of up to tens of thousands of points and must fit in a small volume in order to be integrated into a plane, while a software-defined radio (SDR) requires a high level of virtualization, with very low power consumption for handheld devices.

For systems with a high level of safety, further validations and certifications are performed. It is the case for systems that may affect the life of human beings. The ED80/DO254 for hardware and ED12B/DO178B for software, in civil aviation, are among the most restrictive standards [20]. They impose to strictly validate and demonstrate that the application requirements are fulfilled in any possible situation, and identify the possible failure modes.

3. DYNAMIC PARTIAL RECONFIGURATION

Current FPGAs are composed by a user programmable logic plane, configured by an underlying memory plane. The logic plane holds fine-grain customizable logic made of LUT and interconnection resources, but also optimized macroblocks like SRAM arrays, DSP accelerators, clock tree managers, I/O modules, and so forth. For most of the components, the configuration memory is filled at startup by a bitstream through a reconfiguration interface. In order to do this, several interfaces are implemented; they are connected to the reconfiguration engine accessing the configuration memory. In Xilinx FPGA, the bitstream is a packet sequence, each packet containing a header and its data. The header holds commands and information for the configuration engine, so that very few external control signals are required during the reconfiguration process.

In some components of the high-end Virtex family from Xilinx, an internal interface internal configuration access port (ICAP) allows accessing the configuration engine directly from the user logic plane. Thanks to this interface, it

is possible for an application to perform self-reconfiguration while it is running. Moreover, according to the vendor, the internal implementation of the configuration ensures that (i) modifying a region of the component does not affect the configuration memory of other, unmodified, regions and (ii) when the content of the configuration memory is overwritten by the same content, its corresponding logic can operate normally without being affected. In order to perform a dynamic partial reconfiguration (DPR), a partial bitstream is written into the ICAP. Since it is made of raw configuration data highly component dependent, special tools are required for its generation.

3.1. New DPR possibilities in latest Xilinx Virtex component

The Virtex family encompasses the highest performances and biggest FPGA from Xilinx. Furthermore, DPR was introduced in this family with the Virtex II component. Its layout was organized in columns, defining an entire column of the component as the basic configuration granularity. Therefore, the DPR had to cope with severe hardware constraints that lessen its use in professional applications.

Since the Virtex 4, several improvements facilitate the use of DPR, making it a credible solution for some specific applications like software-defined radio (SDR). The improvements for DPR are on the layout architecture, the clock tree, and the ICAP. The component is still organized in columns but the partial reconfiguration granularity is reduced to a frame, which is here only a part of a column. Therefore, the partial reconfigurable region (PRR) can be almost any height and width. Clock regions are rectangular, allowing to be matched by a PRR. This leads to better timing performances and clock tree management. Moreover, the output frequency and phase shift of the digital clock manager (DCM) can be modified using DPR.

Finally, the width of the ICAP port is extended to 32 bits, and its speed is increased up to 100 MHz. This significantly speeds up the reconfiguration time, which is a main concern when using DPR. Indeed, the HW in the PRR cannot be used during the reconfiguration process. With all those improvements, the HW capabilities are far less a limitation for using DPR in real applications.

3.2. New tools dedicated for reconfiguration

The implementation of DPR in Virtex FPGA requires a nonstandard flow [21]. It is driven by constraints on the area and primitives location. It uses special dedicated resources, bus macro (BM) for communications, and uses the PlanAhead tool.

Area constraints make it possible to partition a design in a fixed part and a reconfigurable part. They are required to force a design to be placed & routed (P&R) in a predefined region composed of a fixed set of frames.

The BMs are slice-based prerouted elements made of simple LUT. They enable the communication between the fixed and the dynamic parts of the design. Indeed, since placement constraints cannot be directly applied to routing resources, they lock the starting or the ending point of the wires to ensure their correct connection after reconfiguration. They can be located on any edge of a PRR (left, right, top, bottom). They can be asynchronous or clocked, have a fixed direction (in or out); they are device dependent. Their main drawbacks are that they consume a significant amount of resources, and their automatic placement is not supported by the tools.

PlanAhead is a proprietary tool from Xilinx that is used to manage the DPR constraints and to drive the implementation process. It is a production tool with graphical interface allowing to handle constraints and area location of a PRR at component level. It provides a hierarchical design view at netlist level and a resource view at component level. Moreover, it makes a rough resources and bitstream size estimation, performs some design rule checking, and exports the results for implementation.

Finally, a specific partial flow is used to P&R the different portions of the design. It is based on a modular flow available in ISE [22]. It requires special patch updates that are currently available for selected customers and research centers. The flow is compatible with the embedded processors (PPC and MicroBlaze). At the end, it provides full and partial bitstreams that can be directly written into the ICAP. Note that with this flow, the modules do not have to follow the restrictive constraints of the XAPP290 [23] anymore.

3.3. The next generation

The latest high-end FPGA available in Xilinx is the Virtex 5. It is fabricated in a 65 nm technology, and it introduces innovations like diagonal routing capabilities and 6-entry LUT. It has new configuration ports and can manage multiple configurations. Regarding DPR, the component will support the features available on Virtex 4. Since DPR is today highly component dependent, specific tools and patches are required on top of the standard flow as for Virtex 4. However, it is not sure that the upcoming components will support DPR, and there are today no clear vendor roadmap regarding this technology.

4. ADVANTAGE FOR USING DPR IN PROFESSIONAL ELECTRONICS

The DPR allows using more hardware than that physically present in the FPGA. This can be used to reduce the size of the FPGA and its overall power consumption. This also permits to execute an algorithm with an optimized implementation depending on its parameters and data set. Furthermore, upon the usual speed and power research goals, DPR offers system-level advantages for professional electronics [19]. The advantages considered are the following.

(a) Task speed. By shifting the partitioning between hardware and software toward (faster) hardware. More functions can be implemented in hardware without being limited by the size of the component.

(b) Power reduction. By having less hardware instantiated and running, and by using a smaller FPGA.

(c) Survivability. By allowing selection and operation in a degraded mode when a part of the system is damaged. It is necessary for applications running in harsh environments where environmental conditions can exceed the normal operating range.

(d) Mission change. By allowing to configure the application for an entire mission without interrupting services. The real-time issue is not critical here. The application is configured for a long period. DPR provides an easy and safe way to strongly modify an entire system without having the complexity of implementing all the functionalities in one design. It is very useful and it facilitates the validation of applications interconnected in a complex system.

(e) Environment change. During operation, the application can be developed specifically for several environments and switch dynamically. Here, the real-time issue is critical.

(f) Adaptive algorithm change. By adapting dynamically an algorithm depending on the external conditions. It is a lower granularity than environment change.

(g) Online system test. A system in harsh environment can be damaged. For critical systems, it is necessary to know its level of functionality, for example, to decide to power on a redundant one.

(h) Hardware virtualization. By having more hardware available than that physically present in the FPGA. It allows to manage a set of hardware modules as a component library. It is used for example in SDR applications.

5. IMPLEMENTATION OF DPR IN PROFESSIONAL APPLICATIONS

In order to implement DPR in real applications, some issues need to be carefully handled. The most important ones are the design flow, the constraints brought by the use of DPR on the application board, and the minimum development required for its integration.

5.1. The design flow issue

The deep validation requirements or even the certification in critical applications is not only performed on the final product, it also imposes the use of a validation methodology during the whole development process. In this scope, the design flow is a key point. For severe requirements, it must be certified [20]. The validation imposes that at each step of the development, from the first specifications to the final tests, one must be able to verify that the application meets the requirements. For this, it is necessary to have precise simulations and modeling capabilities and to have efficient tools, at least for productivity reasons.

The standard design flow for a digital system is based on a top-down approach. The main steps are the following.

(a) System specification.

(b) Functional modeling and simulation.

(c) Hardware/software partitioning.

(d) Architecture definition.

(e) HW and SW development.

(f) Platform integration.

(g) Validation and qualification.

A complete design is rarely done in one conception pass. Indeed, in order to meet the application requirements, it is often necessary to make several iterations until reaching acceptable performances. The following paragraph focuses on the missing elements for DPR regarding this flow.

5.2. Identification of missing elements for the dynamic reconfiguration flow

For each step of the design flow, it is mandatory to have a simulation model of the DPR. For the first steps until the architecture definition, a model can be built with SystemC. For critical designs or for productivity issues, those steps are usually automated by tools. None of the existing tools supports DPR. Nevertheless, it is important to note that FPGAs are generally not well supported. It is for example difficult to find SystemC models for the complex hard or soft IP in FPGA like the PowerPC 405 or the MicroBlaze.

For the hardware development step, it is necessary to have a behavioral model of the ICAP, the BM, and the configuration process. Indeed, a hardware-level model is required to make functional simulation, ensure real-time constraints, and for debug. Without a behavioral model of the hardware, it is only possible to simulate the modules independently and verify the design when committing the application to the final on-board tests. Then, it is possible to verify a DPR design only when performing the platform integration. Therefore, if something fails, it is not possible to reproduce the problem by simulation. Moreover, debugging tools like ChipScope from Xilinx [24] are not working in a reconfigurable region. Without complete behavioral models, the validation and qualification step can only be done on the hardware platform. Serious difficulties occur, when it is necessary to take into account the design flow for validation.

The DPR modeling, using a powerful tool like SystemC [25–28], is mandatory in order to implement complex and challenging applications like SDR [29, 30]. Moreover, in order to maintain consistency in the model but also preserve good validation capabilities, the successive refinements of the SystemC model during the design flow should also support DPR.

5.3. The hardware platform and constraints

The board here is the printed circuit board hosting the FPGA. Its constraints are of two kinds. The first is that it requires a large amount of external FLASH memory, in order to permanently store the partial bitstreams, but also fast external memory like DDR in order to load bitstream and perform a reconfiguration at the maximum speed. The second kind of constraints is on the I/O position. For example, with DDR I/Os, the memory controller needs to be placed in front of the I/Os, constraining the placement of the PRR. Regarding those constraints, the standard development boards can easily be used to test DPR on a reference application.

5.4. Developments required for DPR evaluation

The developments required for evaluation are the design of a reconfiguration controller and a scheduler. The controller is based on an interface between the on-chip peripheral bus (OPB) available in the Virtex and the ICAP. It offers the flexibility of the standard bus that allows connecting any type of memory through a standard interface. Furthermore, the bus benefits from DMA services that provide sustained data rate to reach the maximum speed of the ICAP. Since the configuration manager is connected to a bus that can be connected to a MicroBlaze or a PowerPC, the scheduler can be implemented in software, which strongly reduces its development cost. Note that an OPB-ICAP interface is available from Xilinx, but not for all the components, and the full utilization of the ICAP is not supported on the latest one.

6. EXPERIMENTS

The DPR is evaluated in industrial conditions on a broad range of applications mainly in the field of defense applications. The approach is to use a common platform in order to easily share development experiences, make relevant comparisons, and demonstrate the platform flexibility offered by the FPGA implementation. The broad range of applications is required by the diversity of applications and constraints encountered in professional electronics.

6.1. The evaluation applications

The experiments are made on real applications or on a representative part of them. They have been chosen in order to have a representative set of challenging applications in the field of signal and image processing in professional electronics. They have not been specially chosen for a good match with DPR. Nevertheless, they all offer opportunities for improvement by using DPR compared to the usual solutions at least with several advantages discussed in Section 4. They are listed in the following with a highlight on their implementation challenge.

(a) *Portable device for remote-control video capture and transfer.* This application realizes image acquisition and transfer with remote control video using wireless communication. Several compression algorithms and several bandwidths have to be supported. The quality of the image and the bandwidth are both adapted depending on the context and the battery charge. All must fit within a minimal HW. The reconfigurable process performs axis motion control, image capture, and data transmission in the same reconfigurable region.

(b) *Blanking management for naval electronic counter measure/electronic support measure systems.* This application generates control signals avoiding interferences between the ECM and ESM systems. The system must operate with a high level of safety and support multiple context switches. Therefore, the design complexity must be as low as possible.

(c) *Real-time image processing unit for missile applications.* It applies several operators on an image with hard real-time constraints. The number of operators is potentially high, and they must be optimized in order to be very fast. The change of operators must be taken within a narrow time slot. The operators are implemented in a reconfigurable region.

(d) *Front end processing for airborne radar.* The maximum speed of the FPGA is used. The functionality must change very quickly, and the scenario depends dynamically on the context of the application. It tests the impact of reconfiguration on the use of high-speed I/O links and the front-end data processing under hard real-time and jitter constraints.

(e) *Short range radio modem.* It is used for local and private data communications and needs to support several data rates. Only the baseband of the modem needs to change but it is a very high computational task with only signal processing. The goal is to obtain the highest data rate in a given situation. The baseband functions of the modem are implemented in a reconfigurable region.

(f) *Software defined radio transmitter.* Only the modulation waveforms are implemented in a reconfigurable region. It tests the waveform parameterization and change. As the SDR application is highly reconfigurable, the maximum variability of the modules is required.

(g) *Software defined radio receiver.* This application focuses on all the upper layers of the SDR. Since they are usually implemented in software, the HW/SW interface with a high variability support is a main challenge when using dynamic hardware. It evaluates the hardware virtualization brought by the dynamic reconfiguration and demonstrates the partial reconfiguration of the FPGA by software from the SCA core framework. It has been published in [31].

Each demonstrator covers its own area of assessment, but together, they cover a large panel of activities and techniques. The tests are carried out to emphasize the different areas where the use of DPR can be interesting. It is not possible to detail them all here. Nevertheless, we will highlight features of the software defined radio application.

6.2. The software-defined radio experiments

The opportunity for SDR is to use the virtualization brought by DPR in order to be implemented in hardware, rather than in software. It is thus a very good candidate to take benefit of the dynamic reconfiguration. The experiments conducted with the SDR transmitter for multiband, multimode radio, are performed by switching between two different waveforms, the D8PSK and the 16QAM. The reconfiguration is controlled by the internal Power PC core of the FPGA. The mappings of data patterns to symbol, and symbols to in-phase and quadrature components, are done in the PRR. The pulse-shaping filter is also implemented in the PRR. The experiments with the receiver implement the SCA layer of the SDR in an FPGA platform using DPR.

6.3. The evaluation platform

The platform targeted for all the experiments is the ML410 development board from Xilinx. It hosts an XC4V-FX60 with sufficient external storage resources for storing the partial bitstreams for all the applications. No other specific board

development is needed by using the platform. Nevertheless, the boards ML403 and ML405 are also used for some experiments (due to the late availability of the ML410 board). The components used by all the experiments are the XC4V-FX12, FX20, FX60, and LX60.

6.4. The reconfiguration controller

For hard real-time applications that need to change frequently the reconfigurable module, a high bandwidth through the ICAP port is required. It is for example the case for the image processing application, which applies sequentially a set of reconfigurable functions for each image in a narrow time frame. The theoretical speed of the ICAP is 100 MHz, and its width can be programmed as an 8-bit or a 32 bits port allowing a bandwidth of 0.75 or 2.98 Gbit/s. The ICAP is usually connected to an OPB bus. This allows to use easily all the existing memory resources of the board by means of standard interfaces. The bus is driven by a microprocessor that can be a hard embedded PPC core or by a soft MicroBlaze core. Unfortunately, when the experiments were carried out, the interface provided by the vendor was able to operate at no more than 40 MHz in the 8-bit mode, and was designed for the Virtex II component. Those performances are not sufficient to meet the real-time constraints of the image processing application. Therefore, an improved version based on the OPB-ICAP interface for Virtex II was developed. The improvements allow working with the ICAP in the 32 bits mode at 100 MHz. Moreover, it also provides DMA services allowing to use the OPB bus with the required bandwidth.

Figure 1 shows a functional description of the reconfiguration controller implementing the OPB-ICAP interface. It is viewed as an OPB peripheral through an IPIF from the bus.

It is written in VHDL, and generic parameters allow selecting the memory size as well as the ICAP mode of 8 or 32 bits. A control and status register allows writing and reading bits to command and control the reconfiguration controller. It implements a DMA able to steer data from the bus at a sustained rate. The parameters are first written in two registers, a DMA_START_ADDR register that holds the base address of the data segment and a DMA_BURST_SIZE register that defines its size (max 1024 words). Then, a start bit in the CTRL/STATUS register is set to start the transfer. When it is finished, a status bit is set in the same register. The BRAM memory is used as a buffer. It was originally implemented in the Virtex II controller to convert the data from a 32 bits stream from the bus to an 8-bit stream toward the ICAP. Nevertheless, its instantiation is not mandatory. Indeed, the ICAP can be controlled at data word level by an enable signal, thus a single 32 bits register can be used to split the 32 bits data in four 8-bit words. When using the DMA, the buffer is bypassed. The reconfiguration controller supports read back, allowing to transfer the content of selected configuration frames to the bus. The DMA is not supported with this feature. Configured in the 32 bits mode, the reconfiguration manager occupies 973 slices and 1 BRAM. It corresponds to 3.7% of the mid-range XC4VLX60 Virtex 4 component.

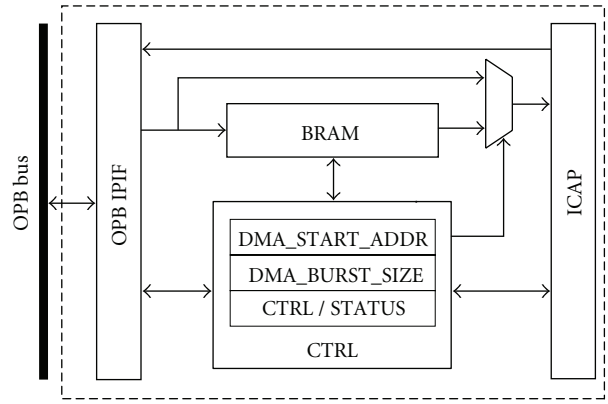


Figure 1: Reconfiguration controller.

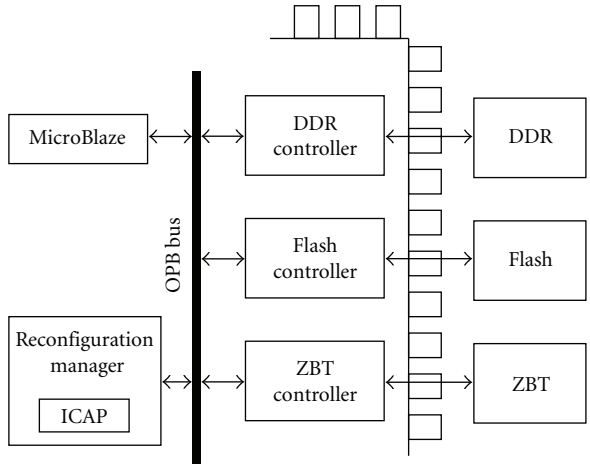


Figure 2: Memory organization.

The memory organization for the image processing application is given in Figure 2. The external flash is used to permanently store the bitstreams. The DDR and zero bus turnaround (ZBT) are used as fast memories. The bitstream size for the reconfigurable region is around 300 KB for the biggest. In order to obtain high-speed transfer, the partial bitstreams need to be stored in a fast external memory. Indeed, the permanent flash memory is very slow; it can only deliver 8 bits at 10 MHz. Therefore, the ZBT SRAM or the DDRs are used since they are the only memories on the board able to sustain the 100 MHz, 32 bits, throughputs. When the application starts, the bitstreams are first copied from the flash to the ZBT SRAM or the DDR depending on the speed requirements of the application.

7. RESULTS

The results are presented here from a system-level approach with highlights on the virtualization in the SDR experiments. Finally, measures of the performances of the ICAP are detailed.

The characteristics of the seven applications are given in Table 1. The first column gives the XC4V component

Table 1: Applications characteristics.

	Component	N. reconf. region	PRR size [%]	Reconf. time [ms]	Bitstream size [KB]	Real-time constr.
(a) Im. acq.	FX60	1	16	2.2	330	20 ms
(b) Naval	FX12	1	28	195	90	1 s
(c) Im. proc.	FX60	1	13	1.3	17	5 ms
(d) Radar	FX20	1	33	0.47	166	1 ms
(e) Modem	FX12	1	32	3.5	241	1 s
(f) SDR Tx	FX12	1	9	31.6	38	1 s
(g) SDR Rx	FX12	2	12	15.5	46	1 s

used, then the number of reconfigurable regions and the relative size of the reconfigurable regions counted in slices. The fourth column gives the reconfiguration time followed by the maximum size of all the partial bitstreams targeted to a region. Note that not all the applications use the fast reconfiguration manager. Moreover, they can use 8-bit or 32 bits interfaces, and their bus may be interrupted by other tasks. Therefore, the reconfiguration time summarizes the application performances for doing a reconfiguration while respecting its real-time constraints. The last column gives the real-time constraint for performing a reconfiguration.

7.1. System-level assessment

Most of the advantages listed in Section 4 are tested by the experiences. The results are given in Table 3. For each advantage, a rating between −2 and +2 evaluates its benefit in the application. An empty evaluation means that the advantage was not tested during the experiments by the application. A rating between parenthesis means that the advantage was not directly evaluated during the experiments but rather estimated. Results show that the most interesting advantages are virtualization, environment, and mission change. They allow changing the functionality of a system under weak real-time constraints. Moreover, in all the applications using this feature, the architecture of the system remains the same while an instance of a particular function is changed. This leads to add more functionalities without increasing the complexity of the data path of the application. The telecommunication applications change their waveforms simply by reconfiguring a region. The adaptive algorithm change is obtained by the virtualization advantage. It was only lightly tested by the experiments, since this feature is today rarely used in the applications due to its novelty. There was no increase of the tasks' speed of the application, except for the SDR applications, where more accelerators can be instantiated in the FPGA instead of being executed on a DSP. For the other applications, the accelerators are always instantiated in hardware. The power reduction is then obtained by using a smaller FPGA when using DPR, reducing the leakage and the dynamic power consumption generated by unused hardware. The survivability possibility of the application is lower for the SDR application because the DPR brings new failure modes to the system. Since they are not yet well characterized, they have to be cautiously handled. The online system test is experienced by the radar and the image processing application where every reconfigurable module has its own communication lines. Therefore, it is possible for the application to monitor the good operation of the functions and the reconfiguration process.

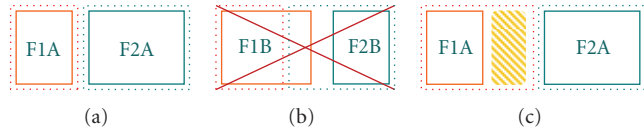


Figure 3: PRR reservation.

Most the evaluated features bring the promised advantages to the application. Nevertheless, a drawback comes from the size of the design. Indeed, where few virtualization is used, there is a significant size overhead when using DPR compared to the static implementation.

7.2. The hardware virtualization

Specific hardware virtualization tests are carried out in the SDR receiver application. The experiments compare implementations of crypto algorithms for an SDR application. The radio has two channels implemented and running at the same time, one requiring encryption and the other decryption. The algorithms can be changed dynamically and can be different for each channel. Three functions are considered, PLAIN, SCRAMBLER, and DES. Plain returns the clear message (no encryption), scrambler performs basic bit scrambling, and DES is a standard symmetric algorithm. The implementation is done with two PRR and a static region. Since the PRRs are statically defined, they need to be sufficiently big to fit the biggest module, and a significant amount of resources can be wasted. This problem is illustrated in Figure 3 for two PRRs in three situations. In Figure 3(a), the PRRs are dimensioned to hold F1A and F2A. When reconfiguring with F1B and F2B in Figure 3(b), the place left by F2B cannot be used by F1B. Therefore, the PRR reserved for F1 in Figure 3(c) needs to be maximal.

The resources consumption for the three modules and the static part of the design for an XC4VFX12 component is presented in Table 2. "The static + max PRR" is the DPR version of the design; the static part and two PRRs are implemented. In the "static + all modules" version, all modules are implemented statically. The resource gain shown in Table 2 is not very important when using DPR.

Table 2: Resource consumption.

	Plain		Scrambler		DES		Static	Static + max PRR		Static + all modules		Avail
	Encr	Decr	Encr	Decr	Encr	Decr						
Slice	53	53	69	69	379	379	2674	3432	63%	3676	67%	5472
RAMB 16	1	1	1	1	8	8	17	33	92%	37	103%	36
DSP48	0	0	0	0	0	0	0	0	0%	0	0%	32
PPC405	0	0	0	0	0	0	1	1	100%	1	100%	1
Bitstream (KB)	31,7	31,7	33,7	33,4	46,3	46,1	—	—	—	—	—	—

Table 3: Advantages evaluation.

	Task speed	Power reduction	Survivability	Mission change	Environment change	Algorithm change	Online system test	Virtualization
(a) Im. acq.		+1		(0)				+2
(b) Naval				0		+1		
(c) Im. proc.	0	+1	(0)	(0)	(+1)	(+1)	(+1)	+1
(d) Radar	0			+2	+2		+2	
(e) Modem	0	+1		+2	+2	+1		+2
(f) SDR Tx	+1	(+1)		+2	+2			+2
(g) SDR Rx	(+1)	(+1)	−2	(0)	+1	+1		+2

This is due to the choice of the algorithms implemented. Indeed, the plain and scrambler are very small compared to the DES that directs the size of the PRR. Nevertheless, even for that, the RAMB 16 consumption for the static design overuses the resources available in the component by 103%. Thus, without DPR, it cannot fit into the component. Furthermore, there are only three algorithms supported here. For applications with a reasonable level of flexibility as it is the case for SDR, many more algorithms need to be supported, like 3DES, AES, and so forth. For those applications, it is not possible to use a static design.

7.3. The ICAP performances

For industrial and security reasons, the bitstreams need to be encrypted. In the components used for the experiments, the ICAP is not usable when the bitstream encryption is enabled. The ICAP throughput is an issue for the designs using DPR in hard real-time applications, and it is carefully measured by the experiments. The ICAP was successfully tested at 100 MHz in 32- and 8-bit modes. The write and read-back modes were tested. Note that for reaching this speed, we found that the data needs to be sent on the falling edge of the ICAP clock. This is maybe due to a clock skew between the user logic and the ICAP. Table 4 summarizes the performances measured by connecting the ICAP to a basic GPIO, with the OPB-ICAP provided by Xilinx for Virtex II and with our reconfiguration manager. The partial bitstreams are stored in a DDR with a clock at 100 MHz. The Xilinx interface is always in 8-bit mode. The maximum theoretical bandwidth of the ICAP is 2.98 Gbit/s. It is not reached due to DMA overhead. Nevertheless, our custom OPB-ICAP is 84 times faster, compared to the vendor's module. 4 times are due to the data word size, 2.5 times due to frequency, and 8.4 times to the DMA accesses.

8. DISCUSSION

The development of applications for the experiments shows that designs using DPR must handle specificities like communication interfaces between modules, the execution scenarios, and the bitstream handling. Even if DPR has many open technology issues, all the applications tested can potentially gain from its use. For this, there are still many researches to do in order to bring DPR and dynamic HW at sufficient maturity level.

8.1. Communication interface

A high-level interface must be designed to handle the communication between the fixed part and the dynamic modules. This interface is similar to the one separating components in a static design but has to be uniform for all the modules targeted to be instantiated in a given PRR. It provides an abstraction of the accesses to dynamic modules allowing their use, as they were statically instantiated.

During the development of the experiments, the opportunity to use a dedicated interface for dynamic reconfiguration appears clearly. Nevertheless, regarding the number of application tested and their specific needs in terms of connections, latencies and throughputs, it was not possible to find out a solution to this problem. Therefore, the classical methodology for modeling and implementing interfaces was used.

The SDR application requires a communication transparency between the functional blocks of an application. This

Table 4: ICAP performances.

ICAPmode	GPIO	Xilinx OPB-ICAP	Custom OPB-ICAP	Theor.
8 bits	—	32.4 Mbit/s	0.56 Gbit/s	0.75 Gbit/s
32 bits	22.2 Mbit/s	—	2.8 Gbit/s	2.98 Gbit/s

requires the support of innovative middleware services for reconfigurable platforms.

8.2. Execution management

One of the main challenges when using dynamic reconfiguration is the management of the dynamic modules. This includes spatial and temporal management. Moreover, with hard real-time constraints, the bitstream is loaded from fast memory locations that have also to be managed.

In all the experiments carried out, the management is done at the operating system level. Nevertheless, since this solution adds substantial complexity to the operating system, the ideal solution would provide all those services transparently to the application.

The latency for loading a bitstream is important regarding the operating frequency of the component. A real-time application must have the control of the latencies to avoid undetermined behavior. Therefore, the reconfiguration management must be carried out by a scheduler controlled by the operating system of the application, which needs to access all the relevant parameters, like the reconfiguration latencies and status.

For preemptive scheduling, the context switch must be supported in the PRR. Therefore, the state hold in the PRR must be saved and restored.

8.3. Bitstream handling

The size of a complete bitstream reaches several megabytes for a high-end component, and for a partial bitstream, it is of the order of hundreds of kilobytes. For experiments from hard real-time applications, the reconfiguration speed can be a key factor for improving the performances by using DPR. This leads to use a high-performance reconfiguration manager using DMA to fetch the bitstream from a fast external memory at the maximum throughput. The configuration manager is controlled by a processor running the scheduler. This approach is used to release the pressure on the processor when transferring bitstream at word level. Moreover, a dedicated bus connecting memories release the processor bus allowing its use for other tasks. No other high-level abstraction for handling the bitstream is used by the experiments. This basic low-level solution has the disadvantage of mixing the bitstream handling with the application, which strongly complexifies the design.

8.4. Technology issues

There is today a single vendor providing large FPGA matrices supporting DPR. This causes several issues for its use in professional electronics applications. Upon the commercial and strategic ones, it lessens its use for safety applications. Indeed, a good level of safety is often reached by using redundant systems with the same specifications but designed by separate teams using different components coming from different vendors.

There is also no clear roadmap, from the vendor, regarding the future support of this technology in the upcoming high-end components. Therefore, this increases the risk of using it in real professional applications. Indeed, after their first release most of those applications must be maintained and upgraded during decades. On the contrary, consumer electronics products last no more than few years with barely any updates. The maintenance is performed to correct bugs, adapting the products with new client specifications or replacing old and damaged components. For this, the original (or equivalent) components need to be used to avoid going again into the heavy validation process. In this scope, the obsolescence and future compatibility of the components are a key point that needs to be addressed by the vendor.

The design flow provided is weak and experimental. It is not possible to model DPR during all the steps of an application development. SystemC can be used for first high-level steps but then it is difficult to use other tools, like for HW/SW partitioning, since DPR is not integrated by the tool vendors. For the low-level steps, there is a lack of behavioral and HW model, allowing to simulate and validate the designs before the platform integration into the final board. All those drawbacks are due to the novelty of this technology and to the fact that it brings the completely new concept of dynamic HW. Indeed, all the HW models used so far are static models, and none of the HDL languages used for description implement dynamic HW. Moreover, DPR is transverse to all abstraction layers of an application, from the reconfiguration port and wires at bit level to the scheduler at operating system level. Therefore, there are many research opportunities on dynamic HW at all abstraction levels. For its use in real applications, an IP library with components like reconfiguration manager, scheduler, and communication interface is necessary to abstract the additional complexity and minimize the increase of development efforts. Furthermore, since the applications become quite complex as the design effort for implementing and validate SoCs in large components, the most important challenge for the DPR is certainly its transparency. The designer should not worry about the reconfiguration details.

8.5. Application benefits

Many benefits can be taken from DPR in real professional electronics applications. Indeed, upon the criteria listed in Section 4 and directly evaluated by the experiments, DPR

brings a new dimension of flexibility on the HW allowing its virtualization like with SW library. All experiments are taken from highly constrained applications, often integrated in complex systems. Therefore, the flexibility brought by the DPR enables to ease their development, integration, maintenance, or evolution. The use of very large components as high-end FPGA is enabled thanks to the system-on-chip approach based on IP assembling and reuse. In this scope, DPR brought the flexibility to modify IP dynamically.

The improved reconfiguration manager used by some experiments makes possible to use DPR directly in constrained hard real-time algorithm and applications. Nevertheless, using the maximum reconfiguration speed leads to a huge memory traffic causing internal and external bus congestion and therefore to power consumption. Some experiments mitigate bus congestion by using dedicated bus connecting the reconfiguration controller to memories. However, the power consumption is still important.

When the dynamic modules are changed too frequently, the power consumption increases and a time overhead appears because the PRR cannot be used during a reconfiguration process. This is the case when the reconfiguration is performed in a hot loop body of the algorithm. The most significant advantage pointed out by the experiments is the use of DPR for hardware virtualization when the reconfiguration is not part of the algorithm but is rather used to select the more suitable one in a given situation, leading to adaptive systems. The experiments show that adaptive algorithms are not frequent. Indeed, the algorithm designers are used to consider a static implementation. Here again, there is a need for a model of dynamic machine for algorithms designers.

Finally, the classical area, speed, and power metrics are not necessary improved when using DPR. The area is always improved when using DPR for virtualization where a significant part of dynamic HW resides in large- and low-cost external flash memory. Hence, the static power consumption is reduced compared to a static solution using a bigger component. However, the dynamic power consumption and speed improvements depend on the application.

9. CONCLUSION

During the last decade, DPR has already been widely studied as a research topic. However, it exists today very few real applications using it. This paper evaluates the use of DPR for real signal and image processing applications in professional electronics. An emphasis is put on the SDR that is the main target application for DPR as announced by Xilinx. Moreover, it provides relevant feedback in order to improve its applicability. For this, it is based on a set of seven real applications in signal and image processing with experiments carried out under real conditions. As the design flow is a key point for professional electronics applications, it makes a precise analysis of the current flow available and highlights its missing elements. It shows how SystemC can be used to supplement the first steps of the flow. Those steps are crucial for the modeling and validations of the SDR application.

Upon the classical area, speed, and power metrics, professional electronics applications can also benefit from DPR for online system test, mission/environment changes, algorithms adaptive changes, survivability, and hardware virtualization. The paper provides research directions to improve the applicability of DPR, the modeling of dynamic hardware, the definition of a generic reconfiguration interface, the state handling with preemptive scheduling, and the transparent bitstream handling by the application.

Finally, it shows how the reconfiguration interface provided by the vendor can be improved by using DMA and resolving a clock jitter issue. Thanks to those optimizations; it is possible to achieve a reconfiguration speed of 2.8 Gbits/sec close to the theoretical limit.

ACKNOWLEDGMENTS

This research is part of the RECOPS project. It is a EUROPA contract from the European Defense Agency, funded by French and Italian Ministries of Defense and the Walloon Region in Belgium. B. Rousseau holds an F.R.S.-FNRS fellowship (Belgian Fund for Scientific Research).

REFERENCES

[1] K. De Bosschere, W. Luk, X. Martorell, et al., "High-performance embedded architecture and compilation roadmap," in *Transactions on High-Performance Embedded Architectures and Compilers I*, vol. 4050 of *Lecture Notes in Computer Science*, pp. 5–29, Springer, Berlin, Germany, 2007.

[2] M. J. Flynn and P. Hung, "Microprocessor design issues: thoughts on the road ahead," *IEEE Micro*, vol. 25, no. 3, pp. 16–31, 2005.

[3] J. Song, T. Shepherd, M. Chau, et al., "A low power open multimedia application platform for 3G wireless," in *Proceedings of IEEE International Symposium on Systems-on Chip (SOC '03)*, pp. 377–380, Tampere, Finland, November 2003.

[4] M. Hammes, C. Kranz, D. Seippel, J. Kissing, and A. Leyk, "Evolution on SoC integration: GSM baseband-radio in 0.13 μm CMOS extended by fully integrated power management unit," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 236–245, 2008.

[5] A. C. Tribble, "The software defined radio: fact and fiction," in *Proceedings of IEEE Radio and Wireless Symposium*, pp. 5–8, Orlando, Fla, USA, January 2008.

[6] http://www.xilinx.com/prs_rls/dsp/0626_sdr.htm.

[7] J.-P. Delahaye, J. Palicot, C. Moy, and P. Leray, "Partial reconfiguration of FPGAs for dynamical reconfiguration of a software radio platform," in *Proceedings of the 16th IST Mobile and Wireless Communications Summit*, pp. 1–5, Budapest, Hungary, July 2007.

[8] C. Kao, "Benefits of partial reconfiguration," *Xilinx Xcell Journal*, vol. 2005, no. 55, pp. 65–67, 2005.

[9] N. McKay, T. Melham, and K. W. Susanto, "Dynamic specialisation of XC6200 FPGAs by partial evaluation," in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 308–309, Napa Valley, Calif, USA, April 1998.

[10] P. Merino, M. Jacome, and J. C. Lopez, "A methodology for task based partitioning and scheduling of dynamically reconfigurable systems," in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 324–325, Napa Valley, Calif, USA, April 1998.

[11] S. R. Park and W. Burleson, "Reconfiguration for power saving in real-time motion estimation," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '98)*, vol. 5, pp. 3037–3040, Seattler, Wash, USA, May 1998.

[12] J. Becker, A. Donlin, and M. Hübner, "New tool support and architectures in adaptive reconfigurable computing," in *Proceedings of IFIP International Conference on Very Large Scale Integration (VLSI-SoC '07)*, pp. 134–139, Atlanta, Ga, USA, October 2007.

[13] M. Alderighi, F. Casini, S. D'Angelo, M. Mancini, S. Pastore, and G. R. Sechi, "Evaluation of single event upset mitigation schemes for SRAM based FPGAs using the FLIPPER fault injection platform," in *Proceedings of the 22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT '07)*, pp. 105–113, Rome, Italy, September 2007.

[14] H. Wang, J.-P. Delahaye, P. Leray, and J. Palicot, "Managing dynamic reconfiguration on MIMO Decoder," in *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–8, Long Beach, Calif, USA, March 2007.

[15] P. Sedcole, B. Blodget, J. Anderson, P. Lysaghi, and T. Becker, "Modular partial reconfigurable in Virtex FPGAs," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL '05)*, pp. 211–216, Tampere, Finland, August 2005.

[16] K. Wu and J. Madsen, "Run-time dynamic reconfiguration: a reality check based on FPGA architectures from Xilinx," in *Proceedings of the 23rd IEEE Norchip Conference (NORCHP '05)*, pp. 192–195, Oulu, Finland, November 2005.

[17] T. Pionteck, C. Albrecht, and R. Koch, "A dynamically reconfigurable packet-switched network-on-chip," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '06)*, vol. 1, pp. 1–4, Munich, Germany, March 2006.

[18] C. Claus, F. H. Müller, J. Zeppenfeld, and W. Stechele, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS '07)*, pp. 1–7, Long Beach, Calif, USA, March 2007.

[19] P. Manet, D. Maufroid, L. Tosi, et al., "RECOPS: reconfiguring programmable devices for military hardware electronics," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '07)*, pp. 1–6, Nice, France, April 2007.

[20] RTCA DO-254/EUROCAE ED-80, "Design assurance guidance for airborne electronic hardware," April 2000.

[21] P. Lysaght, B. Blodget, J. Mason, J. Young, and B. Bridgford, "Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs," in *Proceedings of International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 1–6, Madrid, Spain, August 2006.

[22] http://www.xilinx.com/ise/.

[23] Xilinx XAPP290, "Two flows for partial reconfiguration: module based or difference based," September 2004.

[24] http://www.xilinx.com/ise/optional_prod/cspro.htm.

[25] F. Ghenassia, *Transaction-Level Modeling with SystemC*, Springer, New York, NY, USA, 2005.

[26] Y. Qu, K. Tiensyrja, and J.-P. Soininen, "SystemC-based design methodology for reconfigurable system-on-chip," in *Proceedings of the 8th Euromicro Conference on Digital System Design (DSD '05)*, pp. 364–371, Porto, Portugal, August-September 2005.

[27] A. Schallenberg, W. Nebel, and F. Oppenheimer, "OSSS+R: modelling and simulating self-reconfigurable systems," in *Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL '06)*, pp. 1–6, Madrid, Spain, August 2006.

[28] A. Herrholz, F. Oppenheimer, P. A. Hartmann, et al., "The ANDRES project: analysis and design of run-time reconfigurable, heterogeneous systems," in *Proceedings of the 17th International Conference on Field Programmable Logic and Applications (FPL '07)*, pp. 396–401, Amsterdam, The Netherlands, 2007.

[29] G. Gailliard, E. Nicollet, M. Sarlotte, and F. Verdier, "Transaction level modelling of SCA compliant software defined radio waveforms and platforms PIM/PSM," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '07)*, pp. 1–6, Nice, France, April 2007.

[30] G. Gailliard, B. Mercier, M. Sarlotte, B. Candaele, and F. Verdier, "Towards a systemC TLM based methodology for platform design and IP reuse: application to software defined radio," in *Proceedings of the 2nd International European Workshop on Reconfigurable Communication-Centric Systems-on (RECOSOC '06)*, pp. 131–138, Montpellier, France, July 2006.

[31] M. Sarlotte, B. Counil, P. Gelineau, R. Chau, and D. Maufroid, "Partial reconfiguration concept in a SCA approach," in *Software Defined Radio Technical Conference and Product Exposition (SDR '07)*, Denver, Colo, USA, November 2007.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 349465, 17 pages
doi:10.1155/2008/349465

Research Article

Using High-Level RTOS Models for HW/SW Embedded Architecture Exploration: Case Study on Mobile Robotic Vision

François Verdier, Benoît Miramond, Mickaël Maillard, Emmanuel Huck, and Thomas Lefebvre

ETIS Laboratory, CNRS UMR 8051/University of Cergy-Pontoise/ENSEA, 6 avenue du Ponceau, 95000 Cergy-Pontoise Cedex, France

Correspondence should be addressed to François Verdier, verdier@ensea.fr

Received 1 March 2008; Revised 19 June 2008; Accepted 22 July 2008

Recommended by Guy Gogniat

We are interested in the design of a system-on-chip implementing the vision system of a mobile robot. Following a biologically inspired approach, this vision architecture belongs to a larger sensorimotor loop. This regulation loop both creates and exploits dynamics properties to achieve a wide variety of target tracking and navigation objectives. Such a system is representative of numerous flexible and dynamic applications which are more and more encountered in embedded systems. In order to deal with all of the dynamic aspects of these applications, it appears necessary to embed a dedicated real-time operating system on the chip. The presence of this on-chip custom executive layer constitutes a major scientific obstacle in the traditional hardware and software design flows. Classical exploration and simulation tools are particularly inappropriate in this case. We detail in this paper the specific mechanisms necessary to build a high-level model of an embedded custom operating system able to manage such a real-time but flexible application. We also describe our executable RTOS model written in SystemC allowing an early simulation of our application on top of its specific scheduling layer. Based on this model, a methodology is discussed and results are given on the exploration and validation of a distributed platform adapted to this vision system.

1. INTRODUCTION

Today, real-time visual scene processing represents one of the major problem for autonomous robots. Lots of robot behaviours are based on this processing: navigation, object recognition and manipulation, target tracking, and even social interactions between human and robots. Currently, visual systems require large computing capabilities making them hard to embed. Indeed, most of such heavy vision tasks are often performed by distant host computers via network connections.

However, for several years, new approaches developed for visual processing have been proposed. The visual system is not considered isolated anymore but as part of an architecture integrated in its environment. They take into account more and more parameters related to the dynamic properties of the systems they belong to (see active vision [1]). These new visual processing algorithms strongly depend on the dynamics of interactions between the whole system and its environment by continuous feedbacks regulating even the low-level visual stages (see, e.g., the attentional mechanisms in biological systems).

The studied application consists in a subset of a cognitive system allowing a robot equipped with a charge-coupled device (CCD) camera to navigate and to perceive objects. The global architecture in which the visual system is integrated is biologically inspired and based on the interactions between the processing of the visual flow and the robot movements (Per-Ac architecture [2]). The learning of the sensorimotor associations allows the system to regulate its dynamics [3] and, therefore, navigate, recognise objects, or create a visual compass [4].

In this paper, we aim at designing an embedded visual processing system in the form of a single chip that could be used for building the CCD-based smart camera of our robot. On one hand, the embedded processing part should be flexible enough in order to allow a variety of navigation missions. It also needs to adapt to evolutive constraints due to the global system intrinsic dynamics (see Figure 1). On the other hand, the architecture should also provide intensive computation capabilities to deal with low-level

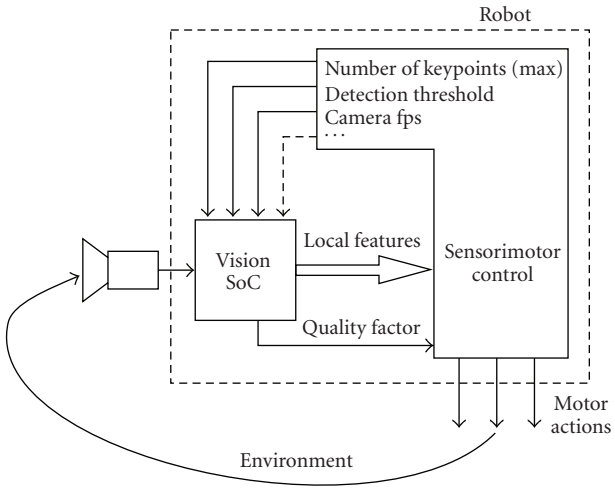


Figure 1: The dedicated SoC architecture used in a robot global sensorimotor loop.

image processing. One of the solutions to provide efficiency and flexibility may consist in implementing the application both in hardware and software. All regular and intensive computation tasks should be implemented in pipelined hardware modules and all irregular and control tasks should be mapped onto classical processing elements. These design choices implicitly lead to a heterogeneous and probably distributed application composed of multiple computation tasks. Such an application will be managed by a dedicated real-time operating system (RTOS). In our case, the use of an RTOS becomes essential in a domain where applications exhibit dynamic and adaptive behaviours.

1.1. Related works

When designers are faced with an SoC implementation of a new application, the hardware and software parts of the SoC must be designed according to the computing properties of the application. Precisely, our vision application has a specific dynamical real-time behaviour. As a result, we advocate for the use of high-level system models to early validate architecture alternatives and the corresponding real-time behaviours according to the constraints. Practically, the definition of a custom SoC architecture following a high-level design methodology is based on [5] the following:

(i) high-level modelling of the hardware and software components of the SoC,

(ii) exploration of the design space,

(iii) validation of the selected design solutions.

Some system level methodologies and tools already help to design SoC architectures composed of hardware and software processing elements. These methodologies are often based on system level design languages (SLDLs) such as SystemC [6] or SpecC [7] and are presented in the following. Unfortunately, only few of the proposed techniques include the definition of a dedicated RTOS into their design flow. Indeed, hardware/software codesign now includes the problem of the RTOS design which is considered as the main component responsible for the control of the global system.

For example, some works have proposed efficient solutions for the automatic synthesis of distributed implementations of dataflow-dominated applications under real-time constraints. For software-based applications, the SynDex tool [8] allows, for example, to describe an application as a hierarchical dataflow graph (and the corresponding code of elementary blocks) and to automatically generate a fully static software implementation on a heterogeneous multiprocessor architecture. With this kind of scheduling solution, the execution order of the elementary blocks must be defined at compile time. It thus cannot be used efficiently in application domains where potential parallelism can change dynamically according to input data (except if a worst-case architectural dimensioning is done). Unfortunately, visual computations such as the one found in autonomous robotics cannot always be predetermined. The degree of parallelism can vary during its execution. Static scheduling solutions are thus inappropriate in a real-time context and a dynamical scheduling must be defined online by a real-time operating system.

In the codesign context, many automatic HW/SW synthesis methods have been proposed [9, 10] but do not provide anymore dynamic behaviours management. Some codesign methods such as COSYN [11] deal with multitask application specifications but in the same static context. Realistic embedded systems design need methods to rapidly define RTOS in an application-specific way. This need has been identified by recent research works. First of all, in the context of high-level design methods, solutions have been proposed to model an RTOS at a high level. Gerstlauer et al. in [12] have recently initiated this research activity by presenting an RTOS model on top of the SpecC SLDL. As SystemC or SpecC SLDLs allow timed simulations of written models, the work in [12] takes advantage of SpecC primitives to explicitly model dynamic behaviour of multitasks systems at high levels of abstraction.

SoCoS in [13] is a C++ library for system-level design providing the user automatic linking with operating system (OS) services. The main difference with [12] is that SoCoS requires its own proprietary simulation engine.

In [14], Le Moigne et al. describe a SystemC model of a multitask RTOS. This model is a part of the Cofluent tool [15] which allows timing parametrisation and validation of the RTOS model by configuring context load, context save, and scheduling duration.

After modelling and simulating high-level RTOS representation, another problem addressed by Gauthier et al. is the automatic generation of RTOS code. In [16], they present a method of automatic generation of operating systems for a target processor. This method finds OS services required in the code of the application SW and generates the corresponding code deduced from dependencies between services in an OS service library.

Putting aside the works of [12], none of the existing RTOS modelling approaches deals with creation of dynamic processes. However, as it will be explained in this paper, this

property is needed to early validate the real-time behaviour of our application. Hence, our method addresses this design challenge by introducing a high-level RTOS model for custom SoC design. Working at a high level of abstraction allows the designer to jointly explore the RTOS architecture in terms of custom services adapted to the application and the parallel SoC architecture. Both dynamic behaviour control and embedded constraints satisfaction problems can thus be solved by a single approach.

Contributions of this work consist in proposing a SystemC functional accurate dynamical RTOS model allowing a high-level simulation of a distributed architecture. This simulation is done at a *Service Accurate* level in the sense that allows functional and timed verification without the need of modelling explicitly processing resources. By working at this level of abstraction, an early exploration of the architecture dimensioning and the validation of application real-time constraints are feasible.

1.2. Paper organisation

The rest of the paper is organised as follows. Section 2 presents our robotic vision application in more details and stresses its dynamical properties. Section 3 describes the proposed RTOS modelling approach based on the system-level design language SystemC. Results are also given on the corresponding dynamical implementation of the vision application.

We then discuss in Section 4 how our simple RTOS model can be used for building a parallel and distributed multiprocessor architecture coupled with dedicated hardware accelerators. In Section 5, we give the results of the proposed Hw/Sw exploration process permitted to define the multiprocessor system-on-chip (MPSoC) platform dedicated to our vision application. Finally, we conclude and discuss some perspectives in Section 6.

2. A VISION APPLICATION FOR ROBOT PERCEPTION AND NAVIGATION

In the following, we first describe the considered vision application. This application mainly consists in applying classical filtering, subsampling, and extraction operators, and it can be considered as a pure data flow process. However, we will detail in Section 2.2 how this application is inserted in a global dynamic and adaptive regulation loop (see Figure 1). We will then show why that forbids the use of classical implementation flows in this specific context.

2.1. Application description

The current visual system here is close to the one used by Leprêtre et al. in [17] and integrate a multiscale approach to extract the visual primitives. In doing so, it also allows a wider range of applications. Roughly the visual system provides a local characterisation of the keypoints detected on the image flow of an 8-bit gray-scale CCD camera (382×286 pixels). This local characterisation feeds a neural network which can associate motor actions with visual information: this neural network can learn, for example, the direction of a displacement of the robot as a function of the scene recognition. The studied visual system can be divided into two main modules:

(i) a multiscale mechanism for characteristic points extraction (keypoints detection),

(ii) a mechanism supplying a local feature of each keypoint.

2.1.1. The multiscale keypoints detection

The multiresolution approach is now well known in the vision community. A wide variety of keypoints detectors based on multiresolution mechanisms can be found in the literature. Amongst them are the Lindeberg interest point detector [18], the Lowe detector [19]—based on local maxima of the image filtered by difference of Gaussians (DoGs)—or the Mikolajczyk detector [20], where keypoints correspond to those provided by the computation of a 2D Harris function and fit local maxima of the Laplacian over scales.

The visual system described here is psychophysically inspired in the sense that it takes into account the work done on the Müller-Lyer illusions in [21]. The used detector extracts points in the neighbourhood of the keypoints, which are sharp corners of the robot's visual environment. More precisely, the keypoints correspond to the local maxima of the gradient magnitude image filtered by DoGs (Figure 2). Moreover, the detector remains computationally reasonable and is characterised by a good stability. It also automatically sorts the keypoint lists. The gradient magnitude Grad is computed by the following equation (where $I(x, y)$ corresponds to the pixel magnitude at coordinates (x, y)):

$$\mathrm{Grad}(x, y) = \sqrt{\frac{\left(I(x+1, y) - I(x-1, y)\right)^2 + \left(I(x, y+1) - I(x, y-1)\right)^2}{2}}. \quad (1)$$

Keypoints are detected in a sampled scale space based on an image pyramid. Pyramids are used in multiresolution methods to avoid expensive computations due to filtering operations. The algorithm used to construct the pyramid is detailed and evaluated in [22]. The pyramid is based on successive image filtering with 2D Gaussian kernels ($G_\sigma(x, y)$) normalised by a factor S:

$$G_\sigma(x, y) = \frac{e^{(-(x^2+y^2)/2\sigma^2)}}{S}. \quad (2)$$

These operations achieve successive smoothing of the images. Two successive smoothing are carried out by two Gaussian kernels with variance $\sigma^2 = 1$ and $\sigma^2 = 2$. The scale factor doubles (achievement of an octave) and thus the image is decimated by a factor of two without loss of information. The same Gaussian kernels can be reused to continue the pyramid construction. Interestingly, the kernel sizes remain small (half-width and half-height of $3 \times \sigma$) allowing a fast

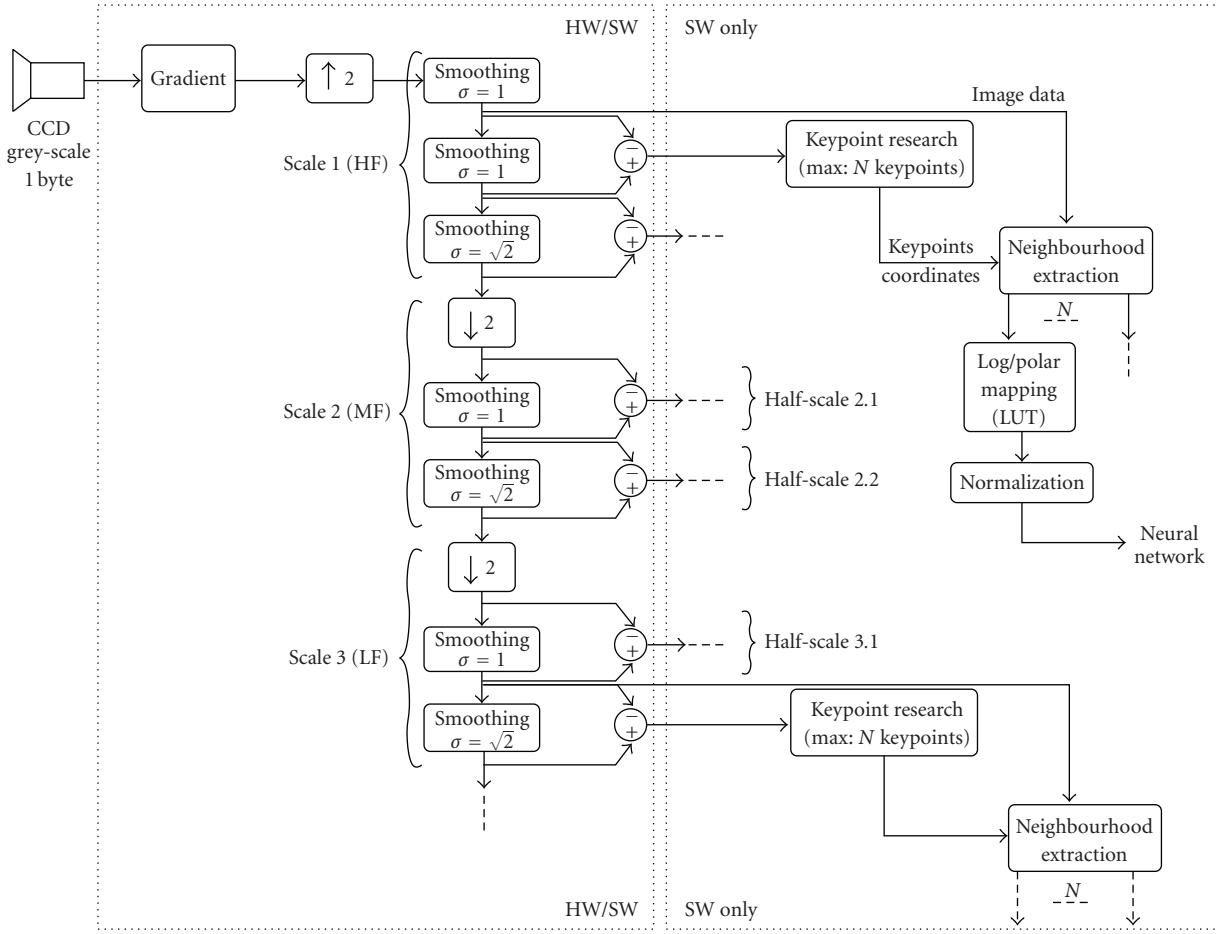


Figure 2: Global architecture of the algorithm. Local features are extracted from the neighbourhood of characteristic points detected on each image of the pyramid.

computation of the pyramid. Finally, the images filtered by DoGs in the pyramid can be simply obtained by subtracting two consecutive images.

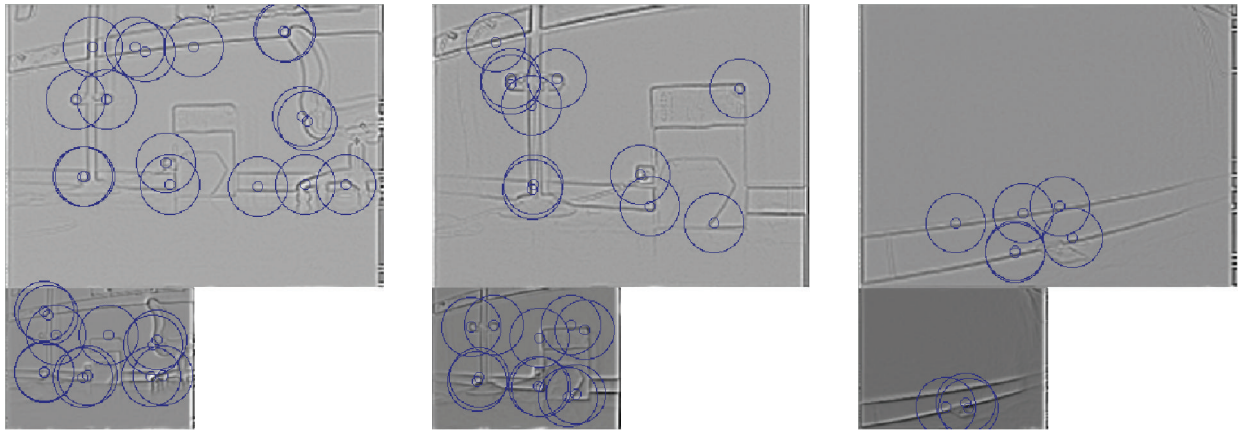
Keypoints detected on the images are the first N local maxima existing in each DoG image of the pyramid. Thus, the keypoint research algorithm orders the N first local maxima according to their intensities and extracts their coordinates. The shape of the neighbourhood for the research of maxima is a circular region with a radius of 20 pixels.

The number N which parametrises the algorithm corresponds to a maximal number of detections. Indeed, the robot may explore various visual environments (indoor versus outdoor) and particularly more or less cluttered scenes may be captured (e.g., walls with no salience versus complex objects as illustrated in Figure 3). A detection threshold (γ) is set to avoid nonsalient keypoints (Figure 3 illustrates the effect of this parameter on different images). This threshold is based on a minimal value of the local maxima detected. The presence of this threshold is even more important in the lowest resolutions since the information is very coarse at these resolutions. This particularity of the algorithm confers it a dynamical aspect. Precisely, the number of keypoints (and consequently the number of local features) depends on the visual scene and is not known a priori. Furthermore, the threshold γ could be set dynamically through a context recognition feedback but discussing here this mechanism is not our purpose (see an example of context recognition in [23]). However, even if this threshold is considered as a constant value, the number of detected keypoints varies dynamically according to the input visual scene. Consequently, the number of computations (neighbourhood extractions) also depends on the input data.

2.1.2. The local image feature extraction

At this stage, the neighbourhood of each keypoint has to be characterised in order to be learnt by the neural network. Existing approaches to locally characterise keypoints are numerous in the literature: local jets, scale invariant feature transform (SIFT) and its variants, steerable filters, and so forth (see [24] for a review of local descriptors).

In the current application, we simply reuse a view-based characterisation where keypoint neighbourhoods are represented in a log-polar space. This representation has good



(a) (b) (c)

Figure 3: Keypoints detected on different visual viewpoints. The same detection threshold is used in the three cases. Keypoints coming from the same octave are gathered on one image (2 octaves represented): (a) cluttered scene (29 keypoint detected); (b) same scene but closer, less cluttered (22 keypoints); (c) view captured during a wall following (9 keypoints).

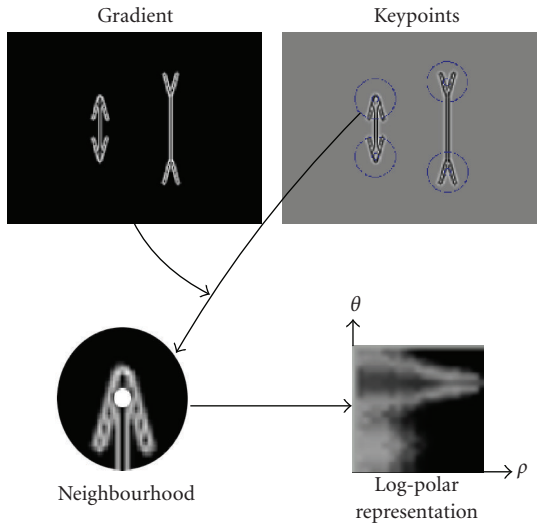


Figure 4: Local features extracted at one scale on Müller-Lyer's illusion.

properties in terms of scale and rotation robustness [25]. This kind of mapping is also used by the *GLOH* descriptor [24]. The local feature of each keypoint is, therefore, a small image resulting from the log-polar transformation of the neighbourhood (Figure 4). This transformation is done by a lookup table (LUT) allowing the mapping: $(\rho, \theta) = f(x, y)$.

The neighbourhoods are extracted from the gradient magnitude image at the scale the keypoint was found by the detector. Each neighbourhood extracted is a ring of radius $(5, 36)$ pixels. Excluding the small interior disc avoids multiple representations of the central pixels in the log-polar coordinates.

The angular and logarithmic radius scale of the log-polar mapping are sampled with 20 values. Each feature is thus an image of dimension 20×20 pixels. The sizes of the rings and feature images have been determined experimentally for an indoor object recognition. The given parameters represent a tradeoff between stability and specificity of the features [26]. Finally, the small log-polar images are normalised before their use by the rest of the neural architecture. By associating the data provided by the visual system with actions, the global system allows the robot to behave coherently in its environment [3].

Generally, the visual system must not be considered isolated but integrated in a whole architecture whose modules interact dynamically with each other and through the environment [27]. Hence, the evaluation of the parameters of the visual system depends on the rest of the robot system architecture.

2.2. *A custom RTOS as a solution for domain-specific implementation*

When integrated in mobile robots with navigation or object recognition objectives, this application must obviously satisfy some real-time constraints. For example, the local image features extracted in the neighbourhood of keypoints may be used for obstacle perception and, therefore, for trajectory guidance. In the general case, this vision subsystem must match real-time constraints if it is used in a global sensorimotor loop. Robot and environment integrities depend on these constraints.

However, due to the complex dynamic behaviour of our vision application, a precise characterisation of its timing behaviour is not trivial a priori. Expressing a global application deadline or period matching all context conditions (robot motivation and internal state, nature of the visual scenes, etc.) is impossible at compile time and mainly depends on the global system dynamics (see Figure 1). In our use case, the period constraint (the camera fps) is a parameter, amongst other regulation signals (maximum number of extracted keypoints N, detection

threshold γ), given by the external control system. All of these parameters will dynamically vary during the system lifetime.

More precisely, let us suppose the rough functional partitioning of our application into separate tasks as given in Figure 2 (at this step we do not consider the Hw/Sw partitioning). In this case, the number of concurrent computation subtasks (number of extracted keypoints thus the parallelism degree) and their deadlines have different configurations according to the current system mode. In these conditions, all classical implementation tools are clearly inappropriate for the design of our SoC architecture. Since deadlines are variable, all static scheduling-based compilers or synthesisers will be inapplicable except if the number of processing resources corresponds to the maximum degree of parallelism. The latter is of course unthinkable in embedded systems.

As an example, we have done a temporal profiling of the vision application and measured the execution time of the application tasks. This experiment has been done with a pure software version of the application executed on a single Nios2-embedded 32 bits microprocessor from Altera with a 100 Mhz clock frequency. Application tasks can be divided in three groups according to the execution time:

(i) intensive data-flow computation tasks that execute in a constant time,

(ii) tasks with execution time correlated with the number of interest points (Figure 5),

(iii) tasks with unpredictable execution time (Figure 6).

These preliminary results obviously show that intensive computation tasks would interestingly be implemented as pure (eventually pipelined) hardware modules. On the other hand, execution results of the search, sort, and extraction tasks confirm a software implementation poorly adapted to a static scheduling.

It is well known that multiscale systems take advantage of distinguishing the visual processing done at the different scales. In the classical but limited *coarse-to-fine* approach, lowest resolutions are considered with highest priorities. Visual data from these resolutions are integrated quickly in the system to get a first coarse description of the environment. This description is then refined by other scales. More reasonably, the visual system behaves in a more flexible way but remains still based on these different priorities. It may favour the utilisation of different frequency domains according to its objectives. For example, the recognition of facial human expressions can be done *selectively* in different frequency domains (see [28] for a more detailed discussion on frequency selection). In a navigation task, this approach can be extended: in noncluttered environments, a priority to the low scales can be given to navigate coarsely. The robot speed and/or the camera acquisition speed (the frame per second) can be increased. At the opposite, to precisely recognise objects, keypoints from other scales must also be taken into account (more scales would have high priorities) and the robot and camera speeds must be reduced accordingly.

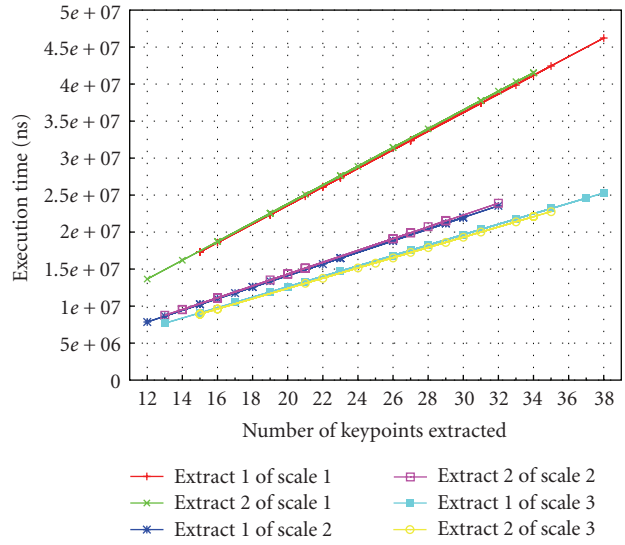


Figure 5: Tasks with execution time linearly dependent with the number of keypoints processed on representative samples.

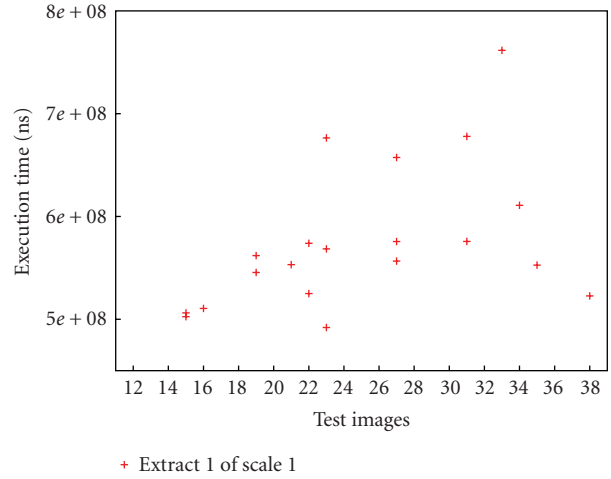


Figure 6: Tasks with unpredictable execution time on representative samples.

Yet designing a hard real-time system imposes the static knowledge on upper bounds of application parameters. So our application, with its particular properties, seems not adapted to execute in real time with deterministic and predictive times. For that reason, we have defined three modes (ranges of parameters and constraints) based on the selective coarse-to-fine approach, which correspond to the three main types of behaviour for the robot.

(i) Fast mode: the robot moves around in a learnt environment, it only needs coarse description of the landscape but at a high frame rate. Thus, only the lower scales are processed but the keypoints of the current image must be provided to the neural network (sensorimotor control in Figure 1) at the rate corresponding to the robot speed.

Table 1: Application modes.

Mode	N	FPS	deadline	mission
Fast	10	20	50 ms	Obstacle detection in known environment
Intermediate	30	5	200 ms	Exploration of unknown environment
High detail	120	1	1s	Static detailed analysis of environment

It is the reason why in this first mode we fix the maximum number of extracted keypoint $N = 10$, and the number of frames processed per second (fps) to 20.

(ii) Intermediate mode: the robot moves slowly, for example, when the passage is blocked (obstacle avoidance, door passage) and needs more precision on its environment. In this mode, the middle and the lower scales are processed. So, keypoints with information on a larger frequency band are provided to the neural network.

In the intermediate mode, we fix $N = 20$ and the system works at a rate of 5 fps.

(iii) High-detail mode: the robot is stopped in a recognition phase (object tracking, new place exploration). All scales are fully processed and full information on the visual environment is provided at the expense of the processing time.

For this last mode, we fix $N = 120$ and the rate of the system to 1 fps.

For all of these modes, summarised in Table 1, we consider a constant value for the detection threshold $\gamma = 200$. The system parameters, such as the number of frames per second processed by the system (and consequently the period), have been defined from measures on a representative sample of images (acquired by the robot) on the embedded softcore processor we use to determine realistic times (the Nios II introduced previously). Moreover, the upper, average, and lower bounds of execution times for those three modes have been analysed in order to extract predictive behaviours from the global dynamics. Exactly this approach limits the system dynamics so that in each mode the system would now be under hard real-time constraints. It also allows to explore and define the dimensioning of the embedded architecture.

Inside each mode, and from a task-scheduling point of view, these mechanisms can be modelled by attributing different priorities to the local features extraction and normalisation tasks. Moreover, in some cases. the lowest priority tasks may not be executed without significantly damaging the robot behaviour. According to the external conditions, scheduling or not the lowest priority tasks must be integrated in a quality-of-service (QoS) information. This information will be necessary to insure an efficient global sensorimotor loop. Such a *quality factor* would easily be computed locally with the number of extracted features versus the total number of keypoints ratio. This execution scenario can only be achieved by an embedded dynamical real-time operating system.

So far, we are faced with a twofold problem. First, to integrate the development of an embedded RTOS in an HW/SW design flow. Second, to propose a design methodology for exploring application-specific dynamical scheduling strategies. Such a SoC design approach requires a methodical design process based on the concept of high-level modelling which allows designers to explore and validate application deployment alternatives on a dedicated architecture. It permits also to incrementally refine this model towards the final hardware and software implementation. Today, developing a high-level RTOS model constitutes a challenge and exploring dedicated dynamical scheduling policies is not addressed by existing approaches such as [11, 14, 16].

Since version 2.1 of the SLDL SystemC simulation engine, we developed the basic mechanisms for building such a model of an embedded RTOS. Those mechanisms will be essential for the development of our future SoC platform. From these basic blocks, we developed a modular SystemC RTOS model. It provides the RTOS services responsible for the dynamic control of the execution sequencing and the possibility to dynamically create applicative processes. The high-level model of this is described in the Section 3. Section 4 will bring the first guidelines for using our model as a basis for the exploration and validation of a dedicated distributed embedded platform needed by our application.

3. BUILDING THE SYSTEMC MODEL OF A REAL-TIME OS

We present in this section how it is possible to define a SystemC model of a relatively simple but realistic real-time operating system layer. We will particularly focus on the dynamical mechanisms of the RTOS (i.e., the dynamical creation and preemption of processes) needed by our vision application.

For the sake of illustration, Figure 7(a) gives a subset of the vision application, partitioned into several tasks, that will be used to highlight the mechanisms we propose. `Sampling`, `Smoothing` and `DoG` tasks correspond to the subsampling, Gaussian filtering, and difference of Gaussian operations. `Search` task searches, sorts, and extracts the coordinates of the first keypoints above the detection threshold. It takes as data input the result of `DoG` and its associated `Smoothing`, and its parameters are N and γ. `Extract` task builds the corresponding neighbourhoods and `Norm` task transforms and normalises the local features. The exact way the application is partitioned into tasks is out of the scope of this work. Our purpose here is only to illustrate how we can model and simulate the execution of a multitask application on top of an embedded real-time OS.

In order to obtain the attended dynamic behaviour, our application needs to create, depending on some parameters or data, new treatments with different parallelism degrees. This behaviour cannot be estimated during development nor at compile time. The SystemC simulation kernel can neither

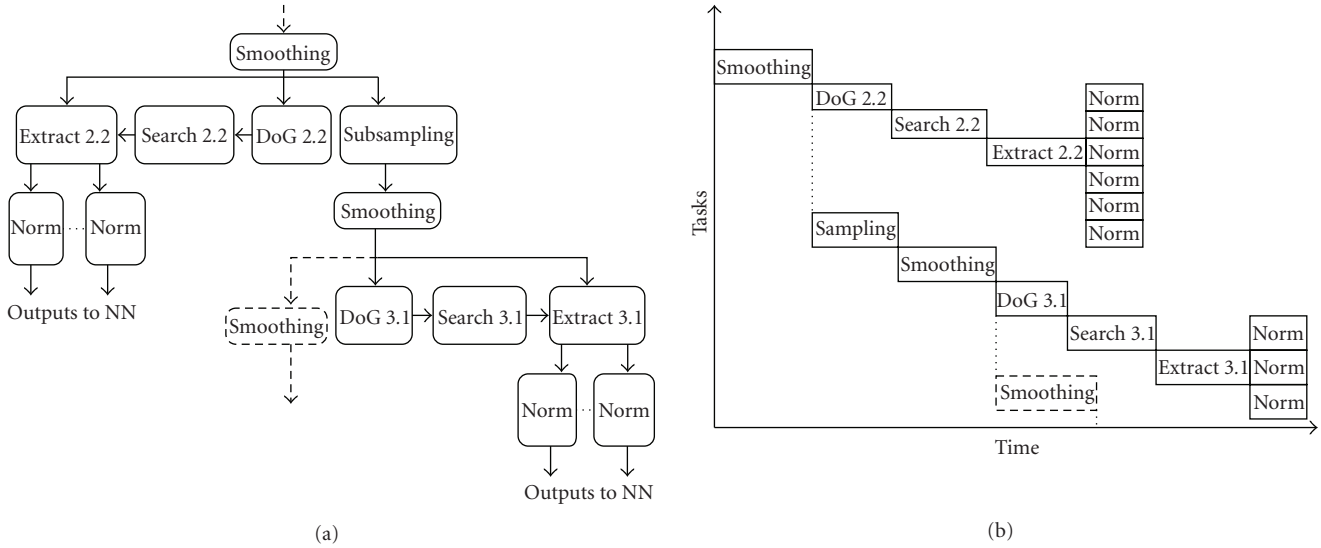


Figure 7: Example of the vision application partitioned into a task graph in (a). Only two half scales belonging to two different scales are illustrated. If an infinite number of resources are available and an ASAP (as soon as possible) scheduling is used, this graph would be scheduled as illustrated in (b) where the two half-scales contain, respectively, 6 and 3 keypoints.

handle such an execution model. Indeed, standard SystemC processes must be declared before starting a simulation (at elaboration time) and cannot vary until the end of simulation. This constitutes the first limitation of modelling dynamical-embedded software as classical processes in SystemC 2.0.

The second problem of modelling an OS in SystemC consists in finding a mechanism allowing an explicit scheduling of applicative tasks instead of the native event-based SystemC scheduler. The native SystemC simulation kernel acts as an abstraction layer and provides to the developer a model of a pure virtual architecture with an infinite number of resources. It thus simulates the execution of all processes in a parallel way (see the example of Figure 7(b)). If an architecture with a finite number of resources (processors) is targeted, the exact behaviour of an application composed of multiple tasks sharing the same resource cannot easily be simulated.

As a third keypoint, scheduling a task in an RTOS not only consists in starting or killing it but also, if preemptive schedulers are targeted, in interrupting tasks and switching between contexts. SystemC does not allow interrupting a thread without an explicit declaration of breakpoints by using the SystemC `wait()` primitive. Unfortunately, the `wait()` call cannot take into account several preemption and resume cycles. Thus a specific mechanism must be developed for modelling such an execution scheme.

3.1. Modelling the bare mechanisms

Dynamical creation of tasks—creation of SystemC processes—is now possible by using the new SystemC kernel version 2.1 that comes with the public `boost` library. This library offers primitives for spawning processes and attaching them to SystemC modules after the elaboration phase, thus dynamically during simulation. Moreover, thanks to the dynamic lists of sensitivity, SystemC threads can be started and resumed depending on dynamically created events.

3.1.1. Task creation service

As shown in Figure 7(b), the potential parallelism of the normalisation tasks (`Norm`) depends on the dynamical threshold γ set by an external context recognition feedback (see also Figure 1). The `Extract` task thus needs to dynamically create a variable number of `Norm` subtasks depending on the results of `Search` task.

We have implemented the `create_task()` service of the RTOS with the `sc_spawn()` primitive of the kernel. The `create_task()` service allows an applicative task to dynamically create several instances of new tasks. Our RTOS model implements tasks as C++ classes containing several data as member variables: priority, period, deadline, function code, and so forth. It is important to note that our task objects are not modelled as SystemC modules. Instead, each task is given as a global function that comes in a separate C++ file linked with our RTOS model. `create_task()` creates an instance of a new thread attached to the RTOS model. It creates also a new event that will be used by the scheduler for starting and resuming the thread. Finally, it creates the corresponding C++ structure containing all task data (e.g., task control block) for an easy manipulation by the RTOS.

By specifying tasks as functions instead of SystemC modules, application engineers do not need to take care about detailed implementation of the model. Moreover, as it will be mentioned in Section 4, the same application code could be used in a refined architectural model without any modification of the source code. For example, it could be linked with a more complicated platform model having

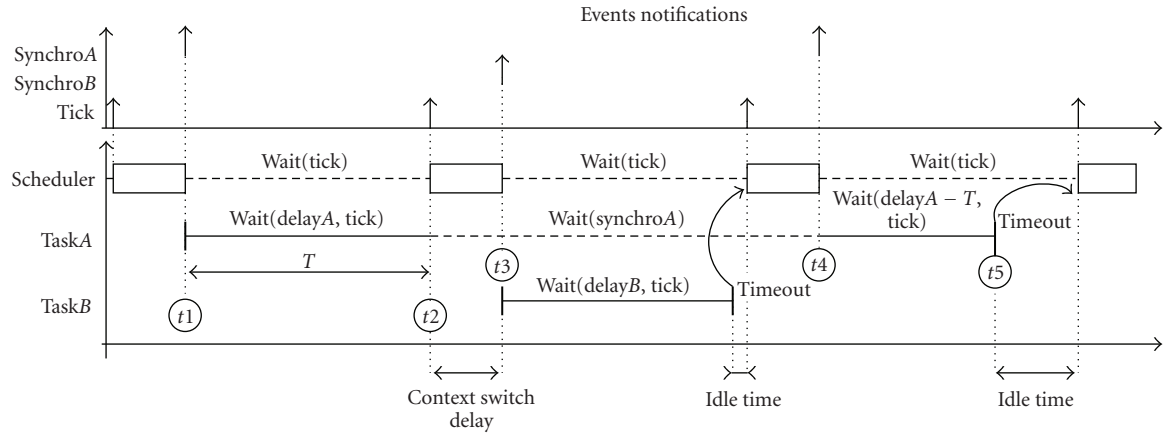


Figure 8: Details of the preemption mechanism provided by the `os_wait()` service.

several RTOS instances, thus modelling a multiprocessor architecture.

3.1.2. Scheduling

The main module of our RTOS model is the scheduler function. This function is developed as a member method of the scheduling module inside the RTOS model. Thanks to the object-oriented SystemC kernel, modifications of the scheduler can be done by using inheritance. Method overload is a light way for exploring different scheduling strategies. Even without the preemption mechanism, the model can help the developers to set the right task priorities and explore the scheduling algorithm. However, with the execution time modelling, a refined preemptive scheduling could be simulated and could give results on the OS overhead (number of context switches).

3.1.3. Tasks preemption

For modelling task interruption and preemption, we have created a dedicated `os_wait()` service that gives control to the OS model and allows taking into account preemption and context switches. Tasks are viewed as sequence of functional code and system calls. So their code is splitted into unbreakable portions of code each having their own estimated execution time, mentioned by using the `os_wait()` primitive. The `os_wait()` primitive has been written as a blocking call that returns after a specific time delay and allow to model preemption. This delay is the estimated task duration added dynamically by the sum of all interrupt treatments or scheduler invocations durations suspending the task. In this manner, `os_wait()` service acts as a way to model exact execution time and concurrency of tasks.

As an illustration, Figure 8 gives details about how preemption is modelled on a single processor architecture. The call to the SystemC `wait` primitive is encapsulated into the `os_wait()` service. In this example, when task*A* is launched, it first runs its functional code portion in zero simulation time, then it simulates its execution time by the `os_wait()` call instead of the classical `wait()` with its duration given in parameter (delay*A*). In fact, the corresponding service code waits for both the duration timeout and any interruption event like the real-time clock tick which announce the wakeup of the scheduling service. As the execution time of Task*A* is greater than the clock period, the elapsed time T ($t2 - t1$ on Figure 8) is taken into account and subtracted to the attended execution time.

At date *t2*, the scheduler is executed and launches Task*B* which has the same priority, following the round-robin policy. After the simulated context switching duration ($t3 - t2$), the synchronisation event of Task*B* is notified and the schedule process waits until a new clock tick. At time, *t3* Task*B* executes its functional code till the first encountered `os_wait()` call. This functional code is executed in zero time in the SystemC simulation engine. At its turn, Task*B* calls the `os_wait()` service for a duration lower than the tick period and terminates. At date *t4*, Task*A* is scheduled again and completes its execution time simulation (*t5*).

With this preemption model, the number of context switches between SystemC threads is approximately equal to the one in a real-preemptive RTOS kernel [29]. This leads to an efficient simulation with no significant overhead (see Section 3.3).

3.2. Building modular RTOS models

Thanks to the object-oriented nature of the SystemC library, the RTOS model can be developed in a modular way [29]. Adding new or specific services and modules can be easily done by using objects aggregation or inheritance and objects relationships. The RTOS interface (its application programming interface) works exactly the same way and can be updated and augmented according to the application needs.

Our RTOS model is implemented as a single hierarchical SystemC module and instantiates multiple service modules (see Figure 9). Each service module is also a hierarchical SystemC channel (is a SystemC module that inherits from SystemC interfaces). The application programming interface of a given service module is a part of the global RTOS API. This global API provided to the application is constructed

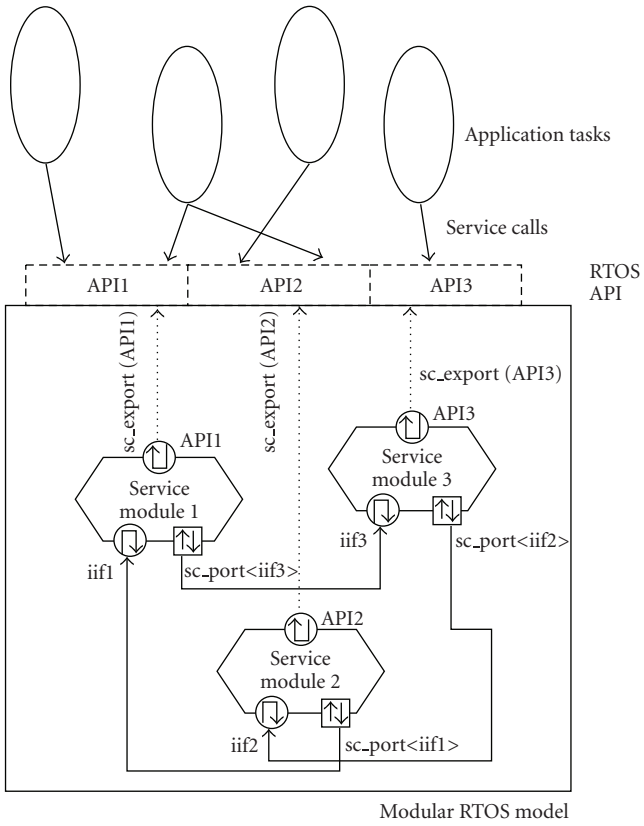


Figure 9: The modular RTOS model is a hierarchical SystemC module containing multiple interconnected service modules. Each service module is a SystemC channel requiring and implementing some dedicated interfaces.

```
// Task management service API
class task_mgr_if: virtual public sc_interface {
public:
        virtual task* create_task (· · · ) = 0;
        virtual void kill_task (· · · ) = 0;
        virtual int get_pid (· · · ) = 0;
        · · ·
};
// Task management service internal interface
class task_mgr_iif: virtual public sc_interface {
public:
        virtual void change_task_state (· · · ) = 0;
        · · ·
};
// Task management service module header
class task_mgr: public sc_channel,
                public task_mgr_if,
                public task_mgr_iif {
private:
        · · ·
}; // End class taskmgr


// RTOS SystemC module
class my_OS: public sc_module{
public:
        // export all services API
        sc_export⟨task_mgr_if⟩ TMGR_IF;
        · · ·
SC_HAS_PROCESS (my_OS);
· · ·
// instanciates all service modules
task_mgr        my_task_mgr;
scheduler       my_sheduler;
· · ·
// inter_module binding
my_scheduler·task_mgr_port (my_task_mgr);
· · ·
// modules/exports binding is done in
// the constructor
· · ·
}; // End class my_OS
```

Algorithm 1: Example of the modular construction of the RTOS module. This example shows how a service module is declared with both a global interface exported to the application and an internal interface offered by this module to other service modules. The whole API is built by exporting each service's interface.

in a modular way in the sense that it is the union of the all API interfaces instantiated by service modules. This can be done automatically at design time thanks to the `sc_export()` facility.

In addition, each service module of an RTOS model can provide an internal interface through which other service modules can interact. Respectively, each service module can have an internal communication port (`sc_port⟨⟩`) requiring the internal interface of another service module. Communication between modules are for the moment modelled as simple method invocations. An example of such a communication is the internal `change_task_state()` service call offered by the task management module: it allows other modules (as for the scheduling module for example) to change the internal state of an active task (see Algorithm 1 for the SystemC skeleton).

In order to propose a generic RTOS structure, we have defined several service categories by taking into account the potential locality of shared internal data as an important criterion. It is important to notice that minimising the service modules granularity—by building elementary modules—could enhance the exploration capability of the OS model at the expense of more complex communication infrastructure between modules. Our nonexhaustive service module list is then mainly composed of the following services: task management, interrupt requests management, semaphore service, scheduling, timer, and real-time clock management.

3.3. Validation of the abstract OS

Our RTOS model has been written in SystemC and was firstly validated under a single processor assumption with all of the presented services and RTOS modules. Two schedulers with different strategies have been tested in the model (fixed priorities without preemption and real-time round-robin).

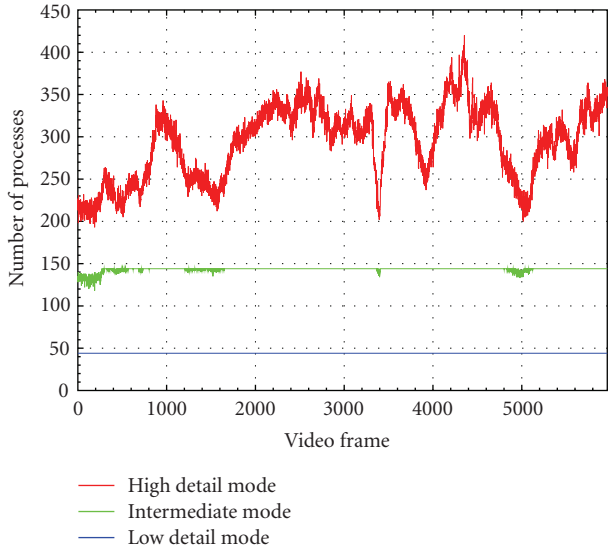


FIGURE 10: Number of process created and managed by the OS model during the application simulation.

The vision application source code has also been validated on top of the RTOS model. One can notice that porting this source code on the RTOS API did not represent a significant effort. The simulation of our application with the RTOS layer produced interesting results: a graphical C++ library (`gtk` + 2.0) has been included in the SystemC code and allowed us a quick functional verification (Figure 11). In addition, the model produces execution traces that allows a deep examination of the application behaviour. For example, simulation results presented in Figure 10 illustrate the dynamic behaviour of the RTOS model. Figure 10 shows the evolution of the number of active process in the different modes according to the complexity of the visual environment. In high-detail mode, all scales are processed and the number of process depends on the number of keypoints in the video frames. In this example, the OS dynamically creates, schedules, and deletes up to 420 processes. In the intermediate and fast modes, the number of process is, respectively, bounded to 30 and 10 processes by half-scale.

Moreover, a timed simulation is possible with this model. Indeed, timing data can be inserted in the SystemC code. Execution time estimations can be added in the RTOS model by using the dedicated `wait()` function inside the each RTOS module for modelling durations of system calls. By doing so, it becomes possible to simulate the timing overhead produced by the RTOS calls and scheduling operations. In addition, a global performance evaluation is also possible by giving worst-case execution time (WCET) estimations of each code portion as parameters to the `os_wait()` pseudoservice calls in the application code.

We evaluated the accuracy of our modelling approach by performing several sets of experiments and comparing the simulated execution times relatively to actual board measurements for multiple sets of data. The average application times are depicted in Table 2.

According to these results the high-level simulation, accuracy is within 3-4% of board measurements. This accuracy is acceptable at this level of description where the processing elements are abstracted. In addition, the exact task ordering and preemption realised by μC/OS-II on the board is modelled [29] thanks to the preemption modelling described precedently.

We have also compared the simulation duration of our application with or without the RTOS model. The simulation duration is about 2 minutes 53 seconds for the application described as pure functional C code (using Linux host API) and 3 minutes 12 seconds when the application runs over 4 SystemC RTOS models. These results have been obtained with a simulation host machine equipped with an Intel Dual-core at 1.66 GHz and on a set of 1000 images.

These results demonstrate that an annotated OS-based exploration methodology responds to the tradeoff between simulation time and estimation accuracy at early design steps in order to model and explore concurrency without taking into account specificities of the processing elements architecture. These properties will help us to explore the architecture dimensioning in Section 5.

4. TOWARDS A GLOBAL PLATFORM EXPLORATION AND DESIGN FLOW

In order to finalise the SoC platform, we must follow a dedicated methodology that includes the RTOS exploration and validation. We thus propose a global exploration and design flow based on successive refinement steps of both the architecture and the RTOS implementation strategies. This flow is partially inspired by the works of Gajski et al. and Jerraya et al. and is illustrated in Figure 12. A three-step modelling approach (specification, architecture, and implementation levels) has been proposed in [12]. The main drawback of this proposal is that there still exists a design gap between the architectural model (the one exhibiting the OS model) and the final implementation model (with processors modelled as instruction set simulators (ISSs)). For bridging this gap, we propose an intermediate distributed architecture model where the parallelism can be explored. The presented high-level RTOS model is used both in the second and third steps of this flow.

4.1. *The SoC modelling flow*

The proposed methodology then consists in four successive modelling levels (specification, executive, distribution, and implementation models) described in the following.

4.1.1. *Specification model*

This model is written as a pure SystemC model of the application where each applicative task is a SystemC thread or process and where intertask communications take place through the SystemC primitive channel (`sc_mutex`, `sc_fifo`, etc.). Tasks are synchronised with `sc_events`. The whole application is specified as a single `sc_module` with all tasks as member functions. A complete functional verification of the

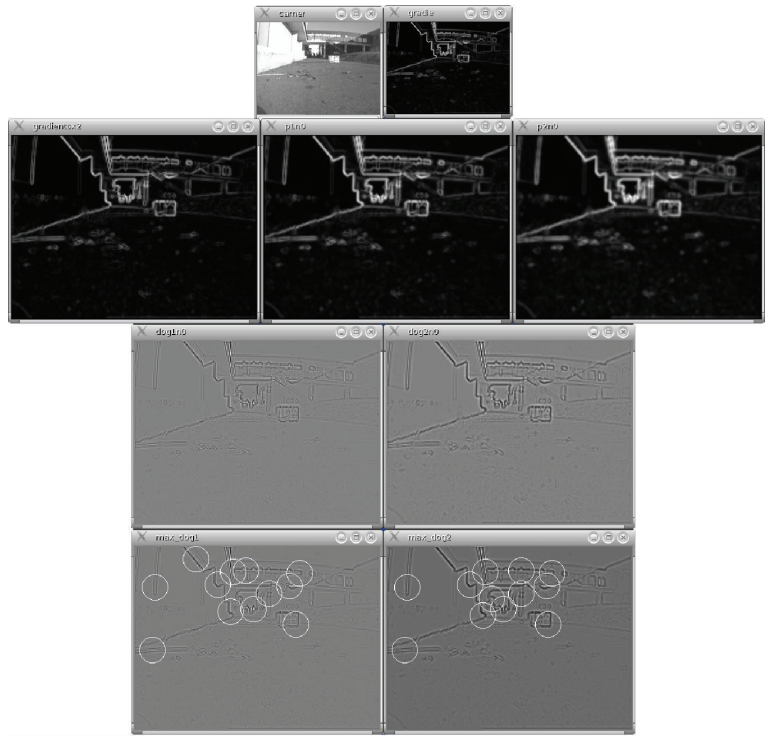


Figure 11: Graphical results of the SystemC functional model simulation obtained by using the `gtk` + 2.0 C++ library.

Table 2: Comparison between accuracy and simulation time of the OS model.

Board measurements	Simulation estimations	Error percentage	Simulation duration per image
29268.570400 ms	28369.598240 ms	3.07%	192 ms

application is possible at this level. The SystemC kernel and its API model the pure virtual architecture that executes the application. Due to the SystemC concurrency principle, an infinite number of resources is simulated at this level. Moreover, additional C++ libraries can be used for debug purpose and can give a functional trace of the application similar to the one illustrated in Figure 11.

4.1.2. Executive model

The second model refines the previous one by adding an explicit RTOS layer in the application model. At this level, a SystemC module models the whole application. This main module only contains the first main task of the application that acts as the boot code of a real-embedded RTOS: it initialises the scheduler (`OSInit()` service), creates the first task, and launches the OS scheduler (`OSStart()` service). The application source code has access to the RTOS services through the whole RTOS API. All accesses to SystemC primitive channels are thus replaced by OS calls. This can be done automatically by a simple C/C++ compiler preprocessing step. In addition, execution time of tasks may be modelled by `os_wait()` calls. No further modifications of the application code are necessary and a very fast validation can then be done. At this level, a number of strategies can be explored for customising the needed RTOS layer (scheduling policies, preemption model, and implementation of some optional services). Exploring the software or hardware implementation of the applicative tasks is not yet addressed at this level. The main objective is to explore the global sequencing of operations.

4.1.3. Distribution model

The distribution model is built by instantiating multiple (eventually different) OS models (see Figure 13). All models have access to the applicative C++ functions given in the specification source code and a global multiprocessor evaluation is allowed. In this case, each RTOS node integrates some specific modules dedicated to inter-OS synchronisation and communication. The associated RTOS services can then be explored at this level. One can note that simulating the architecture at this level allows an exploration of the number and type of resources without explicitly modelling processing resources (microprocessors). Up to this level, communication infrastructure refinement will be done by following the transaction level modelling (TLM) supported by the SystemC language [6].

4.1.4. Implementation model

This model is the last step before the physical synthesis of the platform. Hardware parts (memories, I/O devices, interrupt

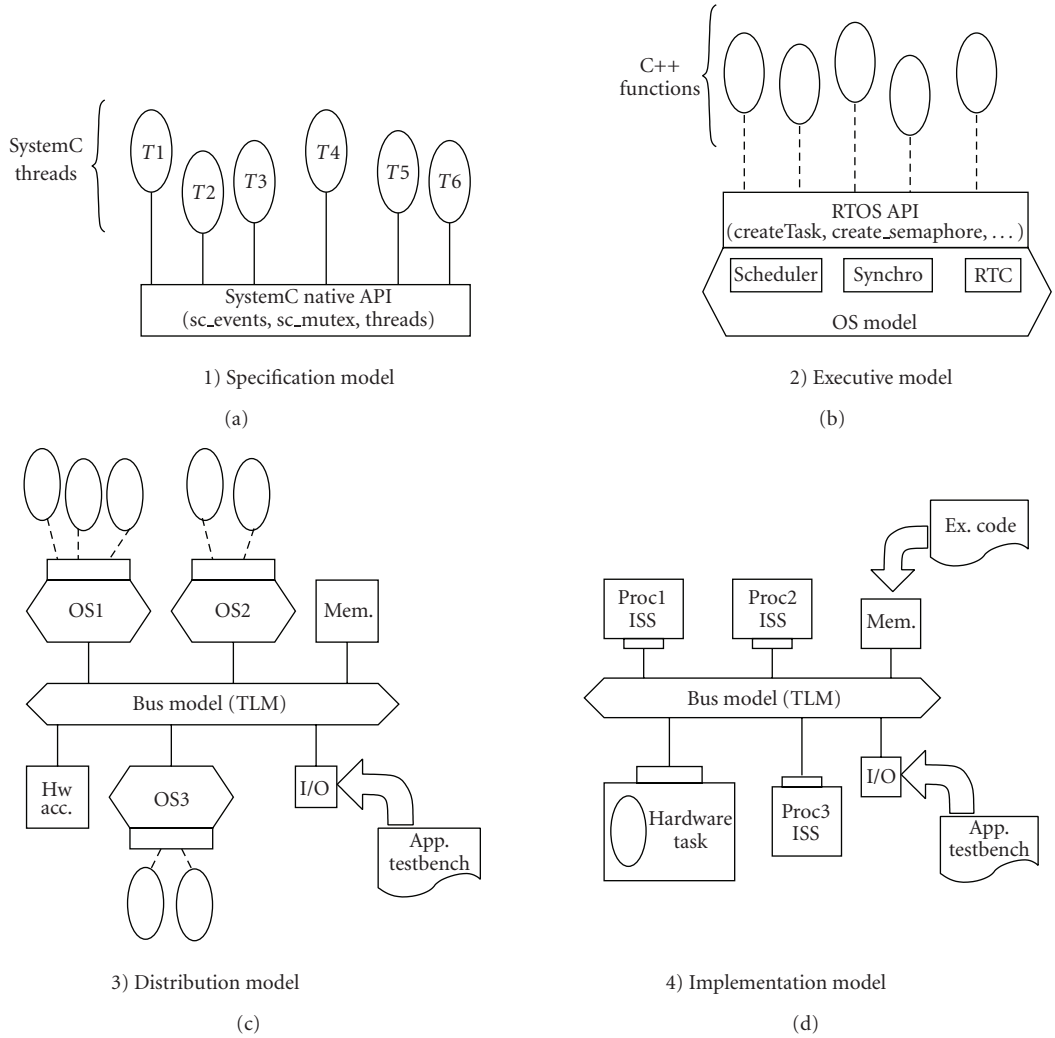


Figure 12: The proposed platform refinement flow starts with the specification model and provides at least four refinement steps until the final implementation model.

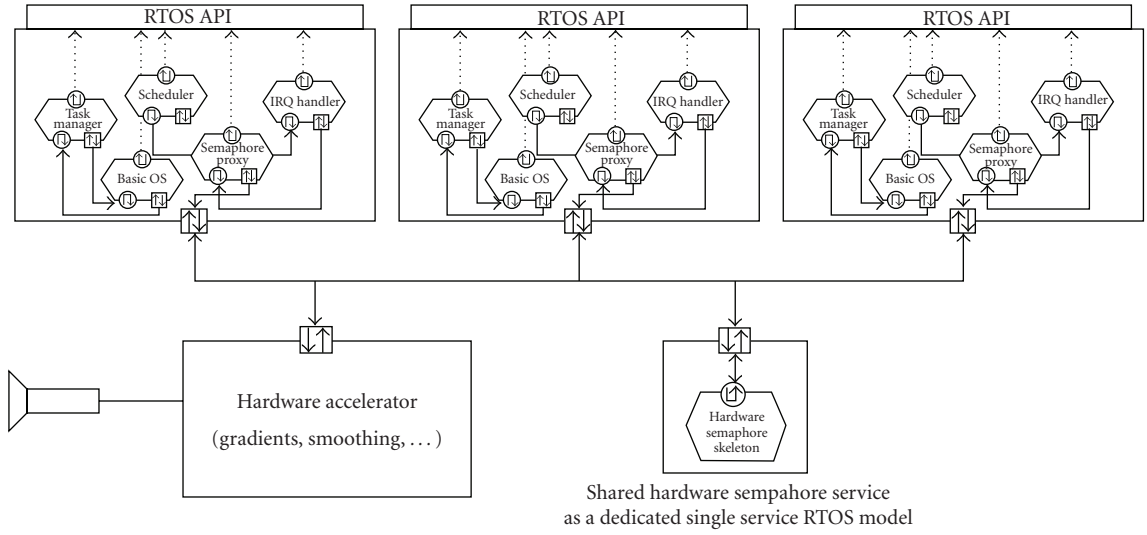


Figure 13: MPSoC platform modelled as a collection of interacting RTOS models.

handlers, hardware accelerators, etc.) of the platform are described as register transfer level (RTL) models. Processors are represented by their instruction set simulators. The application source code must be compiled and inserted in the platform model in the form of executable binaries. Simulation results can then be accurate at the clock cycle level (Cycle Accurate) or even at the signal level (Bit Accurate). At this level, the exploration and validation of a number of strategies have been done and the following physical synthesis of the platform can be realised by classical hardware design tools. We are not concerned in developing models at this level of abstraction.

This exploration methodology can be applied to a lot of different platform architectures and application domains. This flow is particularly well adapted to dynamic contexts, applications, and environments.

4.2. Modelling a distributed architecture

This part presents the latest results obtained when using the RTOS model presented in Section 3 for modelling a distributed multiprocessor architecture for the vision application.

Based on the presented SystemC RTOS model, and by taking advantage of modularity and genericity of SystemC models, we have developed a multi-RTOS model. At this level of abstraction, each processing element is represented by a single RTOS model. Each RTOS is responsible in executing its own part of the application (we assume that the application partitioning is done at design time). The application thus comes in the form of one main task per execution node. These main tasks are responsible in the creation of all application tasks and interrupt handlers. Each RTOS is also augmented by a communication port through which it can communicate with external world.

A common shared service is necessary to insure communication and synchronisation between tasks executed on different nodes. A shared semaphore module has been easily modelled by adding an extra RTOS model representing a unique semaphore service module. This module would be either refined as a single processor running a dedicated semaphore service or a hardware interprocessor synchronisation block. Figure 13 illustrates how multiple RTOS models can interact by using this shared module. Each local semaphore service module has been replaced by a proxy module implementing multi-RTOS communications. We inspired from the CORBA philosophy where proxies are used to implement a *service invocation* and skeletons are in charge of implementing the service itself. Distant communications between proxies and skeletons are modelled as simple method invocations and can be managed by any transport layer as for a TLM infrastructure for example.

5. EXPLORING THE APPLICATION ARCHITECTURE

Based on the presented design flow, we use our modelling framework to explore the architecture of the vision application. As described in Section 2.2, we made the profiling of the entire application on an embedded platform. We also built the profile of the μC/OS-II real-time services (deterministic). The timing data were measured and back-annotated into the high-level model in order to explore and evaluate the architecture dimensioning and the implementation strategies: tasks distribution, services distribution, scheduling algorithms, and so forth.

Table 3: Software profiling of the significant application tasks Average execution times on 20 representative images.

Task	Average execution time (ms)	Percentage
Gradients	13915.4	49%
HF Gaussian filtering	9795.8	35%
LF Gaussian filtering	426.0	1.5%
HF DoGs	443.0	1.6%
LF DoGs	15.3	0.05%
HF Searches	1286.1	4.6%
HF Extracts (+ norm.)	58.8	0.21%
LF Searches	45.1	0.16%
LF Extracts (+ norm.)	33.1	0.12%
Others (MF tasks, sampling)	2238.2	7.9%
Total	28257.2	100 %

As illustrated in Table 3, the on-board measurements helps to determine the critical portions of the application. In high-detail mode the gradient and the HF Gaussian filters represent more than 80% of the total execution time. Moreover, on the software implementation, all the treatments realised on high frequencies, except the extract task, are very critical and exceed the real-time constraint of the high-detail mode of the application: 1000 milliseconds. More generally, the actual software implementation of the common gradient task is incompatible with any of the three identified real-time behaviours.

For these reasons, we used the modularity advantage of the OS model to evaluate the gain of parallelism on the execution time of the three identified modes. Figure 14 shows the potential gain using multiple processors (from 2 to 5) for the high-detail mode. The better software partitioning among the explored ones is depicted on Figures 2 and 16; but the parallelisation has no significant effect beyond 2 processors. Indeed, the concurrency in the application only appears between the sequential Gaussian pyramid and the different scales (searches and extractions tasks), and between the normalisation tasks inside each scale, the latters only represent a small percentage of the total software application time. In addition, as illustrated in Table 3, the difference of complexity between scales explains the nonsignificant gain for the parallelisation of the normalisation tasks on a third processor.

Hence, only a hardware and software implementation could thus respect our variable constraints by accelerating the Gaussian pyramid.

According to the results of the first exploration phase, the identified critical and regular treatments (gradient, HF Gaussian filters, and DoGs) are candidates to a static

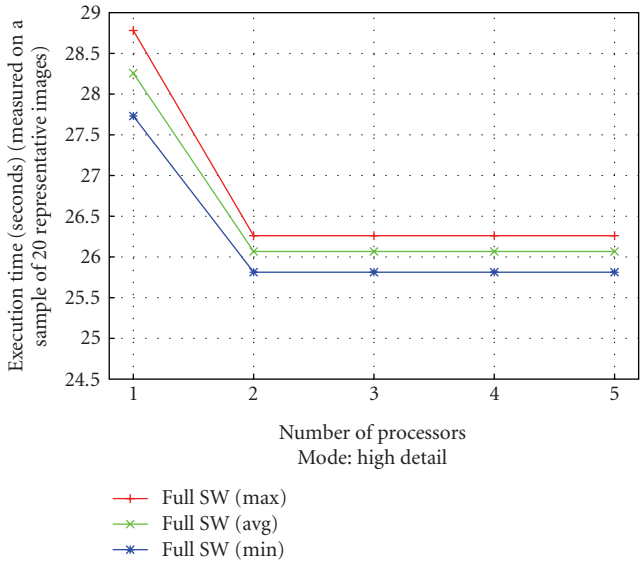


FIGURE 14: Execution times for different numbers of processors, in a full SW implementation.

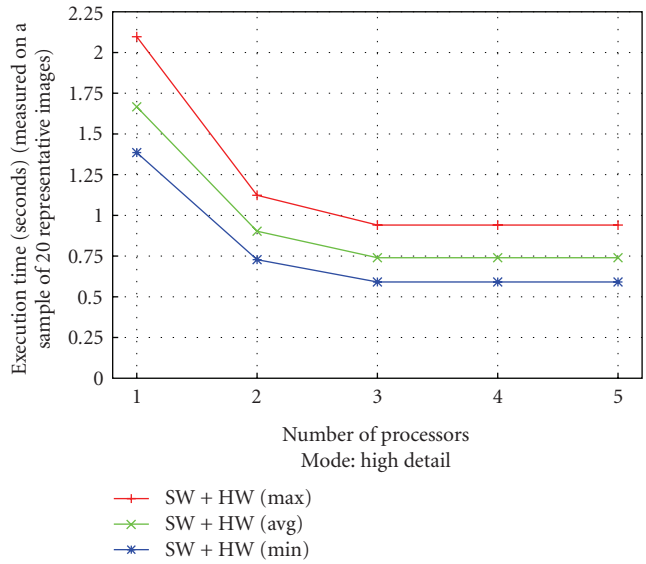


FIGURE 15: Execution times for different numbers of processors, on our hybrid HW/SW architecture.

hardware implementation as dedicated accelerators. These results conduct to a first-refinement process. Indeed, in order to evaluate the acceleration, we developed a VHDL description of the selected tasks. The temporal characteristics reported by the hardware synthesis tool (Altera's Quartus II for our example) were integrated into the model. The corresponding hardware tasks were modelled as independent and concurrent SystemC threads with back-annotated execution times. In addition, each hardware block provides an interruption line for synchronisation with the software part when data are produced. From the identified Hw/Sw partitioning, we led a second set of experiments resulting on the performance evaluations depicted in Figure 15 (the figure only represents results for the high-detail mode). Comparing Figures 14 and 15, we found a speed up factor of x17, thanks to the hardware acceleration. The hardware implementation of the pyramid also makes the parallelisation effects more significant on the third processor.

However, the challenge of the exploration was to find an architecture respecting the constraints corresponding to the three application modes. Exactly each mode executes different treatments under different rates. The variation of constraints finally leads to three embedded architectures that are presented in Figure 16. In fast mode, the lower scales are implemented on two processors. If two processors or more are used, the worst-case execution time of the application is about 48 milliseconds thus corresponding to the robot speed. In intermediate mode, 3 processors are needed leading to a total execution time under 150 milliseconds. In this mode, since the period constraint is relaxed, the two half lower scales can be implemented on the same processor. Finally, in the high-detail mode, the search and extract tasks of the two high-frequency half scales are processed on separate processors while low and medium frequencies are executed on a single processor. The maximum total execution time in this mode is 950 milliseconds on our three processor model. In the three cases, the architecture contains at least two common components: the hardware-gradient accelerator and a processor executing at least the low-frequency tasks. The performance results obtained with the following Hw/Sw solution are summarised in Figure 17.

The final system meets all the application requirements in each mode.

As a conclusion, we have successfully modelled a realistic multiprocessor platform with our distributed OS model. Thanks to the properties of the model, we have explored and defined the architecture adapted to the application requirements. The considered target platform is a System on Programmable Chip (SoPC) from ALTERA [30].

With our conclusions, an FPGA will be configured with 3 RISC microprocessors (Nios II) and the selected hardware blocks (accelerator and semaphore). Each processor will run an instance of a custom RTOS refined from the OS model. The interprocessor synchronisation will be realised through the shared hardware semaphore service. The entire application can thus be implemented on a single chip SoC.

In order to optimize the mode changes, we are now interested in refining the OS in order to support new specific services such as online software migration and dynamic voltage scaling (DVS) scheduling since only two of the three processors are useful in fast mode. These mechanisms will be managed by the dedicated and distributed OS. Another interesting perspective is the management of dynamically hardware reconfigurable units in order to propose a hardware adaptive architecture.

6. CONCLUSION

We have presented in this paper a particular design problem dealing with the SoC implementation of a visual system

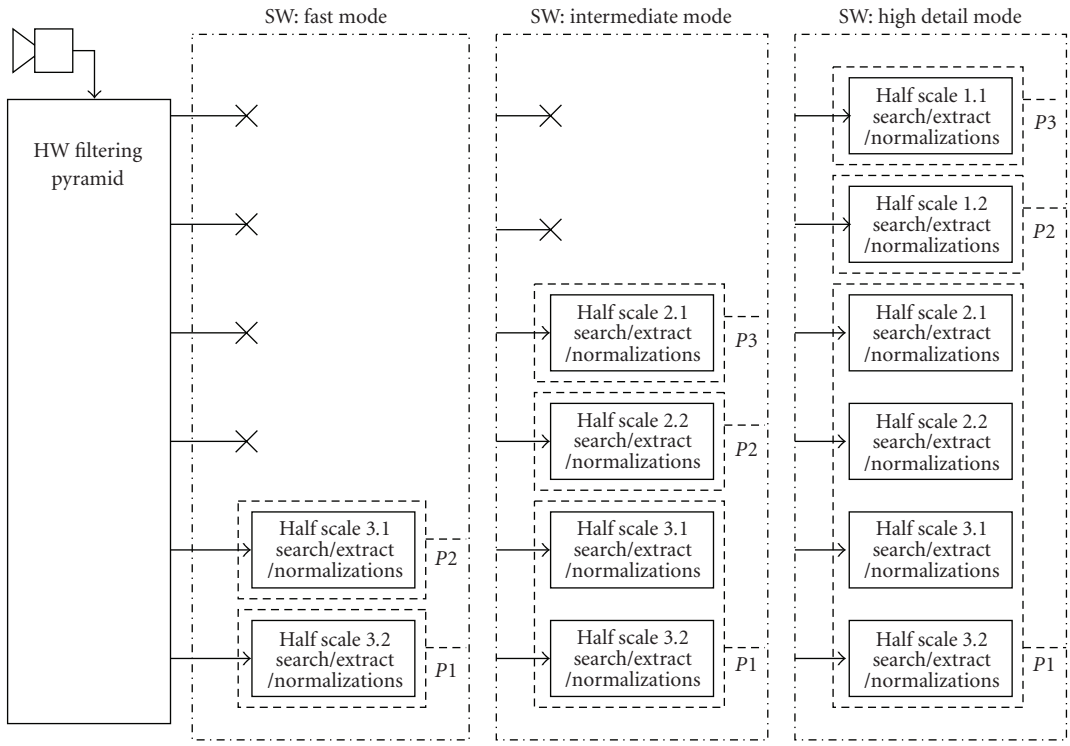


Figure 16: Architectures corresponding to the 3 application modes.

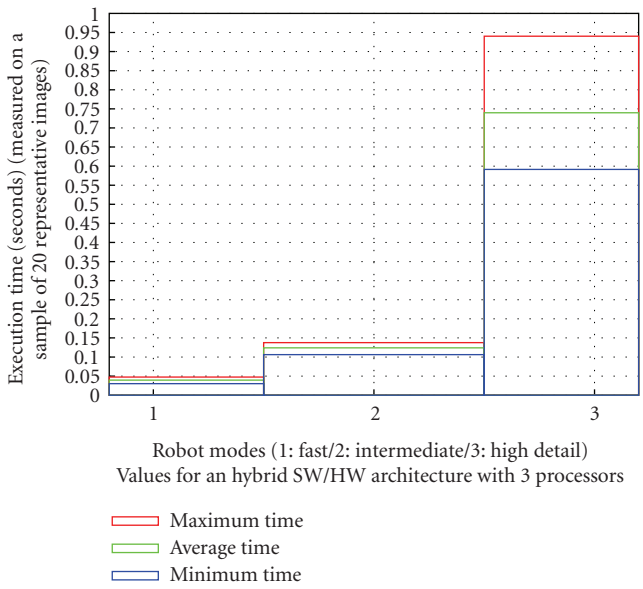


Figure 17: Execution times comparison on different modes on our hybrid architecture (HW + multiprocessor SW).

embedded in a mobile robot. The image-processing application works on a multiscale pyramidal decomposition of the images and extracts local features in the neighbourhoods of interest points. Such an application could be used for objects localisation, tracking, or recognition.

Since this application is inserted into a biologically inspired sensorimotor loop, it participates to the global system dynamics. As a first contribution, we precisely analysed the impact of the dynamics on the application behaviour. It has been shown that a lot of characteristics dynamically vary according to the regulation process imposed by the global system.

We have also shown that these variations are unpredictable without an entire a priori knowledge on the environment. We have thus demonstrated that classical design techniques based on static scheduling will fail to efficiently implement such an application. Only a custom RTOS with dedicated services could manage the adaptation of the architecture to the environment variations. The exploration and the definition of the required architecture have been made possible thanks to a high-level executable model of RTOS. This model facilitates early system dimensioning and application partitioning.

Our second contribution consists in detailing the bare dynamic mechanisms necessary to build a SystemC RTOS model. These mechanisms mainly provide the dynamical creation of SystemC processes and the preemption and execution time modelling. We have chosen the SystemC language for its ability to model and simulate both hardware and software systems at multiple levels of abstraction. We have also presented in this paper an operational executable RTOS model including these mechanisms. This model has been used to simulate the vision application on a representative set of data.

Finally, we have described how this RTOS model can be modified in order to be used for building a distributed architecture model. We have constructed the model of a realistic MPSoC platform containing multiple execution

nodes and a shared-hardware semaphore. The application partitioning and the architecture dimensioning have been made by using and customizing our generic model during the proposed exploration process. The obtained results constitute a promising way for the final SoC design. More generally, this work falls into the scope of the OveRSoC project [31] that aims at developing an exploration methodology adapted to the design of dynamically reconfigurable systems. The high-level SystemC RTOS model presented in this paper is expected to be used also for the exploration of custom RTOS services dedicated to the management of these particular dynamically reconfigurable resources.

REFERENCES

[1] D. H. Ballard, "Animate vision," *Artificial Intelligence*, vol. 48, no. 1, pp. 57–86, 1991.

[2] P. Gaussier and S. Zrehen, "PerAc: a neural architecture to control artificial animals," *Robotics and Autonomous Systems*, vol. 16, no. 2–4, pp. 291–320, 1995.

[3] M. Maillard, O. Gapenne, L. Hafemeister, and P. Gaussier, "Perception as a dynamical sensori-motor attraction basin," in *Proceedings of the 8th European Conference on Advances in Artificial Life (ECAL '05)*, M. S. Capcarrère, A. A. Freitas, P. J. Bentley, C. G. Johnson, and J. Timmis, Eds., vol. 3630 of *Lecture Notes in Computer Science*, pp. 37–46, Canterbury, UK, September 2005.

[4] P. Gaussier, C. Joulain, J. P. Banquet, S. Leprêtre, and A. Revel, "The visual homing problem: an example of robotics/biology cross fertilization," *Robotics and Autonomous Systems*, vol. 30, no. 1-2, pp. 155–180, 2000.

[5] A. A. Jerraya and W. Wolf, "The what, why, and how of MPSoCs," in *Multiprocessor Systems-on-Chips*, chapter 1, pp. 1–18, Morgan Kaufmann, San Francisco, Calif, USA, 2004.

[6] "SystemC standard," http://www.systemc.org/.

[7] "SpecC language," http://www.specc.org/.

[8] Y. Sorel, "SynDEx: system-level cad software for optimizing distributed real-time embedded systems," *Journal ERCIM News*, vol. 59, pp. 68–69, 2004.

[9] R. K. Gupta, *Co-Synthesis of Hardware and Software for Digital Embedded Systems*, Kluwer Academic Publishers, Norwell, Mass, USA, 1995, foreword By-Giovanni De Micheli.

[10] G. De Micheli, Ed., "Special issue on hardware/software co-design," *Proceedings of IEEE*, vol. 85, no. 3, 1997.

[11] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: hardware-software co-synthesis of embedded systems," in *Proceedings of the 34th Design Automation Conference*, pp. 703–708, Anaheim, Calif, USA, June 1997.

[12] A. Gerstlauer, H. Yu, and D. D. Gajski, "RTOS modeling for system level design," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, pp. 130–135, Munich, Germany, March 2003.

[13] D. Desmet, D. Verkest, and H. De Man, "Operating system based software generation for systems-on-chip," in *Proceedings of the 37th Design Automation Conference (DAC '00)*, pp. 396–401, Los Angeles, Calif, USA, June 2000.

[14] R. Le Moigne, O. Pasquier, and J.-P. Calvez, "A generic RTOS model for real-time systems simulation with SystemC," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '04)*, pp. 82–87, Paris, France, February 2004.

[15] "Cofluent Studio™," www.cofluentdesign.com.

[16] L. Gauthier, S. Yoo, and A. Jerraya, "Automatic generation and targeting of application specific operating systems and embedded systems software," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '01)*, pp. 679–685, IEEE Press, Munich, Germany, March 2001.

[17] S. Leprêtre, P. Gaussier, and J. Cocquerez, "From navigation to active object recognition," in *Proceedings of the 6th International Conference on Simulation of Adaptive Behavior (SAB '00)*, pp. 266–275, Paris, France, August 2000.

[18] T. Lindeberg, "Feature detection with automatic scale selection," *International Journal of Computer Vision*, vol. 30, no. 2, pp. 79–116, 1998.

[19] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[20] K. Mikolajczyk and C. Schmid, "Scale & affine invariant interest point detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, 2004.

[21] M. Seibert and A. M. Waxman, "Spreading activation layers, visual saccades, and invariant representations for neural pattern recognition systems," *Neural Networks*, vol. 2, no. 1, pp. 9–27, 1989.

[22] J. Crowley, O. Riff, and J. H. Piater, "Fast computation of characteristic scale using a half-octave pyramid," in *Proceedings of the International Workshop on Cognitive Computing (Cogvis '02)*, Zurich, Switzerland, October 2002.

[23] A. Torralba and A. Oliva, "Statistics of natural image categories," *Network: Computation in Neural Systems*, vol. 14, no. 3, pp. 391–412, 2003.

[24] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '03)*, vol. 2, pp. 257–263, Madison, Wis, USA, June 2003.

[25] E. L. Schwartz, "Computational anatomy and functional architecture of striate cortex: a spatial mapping approach to perceptual coding," *Vision Research*, vol. 20, no. 8, pp. 645–669, 1980.

[26] D. Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, W.H. Freeman, San Francisco, Calif, USA, 1982.

[27] R. A. Brooks and L. A. Stein, "Building brains for bodies," *Autonomous Robots*, vol. 1, no. 1, pp. 7–25, 1994.

[28] P. G. Schyns and A. Oliva, "Dr. Angry and Mr. Smile: when categorization flexibly modifies the perception of faces in rapid visual presentations," *Cognition*, vol. 69, no. 3, pp. 243–265, 1999.

[29] E. Huck, B. Miramond, and F. Verdier, "A modular systemC RTOS model for embedded services exploration," in *Proceedings of the 1st European Workshop on Design and Architectures for Signal and Image Processing (DASIP '07)*, Grenoble, France, November 2007.

[30] Altera Corp, "Creating Multiprocessor Nios II systems, ver. 1.3," December 2007, http://www.altera.com/literature/lit-nio2.jsp.

[31] I. Benkhermi, A. Benkhelifa, D. Chillet, S. Pillement, J.-C. Prévotet, and F. Verdier, "System-level modelling for reconfigurable SoCs," in *Proceedings of the 20th Conference on Design of Circuits and Integrated Systems (DCIS '05)*, Lisboa, Portugal, November 2005.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 685139, 16 pages
doi:10.1155/2008/685139

Research Article

A Platform for the Development and the Validation of HW IP Components Starting from Reference Software Specifications

Christophe Lucarz,[1] Marco Mattavelli,[1] and Julien Dubois[2]

[1] *Multimedia Architecture Research Group, Microelectronic Systems Laboratory, Ecole Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland*
[2] *Laboratoire Electronique, Informatique et Image (LE2I), Université de Bourgogne, 21000 Dijon, France*

Correspondence should be addressed to Christophe Lucarz, christophe.lucarz@epfl.ch

Received 6 March 2008; Revised 20 July 2008; Accepted 25 November 2008

Recommended by Guy Gogniat

Signal processing algorithms become more and more efficient as a result of the developments of new standards. It is particularly true in the field video compression. However, at each improvement in efficiency and functionality, the complexity of the algorithms is also increasing. Textual specifications, that in the past were the original form of specifications, have been substituted by reference software which became the starting point of any design flow leading to implementation. Therefore, designing an embedded application has become equivalent to port a generic software on a, possibly heterogeneous, embedded platform. Such operation is getting more and more difficult because of the increased algorithm complexity and the wide range of architectural solutions. This paper describes a new platform aiming at supporting a step-by-step mapping of reference software (i.e., generic and nonoptimized software) into software and hardware implementations. The platform provides a seamless interface between the software and hardware environments with profiling capabilities for the analysis of data transfers between hardware and software. Such profiling capabilities help the designer to achieve different implementations aiming at specific objectives such as the optimization of hardware processing resources, of the memory architectures, or the minimization of data transfers to reach low-power designs.

1. INTRODUCTION

Video and multimedia algorithms have changed a lot since their first achievements in the early nineties. If we take the example of ISO/IEC video coding standards (better known as MPEG), we can observe that each successive codec generation released by MPEG has been substantially more complex than the previous, typically yielding twice the compression efficiency of its predecessor. Due to the growing complexity, the textual specification of recent standards (since MPEG-4) [1] has lost its normative role, being replaced by the reference software implementation as the true normative specification. Any ambiguous interpretation is solved by referring to the software description. Therefore, the generic and nonoptimized sequential software description became the starting point for any implementation of a video standard. Unfortunately, working and reasoning on architectural solutions, such as SW/HW partitioning, on a few tenth of thousands of reference software lines of code is a very time- and resource-consuming task [2]. In the traditional way of designing hardware blocks, designers must find out a suitable architecture and isolate the candidate hardware components. Appropriate test vectors must be also generated to test each of these processing elements. Such sequence of tasks could result in more resource demanding than the pure hardware development itself. Realizing this fact, two initiatives have been taken within the MPEG standardization committee. The first was to develop a generic-optimized reference software version of the standard (MPEG-4 Part 7) [3]. The second was to derive mixed SW/HW descriptions from the reference software. The blocks described in a hardware description language (HDL) are now included in the standard (MPEG-4 Part 9 [4]).

Nowadays, reference software is the true starting point for the implementation of video codecs on embedded systems and usually the reference software is written in C/C++. The first step of such process is to identify from the reference software candidate IP components for implementation in

HW and separate them from what should remain SW and should be optimized for the target-embedded platform. The platform described in this paper supports the designer in the identification and development of the different IP components extracted from the reference software and in the validation and test under the real test vectors. The validation and test is not a simulation stage, but the real execution of the partitioned system on a generic SW/HW architecture. The result does not constitute the final algorithm implementation, but provides useful information for the exploration of the SW/HW design space.

The paper is organized as follows. Section 2 exposes the state-of-the-art of the existing platforms supporting rapid prototyping of video and multimedia designs for embedded implementations. Section 3 presents the concept of the virtual socket platform. Section 4 describes in detail the implementation of the virtual memory extension. Section 5 explains how useful profiling information can be extracted by executing the algorithm on the platform. Section 6 outlines a design methodology for the development of heterogeneous implementations according to different optimizations starting from a reference software by exploiting the features of the virtual socket platforms. Section 7 demonstrates the usage of the platform and associated tools in a real case study: the design and optimization of an MPEG-4 motion estimation module. Section 8 concludes the paper.

2. STATE-OF-THE-ART

Even if C/C++ language is not an appropriate language to specify and model signal processing algorithms in the early steps of the design flow of embedded systems, it is still widely used in several contexts. Consequently, the first problem embedded systems designers have to face is the conversion of the generic sequential C/C++ reference software into a form that begins to include architectural features of the selected embedded platform. For instance, this means to identify functions that can be processed by software or hardware coprocessor units if available and to express them in a form that exploits the available parallelism.

In the early stages of the design flow, designers should have a feeling about the architecture of the system, that is, decide which part of the algorithm must be in hardware or software. Unfortunately, the intuition of the designer becomes not reliable when dealing with complex systems. It may lead to wrong initial decision that affects all other stages of the design states. The possibility of quickly testing possible solutions is a clear advantage to try to find the best architecture. In platform-based design, the entire algorithm must be mapped on the development platform, mapping SW parts on embedded processors and HW parts on FPGA. Platforms are composed of processors, dedicated hardware, and reconfigurable hardware (FPGA). For example, the XUP Virtex-II Pro Development System [5] provides an advanced hardware platform that consists of a high-performance Virtex-II Pro Platform FPGA (with PowerPC 405 cores) surrounded by a comprehensive collection of peripheral components that can be used to create a complex system. Another example of such platform is the Celoxica RC1000-PP PCI board [6].

From such class of platforms, in which the entire algorithm is mapped on the platform itself, the sharing of data between a host PC and the platform is possible [7]. The main program still lies in the embedded processor and data on the host are easily available by means of virtual serial ports. However, the plugging of hardware modules inside the reference software running on the host remains the most difficult task.

The more advanced step is reached by the work of Martyn Edwards and Benjamin Fozard [8]. An FPGA-based algorithm (implemented on an external platform) is activated from the host PC, directly from the reference software. Such platform is based on the Celoxica RC1000-PP board [6] and communicates with the host by using the PCI bus. The main program is on the host processor, sends control information to the FPGA, and transfers data in a small shared memory which is part of the hardware platform. In such case, the designer must explicitly transfer the data necessary for the processing (on the platform) from the host to the local memory. Other works addressing coprocessors and relative coprocessors interfaces have been reported in literature. Some examples are given in [9, 10]. However, the problem of seamlessly plugging hardware modules is not yet solved and the specification of the data transfers remains to the charge of the designer.

The problem appears when dealing with large amount of data and irregular accesses, as it occurs in complex data-dominated video or multimedia systems. Explicitly specifying all the data transfers might be a very burdensome task. In some works on coprocessors, data transfers can be generated automatically by the host like, for instance, in the case reported in [11]. However, data are copied in the local memory at a predefined location. Thus, the HDL module must be aware of the physical addresses of the data in the local memory. Again, the management of the addresses can be a nontrivial and resource consuming task when dealing with complex algorithms.

The virtual socket concept has been presented in [12–14] and has been developed to support the mixed specification of MPEG-4 Part 2 and Part 10 (AVC/H.264) specifications in terms of reference SW including the plug-in of HDL modules [4]. The platform is constituted by a standard PC, where the SW is running and a PCMCIA card that contains an FPGA and a local memory. In this platform, HDL modules on the platform can be started directly from the reference software, but the data transfers between the host memory and the local memory on the platform must be explicitly specified by the designer/programer.

In conclusion, testing the behavior of the implementation in HW of parts of a reference software is not a trivial task. It is very resource-consuming in term of design efforts, but is very attractive as a design exploration methodology. Designers need a framework in which it is easy to make seamless call the hardware components directly from the reference software and test the performances of the HW modules without worrying too much about low-level implementations details. This is possible only if the

hardware components are closely linked to the reference software environment. Some HW/SW codesign platforms can be used to support the algorithm-architecture adaptation methodology [15], but all of them lack a simple procedure capable of plugging seamlessly hardware modules described in HDL within a pure reference software. The problem of the current platforms is that either the memory management is a burdensome task or the call to the hardware modules is done by an embedded processor on the platform.

3. OVERVIEW OF THE VIRTUAL SOCKET PLATFORM

As a possible solution to the problem described in Section 2, this paper presents an extension of the virtual socket platform [12–14] capable of easily plugging and calling hardware modules directly from the reference software without worrying about the data transfers between the host and the hardware modules. Specifying explicitly the data transfers would not constitute a burdensome task when dealing with simple deterministic algorithms for which the data required by the HDL module are known exactly. Data transfers cannot be explicitly specified in advance by the designer in the case of complex designs, where design tradeoffs are much more convenient and viable than worst case designs. The idea is not to build a final embedded system because the underlying hardware used in the platform (wildcard and PC processor) is very different from the hardware components of the platform used for a real implementation. The idea is to provide a framework to transform seamlessly C/C++ reference software into a mixed HW/SW representation (C/C++ and VHDL). The step-by-step transformation method is especially interesting when dealing with large reference software packages composed of several tens of thousand lines of codes. The work described here allows the designer to transform step by step a large reference software package into a mixed HW/SW without worrying about data transfers between the software and hardware environments.

Given a reference software and a given partitioning between SW and HW, so as to test each HW candidate module separately, the designer executes some parts of the reference algorithm using the host processor and runs the HW module on the virtual socket platform which is the implementation support for the hardware. So as to easily handle such mixed HW/SW specifications, it is very convenient that the HDL module and the C/C++ functions have access to the same user memory space. This latter is part of the host software and contains all the data to be processed. Such host memory space is trivially available by the processor which executes the reference software, but it is much less evident for the virtual socket platform which is the support for the HW modules and lies on the FPGA.

The extension consists in adding virtual memory capabilities to enable automatic data transfers between the host, running the SW part and the platform, running the HW modules. Thus, a portion of software reference code can be easily replaced and executed by an HDL module without the need of specifying any data transfer explicitly. HDL modules are started directly from the reference software.

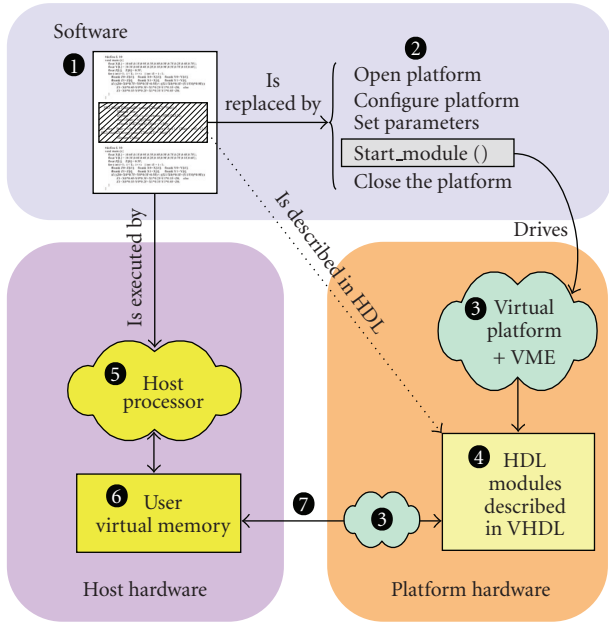


Figure 1: Relations between the unified virtual memory, the reference software algorithm, and the HDL module.

Having a shared memory-enforced by a cache-coherence protocol—between the host PC running, the SW sections, and the platform running HW modules—avoids the need of specifying explicitly all the data transfers. The clear advantage of this solution is that the data transfer needed to feed the HDL module can be profiled so as to explore different memory architecture solutions. Another advantage of such direct map is that conformance with the original SW specification is guaranteed at any stage and the generation of test vectors is naturally provided by the way the HDL module is plugged to the SW section. This platform can help in the design of low-power designs thanks to the profiling of data transfers [16] and enable algorithm-architecture codesign methodology to build complex multimedia systems [17].

3.1. *Principle*

Figure 1 illustrates the principle of the platform with its extension and shows the interactions between the unified virtual memory (6), the reference software (1), and an HDL module (4).

For a given algorithm described in a mixed C/C++ and HDL form (1), the software parts are executed by the host processor (5). The hardware parts are executed by the HDL modules described in VHDL (4) implemented on the FPGA. In the original version of the platform [12–14], hardware modules only have access to the local memory on the platform and all the data transfers from the host to the local memory have to be explicitly specified in advance by the designer. It is needless to underline how difficult this task can become when the designer deals with complex signal processing algorithms such as MPEG video codecs. Such operations can be especially error prone when the volume

of data is large compared to the small size of the local memory. With the extension, the HDL modules (4) on the FPGA and the C/C++ functions (1) on the host processor (5) have access to the same user memory space (6). The part of C/C++ code of the reference software the designer intends to execute in hardware is replaced by the hardware call function (2). The `Start_module ( )` function drives the virtual socket platform and its extension (3) to trigger the execution of the HDL module (4). The platform (3) manages the data transfers between the main memory (6) and the local memory on the platform (3-4).

3.2. Architecture of the platform

The virtual socket platform is composed of a PCMCIA card and a set of software libraries in C to enable communications between the PC and the platform. The PCMCIA card contains an FPGA, memories, and an interface for the host computer (PC). The virtual socket platform handles the communications between the HDL modules implemented on the FPGA on the PCMCIA card and the host PC. Figure 2 shows the connections between the HDL modules, the host PC, and the virtual socket platform. The virtual memory extension comprises the window memory unit (WMU) and the virtual memory controller (VMC) hardware blocks and a software library (virtual manager window) not represented on the figure.

The extension is capable of automatically handling the data transfers and is implemented with three components. The window manager unit (WMU) hardware module translates virtual addresses into physical addresses. The virtual memory controller (VMC) hardware module is a kind of switch in order to manage two different modes: the original and virtual modes. These modes correspond to the type of access the HDL module asks for. If the HDL module sends a physical address, the explicit mode is active. This mode is the one implemented in the original platform. For further detail, the reader can refer to [12–14]. If the HDL module sends a virtual address to the virtual socket platform, the virtual mode is active. In the virtual mode, the cache-coherence protocol guarantees the coherency between the main and local memories. This protocol is implemented in the window manager unit (WMU) using a translation lookaside buffer (TLB) in cooperation with the virtual manager window (VMW) software library. The designer is free to choose one of the two modes for requesting data from the HDL module.

3.3. Integration of an HDL module into the platform

The platform supports the call of an HDL module from the reference software running on the PC. The first part explains how such a call is written inside the reference software. The second part explains how the VHDL code from the designer must be inserted in the platform.

3.3.1. Call of the HDL module from the reference software

the segment of C code of the reference software the designer wants to execute in hardware using a specific module is replaced by the hardware call function with optional parameters to configure the module. The hardware call function is composed of the following simple steps.

(i) *Open and configure the platform*: the designer configures the platform by using the `Platform_Init ( )` and `VMW_Init ( )` functions from the virtual socket API and VMW API.

(ii) *Parameters*: the designer sets a given number of parameters needed for the configuration of the HW module. Sixteen parameters are available for each HW module.

(iii) *Start of the HDL module*: the HDL module is started with the `Start_module ( )` function which sends first the parameters to the HDL module and then triggers the execution. This function is part of the VMW API. During the execution, the module sends requests to the platform to obtain data. These requests are treated by the virtual socket platform and the virtual memory extension to feed the HDL with the correct data coming from the unified virtual memory.

(iv) *Close the platform*: when the entire job is finished, the platform is closed.

The hardware call function is shown as in Algorithm 1.

3.3.2. The wrapper around the hardware module

the designer intends to test a hardware module corresponding to a piece of reference software. This hardware module has specific inputs and outputs. Around this hardware module, a wrapper provides the link between the hardware module and the virtual socket platform. Such wrapper is connected to the virtual socket platform with the standard interface and to the hardware module under test with its specific inputs and outputs. The wrapper appears as a finite state machine (FSM) and feeds the hardware module with data coming from the virtual socket platform through the standard interface. The wrapper is specific to each hardware module under test because only the designer knows how to feed the data into this hardware module. The FSM must respect the constraints of the hardware module in terms of data and timing. Consequently, there are several ways to integrate the hardware module into the platform.

(i) Designers can use FIFO in front of the module to test the true performances of the hardware module. FIFOs are necessary to avoid that the hardware module waits for data. Using the profiling feature of the platform, designers can test the performances of the module.

(ii) Designers can connect directly the interface of the platform to the inputs and outputs of the hardware module in order to test the functionality. Thus, the profiling tool indicates at which moment the data are accessed. In this case, the hardware module under test must be able to wait for data at their inputs.

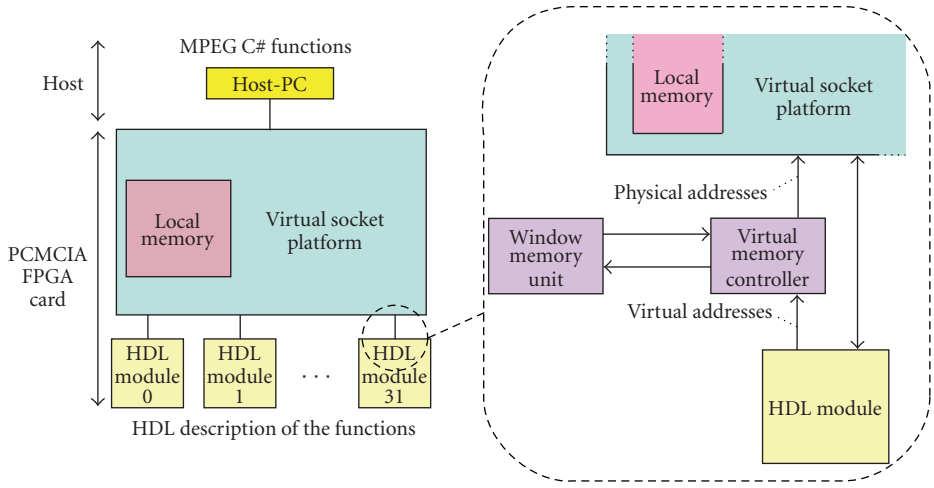


Figure 2: Overview of the virtual socket platform.

```
int main (int argc, char *argv[]) {
/* [...] Reference Software Algorithm stops here */
/* Beginning of the HDL module calling procedure */
/******* OPEN AND CONFIGURE THE PLATFORM *******/
  Platform_Init ( ) ;     // Virtual Socket
  VMW_Init ( ) ;          // Virtual Memory Extension
/******* PARAMETERS *******/
  Module_Param.nb_param = 4 ;     // number of parameters
  Module_Param.Param [0] = A ;    // parameter 1
  Module_Param.Param [1] = B ;    // parameter 2
  Module_Param.Param [2] = C ;    // parameter 3
  Module_Param.Param [3] = D ;    // parameter 4
/******* HDL MODULE START *******/
  Start_module (1, &Module_Param) ;
/******* CLOSE THE PLATFORM *******/
  VMW_Stop ( ) ;          // Virtual Memory Extension
  Platform_Stop ( ) ;     // Virtual Socket
/* End of the HDL module calling procedure */
/* [...] the Reference Software Algorithm continues */
}
```

Algorithm 1

Figure 3 illustrates the links between the wrapper, the virtual socket platform, and the hardware module under test. In order that a hardware module can be tested using this platform, minor constraints need to be considered.

(i) The HDL module must have a `start_process` signal as an input to trigger its execution.

(ii) The HDL module must have a `process_finished` signal as an output to indicate when the module finishes.

(iii) In the case where the designer tests the functionality of the module, this latter must accept latencies, the time for the data to come from the main memory through the virtual memory mechanism.

(iv) In the case in which the designer tests the performances of the module, the wrapper must contain FIFOs at the inputs and outputs so that the hardware module does not presents latencies.

4. IMPLEMENTATION OF THE VIRTUAL MEMORY EXTENSION

This section describes the implementation of the virtual memory extension (VME) based on the original virtual

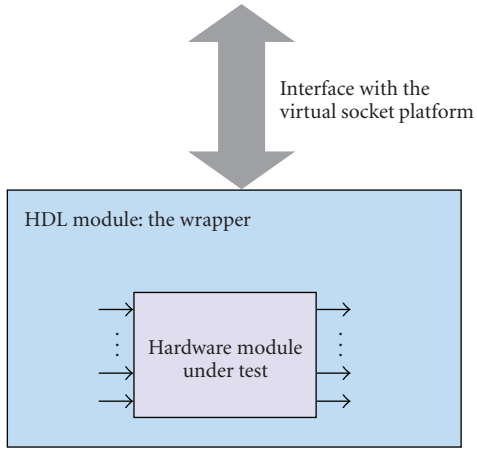


Figure 3: The HDL module comprises a wrapper and the hardware module under test.

socket platform. The first and second parts describe the window manager unit (WMU) and the virtual memory controller (VMC) hardware modules, respectively. A third part describes the virtual memory window (VMW) software library. Finally, the last part explains in detail how the virtual memory extension operates.

4.1. The window manager unit (WMU) hardware module

The WMU is a component designed by Vuletić et al. [18]. It translates a virtual address into a physical address and is the main component responsible for the cache-coherence protocol, maintaining memory coherence between the main memory of the host and the local memory on the platform. In our framework, the virtual addresses refer to the unified virtual memory space ((6) in Figure 1) and the physical addresses refer to the local memory on the platform (see left part of Figure 2). The WMU is composed of one major element: the translation lookaside buffer (TLB) which is built with a content addressable memory (CAM). It stores the status of the data of each page. When a virtual address inputs the WMU, this latter is translated into a physical address. This latter is composed of an offset and the page number (among the 32 pages available). The offset corresponds to the location of the data in a given page. If the CAM search yields a match between the page number of the physical address and an entry of the CAM, it means that the data are already present in the local memory. If no match exists, the VMW library will intervene in order to copy the data required and write the new entry in the page table in the TLB.

4.2. The virtual memory controller (VMC) hardware module

The VMC manages the two available modes: the virtual and explicit modes. In the virtual mode, the virtual addresses must be translated into physical addresses. Additional operations must be done compared to the original protocol (the address translation, e.g.). The VMC intercepts some signals in the standard interface between the HDL module and the platform and sends these signals to the WMU which executes the translation of the addresses. The signals intercepted by the VMC are the address, the count (amount of data requested by the HW module), and the strobe signals.

4.3. The virtual memory window (VMW) software library

The VMW library is built on top of the FPGA card driver (wildcard II API), the virtual socket API developed by Yifeng Qiu and Wael Badawy (based on [12, 13]), and the WIN32 API. The VMW library is in charge of transferring the data from the main memory of the host to the local memory on the platform. The WMU raises an interrupt when such a transfer is necessary. In this case, the VMW fills the pages of the local memory with the requested data coming from the virtual memory. The local memory is composed of 32 pages of 2 kB. When the data on the local memory is dirty and must be replaced by new data, the old data are copied back in the main memory.

4.4. Mechanism

The goal of the virtual memory extension is to support the direct access of HW modules to the software virtual memory space located on the host PC. The HDL module can ask for four types of access.

(i) Read data from the local memory on the platform.

(ii) Write data to the local memory on the platform.

(iii) Read data from the unified virtual memory space on the host.

(iv) Write data to the unified virtual memory space on the host.

The first two requests belong to the *explicit mode* in which the HDL module sends physical addresses (relative to the local memory). This mode is already implemented in the original version of the platform. It will not be detailed in this paper, the reader can refer to [12–14] for further information.

The last two requests belong to the *virtual mode* in which the HDL module sends virtual addresses (relative to the unified virtual memory). In the virtual mode, the addresses are the one of the data in the unified virtual memory space. In other words, it enables the HDL module to have a transparent access the host memory.

The following paragraphs explain in more details how data in the virtual memory of the host is accessed from the HDL without any intervention of the designer. The protocol to write and read in the unified memory space are very similar and for the sake of clarity, only the read protocol is described.

4.4.1. Send the pointers

the piece of reference code the designer wants to execute in hardware is replaced first by the hardware call function (see Section 3.3.1) in order to call the hardware module. This latter needs data to work and the designer must specify which data must be used. Pointers of the data are sent to the HDL module by using parameters. For example, if the HDL module needs two images as input, the designer specify the pointers to the images using the parameters in the hardware call function as in Algorithm 2.

When the execution of the hardware module is triggered with the `Start_module ( )` function (part of the hardware call function), the wrapper of the HDL module receives the parameters (pointers) because it is the wrapper which is connected to the virtual socket platform (see Figure 3). According to these pointers, the wrapper can generate the requests to the virtual socket platform to get the data.

4.4.2. The requests of the module

the wrapper can generate the requests (i.e., the addresses) from the pointers it just received. The way the requests are done depends on how the data must be inputted in the HDL module. In image processing, for example, there are different ways to read an image, raster or blocks. If the HDL module needs a raster format, the wrapper sends the necessary requests in order that the data received from the platform are in a raster format. The HDL module can send either an explicit or a virtual request, that is, physical addresses or virtual addresses. The use of the two modes is further detailed in Section 6. The wrapper uses the existing protocol of communication with the virtual socket to make the requests (explicit or virtual). A read request consists in asserting the following signals of the interface.

(i) `hw_mem_hwaccel`: number of the hardware module.

(ii) `hw_offset`: read address.

(iii) `hw_count`: number of data to read.

(iv) `memory_read`: strobe signal of the request.

A write request consists in asserting the following signals of the interface.

(i) `hw_mem_hwaccel1`: number of the hardware module.

(ii) `hw_offset1`: write address.

(iii) `hw_count1`: number of data to write.

(iv) `output_valid`: strobe signal of the request and the data.

4.4.3. Mode management

according to the type of request sent by the module, the VMC will send the address, count, and strobe signals to the WMU or not. If the request is explicit, the signals go directly to the virtual socket platform and this latter sends the data to the HDL module according to the initial protocol in the original platform (see [12–14]). But if the access is virtual, the requested address is virtual and it must be translated into a physical address so that the platform can send the corresponding data to the HDL module. The WMC catches some signals from the standard interface and sends them to the WMU which translates this virtual address into a physical one.

4.4.4. The translation of the address

the WMU receives a virtual address from a HDL module through the VMC. The WMU translates this virtual address into a physical address. The translation mechanism is illustrated in Figure 4.

The virtual address is 32 bits long. The page offset is 11 bits long because the size of a page in the local memory is 2 kb. Thus, the pattern to be translated is a word of 21 bits. The translation consists in simply replacing the pattern by a page number, knowing that the replacement strategy is FIFO. The first request of the HDL module is going to result in a miss because there is no data in the local memory yet. Thus, the VMW will fill the first page of the local memory with the requested data from the main memory. When the WMU detects a new missed request, the second page of the local memory is filled, and so on. When the 32 entries are complete, the first page is used again; it is an FIFO strategy. If the status of this page is dirty, the VMW will copy back the data to the main memory before replacing the old data by the new one.

Once the address is translated, the WMU checks if the data corresponding to the virtual address is already present in the local memory on the platform. If yes, the WMU sends the corresponding physical address to the virtual socket platform. If not, the WMU raises an interrupt and asks the VMW to copy the requested data in the local memory. The platform knows if the requested data is already present in the local memory thanks to the cache-coherence protocol which guarantees the coherence between the main memory and the local memory. The check is done by verifying the status of the data contained in the local memory thanks to the TLB/CAM (see Section 4.1).

4.4.5. The automatic transfer of the data

if the WMU raises an interrupt, the requested data in the main memory corresponding to the virtual address is copied in the local memory. The VMW library transfers all the data between the main memory (the unified virtual memory) and the local memory. The collaboration of the WMU and the VMW implements a cache-coherence protocol. This latter keeps the status of the data of the local memory on the platform.

(i) Dirty: data copied on the local memory and modified by the HDL module.

(ii) Valid: data copied on the local memory and not modified by the HDL module.

(iii) Invalid: no data copied at this address.

```
Module_Param.nb_param = 2 ;      // number of parameters
Module_Param.Param[0] =  &img1  ;  // pointer to image no. 1
Module_Param.Param[0] =  &img2  ;  // pointer to image no. 2
```

Algorithm 2

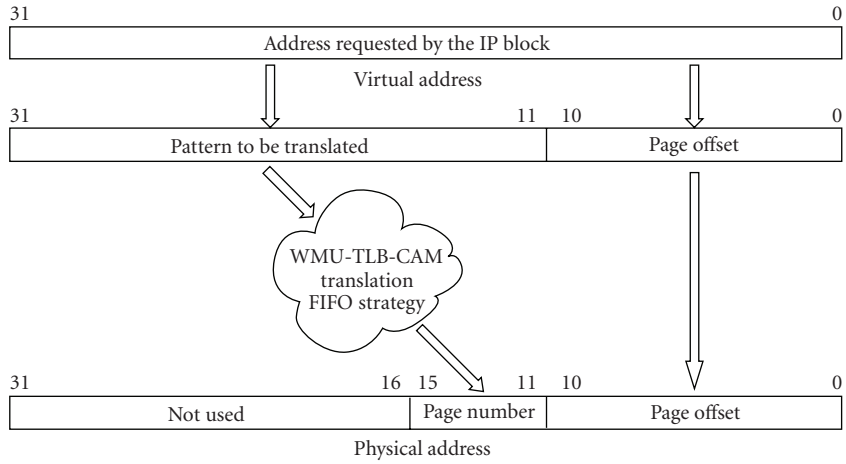


Figure 4: The translation of a virtual address into a physical address.

4.4.6. Answer to the request of the module

when the data has been transferred, the WMU sends the corresponding physical address to the virtual socket platform which sends the data (which are now in the local memory) to the HDL module. When the job of the HDL module is entirely finished, the data processed by the HDL module are copied back in the host memory in order that the reference software can continue running with updated data, created and/or modified by the execution of the HDL module. Thanks to the VME, the data are updated without any intervention of the designer.

Finally, from the designer point of view, using the virtual memory extension, the whole process of data transfers is completely transparent. The only issue the designer has to care for is to generate the virtual addresses accordingly to the data contained in the host memory space. The whole task of transferring data to local memory is done by the platform and its software support.

5. PROFILING FEATURES

Once the HDL module is correctly called by the reference software and executed on the platform, it would be interesting to get useful information on the execution of the hardware module. The profiling feature of the platform can provide such information by recording the data accesses between the HDL module and the virtual socket platform. Since in the field of signal processing, algorithms are essentially data driven, optimizing the data transfers between the different components is crucial for such signal processing systems. Data transfers provide also a relevant contribution to the overall power dissipation in embedded systems. Consequently, these transfers must be carefully optimized in order to be successful in low-power designs.

Table 1 is an example of profiling information given by the platform. The column **Rd/Wr** indicates if the request of the module is a read (r) or a write (w). The column **Addr** contains the address of the data requested by the module. The column **Count** indicates how much data is requested by the module from the address specified in the previous column (kind of a burst). The column **Clk** indicates at which clock cycle the request appends. The origin corresponds to the time at which the module begins its processing. The column **event** indicates the type of event happening. "v" corresponds to a virtual access, "o" corresponds to an explicit access, and "d" indicates the end of the job of the HDL module.

The example corresponding to Table 1 consists of copying 1000 dwords from one place in the main memory to another place in the same memory. The HDL module asks for reading 255 data from the address 0×366408 of the main memory. Once the HDL module received the data, it copies them to the new place in a main memory by asking for a writing request at address $0 \times 3673D8$. The 1000 dwords are copied by sets of 255 dwords. Each virtual access requests a time overhead. In virtual mode, each virtual access requests 4 cycles. Therefore, between a read and write operations 1024 cycles are requested. The time (derived from the clock cycles information) recorded by the platform is the time at which the HDL module asks for data. When a request is sent by the hardware module to the platform, the time is "stopped" until the platform answers and data are inputted in the HDL module. Thus, the profiling tool does not take into account the time taken by the platform to feed the HDL module with

Table 1: Example of profiling information given by the platform.

Rd/Wr	Addr. (hex)	Count	Clk	Event
r	0x366408	255	0	v
w	0x3673d8	255	1024	v
r	0x366804	255	2048	v
w	0x3677d4	255	3072	v
r	0x366c00	255	4096	v
w	0x367bd0	255	5120	v
r	0x366ffc	239	6144	v
w	0x367fcc	239	7168	v
–	–	–	8192	d

the requested data. It is as if the data arrived immediately after the request. However, the time elapsed for the data transfer from the local memory to the module is taken under account. The profiling tool aims at studying how the HDL modules asks data and not how it communicates with the rest of the system because the architecture of the platform will never be implemented in a real embedded system.

The profiling feature is implemented in hardware (WMU) and in software (VMW). The WMU raises interruptions for each data transfers between the HDL module and the virtual socket platform. The VMW library handles these interruptions and records the profiling information in a simple text file. It catches the information presented in Table 1.

6. DESIGN METHODOLOGY

6.1. Objectives

The main goal of the virtual socket platform and its extension is to ease the communications between an SW environment (i.e., a PC) and an HW module (i.e., an FPGA) for the rapid evaluation and profiling of IP blocks derived from a reference software specification. These IP blocks are supposed to be used in the final implementation. In other words, this platform is a support in the process converting large C/C++ reference software packages into mixed HW/SW description/implementations. A step-by-step approach is the methodology adopted so as to reach the desired final implementation. By using the virtual memory extension, the data transfers between the SW environment and the HW modules are handled automatically. If the designer desires to convert entirely the reference software into a hardware implementation, he can replace step-by-step pieces of the C/C++ code with HW modules and proceed with this approach until the entire SW algorithm is converted and integrated in hardware. If the designer desires to target the conversion to a mixed representation of the reference software, he can stop at any intermediate step. Section 7 describes a methodology based on the profiling features of the virtual socket platform to reach optimized implementations.

6.2. Methodology

Figure 5 illustrates the different steps of the proposed methodology.

The *first step* (called "*Building the module*") consists in partitioning the C/C++ reference software into software and hardware partitions. The methodology that identifies these partitions is not the scope of this work and is not further discussed here. The second task consists in writing the hardware modules in HDL. This can be done manually or by using any other C to HDL tools and methodologies, according to the preferences of the designer. The third task consists in inserting the HW module by replacing the corresponding code by the hardware call function (see Section 3.3.1) with parameters if necessary (e.g., the virtual addresses of the data used by the module). The HDL module must be also wrapped in order to satisfy the communication protocol with the platform (see Section 3.3.2).

The *second step* (called "*Conformance tests*") consists in checking the conformance of the hardware module with respect to the reference software. The equivalency of the C/C++ and HW modules is checked relying on the virtual socket platform and the virtual memory extension in the virtual mode. The virtual memory feature simplifies the conformance tests procedures because the designer does not have to care about the data transfers between the main memory and the local memory because this stage is guaranteed by the platform. The conformance check is done by comparing the results generated by the reference software and the HW module. Such verification is done directly in the reference software environment. No profiling is needed at this step because the designer only checks the conformance of the module and not its performances.

The *third step* (called "*Optimization*") consists in optimizing the HW module by using the virtual mode and the profiling features of the platform. The designer can grasp an overview of the data transfers exchanged between the platform and the HW module. The way data are accessed can be the object of further optimization steps. Updates of the internal algorithm of the module can affect the way data are accessed, the amount of requested data and the timing at which the transfers are done. This phase helps the designer to refine the HDL code of the HW module. Using the support provided by the virtual memory extension, the designer can forget about the data transfers between the reference software and the HW module and has only to specify to the module the virtual addresses of the data used by the module. According to the profiling results, the designer faces different cases. The time elapsed for the data transfers and the processing time can be measured by the platform. According to these values, three cases are possible.

(i) Transfers < computations: the processing is too long compared to the data transfers. The optimization effort must focus on the processing in order to reduce the processing time.

(ii) Transfers = computations: the system is balanced. While the transfers occur, the processing is executed. When this latter finished, a new set of data is

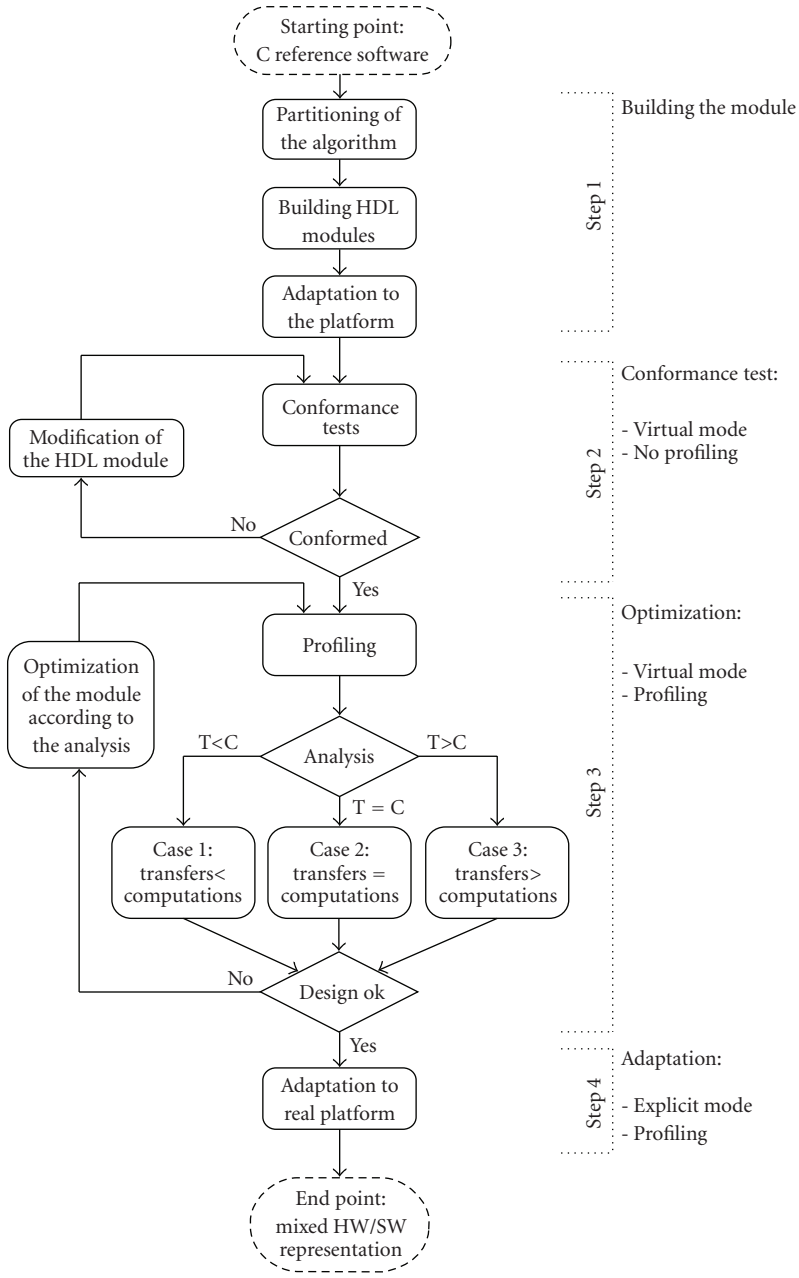


Figure 5: Flow chart representation of the design flow steps using the virtual socket platform.

immediately available at the same time. There is no loss of efficiency.

(iii) Transfers > computations: the module is waiting for data. The designer can choose either to reduce the amount of data used by the module or to implement an equivalent processing that is slower (by adding algorithmic complexity so as to get better results if possible and convenient) or by reducing resource usage or power dissipation whenever appropriate.

According to the overall design policy of the system, the designer can take appropriate design decisions by analyzing profiling results given by the platform. Either the design fills the requirements and the designer continues with the last step, or the design does not fill the requirements and he modifies the module until it is satisfactory (iteration of the third step).

Once the design is satisfactory, the designer can enter the *fourth step* (called "*Adaptation*"). This step consists in adapting the HW module to a real target platform. The explicit mode is used at the place of the virtual mode. At this step, the data exchanged by the HW module and the platform are well-defined. Data transfers are expressed explicitly using physical addresses. At the end of this step, the HW module sends physical addresses, relative to the final implementation design.

7. CASE STUDY: DESIGN OF AN MPEG-4 MOTION ESTIMATION MODULE

This design case intends to show how the virtual socket platform can be used to explore algorithmic/architecture tradeoffs so as to achieve the highest possible performances under bandwidth and processing constraints. In this context, an example of the implementation of a motion estimation with a reduced search strategy HW component is developed. The platform structure, and the processor possibilities, represents a real advantage for the effective implementation of a reduced search strategy algorithm that matches the HW platform capabilities. The performances of an HW block can be considerably increased and optimized using the described methodology and the profiling results obtained by the platform. The profiling information extracted by the virtual socket platform (specifically the timing performance) is investigated by using the virtual memory accesses mode.

7.1. Motion estimation in a video encoding context

The motion estimation is well known to be the most computation-intensive stage of video coding process and has been the subject of many research works (on both algorithmic and architecture sides) which aim at reducing the implementation complexity. For brevity, we cannot here provide an accurate review of the wide literature on the subject, we suggest referring, for instance, to [19, 20], and their references, for an updated review of the state-of-the-art. We just remind the main families of approaches developed so far.

The first family is the so-called "reduced search algorithms," which aims at reducing the complexity by limiting the measure of the matching criterion to only a (small) subset of candidate vectors in the search window. The key element here is the "intelligence" of the search algorithm that may completely change depending on the requirements of the video coding application (portable video telephony, HDTV for sport events, etc.).

A second family is constituted by the approaches in which the complexity reduction is achieved by using multiresolution searches on subsampled image search windows.

A third approach is to use simplified matching criteria in place of the classical maximum absolute difference (MAD) criterion.

A fourth family of approaches is based on various preprocessing stages that reduce the images to binary images for which simple XOR operations are used for the evaluation of the MAD.

7.2. On the choice of the design case study

For most of the algorithms, composed of one or more algorithmic elements from the different families sketched above, dedicated optimized implementations using systolic arrays and/or specific data flows handling are needed to achieve effective performances. While a lot of effort has been devoted to developing such reduced complexity solutions (search algorithm plus data flow implementation), much less effort has been devoted to study how to be able to scale the solutions versus the different parameters such as the size of the search window, the available memory bandwidth (that usually is an off-chip memory), and the processing power. Most of the solutions require a close coupling between data flow handling and the dedicated hardware. Nowadays, with the appearance of powerful processors with specialized instruction sets and new families of FPGA with embedded arithmetic for which the MAD evaluation is no longer a difficult burden, some of the reduced complexity solutions presented in the past have lost their interest. Modularity, flexibility to cover the different coding applications, and the possibility of upgrading using the more recent results and improvements are more desirable and possible implementation objectives.

For such reasons in this case study, we focus on architectures that present a wide degree of flexibility and modularity of the different elements versus the various performance and implementation parameters of motion estimation so as to be able to cover different applications ranging from high-quality HDTV up to mobile video. The family of approaches supported is the reduced search algorithms on a specific search window. The recent results, including the comparison with other approaches, have shown that using appropriate (for the video application) reduced search algorithms is possible to achieve optimal coding results [21]. The main idea of the architecture so as to achieve the desired level of flexibility and reprogramability for the search is thus to separate the data access and the matching criterion implementation stages from the intelligence of the algorithmic search. Therefore, the coprocessor architecture, depending on the support platform and matching criterion used, can be programed freely by the user at high level with his preferred intelligent search, exploiting the resource budget in terms of available candidate vectors for each search. The implementation objective of the architecture is not only an efficient hardware implementation not only to speed up matching operations, but also to provide the possibility of randomly matching any area in a search window without any other limitation on the access sequence. The random-block access in the search window is obtained with a specific data flow architecture and address generation strategy. This accelerator has been presented in [19]. The flexibility of the hardware accelerator supports all the different motion estimation modes which appear in the recent extension of the MPEG video standard family such as AVC. The search-window size is parametric so that it can be configured according to the search window size of the profile under consideration or reduced according to the desired application.

7.3. Design flow

This section describes the application of the proposed methodology supported by the virtual socket platform to the design of the motion estimation module that satisfies the objectives described above. The sections follows the design flow described in Figure 5.

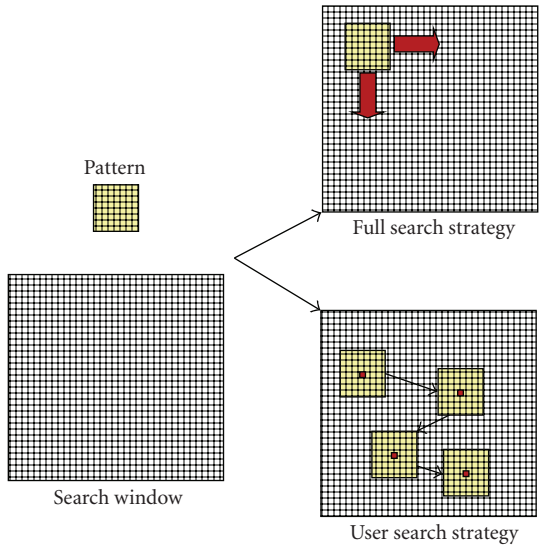


Figure 6: Full-search and reduced-search strategies.

7.3.1. Building the module

the conception and the integration of the HW accelerator represent the first step of the design flow. Therefore, the first step in the design flow must be the partitioning of the algorithm. The motion estimation takes place in the MPEG encoder, described in the reference software which is the starting point of the implementation. The piece of code relative to the motion estimation process is identified in the reference code, and then implemented. A full-search algorithm is implemented in a first time. The motion estimation process is regular, as illustrated in Figure 6, and all possible positions of the pattern in the search window are searched. In this case, the number of matching depends on the sizes of the pattern and of the search window (i.e., for 16 × 16 pattern and a 56 × 40 search window, 1025 matching are needed). Obviously, with such high number of operations (i.e., >1000 matching), the motion estimation represents then the most computation intensive stage of the video encoding. During a matching process, the architecture does not support any interruption. Therefore, the wrapper of the module requests an FIFO memory in front of the data interfaces (input and output). Finally, the HW module is integrated into the platform. The piece of code in the reference software relative to motion estimation is replaced by the hardware call function (see Section 3.3.1). The fundamental information, as the sizes the addresses of the pattern and the search window, is transferred to the HW module through the hardware call function and its parameters structure.

7.3.2. Conformance tests

since the HW module has been correctly designed and produces the requested results, the second step is completed. The conformance of the module is tested in the complete software environment by testing the results of the reference software using the results of the HW module. As far as the results of the HW module are not matching the reference software, the loop must be iterated by debugging and redesigning the module until it is correct. This step can be considered as an HW debugging step, and can complete the common simulation phase.

Table 2: Profiling results for the full-search algorithm.

Rd/Wr	Addr. (hex)	Count	Clk	Event
r	0x366408	64	0	v
r	0x366538	560	256	v
w	0x362f90	1	20950	v
–	–	–	20959	d

7.3.3. Optimization—first iteration

the third step consists in analyzing, profiling, and optimizing the HW module. The profiling results with a full-search methodology are obtained for the implementation setting on a search window with a size of 56 × 40 and 16 × 16 blocks. The frequency of the motion accelerator is 50 MHz. The study of the profiling highlights several aspects. Table 2 shows a fraction of the profiling results for this configuration. At the beginning of the processing, the FIFO receives the input data. The pattern is transferred and followed by the search window. The two first rows of Table 2 represent these two transfers. The data is stored on the hard disk space, therefore virtual accesses are required. The pointers of the pattern and the search window are used by the FSM of the wrapper to generate all the required addresses. As 32 bits (i.e., 4 pixels) are transferred at each access, the pattern and the search window are transferred, respectively, in 64 and 560 accesses. Due to the virtual mode overhead, each access is performed in 4 cycles. At the end of the processing, one write access enables the resulting motion vector to be stored into the output FIFO. The whole processing is completed in 20950 cycles. As the data transfers (pattern and search window) are done in 12480 nanoseconds (i.e., 624 accesses) and the whole matching processing (the 1025 possible matching) is finished in 369000 nanoseconds, the profiling confirms that a simple matching is processed at this frequency in 360 nanoseconds.

Moreover, the profiling results confirm that the processing represents the major part of the require time. With a higher resolution (for HDTV, e.g.), the size of the search window increases and the processing time even more. The processing stage must be improved in priority. Consequently, the search strategy is modified and the HW module modified.

7.3.4. Optimization—redesign

the architecture refinements are applied to implement a reduced search algorithm so as to decrease the processing time of the module.

Contrary to the full-search strategy, a reduced search aims at minimizing the number of matching, in this variant

the objective is achieved by using previous matching information (i.e., the values of the vectors previously computed in time and space). As mentioned previously, using appropriate (for the video application) reduced search algorithms, it is possible, at the same time, to achieve optimal coding results [21] and reduce the processing time for each macroblock motion estimation. Figure 6 represents a reduced search (4 matching represented) strategy versus a full-search one.

The list of candidate matching to process is reduced according to the user's search algorithm. The motion estimation accelerator receives this list of candidates and processes only the listed positions. The possibility of the HW module of randomly matching any area in a search window without any other limitation on the access sequence is exploited in a first phase. In other terms, even with a random access, the performance per matching is not changed.

Therefore, the MAD matching implementation has not been modified. The implemented pipelined architecture enables to match a block and a portion of search window, row by row. The modification is done on the data structure to guarantee the data access. The data structure modification is detailed in a previous work [19]. Moreover, the address generation process is modified. The address generation is provided by a flexible unit to enable the translation of the user search strategy into the hardware setup. Contrary to previous work, this unit is not included into the HW accelerator (FPGA or embedded processor), the host processor is in charge of the generation to increase the system flexibility and to take advantage of the virtual memory mode in the different configurations to be tested (e.g., different sizes of the search window).

To integrate the HW module, the wrapper is modified by including an FIFO memory dedicated to matching addresses in front of the address interface. Moreover, a control register is added to start the accelerator or to define the number of matching. The resulting wrapper is presented in Figure 7.

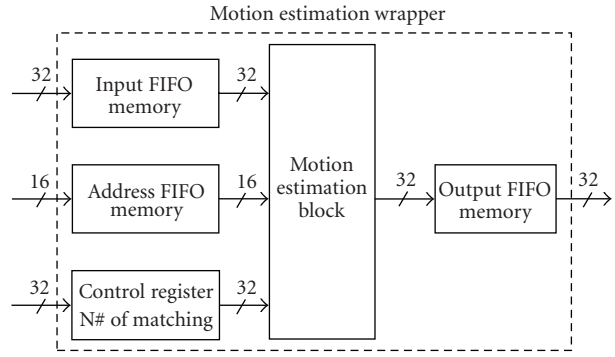


Figure 7: Motion estimation accelerator.

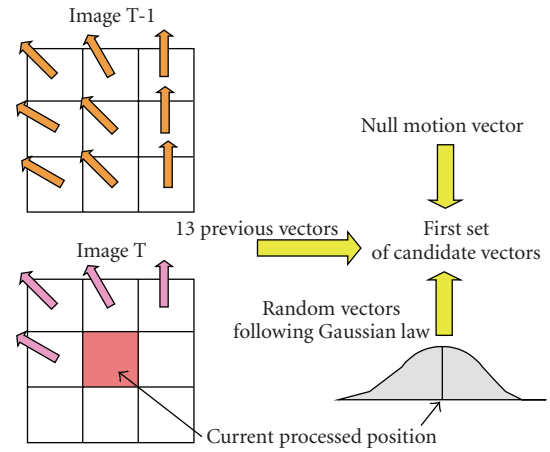


Figure 8: Determination of first set of candidate vectors.

7.3.5. *Optimization—second iteration*

the motion vector is usually highly correlated with the previous vectors. So as to exploit this feature, the user algorithm defines the different matchings to be processed. The proposed algorithm is presented below. The algorithm is split into two phases. As shown in Figure 8, initially nine motion vectors in a 3×3 corresponding neighborhood of the previous image (Image T-1) are listed as candidates. In the current image (Image T), the four motion vectors early processed are considered. The zero motion vector should also always be considered, therefore it is added as a candidate. In view of the profiling results in terms of data transfer and processing time, the user can then determine the number of matching (or motion vectors) that are still available as process budget. Therefore, depending on the amount of the budget left measured by the profiling results, the user can add some random vectors following, for instance, a Gaussian law centered on the current position.

These new candidate vectors are processed. In a second phase, the resulting score of each candidate vector enables them to be ranked (Figure 9). Vector additions of better ranked vectors are executed to define a second set of candidate vectors. Once again, the number of combinations to define new candidates depends on the profiled user budget [21]. Conformance with optimal coding results can be obtained with less than 60 candidate vectors in a 56×40 search window (phase 1 and phase 2 aggregate together) and can be directly validated by executing the HW module and the SW.

The scheduling of the tasks is described in Figure 10. The input data (pattern and search window) and the addresses of the matching to process are transferred to the HW module during task 1. Tasks 2 and 6 represent, respectively, the processing of the first and the second sets of candidate vectors. Tasks 3 and 7 correspond, respectively, to the transfer of the resulting vectors form the HW module to the host processor. Task 4 represents the computation of the second set of vectors. Finally, task 5 corresponds to the transfer from the PC to the HW module of the matching addresses relative to the second vector set. The data are still resident into the internal memory cache of the HW module, therefore are not required.

By using the proposed reduced search strategy, the number of matching has been reduced to 100 (50 candidate vectors for each phase). The profiling tests show that the first phase is achieved in 23 480 nanoseconds (random access and

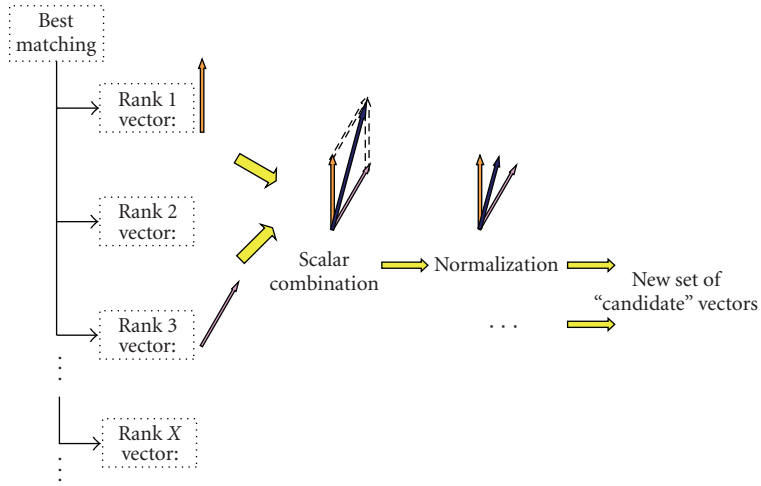


Figure 9: Determination of second set of candidate vectors.

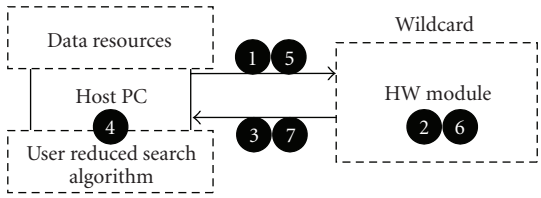


Figure 10: Scheduling of the different tasks.

data transfer included). The second phase requires only the new set of candidate vectors (matching address) as input data. This phase can be finished in 11 000 nanoseconds. Consequently, the whole process, excluding the determination of the second vector set, can be achieved in 34 000 nanoseconds. Depending on the complexity of the search algorithm, the acceleration of the processing, in comparison to the full-search strategy, can be very high (in this example a ratio equals to 10). Obviously, the required time for the determination of the second vector set has to be taken into account.

The data access flexibility of the VMW platform enables to easily implement and explore different search window sizes. With a 80×40 search window, the data transfer is measured to be equal to 17 280 nanoseconds and the processing time for a full search is 585 000 nanoseconds (for the 1625 possible matchings). With a reduced-search strategy of 200 candidate vectors, the processing time for the whole processing (excluding the determination of the second vector set) is then 61 280 nanoseconds. The processing time is much lower than for a full-search strategy (even with a 56×40 search window) and the acceleration achievable by the HW platform is even higher.

7.4. Discussion

Using the VMW platform capabilities and plugging the motion estimation accelerator are possible to easily explore a wide variety of optimization solutions without the need of detailing all necessary data transfers explicitly. The host processor remains in charge of the definition of the list of candidate matching. In summary, the system is highly flexible on three features and provides the following.

(i) A simple data-flow control (with virtual data access).

(ii) An easy integration of the HW accelerator implementation.

(iii) Profiling information on the data flow and the processing time for all tested implementation options.

Moreover, the system does not only provide flexibility of exploring and testing solutions, but also provides direct results of algorithmic-architectures tradeoffs in terms of overall coding performance increases. For instance, the user can scale the resource budget in terms of available candidate vectors for each search in view of the required performances and select the best configuration for the video encoding under test. Using the profiling information measured, the data transfer time and the processing time, the designer can

(i) adjust the number of vectors to obtain optimal coding results or in general select the tradeoff between the coding performances and desired processing time;

(ii) adjust/modify the search algorithm complexity (e.g., three vectors can be considered or different reduced search strategy investigated);

(iii) adjust/increase the size of the search window looking for optimal tradeoffs between coding performances and required memory bandwidths.

The major challenges of the architecture design of the co-processor for motion estimation are to guarantee the random access of any search strategy and to enable the setting of the size of the search window, hence providing sufficient processing resources for the different encoding applications. So as to guarantee the random access in any position of a search window, the search window pixels have

to be accessible to the matching engine and need to be stored in the FPGA to reduce the number of accesses to the external memory, so as not to exceed the available bandwidth. A current investigation aims at minimizing the size of the internal cache in function of the kind of image sequence. The profiling statistics enable the number of virtual accesses to be measured, therefore the number of misses in the local cache. Different cache sizes can be quickly implemented thanks to the virtual access mode, therefore a local cache can be defined for the target sequences. Other parameters can be considered. Indeed, this definition represents a tradeoff between the optimum size and the latency tolerated which is correlated to the ratio between the processing and the data transfer times. Therefore, the cache memory size can be defined in view of the processing time, in other words, with consideration of the selected search algorithm parameters (algorithm, number of matching, or search window size).

8. CONCLUSION

This paper describes the implementation of a platform that enables SW and HW environments to share the same virtual memory space. The platform helps the designer in the different design steps aiming at developing implementations of heterogeneous embedded systems starting from a specification described by a reference software. The platform provides a seamless environment and a clear methodology to help designers to turn C/C++ reference software into HDL modules. The conformance tests become straightforward. The main advantage of the platform is that it provides a step-by-step approach for designing, evaluating, and integrating hardware modules into a heterogeneous environment. The profiling capabilities on the data transfers between the SW and HW components of the platform support the designer to explore different implementation and optimization options at different stages of the design process. Initially, design efforts can be focused on the module functionality without worrying about data transfers. Then, by using the profiled data transfer, designers can focus on appropriate memory architectures, on algorithm/architecture tradeoffs, or on any other design optimizations that match the specific desired criteria of the design that affects or are affected by data transfers between modules.

REFERENCES

[1] "Information technology—generic coding of audio-visual objects—part 2: visual," ISO/IEC 144962-2:2004, June 2004.

[2] K. Denolf, K. Vissers, P. Schumacher, et al., "A systematic design of an MPEG-4 video encoder and decoder for FPGAs," in *Proceedings of the Global Signal Processing Expo and Conference (GSPx '04)*, Santa Clara, Calif, USA, September 2004.

[3] "Information technology—generic coding of audio-visual objects—part 7: optimized reference software," ISO/IEC TR 14496-7:2004, October 2004.

[4] "Information technology—generic coding of audio-visual objects—part 9: reference hardware description," 2nd edition, ISO/IEC TR 14496-9:2007.

[5] The Xilinx XUP Virtex-II Pro Development System, http://www.xilinx.com/univ/xupv2p.html.

[6] Celoxica Handel-C and RC1000-PP PCI board Production Information, Celoxica Ltd., http://www.celoxica.com/.

[7] A. Koch, "A comprehensive prototyping-platform for hardware-software codesign," in *Proceedings of the 11th International Workshop on Rapid System Prototyping (RSP '00)*, pp. 78–82, Paris, France, June 2000.

[8] M. Edwards and B. Fozard, "Rapid prototyping of mixed hardware and software systems," in *Proceedings of the Euromicro Symposium on Digital System Design (DSD '02)*, pp. 118–125, Dortmund, Germany, September 2002.

[9] R. Pradeep, S. Vinay, S. Burman, and V. Kamakoti, "FPGA based agile algorithm-on-demand coprocessor," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '05)*, vol. 3, pp. 82–83, Munich, Germany, March 2005.

[10] C. Plessl and M. Platzner, "TKDM—a reconfigurable co-processor in a PC's memory slot," in *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT '03)*, pp. 252–259, Tokyo, Japan, December 2003.

[11] S. Sukhsawas, K. Benkrid, and D. Crookes, "A reconfigurable high level FPGA-based coprocessor," in *Proceedings of IEEE International Workshop on Computer Architectures for Machine Perception (CAMP '03)*, p. 4, New Orleans, La, USA, May 2003.

[12] T. S. Mohamed and W. Badawy, "Integrated hardware-software platform for image processing applications," in *Proceedings of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC '04)*, pp. 145–148, Banff, Canada, July 2004.

[13] I. Amer, C. A. Rahman, T. Mohamed, M. Sayed, and W. Badawy, "A hardware-accelerated framework with IP-blocks for application in MPEG-4," in *Proceedings of the 5th IEEE International Workshop on System-on-Chip for Real-Time Applications (IWSOC '05)*, pp. 211–214, Banff, Canada, July 2005.

[14] P. Schumacher, M. Mattavelli, A. Chirila-Rus, and R. Turney, "A virtual socket framework for rapid emulation of video and multimedia designs," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '05)*, pp. 872–875, Amsterdam, The Netherlands, July 2005.

[15] L. Kaouane, M. Akil, T. Grandpierre, and Y. Sorel, "A methodology to implement real-time applications onto reconfigurable circuits," *The Journal of Supercomputing*, vol. 30, no. 3, pp. 283–301, 2004.

[16] C. Lucarz and M. Mattavelli, "A platform for mixed HW/SW algorithm specifications for the exploration of SW and HW partitioning," in *Proceedings of the 17th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS '07)*, pp. 485–494, Gothenburg, Sweden, September 2007.

[17] C. Lucarz, M. Mattavelli, and J. Dubois, "A HW/SW codesign platform for algorithm-architecture mapping," in *Proceedings of the Workshop on Design and Architectures for Signal and Image Processing (DASIP '07)*, Grenoble, France, November 2007.

[18] M. Vuletić, L. Pozzi, and P. Ienne, "Virtual memory window for application-specific reconfigurable coprocessors," in *Proceedings of the 41st Annual Conference on Design Automation (DAC '04)*, pp. 948–953, San Diego, Calif, USA, June 2004.

[19] J. Dubois, M. Mattavelli, L. Pierrefeu, and J. Miteran, "Configurable motion-estimation hardware accelerator module for the MPEG-4 reference hardware description platform,"

in *Proceedings of IEEE International Conference on Image Processing (ICIP '05)*, vol. 3, pp. 1040–1043, Genoa, Italy, September 2005.

[20] J.-H. Luo, C.-N. Wang, and T. Chiang, "A novel all-binary motion estimation (ABME) with optimized hardware architectures," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 8, pp. 700–712, 2002.

[21] M. Mattavelli and G. Zoia, "Vector-tracing algorithms for motion estimation in large search windows," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 8, pp. 1426–1437, 2000.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 280347, 12 pages
doi:10.1155/2008/280347

Research Article

A Priori Implementation Effort Estimation for Hardware Design Based on Independent Path Analysis

Rasmus Abildgren,[1] Jean-Philippe Diguet,[2] Pierre Bomel,[2] Guy Gogniat,[2] Peter Koch,[3] and Yannick Le Moullec[3]

[1] *CISS, Aalborg University, Selma Lagerlöfs Vej 300, 9220 Aalborg East, Denmark*
[2] *Lab-STICC (UMR CNRS 3192), Université de Bretagne Sud, Centre de recherche, BP 92116, 56321 Lorient Cedex, France*
[3] *CSDR, Aalborg University, Fredriks Bajers Vej 7, 9220 Aalborg East, Denmark*

Correspondence should be addressed to Rasmus Abildgren, rab@es.aau.dk

Received 15 March 2008; Revised 30 June 2008; Accepted 18 September 2008

Recommended by Markus Rupp

This paper presents a metric-based approach for estimating the hardware implementation effort (in terms of time) for an application in relation to the number of linear-independent paths of its algorithms. We exploit the relation between the number of edges and linear-independent paths in an algorithm and the corresponding implementation effort. We propose an adaptation of the concept of cyclomatic complexity, complemented with a correction function to take designers' learning curve and experience into account. Our experimental results, composed of a training and a validation phase, show that with the proposed approach it is possible to estimate the hardware implementation effort. This approach, part of our light design space exploration concept, is implemented in our framework "Design-Trotter" and offers a new type of tool that can help designers and managers to reduce the time-to-market factor by better estimating the required implementation effort.

1. INTRODUCTION

1.1. Discussion of the problem

Companies developing embedded systems based on high-end technology in areas such as telecommunication, defence, consumer products, healthcare equipment are evolving in an extremely competitive globalised market. In order to preserve their competitiveness, they have to deal with several contradicting objectives: on one hand, they have to face the ever-increasing need for shorter time-to-market; and on the other hand, they have to develop and produce low-cost, high-quality, and innovative products.

This raises major challenges for most companies, especially for small- and medium-sized enterprises (SMEs). Although SMEs are under pressure due to the above-mentioned factors, they are either not applying the latest design methodologies or cannot afford the modern electronic system level (ESL) design tools. By limiting themselves to traditional design methodologies, SMEs make themselves more vulnerable to unforeseen problems in the development process, making the time-to-market factor one of the most critical challenges they have to deal with. A survey released at the Embedded Systems Conference (ESC 2006) [1] indicated that more than 50% of embedded design projects are running behind schedule (i.e., 25% are 1-2 months late, 18% 3–6 months). In the 2008 version of the survey [2], it is again shown that meeting the schedule is the greatest concern for design teams.

Moreover, a workshop [3] held for Danish SMEs working in the domain of embedded systems clearly indicates that there is a need for changing and improving their design trajectories in order to stay in front of the global market. More specifically, this calls for setting modern design, that is, hardware/software (HW/SW) codesign, and ESL design into actual practice in SMEs, so that they can reduce their time-to-market factor and keep up with their competitors by being more efficient in producing embedded systems.

Although HW/SW codesign and ESL design tools (both commercial and academic) have been available for several years, there are several barriers that, so far, have prevented their wide adoption such as the following:

(i) difficulty in transferring the methods and tools developed by academia into industry, because they are mostly developed for experimenting, validating, and proving new concepts rather than for being used in companies; therefore adapting and transferring these methods and tools require additional and tedious efforts, delaying their adoption;

(ii) financial cost in terms of tool licenses, training, and so forth that many SMEs cannot afford, since the cost of a complete commercial tool chain can exceed in excess of 150 k€ per year;

(iii) training cost and knowledge management issues, meaning that switching to a new design trajectory also involves the risk of loosing momentum, that is, loosing time and efficiency because of the training needed to master the new methods and tools;

(iv) many modern design flows are not mature enough to generate efficient and automatic real-time code, and combined with the previous item, cause potential adopters to wait until it is safe to switch.

Considerable research has been undertaken to estimate implementation factors such as area, power, and speed up that are subsequently used in HW/SW partitioning tools with different focuses related to granularity, architecture model, communication topology, and so on. All of these research projects do not include the man-power cost which is the most critical one for many companies, and especially SMEs. This work takes its outset in a research framework facilitating the HW/SW partitioning step for SMEs. It focuses on a light design space exploration approach called "DSE-light" that combines the advances in terms of design methodologies found in academia and the ease of integration required by SMEs, that is, lowering the above-mentioned barriers.

The contribution presented in this paper is the development of a method for estimating the man-power cost (i.e., development time) for implementing hardware components and the integration of this method into our framework, so that HW/SW partitioning decisions can be wiser. A method that used iteratively and systematic will form the engine for precise development schedules. The following subsections present the rationale for this work and the idea enabling this contribution.

1.2. Parameters that influence the implementation effort

A common problem in both SMEs and larger companies is that of estimating the amount of time required to map and implement an algorithm onto an architecture given parameters such as [4, 5] the following :

(i) manpower, that is, the available development team(s) and their size(s),

(ii) quality of the social interactions between the team members and the teams,

(iii) experience of the developers (e.g., years of experience, previously developed projects, novelty of the current project, etc.),

(iv) skills of the developers, that is, their ability to solve problems (this is not the same as experience, which only reflects how often one has tried before),

(v) availability of suitable and efficient tools and how easy they are to learn and use,

(vi) availability of SW/HW IP code/cores,

(vii) involvement of the designers, that is, are they working on other projects simultaneously?

(viii) design constraints, that is, real-time requirements,

This work addresses the issue of adding man-power cost parameter into the cost function and thereby guiding the HW/SW partitioning. More specifically we concentrate on the mapping process, that is, the process of mapping a given algorithm onto a given architecture and the implementation effort (i.e., time) related to the complexity of that algorithm. Our framework also addresses other issues of HW/SW partitioning, for example, [6].

1.3. Idea

In order to understand what makes an algorithm difficult to implement, five semistructured interviews have been conducted with engineers (hardware developers) with very little to 20 years of experience. (Semistructured interview is an information-gathering method of qualitative research. It is also an adequate tool to capture how a person thinks of a particular domain [7].)

From the interviews, it was deduced that several parameters influence on the hardware design difficulty. The hardware developers stated that available knowledge about worst cases, dependencies between variables, and the completeness of the design description of the entire system including all communications are important for the design time. However, according to them, the major parameter influencing a hardware design is the number of connections and signals between the internal components. This should be viewed in the way in which every time a signal enters a component, it means that the component needs to act on it. More signals bring more parameters into the component and that very often leads to an increased complexity.

Based on the interviews, we form our hypothesis, which is that a strong relation exists between what renders an algorithm complex to implement and the number of components as well as the number of signals/paths in the algorithm.

To ensure that not only the number of paths are counted but also that a high number of components is present, we choose to only measure the number of linear-independent paths. Furthermore, this insures that components occuring several times during the execution are counted only once, which better reflects the actual implementation efforts.

The remainder of the paper is organised as follows: Section 2 gives an overview of the state-of-the-art methods for estimating the implementation effort both for software and hardware designs and indicates the need for further work for hardware design. In Section 3 a new metric for estimating the development time is defined and combined with our

research tool "Design-Trotter." Section 4 presents some test cases used to investigate the validity of the above-mentioned hypothesis and of the proposed metric. Furthermore, the experimental results are analysed. Finally we conclude in Section 5.

2. STATE OF THE ART

2.1. Software

Most research about estimating implementation effort is found in the software domain, especially within the COCOMO project [8]. The problem of estimating the implementation effort is twofold. First, a reasonable measure needs to be developed for being able to quantify the algorithm. Second, a model needs to be developed, describing a rational relation between the measure and the implementation effort.

2.1.1. COCOMO

To start with the model, a typical power model has been proposed inside the COCOMO experiment [8, 9]:

$$\text{Effort} = A \times \text{Size}^{b}, \tag{1}$$

where Size is an estimate of the project size, and A and b are adjustable parameters. These parameters are influenced by many external factors which we previously discussed in Section 1.2, but can be trained, based on previous project data.

To use this COCOMO measure, there is a need for expressing the size of the project. Inside the software domain, the dominating metric is lines of code (LOC). Using LOC is not without difficulties, for example, how is a code line defined? Reference [10] discusses this issue and states that LOC is not consistent enough for that use; this is also supported by [11]. Using the LOC metric also has several difficulties, for example, it is not a language independent metric. Furthermore, hardware developers also tend to disapprove this measure, since they do not feel that it is a representative measure for hardware designs.

However, we do not claim that there is no relation between LOC and the implementation effort. It is impossible to write 10 k lines in one day, but for VHDL the relation is not always straightforward. In the experiments that we have performed (data shown in Table 1) there is no unambiguous relation between the LOC in VHDL and the development time.

Reference [11] describes that making "a priori" determination of the size of a software project is difficult especially when using the traditional lines of code measure; instead function points-based estimation seems to be more robust.

2.1.2. Function points analysis

The function points metric was first introduced by Albrecht [12] and consists of two main stages: The first stage is counting and classifying the function types for the software. The identified functions need to be weighted reflecting their complexity, that is determined on the basis of the developers' perception. The second stage is the adjustment of the function points according to the application and environment, based on 14 parameters. The function points can then be converted into an LOC measure, based on an implementation language-dependent factor, and, for example, [11] reports that the function points metric can be used as an implementation effort estimation metric. The function points analysis has been criticised of being too heuristic and [10] has proposed the SPQR/20 function points metric as an alternative. Reference [13] has compared the SPQR/20 and the function points analysis and found their accuracy comparable even though the SPQR/20 metric is simpler to estimate.

2.2. VHDL function points

To the knowledge of the authors, limited research has been carried out in the field of estimating the implementation difficulty of hardware designs.

Fornaciari et al. [14] have taken up the idea from the function points analysis and modified it to fit VHDL. By counting the number of internal I/O signals and components, and classifying these counts into levels, they extract a function point value related to VHDL. They have related their measure to the number of source lines in the LEON-1 processor project, and their predictions are within 20% of the real size. However, as stated previously, estimating the size does not always give an accurate indication of the implementation difficulty, and the necessary implementation time.

By measuring the number of internal I/O signals and components, their work goes along the same road as our initial observations indicate. However, our approach is pointing towards estimating the implementation effort, based on a behavioural description of the algorithm in the C-language. Furthermore, it also takes the designer's experience into account.

3. METHODOLOGY

The proposed flow for estimating the implementation effort is illustrated in Figure 1. It takes its outset in a behavioural description of the algorithm, in C-language (including library function source code), which is intended to be implemented in hardware. From this description, we use the design-Trotter framework to generate a hierarchical control data flow graph (HCDFG) which is then measured to identify the number of independent paths. The resulting measure, combined with the experience of the developers, gives an estimate of the required implementation effort. The method is self-learning in the sense that after each successful implementation, new knowledge about the developers involved can be integrated, and improve the accuracy of the estimates. The HCDFG and the approach for modelling the developers experience are covered later in this section but initially we investigate how the number of paths can be measured.

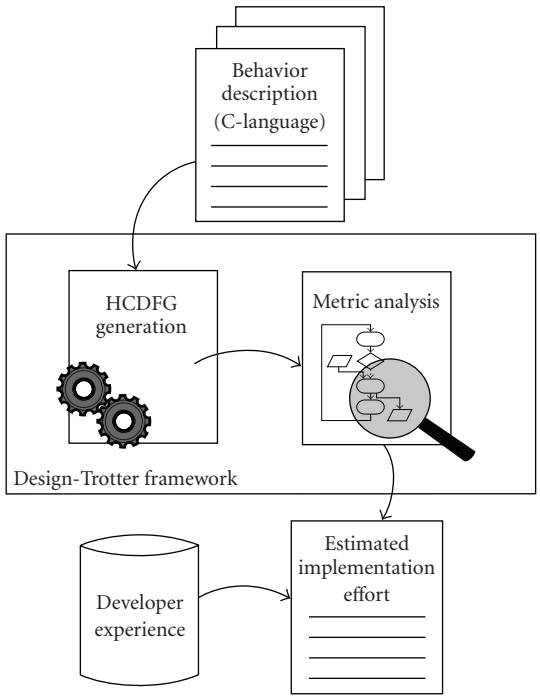
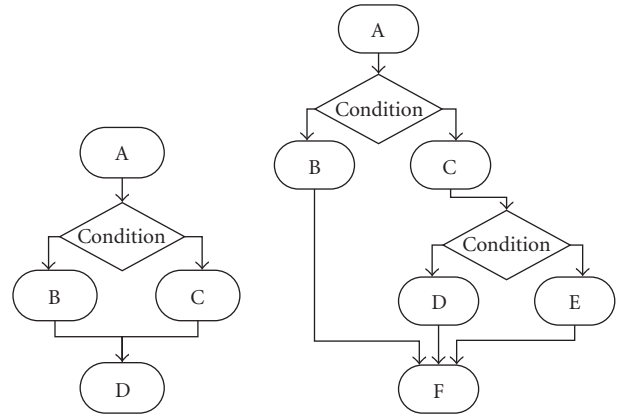


Figure 1: The flow of estimating the required implementation effort. The starting point is a behavioural description in C of the algorithm to be implemented in hardware (e.g., via VHDL). From this description, an HCDFG is generated and measured to identify the number of independent paths in the algorithm. This measure, combined with the experience of the developers, gives an estimate of the required implementation effort (expressed in time).



(a) $P(G) = 2$ (b) $P(G) = 3$

Figure 2: Two examples of graphs for which the cyclomatic complexities have been calculated.

3.1. Cyclomatic complexity

As described in Section 1.3, the number of independent paths is expected to correlate with the complexity that the engineers are facing when working on the implementation. Therefore, finding a method to measure the number of independent paths in an algorithm could help us investigating this issue. A metric measuring is the cyclomatic complexity measure proposed by McCabe [15] which measures the number of linear-independent paths in the algorithm.

The cyclomatic complexity was originally invented as a way to intuitively quantify the complexity of algorithms, but has later found use for other purposes especially in the software domain. The cyclomatic complexity has been used for evaluating the quality of code in companies [16], where quality covers aspects from understandability over testability to maintainability. It has also been shown [17] that algorithms with a high cyclomatic complexity more frequently have errors than algorithms with lower cyclomatic complexity. The cyclomatic complexity has furthermore been used for evaluating programming languages for parallel computing [18], where languages that encapsulate control statement in instructions are receiving higher scores. All use the cyclomatic complexity measure under the assumptions that the complexity has significant influence on the number of paths the developers need to inspect, its correlation to the number of paths that needs to be tested, or a combination of the two.

In the domain of hardware, the cyclomatic complexity has also found use, judging the readability and maintainability in the SAVE project [19]. It is worth noticing that they use a misinterpreted [20] definition of the cyclomatic complexity [21].

All these projects utilise the cyclomatic complexity's ability to measure the number of independent paths and relate them to their individual cases:

$$P(G) = \pi + 1, \tag{2}$$

where π represents the number of condition nodes in the graph G representing the algorithm being analysed. Figure 2 shows two examples of graphs and the corresponding cyclomatic complexity.

In this work, we propose an adapted version of the cyclomatic complexity definition to estimate, a priori, the number of independent paths on a hierarchical control data flow graph (HCDFG), defined in the following section. The cyclomatic complexity for an HCDFG is obtained by examining its subgraphs as explained in Section 3.3.

3.2. HCDFG

For this work we use the hierarchical control data flow graphs (HCDFGs), which are introduced in [22, 23]. The HCDFGs are used to represent an algorithm with a graph-based model so the examination task of the algorithm is eased. Control/Data Flow Graphs (CDFGs) are well accepted by designers as a representation of an algorithm where data flow graphs represent the data flow between different processes/operations, and the control flow layer, encapsulating these data flows and adding control structures to the graphical notation. The hierarchy layered structure is added to help representing large algorithms as well as to enable the analysis mechanism to identify functions/blocks in the graph. Such an identified block can then be seen as a single HCDFG that can be instantiated several times.

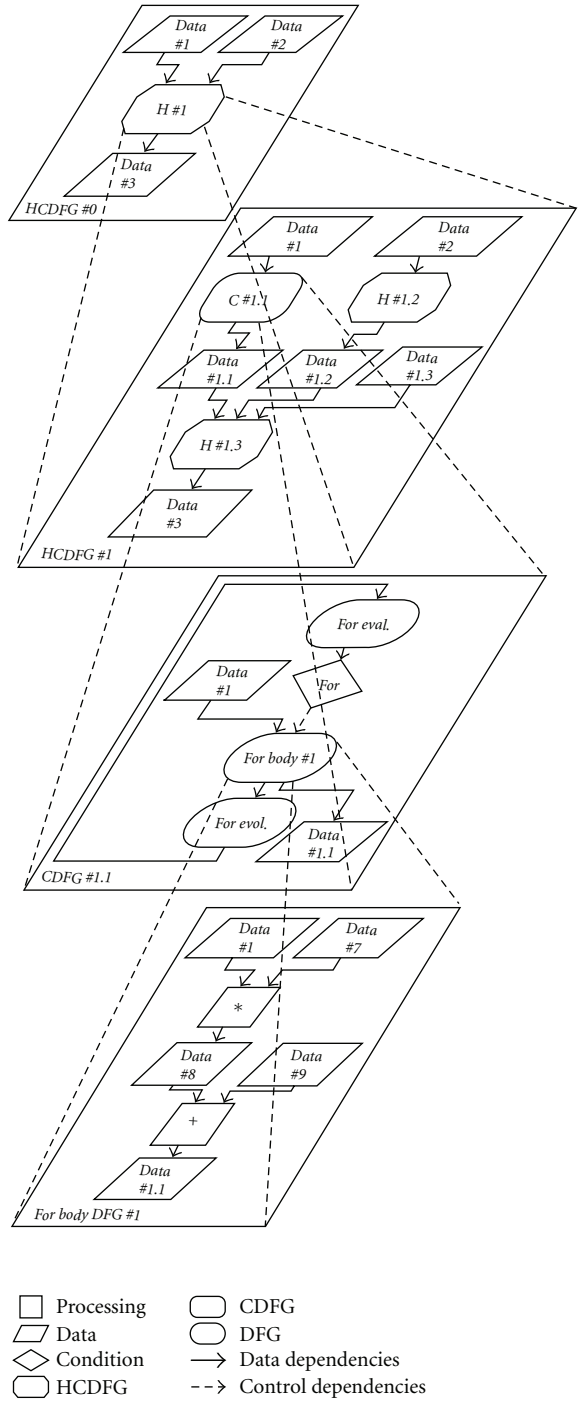


Figure 3: An overview of how the hierarchy in an HCDFG allows analysis of an algorithm on different levels and how the levels are related.

Figure 3 shows an example of a hierarchical control data flow graph.

In this work the design space exploration tool "Design-Trotter" is used as an engine for analysing the algorithms. The HCDFG model is used as "Design-Trotter's" internal representation.

The hierarchy of an HCDFG is shown in Figure 3. An HCDFG can consist of other HCDFGs, Control/Data flow graphs (CDFGs) and data flow graphs (DFGs) as well as elementary nodes (processing, memory, and control nodes). An HCDFG is connected via dependency edges. In this work we only explore the graph at levels above the DFGs, and therefore only concentrate on these when we define the graph types in what follows.

Let us consider the hierarchical control data flow graph, $G_{\text{HCDFG}} = (N_{\text{HCDFG}}, E_{\text{HCDFG}})$, where N_{HCDFG} arethe nodes denoted by $N_{\text{HCDFG}} = \{n_{\text{HCDFG}_1}, \ldots, n_{\text{HCDFG}_m}\}$ and the nodes are $N_{\text{HCDFG}} \in \{G_{\text{HCDFG}} | G_{\text{CDFG}} | G_{\text{DFG}} | \text{Data}\}$, meaning that the nodes in the G_{HCDFG} can be instances of its own type, encapsulated control data flow graphs, G_{CDFG}, encapsulated data flow graphs G_{DFG}, or data transfer nodes, *Data*. The last one is introduced to avoid the duplication of data representations in the hierarchy, when data is exchanged between the graphs. Thereby, data are only represented by their nodes and not by edges as it is common in many other types of DFGs.

The edges, E_{HCDFG}, connect the nodes such that $E_{\text{HCDFG}} = \{e_{n_{\text{HCDFG}_i}, n_{\text{HCDFG}_j}}\}$, where $i \neq j$ and represent the indexes of the nodes, $E_{\text{HCDFG}} \in \{\text{DD}\}$ and where every node can have multiple input and/or output edges. For the G_{HCDFG}, only data dependencies, DD, are allowed, and no control dependencies, CD.

In this way the HCDFG forms a hierarchy of encapsulated HCDFGs, CDFGs, and DFGs, connected via exchanging data nodes. The HCDFG can be seen as a container graph for other graph types such as the CDFG.

We can define the CDFG as $G_{\text{CDFG}} = (N_{\text{CDFG}}, E_{\text{CDFG}})$, where N_{CDFG} are the nodes denoted by $N_{\text{CDFG}} = \{n_{\text{CDFG}_1}, \ldots, n_{\text{CDFG}_m}\}$ and the nodes are $N_{\text{CDFG}} \in \{\text{CC} | G_{\text{HCDFG}} | G_{\text{DFG}} | \text{Data}\}$, where $\text{CC} \in \{\text{if}|\text{switch}|\text{for}|\text{while}|\text{do-while}\}$. In this way the G_{CDFG} is able to describe common control structures, where the actual data processing is encapsulated in either DFGs or HCDFGs. Again, the data exchange nodes are used to exchange data between the other nodes.

The edges, E_{CDFG}, connect the nodes such that $E_{\text{CDFG}} = \{e_{n_{\text{CDFG}_i}, n_{\text{CDFG}_j}}\}$, where $i \neq j$ and represent the indexes of the nodes. If $n_{\text{CDFG}_i} \in \text{CC}$ and $n_{\text{CDFG}_j} \in \{G_{\text{HCDFG}} | G_{\text{DFG}}\}$, then $\{e_{n_{\text{CDFG}_i}, n_{\text{CDFG}_j}}\} \in \{\text{CD}\}$, else $\{e_{n_{\text{CDFG}_i}, n_{\text{CDFG}_j}}\} \in \{\text{DD}\}$.

Beneath the control data flow graphs G_{CDFG}, the data flow graphs G_{DFG} exist but they are of no use in this work so we will not define them further here.

3.3. Calculating the cyclomatic complexity on CDFGs

Now that the HCDFG has been defined, we explain our proposed method for measuring the cyclomatic complexity on the CDFGs.

Since the cyclomatic complexity only considers the control structure in finding the number of independent paths in the algorithm, the DFG part of the algorithm is, as mentioned earlier, of no interest for this task because it only gives a single path. On the other hand, what is of interest is how the cyclomatic complexity is measured on the CDFGs

and HCDFGs which are built by the tool Design-Trotter. This leaves us with the following cases which are described in detail afterwards:

(i) If constructs,

(ii) Switch constructs,

(iii) For-loop,

(iv) While/do-while loops,

(v) Functions,

(vi) HCDFGs in parallel,

(vii) HCDFGs in serial sequence.

3.3.1. If constructs

"If constructs" case is represented as CDFGs, G_{CDFG}, where one node is a control node of type *if* (see Figure 4(a)). Before arriving at the control node, a condition evaluation node $n_{\text{eval}} \in \{G_{\text{HCDFG}} | G_{\text{DFG}}\}$ is traversed to calculate the boolean variable stored in n_{Data} (to maintain simplicity, these are not shown in Figure 4(a)) that is used in the condition node. If the variable is true, the algorithm follows the path through the true body node, $n_{\text{true}} \in \{G_{\text{HCDFG}} | G_{\text{DFG}} | \varnothing\}$. Else it goes to the false body node $n_{\text{false}} \in \{G_{\text{HCDFG}} | G_{\text{DFG}} | \varnothing\}$. Note that in some cases, either the true body or the false body does not exist, but it still gives a path. In this case, according to the cyclomatic complexity measure, the number of independent paths is

$$P(n_{\text{if}}) = P(n_{\text{true}}) + P(n_{\text{false}}) + P(n_{\text{eval}}) - 1. \quad (3)$$

The last part of (3), $+P(n_{\text{eval}}) - 1$ is included in case the evaluation graph is an HCDFG node.

3.3.2. Switch constructs

"Switch constructs" case is represented as CDFGs, G_{CDFG}, and is almost the same flow as the "if constructs" case discussed above. One node is a control node of switch type. Before arriving to the control node, a condition evaluation node $n_{\text{eval}} \in \{G_{\text{HCDFG}} | G_{\text{DFG}}\}$ is traversed. Depending on the output, the switch node leads the algorithm flow to the selected case node: $n_{\text{case}_i} \in \{G_{\text{HCDFG}} | G_{\text{DFG}}\}$. An example is shown in Figure 4(b). According to the cyclomatic complexity measure, the number of independent paths is as follows :

$$P(n_{\text{switch}}) = P(n_{\text{eval}}) - 1 + \sum_{i=1}^{N} P(n_{\text{case}_i}), \quad (4)$$

where N represents the number of cases, i the index to the corresponding node on which the paths are measured.

The same argument goes for the $P(n_{\text{eval}}) - 1$ part of (4); it is included in case the evaluation graph is an HCDFG node, but else it is omitted.

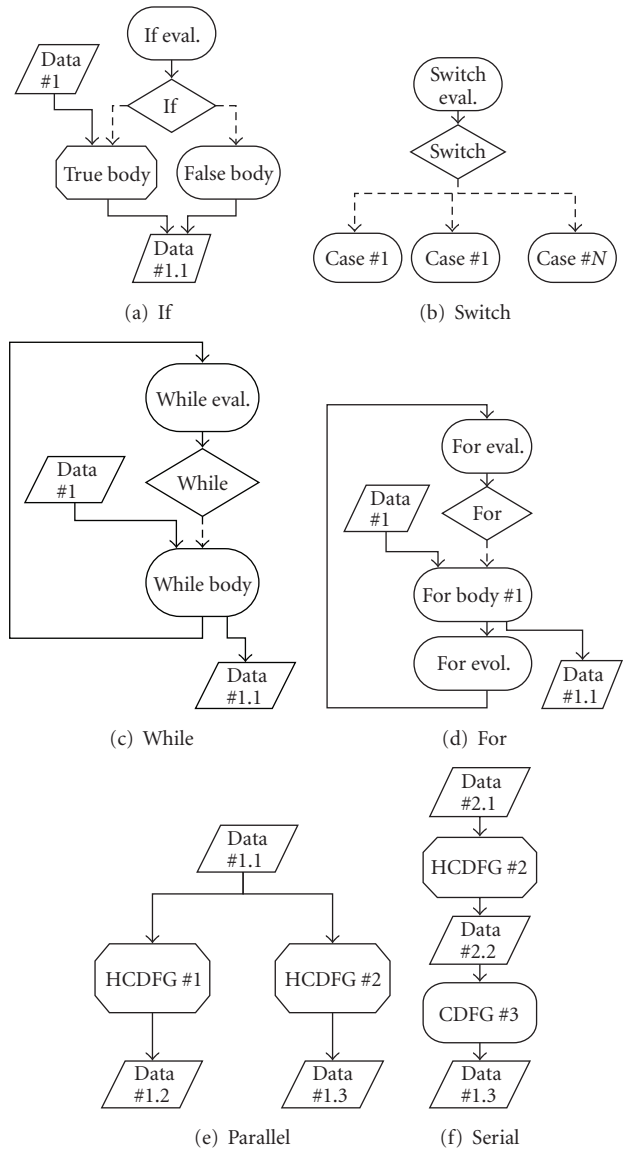


Figure 4: Overview of the different CDFGs and combined HCDFGs, on which the cyclomatic complexity values are measured. Between the (HC)DFGs there is a set of data exchange nodes which are here left out for simplicity. The symbols are similar to those presented in Figure 3.

3.3.3. For-loop

"For-loop" case is the most complex of the control structures. Strictly speaking, a "for loop" consists of three different parts: the evaluation body, the evolution body, and the for body, n_{eval}, n_{evol}, and $n_{\text{for-body}}$, respectively. The control node nfor, determines, based on the output from the evaluation graph, whether the flow should go into the "for loop" or leave it. The evolution node updates the indexes. Since each iteration of the graph needs to pass through the evaluation and evolution nodes, the number of independent paths is calculated as

$$P(n_{\text{for}}) = P(n_{\text{for-body}}) + P(n_{\text{eval}}) - 1 + P(n_{\text{evol}}) - 1. \quad (5)$$

In many cases, the evaluation and evolution part of the "for loop" are quite simple indexing functions, meaning that $n_{\text{eval}} \in \{G_{\text{DFG}}\}$, $n_{\text{evol}} \in \{G_{\text{DFG}}\}$, will leave $P(n_{\text{for}}) = P(n_{\text{for-body}})$. The "for loop" is illustrated in Figure 4(d).

3.3.4. While loops and do-while loops

"While loops" and "do-while loops" cases are described jointly since it is only the entry to the loop structure that separates them and their cyclomatic complexity are equivalent. The "while loops" consist of two main parts: the while body $n_{\text{while-body}} \in \{G_{\text{HCDFG}}|G_{\text{DFG}}\}$, and the while evaluation $n_{\text{eval}} \in \{G_{\text{HCDFG}}|G_{\text{DFG}}\}$. This is illustrated in Figure 4(c). Deciding whether to continue looping is decided by the control node $n_{\text{while}} \in \{\text{while}\}$ based on the output of the n_{eval}. Similarly to the "for loop," each iteration of the graph needs to pass through the evaluation nodes, so the number of independent paths can be calculated as

$$P(n_{\text{while}}) = P(n_{\text{while-body}}) + P(n_{\text{eval}}) - 1. \tag{6}$$

In many cases, the evaluation part of the while loop is a set of simple test functions, meaning that $n_{\text{eval}} \in \{G_{\text{DFG}}\}$, which leaves the $P(n_{\text{while}}) = P(n_{\text{while-body}})$.

3.3.5. Functions

The goal is to identify the number of independent paths in the algorithm/system. For this, reuse in terms of functions/blocks of code is important. When all independent paths through a function are known, reuse of this function does not change the number of independent paths in the system. From an implementation point of view, such functions represent an entity where the paths only need to be implemented once. In HCDFGs, a function/block can be seen as an encapsulated G_{HCDFG}. Therefore, the number of independent paths in function/blocks of reused code should only count once. The paths can be calculated as

$$P(n_{\text{HCDFG}_{\text{function}}}) = \begin{cases} 0 & \text{if reuse,} \\ P(n_{\text{HCDFG}}) & \text{else.} \end{cases} \tag{7}$$

3.3.6. HCDFGs in parallel and serial

Knowing how to handle all the HCDFGs that are identified for reuse (function), together with all the CDFGs, does not give it all. How the hierarchy of graphs should be combined is also of interest. For a parallel combination of two or more HCDFGs/CDFGs, as shown in Figure 4(e), the increase in the number of independent paths is then additive. The number of paths can be calculated as

$$P(n_{\text{HCDFG}_{\text{Parallel}}}) = \sum_{i=1}^{N} P(n_{\text{HCDFG}_i}), \tag{8}$$

where N represents the number of nodes in parallel, i the index to the corresponding node where the paths are measured.

For serial combination of two or more HCDFGs and/or CDFGs, the number of independent paths is a combination of the independent paths of the involved HCDFGs/CDFGs. Remembering that there always needs to be one path through the system, the number of independent paths in a serial combination, is given as

$$P(n_{\text{HCDFG}_{\text{Serial}}}) = \sum_{i=1}^{N} P(n_{\text{HCDFG}_i}) - (N-1), \tag{9}$$

where N represents the number of nodes in serial, i the index to the corresponding node where the paths are measured.

An example of serial combination is shown in Figure 4(f). The number of independent paths for the entire algorithm, $(P(n_{\text{HCDFG}_{\text{Alg}}}))$, is equivalent to the top HCDFG node which includes all the independent paths of its subgraphs.

3.4. Experience impact

The experience of the designer has an impact on the challenge that he/she is facing when developing a system. A radical example is when a beginner and a developer with ten years of experience are asked to solve the same task. They will not see equal difficulty in the same task, and thereby do not need to put the same effort into the development.

Experience is influenced by many parameters but in this work we only focus on the time the developer has worked with the implementation language and the target architecture.

The impact of experience is a factor that slowly decreases over time: consider a new developer, the experience that he/she obtains in the first months working with the language, and architecture improves his/her skills significantly. On the other hand, a developer who has worked with the language and architecture for five years, for example, will not improve her/his skills at the same rate by working an extra year. The impact from the experience is therefore not linear but tends to have a negative acceleration or inverse logarithmic nature, with dramatic change in impact in the beginning, progressing towards little or no change as time increases.

In literature, for example, [24], many studies try to fit historical data to models. An example of a model is a power function with negative slope or a negative exponential function. From the vast variety of models that has been proposed over the years, the only conclusion that can be drawn is that there are multiple curvatures, but they all appear to have a negative accelerating slope, which tends to be exponential/logarithmic.

In order to get the best possible outset for predicting the implementation effort, it is of vital importance to obtain some data of the developers' experiences, and also how they performed in the past. The parameters involved in the experience curve can then be trimmed to create the best possible fit. However, it has not been the purpose of this work to select the perfect nature for a learning curve nor to evaluate the accuracy of such one. The learning curve will be adapted to the individual developers, and as the model is used in subsequent projects, its accuracy will progressively improve. As a consequence, the experience here is only

intended as an element in modelling the complexity and thereby a means for more accurate estimates.

For the experiments in this study we have chosen to use the following model:

$$\eta_{\text{experience}}(\text{Dev}) = \frac{1}{\alpha \log(\text{Experience}(\text{Dev}) + \beta)}, \quad (10)$$

where α and β are trim parameters which can be used to optimise the curve to fit reality, Experience is the number of weeks which the developer, Dev, has worked with the language and architecture. Figure 5 depicts the shape of the experience model.

In this work,our initial experiments have shown that setting $\alpha = 1$ and $\beta = 1$ makes our model sufficiently general, and therefore we have not further investigated the tuning of these two parameters.

4. RESULTS

In order to verify the hypothesis, a classical test has been conducted. The test is dual phased and consists of (i) a training phase using a first set of real-life data, during which the hypothesis is said to be true, and (ii) a validation phase during which a second set of real-life data is used to evaluate whether the hypothesis holds true or not.

4.1. *Phase one—training*

The real-life data used as training data originate from two different application types that are both developed as academic projects in universities in France. The first application is composed of five different video processing algorithms for an intelligent camera, which is able to track moving objects in a video sequence. The second application is a cryptographic system, able to encrypt data with different cryptographic/hashing algorithms, that is, MD5, AES and SHA-1. The system consists of one combined engine [25] as well as individual implementations. These projects were selected since they all follow the methodology of using a behavioural specification in C, as a starting point for the VHDL implementation. Common to this data is that none of the developers has made the behavioural specification in C. For the cryptographic algorithms the behavioural specification comes from the standards, and the video algorithms were based on a previous project.

Using the behavioural description as the starting point of the experiment, the exercise consists of studying the relationship between the complexity of the algorithms (as defined in Section 3) and the implementation effort (i.e., time) required to implement them in VHDL (including testbed and heuristic tests).

The developers involved in these projects have all been Master and Ph.D. students with electrical engineering backgrounds but no VHDL background other than what they obtained during their studies, see Table 2. All developers were taught VHDL by other instructors than the authors, but at our university. Table 3 summaries the training data.

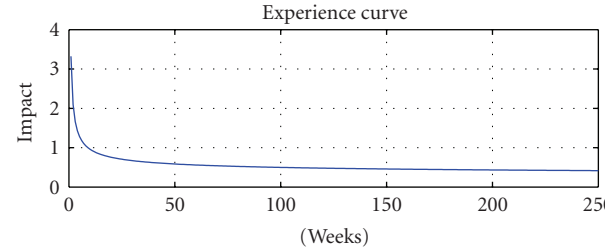


Figure 5: An example of how the lack of experience impacts the difficulty the engineers are facing.

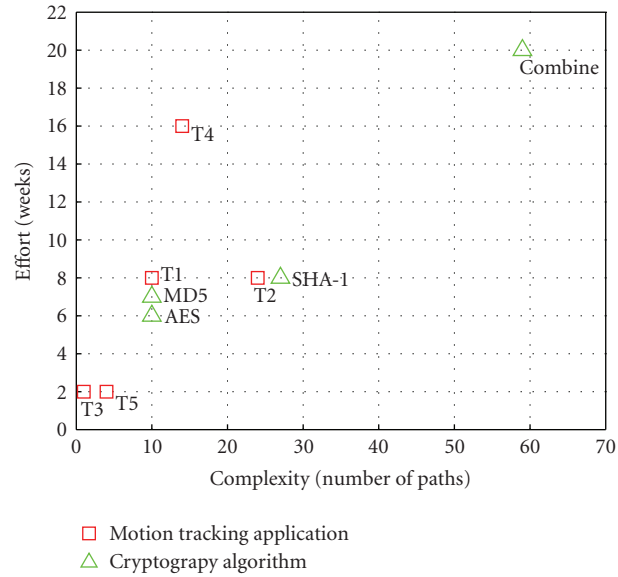


Figure 6: Relation between the implementation effort (number of weeks) and the not corrected complexity (as defined in Section 3).

Figure 6 shows the relation between the implementation effort and the measured complexity for the individual algorithms. Please note that in this graph the complexity values are not yet corrected for the designers' experience.

A first examination of the data points indicates a possible relation between some of them. However many other points are located far away from any relation. These data are not corrected for the designers' experience and, as earlier mentioned, we strongly believe that the experience of the individual designer has a nonnegligible influence on the development time. If we inspect the data more thoroughly, it is clear that the points of greatest divergence are those implementations where the developers have very limited knowledge and experience with the VHDL language.

Applying the proposed equation (10) (nonlinear) experience transform onto the data, results in a significantly different picture as depicted in Figure 7. A clear trend toward a relation is now visible in the plotted data. From the COCOMO II project [8], it is known that the relationship between the implementation time and the complexity measure (in their case lines of code, LOC) can be expressed as a

Table 1: Line of code, area, and time constraints for the validation data.

Algorithm	SS1	SS2	SS3	SS4	SS5	SS6	Ethernet	App 4
Dev. Time (weeks)	3.6	6.4	2.4	16.4	12	17.2	16	2
LOC-VHDL	994	1195	776	1695	760	2088	3973	232
Slices	564	2212	382	888	372	2171	3372	750
FlipFlops	913	2921	1290	1366	1208	2077	6149	942
LUTs	997	3157	6453	1569	6443	3458	18255	567
Time Constraint. (ns)	112	128	360	112	360	248	696	56

Table 2: Facts about the developers. Developers for training data (top) and validation data (bottom).

Developer	Education	Years in the domain
Dev 1	Ph.D. stud.	0
Dev 2	Stud. (EE)	0
Dev 3	Stud. (EE)	0
Dev 4	Stud. (EE)	0
Dev 5	BSc.EE.	9
Dev 6	MSc.EE.	15
Dev 7	MSc.EE.	9
Dev 8	MSc.EE.	8
Dev 9	MSc.EE.	8

Table 3: Training data (top) and validation data (bottom). Algorithms are related to the developers and their experience at the given time. Complexity is not corrected.

Algorithm	Complexity	Developer	Dev. Exp.
T1	10	Dev 1	2
T2	24	Dev 1	10
T3	12	Dev 1	18
T4	14	Dev 2	1
T5	4	Dev 1	20
MD5	10	Dev 3	1
MD5	10	Dev 4	1
AES	10	Dev 4	8
SHA-1	27	Dev 4	14
Combined	59	Dev 4	14
SS1	25	Dev 6, 7	150
SS2	35	Dev 5	150
SS3	17	Dev 5, 6, 7, 8	150
SS4	50	Dev 6	6
SS5	29	Dev 7	3
SS6	25	Dev 5, 6, 7	3
Ethernet app	60	Dev 5, 6, 7, 8, 9	150
App 4	9	Dev 6	150

power function with a weak slope. We showed its nature in (1), and with correction for experience it becomes

$$\text{Effort} = A \times \eta_{\text{experience}}(\text{Dev}) \times P\left(n_{\text{HCDFG}_{\text{Alg}}}\right)^{b}. \quad (11)$$

The parameters A and b are found, via a least square (LS) fit on our training data, to be $A = 0.226$ and $b = 1.103$. In

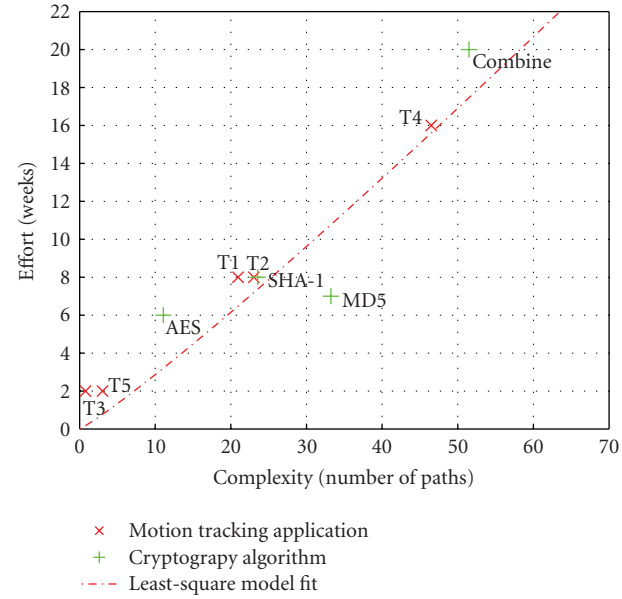


Figure 7: Relation between the implementation effort (number of weeks) and the complexity corrected according to the designers' experience model as shown in Figure 5.

Figure 7 the dashed line illustrates the relationship, with the parameters given above.

4.2. *Phase two—validation*

After having elaborated on a model based on the training data, we proceeded with the validation of its correctness. For this, a new set of data provided by ETI A/S, a Danish SME, is used. The dataset originates from a networking system and consists of Ethernet applications that have been implemented on an FPGA, as well as corresponding testbeds. This Ethernet application is part of an existing system with which it requires interaction. Table 1 shows additional implementation information with regards to these applications. The system is a real-time system with hard-time constraints and all algorithms were implemented as to meet these constraints. Similar to the training data, the development flow for this application has been as follows: a behavioural C++ model of the application has been constructed before the implementation on the FPGA architecture. The behavioural model has been developed by developers separate to those undertaking the implementation. The developers responsible

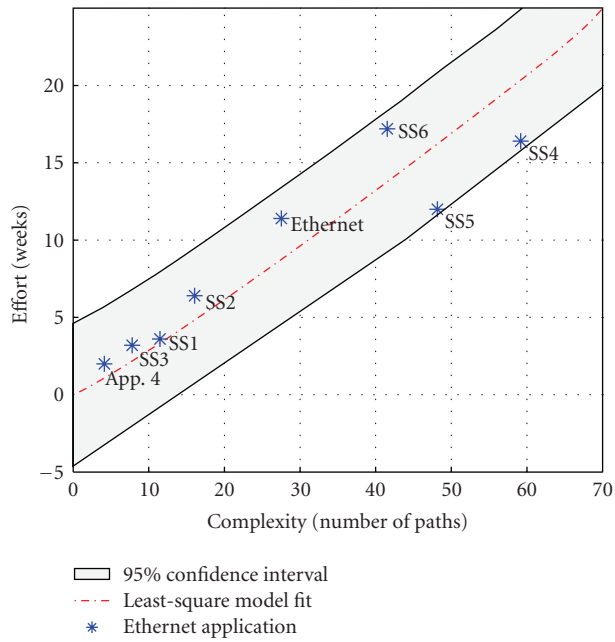


Figure 8: Validation data plot: relation between implementation effort (number of weeks) and complexity, corrected according to the designers' experience model.

for the implementation have obtained their skills in VHDL from a professional course with no relation to our university in Denmark.

The time spent on the implementation process covers: the design and implementation of the VHDL code of the functionalities and testbed as well as the tests of the different modules in the applications. This data is shown in the lower part of Table 3. The time data originate from the company's internal registration for the project, and correspond therefore to the effective time used.

The relation between implementation effort and complexity is plotted in Figure 8. It can be seen that this data, corrected for the designers' experience ($*$) closely follows the model derived from the training data (dashed line). Figure 8 also shows the 95% confidence interval, indicating that, with 95% confidence, future predictions of implementation effort will lie within this interval, given that the model holds true.

Comparing the predicted effort (dashed line) to the real effort ($*$), indicates that there is an estimation error. The values are also shown in Table 4, The average estimation error is 0.2 week with a variance of 8. In the next section, we discuss the validity of the model.

4.3. Validity discussion

Estimating the effort required in implementing an algorithm into hardware involves many parameters. We discussed a number of these parameters in Section 1.2, but could not include them all in this study. The proposed model is therefore devised from the idea of the relation between implementation effort and number of linear-independent paths.

Table 4: Development time and estimated development time measured in weeks together with the error.

Algorithm	Dev. time	Est. dev. time	Error
SS1	3.6	3.3	0.3
SS2	6.4	4.8	1.6
SS3	3.2	2.2	1
SS4	16.4	20.3	−3.9
SS5	12	16.2	−4.2
SS6	17.2	13.8	3.4
Ethernet app	11.4	8.8	2.6
App 4	2	1.1	0.9
Mean (variance):			0.2 (8)

To validate the model, a classical two-phased hypothesis test has been performed and the validity of this test depends on the following important factors: (i) the independence between training and validation data; (ii) the volume and variety of the experiments.

In the first instance, not only different applications were used for training and validation data, but in addition the developers had no relation in terms of education, nationality, work, and so forth. Moreover, the validation data has not been measured before the model was trained. All this strengthens the validity of the results. The only potential connection is that some of the developers who have been involved in the implementation of the training and validation data have also been included within those interviewed. However, this accounts for a minority and we see this as a minimal risk.

Secondly, we should ideally have had a large volume and variety of experimental data for training and validation. However, our set of data originates from a single company and a few developers. So strictly speaking we can only conclude that this model applies to the specific SME setup involved in the study and partially to the academic environment studied.

In order to generalise our model, more cases of validation are needed. However, obtaining all the statistical data for this new methodology is time consuming. We would therefore like to remind the reader that this paper proposes a methodology for estimating implementation effort and the validation of the model concentrates on illustrating its usefulness. Looking at the graphs, we can determine a clear trend in the results. The curve identified in the training data is sustained for the validation data as well: they both fall in line with the underlying rationale, and we are quite confident in the strength of the proposed model.

The results clearly show the necessity for the proposed correction function; the proposed logarithmic nature works well, even though the correction function has not been trimmed to fit the individual developers due to the lack of available data. In this light, our approach must be seen as the engine of a global methodology for the management of design projects, that impose a systematic registration of man-power. With such a registration, a database of the developers' experience can easily be constructed and the

correction function can be trimmed to fit the companies' individual designers. Several iterations of this process would provide convergence towards a more precise estimation of the implementation effort.

The limited data set on which the model is constructed also limits the complexity window to which this model can be applied: having no algorithm with a corrected complexity value larger than 51, extrapolating the model further would weaken the current conclusion. More training data, from larger and more varied projects would allow for a more refined model.

Nevertheless, the results described in this paper are very encouraging with all the real-life cases that we have examined and we are reasonably confident that this model can easily be applied to other types of applications.

5. CONCLUSION

The contribution presented in this paper is a metric-based approach for estimating the time needed for hardware implementation in relation to the complexity of an algorithm. We have deduced that a relationship exists between the number of linear-independent paths in the algorithm and the corresponding implementation effort. We have proposed an original solution for estimating implementation effort that extends the concept of the cyclomatic complexity.

To further improve our solution, we developed a more realistic estimation model that includes a correction function to take into account the designer's experience.

We have implemented this solution in our tool design Trotter of which the input is a behavioural description in C language and the output is the number of independent paths. Based on this output and the proposed model, we are able to predict the required implementation effort. Our experimental results, using industrial Ethernet applications, confirmed that the data, corrected for the designers' experience, follows the derived model closely and that all data falls inside its 95% confidence interval. Using this method iteratively paves the way for an implementation effort estimator of which the accuracy improves continuously after each project.

REFERENCES

[1] D. Blaza, "Embedded systems design state of embedded market survey," Tech. Rep., CMP Media, New York, NY, USA, 2006.

[2] R. Nass, "An insider's view of the 2008 embedded market study," Tech. Rep., CMP Media, New York, NY, USA, 2008.

[3] "Workshop for Danish smes developing embedded systems co-organized by the Danish technological institute, the center for software defined radio (csdr) and the center for embedded software systems (ciss)," Nyhedsmagasinet Elektronik & Data, Nr.1 2008, Aarhus, Denmark, 2008.

[4] O.-H. Kwon, "Keynote speaker: perspective of the future semiconductor industry: challenges and solutions," in *Proceedings of the 44th Design Automation Conference (DAC '07)*, San Diego, Calif, USA, June 2007.

[5] S. McConnell, *Software Estimation: Demystifying the Black Art*, Microsoft Press, Wasthington, DC, USA, 2006.

[6] R. Abildgren, A. Saramentovas, P. Ruzgys, P. Koch, and Y. Le Moullec, "Algorithm-architecture affinity—parallelism changes the picture," in *Proceedings on the Design and Architectures for Signal and Image Processing*, Grenoble, France, November 2007.

[7] M. A. Honey, "The interview as text: hermeneutics considered as a model for analysing the clinically informed research interview," *Human Development*, vol. 30, pp. 69–82, 1987.

[8] B. W. Boehm, C. Abts, A. W. Brown, et al., *Software Cost Estimation with COCOMO II*, Prentice-Hall, Upper Saddle River, NJ, USA, 2000.

[9] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[10] T. Jones, *Programming Productivity*, McGraw-Hill, New York, NY, USA, 1986.

[11] G. C. Low and D. R. Jeffery, "Function points in the estimation and evaluation of the software process," *IEEE Transactions on Software Engineering*, vol. 16, no. 1, pp. 64–71, 1990.

[12] A. J. Albrecht, "Measuring application development productivity," in *Proceedings of the IBM Application Development Symposium*, pp. 83–92, Monterey, Calif, USA, October 1979.

[13] D. R. Jeffery, G. C. Low, and M. Barnes, "Comparison of function point counting techniques," *IEEE Transactions on Software Engineering*, vol. 19, no. 5, pp. 529–532, 1993.

[14] W. Fornaciari, F. Salice, U. Bondi, and E. Magini, "Development cost and size estimation starting from high-level specifications," in *Proceedings of the 9th International Symposium on Hardware/Software Codesign*, pp. 86–91, Copenhagen, Denmark, April 2001.

[15] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, 1976.

[16] D. L. Lanning and T. M. Khoshgoftaar, "Modeling the relationship between source code complexity and maintenance difficulty," *Computer*, vol. 27, no. 9, pp. 35–40, 1994.

[17] T. J. Walsh, "Software reliability study using a complexity measure," in *Proceedings of the National Computer Conference (NCC '79)*, vol. 48, pp. 761–768, AFIPS Press, New York, NY, USA, June 1979.

[18] S. P. VanderWiel, D. Nathanson, and D. J. Lilja, "Complexity and performance in parallel programming languages," in *Proceedings of the 2nd International Workshop on High-Level Programming Models and Supportive Environments (HIPS '97)*, pp. 3–12, Geneva, Switzerland, April 1997.

[19] M. Mastretti, M. L. Busi, R. Sarvello, M. Sturlesi, and S. Tomasello, "VHDL quality: synthesizability, complexity and efficiency evaluation," in *Proceedings of the European Design Automation Conference with EURO-VHDL (EURO-DAC '95)*, pp. 482–487, IEEE Computer Society Press, Brighton, UK, September 1995.

[20] I. Feghali and A. H. Watson, "Clarification concerning modularization and mccabe's cyclomatic complexity," *Communication of the ACM*, vol. 37, no. 4, pp. 91–94, 1994.

[21] B. Henderson-Sellers, "Modularization and mccabe's cyclomatic complexity," *Communication of the ACM*, vol. 35, no. 12, pp. 17–19, 1992.

[22] Y. Le Moullec, N. B. Amor, J.-P. Diguet, M. Abid, and J.-L. Philippe, "Multi-granularity metrics for the era of strongly personalized SOCs," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03)*, vol. 1, pp. 674–679, Munich, Germany, March 2003.

[23] Y. Le Moullec, J.-P. Diguet, N. B. Amor, T. Gourdeaux, and J.-L. Philippe, "Algorithmic-level specification and characterization of embedded multimedia applications with design

trotter," *The Journal of VLSI Signal Processing*, vol. 42, no. 2, pp. 185–208, 2006.

[24] A. Heathcote, S. Brown, and D. J. Mewhort, "The power law repealed: the case for an exponential law of practice," *Psychonomic Bulletin & Review*, vol. 7, no. 2, pp. 185–207, 2000.

[25] S. Ducloyer, R. Vaslin, G. Gogniat, and E. Wanderley, "Hardware implementation of a multi-mode hash architecture for MD5, SHA-1 and SHA-2," in *Proceedings on the Design and Architectures for Signal and Image Processing Workshop (DASIP '07)*, Grenoble, France, November 2007.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 916867, 11 pages
doi:10.1155/2008/916867

Research Article

Multiple Word-Length High-Level Synthesis

Philippe Coussy, Ghizlane Lhairech-Lebreton, and Dominique Heller

Lab-STICC (CNRS), European University of Brittany, The Université de Bretagne-Sud, Centre de Recherche, BP 92116, F-56321 Lorient Cedex, France

Correspondence should be addressed to Philippe Coussy, philippe.coussy@univ-ubs.fr

Received 29 February 2008; Revised 5 May 2008; Accepted 21 July 2008

Recommended by Markus Rupp

Digital signal processing (DSP) applications are nowadays widely used and their complexity is ever growing. The design of dedicated hardware accelerators is thus still needed in system-on-chip and embedded systems. Realistic hardware implementation requires first to convert the floating-point data of the initial specification into arbitrary length data (finite-precision) while keeping an acceptable computation accuracy. Next, an optimized hardware architecture has to be designed. Considering uniform bit-width specification allows to use traditional automated design flow. However, it leads to oversized design. On the other hand, considering non uniform bit-width specification allows to get a smaller circuit but requires complex design tasks. In this paper, we propose an approach that inputs a C/C++ specification. The design flow, based on high-level synthesis (HLS) techniques, automatically generates a potentially pipeline RTL architecture described in VHDL. Both bitaccurate integer and fixed-point data types can be used in the input specification. The generated architecture uses components (operator, register, etc.) that have different widths. The design constraints are the clock period and the throughput of the application. The proposed approach considers data word-length information in all the synthesis steps by using dedicated algorithms. We show in this paper the effectiveness of the proposed approach through several design experiments in the DSP domain.

1. INTRODUCTION

Electronic devices are more and more oriented towards multimedia and communication applications. The design of system-on-chip (SoC) is currently achieved by using system level description, electronic system level (ESL) tools, and by reusing predesigned IP-cores. Typical MPSoC architectures include several processors, memories, I/O devices, communication media (bus, network-on-chip), and dedicated HW accelerators. Indeed, the increasing complexity, the low-power design constraints, and the growing data rates of applications from the digital signal processing (DSP) domain still often require hardwired implementation to be used as a dedicated accelerator in the final SoC. Due to both the complexity of today's DSP applications and the shrinking time-to-market, designers need a more direct path from the functionality down to the silicon. Layered design flow and associated CAD tools to manage DSP system complexity in a shorten time are thus needed. This led to the development of environments that can help the designer to explore the design space thoroughly and to find optimized designs rapidly.

Typically, the design of a DSP application begins by a high-level specification capture of the desired functionality using MATLAB-/Simulink-like environment [1] and/or C/C++ language. This first step consists thus in writing an algorithmic specification with a purely transformational semantics, that is, a function consumes all its input data simultaneously, performs all of computations without any particular timing behavior, and provides all its output data at the same time. At this abstraction level, variables are purely functional (structure, array, etc.), and the data types (typically floating point and/or 16, 32, or 64 bits integer) are not related to the hardware design domain (bit, bit vector). Realistic hardware implementation thus requires the following: (1) to convert the floating-point data types into arbitrary length data types (fixed-precision) while keeping acceptable computation accuracy and (2) to design an optimized hardware architecture starting from this bit-accurate algorithmic specification.

Word-length determination has been early addressed in the literature (see, e.g., [2–4]). This complex and major task in the design flow of DSP applications can be done by using

the following approaches [5–10] when targeting hardware components (DSP, ASIC/FPGA, etc.). Next, the design flow has to take into account bit-width information (i.e., data and operation word-length) in order to carry out an optimized architecture. The design can be done by hand or by using high-level synthesis (HLS) tool. As to be mentioned in Section 2, combining word-length optimization and high-level synthesis allows the hardware implementation cost reduction. This assumes that efficient bit-width aware HLS tools are provided.

In this paper, we present a high-level synthesis approach that inputs specification of digital signal processing application using both bit-accurate integers and fixed point integers. Our approach optimizes area of hardware architectures by taking into account the bit-width information during all the HLS steps. This paper is organized as follows: first, Section 2 presents related work around bit-width aware design steps; Section 3 introduces the general design flow we propose and details the design steps that allow synthesizing optimized multiple word-length architecture; finally, Section 4 shows the effectiveness of our methodology and tool through several experiments in the DSP domain.

2. RELATED WORK

A lot of high-level synthesis work has been presented for two decades [11–13]. However, conventional techniques usually rely on uniform bit-width specification. The works that address the design of multiple bit-width architectures often focus on particular steps of the synthesis flow (see Figure 1). Moreover, to our knowledge, no work has been published on the HLS of fixed-point specification: only bit-accurate integer data type is considered in the literature even if the term "fixed-point" is used.

When the word-length determination is partial, the bit-width refinement, which is the first step of HLS flow, determines bit-width requirements for all integer variables and operations which sizes have not been defined in the input specification. In [14], a forward and a backward propagations are used to infer the minimum bits required.

Next, in order to keep scheduling and binding bit-width unaware (and thus use traditional HLS flows), a clustering is realized. This clustering step groups the operations according to their characteristics into *clusters*. Two operations that belong to the same *cluster* will be able to share an operator if they are not scheduled in the same control step. In order to cluster operations, works from [9, 14, 15] use greedy heuristics while an iterative approach is proposed in [16]. In [14], a bit-width unaware iterative modulo scheduling is performed, afterwards, to satisfy a throughput constraint.

These approaches consider both computational function and area as partitioning criteria and assume latency of operators to be uniform. Unfortunately, larger bit-width operators have longer latency than smaller bit-width operators (see Figure 3): the longest latency will be assigned to all the compatible operations. This can lead to oversized parallel architecture when latency or throughput constraints are considered.

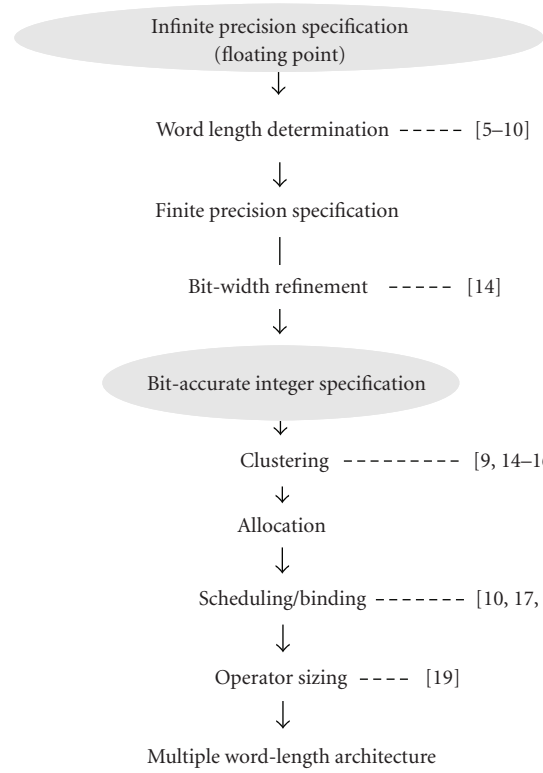


Figure 1: High-level synthesis flow and multiple word-length architecture design approaches.

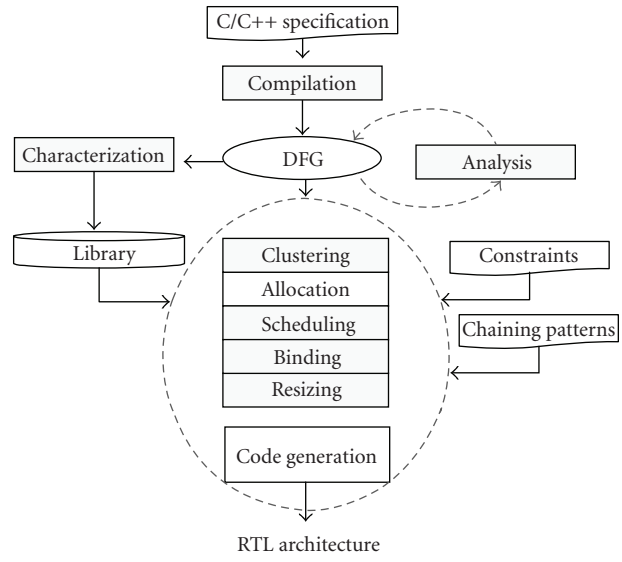


Figure 2: Proposed design flow.

In [10, 16, 17], the word-length optimization is coupled with the high-level synthesis to minimize the hardware implementation cost. In [16], the operations are list-scheduled by decreasing order of timing mobility and no detail on the binding algorithm is provided. Work from [10, 17] is based on a bit-width aware high-level synthesis. A list-scheduling and a register binding algorithm based on the

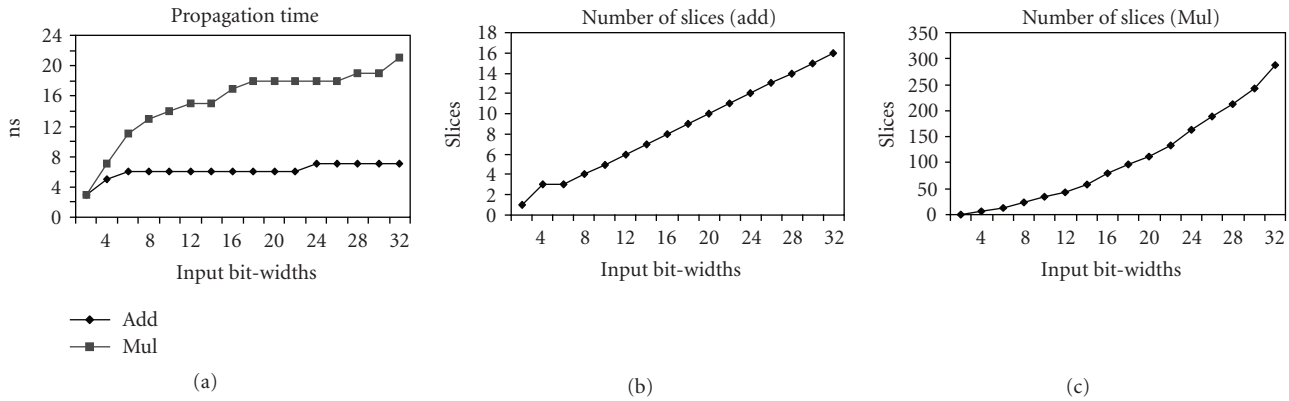


Figure 3: Propagation time and area of multiplier and adder according to the size of their inputs.

clique partitioning are described in [10]. The ready operation list with the largest word-length has the highest priority, and the largest scheduled operations are bound first. In [17], an ILP-based approach and a heuristic are presented. Both proposed methods are introduced to perform scheduling with incomplete word-length information and to combine binding and word-length selection. The refinement of word-length information is based on the critical path analysis.

Work from [18] proposes a bit-width aware HLS flow which does not involve clustering step. Authors first compute an estimated area lower-bound to next schedule and bind operation under latency constraint.

Finally, in [19], authors only propose to resize the operators. They first use an HLS tool which does not take into account bit-width information. The proposed optimization step is done after both the scheduling and the binding tasks. To resize the operators and the registers, a forward propagation of the value range of variables is realized.

To our knowledge, none of the previous works jointly

(i) develop a fully automated design flow that inputs specification mixing both bit-accurate integer and fixed-point variables;

(ii) analyze the specification by combining both bit-width and value range bidirectional propagation;

(iii) use propagation time of operators to cluster multiple word-length operations;

(iv) propose bit-width aware scheduling and binding algorithms;

(v) automate operator reusing between bit-accurate integer and fixed-point operations;

(vi) support operator chaining;

(vii) resize operators to optimize the logic synthesis to generate a multiple-word length architecture.

3. BIT-WIDTH AWARE DESIGN FLOW

The proposed high-level synthesis flow is presented in Figure 2. Starting from a purely functional specification, a technological library and both a throughput (iteration period) and a clock period constraints, our tool extracts the potential parallelism before clustering, allocating, scheduling, and assigning operations. It generates a potentially pipelined architecture composed of a processing unit, a memory unit, a communication and multiplexing unit with a globally asynchronous locally synchronous/latency insensitive system (GALS/LIS) interface (see [20] for more details).

The following subsections detail each step of the bit-width aware HLS flow we propose.

3.1. Compilation and bit-width analysis

The input description is a C/C++ function wherein Algorithmic C^{TM} class library from Mentor Graphics [21] is used. This allows the designer to specify signed and unsigned bit-accurate integer and fixed-point variables by using ac_int and ac_fixed data types. This library, like SystemC [22], hence provides fixed-point data-types that supply all the arithmetic operations and built-in quantization (rounding, truncation, etc.) and overflow (saturation, wrap-around, etc.) functionality. For example, an ac_fixed ⟨5, 2, true, AC_RND, AC_SA⟩ is a signed fixed-point number of the form bb.bbb (5 bits of width, 2 bits integer) and with quantization mode set to rounding and overflow mode set to saturation.

The role of the compiler is to transform this initial specification into a formal representation which exhibits the data dependencies between the operations. The front end of our tool derives GCC 4.2 [23] to extract a data flow graph (DFG) representation of the application annotated with the bit-width information. The code optimizations (such as dead-code elimination, false data-dependency elimination, loop transformations, etc.) performed by the compiler will not be presented in this paper. For the quantization/overflow functionality of a fixed-point variable, the compiler generates dedicated operation nodes into the DFG. As described later in Section 3.4, this allows to share (i.e., reuse) (1) arithmetic operators between bit-accurate integer operations and fixed-point operations and (2) quantization/overflow operators between fixed-point operations. Timing performance optimization is addressed through the operator chaining (see Section 3.4).

```
Inputs:
   DFG, timing constraint II and resource allocation

Output:
   A scheduled DFG

Begin
   c-step = 0;
   Repeat until the last node is scheduled
      Determine the ready operations RO;
      Compute the operation mobility;
      While there are RO
         If there are available resources avail(OPR)
            Schedule the operation ops with the highest priority;
            Remove resource opr from available resource set;
            If the current operation ops belongs to a chaining pattern
               Update the ready operations RO;
               If there are available resources
                  Schedule the operations corresponding to the pattern;
                  Remove resources from available resource set;
               End if
            End if
         Else
            If the operations can be delayed
               Delay the operations;
            Else
               Allocate resources();
               Schedule the operations;
            End if
         End if
      End while
      Bind all the scheduled operations;
      c-step++;
End
```

Algorithm 1: Pseudocode of the scheduling algorithm.

The starting point of the proposed design flow is an already refined specification (the floating-point to fixed-point conversion is not addressed in this paper). However, even if the specification has been already refined (by using existing approaches like [5–10], etc.) the sizes of some variables and constants (#define, int, etc.) and/or variable DFG nodes automatically inferred by the compiler can remain undefined. In order to define the size of such data, the bit-width analysis has been proposed. This step operates on the DFG the two following tasks.

(i) *Constant bit-width definition*: the compiler carries out a DFG representation wherein the constants are represented by nodes with a 16, 32, or 64 bit sizes. This first analysis step defines for each integer (fixed point) constant the exact number of bits needed to represent its value (integer part). We use the following formula for unsigned and signed values:

$$\text{Number of bits} = \lfloor \log_2 |\text{Value}(\text{Cst\_}X)| \rfloor + 1 + S_{\text{igned}}(\text{Cst\_}X), \quad (1)$$

where Value(Cst_X) is the numeric value of the constant Cst_X and S_{igned}(Cst_X) is set to "1" when Cst_X is signed and set to "0" otherwise.

(ii) *Bit-width and value range propagation* infers the bit-width of each variable of the specification by coupling work from [14, 19]. A bit-width analysis is hence performed to optimize the word-length of both the operations and the variables. This step performs a forward and a backward propagation of both the value ranges and the bit-width information to figure out the minimum bits required.

3.2. Library characterization and clustering

Library characterization uses a DFG, a technological library, and a target technology (typically the FPGA model). This fully automated step, based on commercial logic synthesis tools like ISE from Xilinx [24] and Quartus from Altera [25], carries out a library of time characterized operators to be used during the following HLS steps. The technological library provides the VHDL behavioral description

of operators and the DFG provides the set of operation to be characterized with their bit-width information. The characterization step synthesizes each operator from the technological library which is able to realize one operation of the DFG. It next retrieves synthesis results in terms of logical cell number and propagation time to generate a characterized operator library (see Figure 3).

For clustering operations, we propose to combine the computational function and the operation delay. This allows considering indirectly operation's bit-width since the propagation time of an operator depends on its input size. In order to maximize the use of operator, one operation that belongs to a cluster C1 with a propagation time t1 can be assigned to operators allocated for a cluster C2 if the propagation time t2 is greater than t1.

3.3. Resource allocation

Allocation next defines the number of selected operators needed to satisfy the design constraints. In our approach, in order to respect the throughput requirement specified by the designer, allocation is done for each a priori pipeline stage. The number of a priori pipeline stage is computed as the ratio between the minimum latency of the DFG *Latency* (i.e., the longest data dependency path in the graph) and the iteration interval II (i.e., the period at which the application has to iterate): ⌈Latency/II⌉. Thus, we compute the average parallelism of the application extracted from the DFG dated by a as soon as possible ASAP scheduling [11]. The average parallelism is calculated separately for each *type* of operation and for each pipeline stage s of the DFG, comprising the set of the date operations belonging to $[s \cdot \mathrm{II}, (s+1) \cdot \mathrm{II}]$.

This first allocation is considered as a lower bound. Thus during the scheduling phase, supplementary resources can be allocated and pipeline slices may be created if necessary, subsequent to operation scheduling on the previously allocated operators.

3.4. Operation scheduling

The classical list-scheduling algorithm relies on heuristics in which ready operations (operations to be scheduled) are listed by priority order. An operation can be scheduled if the current cycle is greater than its earliest time. Whenever two ready operations need to access the same resource (this is a so-called resource conflict), the operation with the highest priority is scheduled. The other is postponed.

Traditionally, bit-width information is not considered and the priority function depends on the mobility only. The operation mobility (mobility in the following formula) is thus only defined as the difference between the as-late-as possible (ALAP as late as possible) time and the current c-step (see Algorithm 1).

In order to optimize the final architecture area, we modified the classical priority function to take into account the operation bit-width in addition to its mobility. Hence the priority of an operation ops, priority(ops_i), is a weighted sum of (1) the inverse of its mobility mobility(ops_i) (i.e., its timing priority) and (2) the inverse of the overcost inferred by the pseudoassignment of the largest available and compatible operator opr_j with the operation ops_i. The operator opr_j is returned by the maxsize(avail(OPR, ops_i)) function, where OPR is the set of allocated operators and avail(X, y) is a function that returns from the set of operators X, the set of available operators compatible with at least one of the operations y is as follows:

$$\mathrm{priority}(\mathrm{ops}_i) = \frac{\alpha}{\mathrm{mobility}(\mathrm{ops}_i)} + \frac{1-\alpha}{\mathrm{overcost}(\mathrm{ops}_i, \mathrm{opr}_j)}$$

$$\mathrm{opr}_j = \mathrm{maxsize}(\mathrm{avail}(\mathrm{OPR}), \mathrm{ops}_i)$$

$$\mathrm{overcost}(\mathrm{ops}_i, \mathrm{opr}_j) = \mathrm{Min}\left\{\left(\frac{\mathrm{opr}_{j,\mathrm{in1}} - \mathrm{ops}_{i,\mathrm{in1}}}{\mathrm{opr}_{j,\mathrm{in1}}} + \frac{\mathrm{opr}_{j,\mathrm{in2}} - \mathrm{ops}_{i,\mathrm{in2}}}{\mathrm{opr}_{j,\mathrm{in2}}}\right), \left(\frac{\mathrm{opr}_{j,\mathrm{in2}} - \mathrm{ops}_{i,\mathrm{in1}}}{\mathrm{opr}_{j,\mathrm{in2}}} + \frac{\mathrm{opr}_{j,\mathrm{in1}} - \mathrm{ops}_{i,\mathrm{in2}}}{\mathrm{opr}_{j,\mathrm{in1}}}\right)\right\}, \quad (2)$$

where $\mathrm{ops}_{i,\mathrm{in}k}$ ($\mathrm{opr}_{j,\mathrm{in}k}$) is the bit-width of the kth input of the operation ops_i (operator opr_j).

The overcost function returns the lowest sum of the gradients of operation input's bit-width and of operator input's bit-width. This means that for a same mobility, the priority will be given to the operation that best minimizes the overcost. For different mobility, the user defined factor α allows to increase the priority of an operation ops_i having more mobility than an operation ops_k if overcost(ops_i, opr_j) is less than overcost(ops_k, opr_j). In the overcost computation, the reuse of an operator (already used) is avoided through a pseudoassignment made during the scheduling. A pseudoassignment is a preliminary binding which allows to remove the largest operator from the available resource set. Once the operations can be no more scheduled in the current cycle, the resource binding is performed.

Operation chaining

To respect the specified throughput and clock constraints while optimizing the final area, operator chaining can be used. In our approach, the candidates for chaining are identified by using templates in a library. Through a dedicated specification language (see Figure 4), the user defines chaining patterns with their respective maximum delays. The latency constraint is expressed in number of clock cycles which allows to be bit-width independent in the pattern specification.

In order to automate arithmetic operators sharing between bit-accurate and/or fixed-point operations from a specification using the Algorithmic C$^{\mathrm{TM}}$ from Mentor Graphics [21], our compiler generates for fixed-point operations two nodes in the DFG (see Section 3.1): one node for the arithmetic operation and one other for the quantization/overflow functionality. Indeed, the difference between integer and fixed point comes from the operator implementation. Hardware operators have the same architecture and the same area only if one considers the

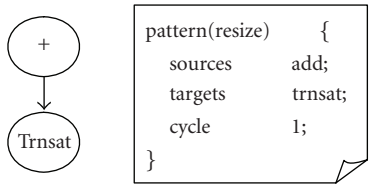


Figure 4: Chaining pattern specification language.

computation part. However, for fixed-point, quantization (rounding, truncation, etc.) and overflow (saturation, wrap-around, etc.) operations can be specified. Of course, the quantization/overflow operations can be part of all the operators (integer and fixed-point). But, this would affect the latency of all the operators and thus the overall latency of the final architecture. This would also increase the overall area. In order to share the computation part only, the compiler has to generate specific operation nodes in the internal representation for the quantization/overflow operations. This allows improving both the area and the timing performance of the final area.

Figure 5(a) depicts a fixed-point dedicated operator, where computational part is merged with the quantization/overflow functionality. This kind of operator architecture allows sharing neither the arithmetic logic nor quantization/overflow part between bit-accurate and/or fixed-point operations.

Figure 5(b) shows the resulting architecture when the compiler generates dedicated nodes for a fixed-point operation and that chaining is not used.

Figure 5(c) presents an architecture wherein the arithmetic part and the quantization/overflow functionality have been chained by coupling both the compiler results and fixed-point templates.

3.5. Resource binding

The assignment of an available operator with a candidate operation has to respond to the minimization of interconnections (steering logic) between operators and to the minimization of the operator's size (see Figure 3 and [18]). Given the set of allocated operators, our binding algorithm assigns all the scheduled operations of the current step (see Algorithm 1). The pipeline control of each operator is managed by a complementary priority on assignment. When an operator is allocated, but not yet used, its priority for assignment is primarily inferior to that of an operator already utilized.

The first step consists in constructing a bipartite weighted graph $G = (U, \text{avail}(\text{OPR}, U), E)$ with:

(i) U, the set of operations in c-step S_k of the DFG;

(ii) avail(OPR, U) is a function that retruns from the set of operators OPR, the set of available operators compatible with the operations from U;

(iii) E, the set of weighted edges (U, avail(OPR)) between a pair of an operation $\text{ops}_i \in U$ and an operator $\text{opr}_j \in \text{avail}(\text{OPR}, U)$;

The edge weight $w_{\text{ops}_i,\text{ops}_j}$ is given by the following equation:

$$w_{\text{ops}_i,\text{opr}_j} = \beta * \text{con}(\text{ops}_i, \text{opr}_j) + (1-\beta) * \text{dis}(\text{ops}_i, \text{opr}_j), \tag{3}$$

where

(i) $\text{con}(\text{ops}_i, \text{opr}_j)$ is a weighted summation of the maximum number of existing connections between opr_j and each operator assigned to the predecessors of ops_i, and a possible future connection with an operator that could be assigned to the successors of ops_i (i.e., operation/operator compatibility);

(ii) $\text{dis}(\text{ops}_i, \text{opr}_j)$ is the reciprocal of the positive difference between bit-widths of ops_i and opr_j inputs;

(iii) β is user defined factor which allows minimizing either steering logic area or computational area.

The second step consists in finding the maximal weighted edge subset by using the maximal bipartite weighted matching (MBWM) algorithm described in [26].

Let us now consider the following example.

Assuming

(i) the scheduling and binding of the operations of the DFG in Figure 6(a) on c-step1 and c-step2 have been already done,

(ii) the operations ops_1 and ops_4 have been scheduled in c-step3,

(iii) allocated operators are opr_1, opr_2, and opr_3,

(iv) ops_9, ops_7 have been bound to opr_1,

(v) ops_3, ops_0 have been bound to opr_3,

we will focus on ops_1 and ops_4 binding. Our algorithm first constructs the bipartite weighted graph (Figure 6(b)) taking β equal to 1 for the sake of simplicity (i.e., only steering logic is considered). Afterwards, the MBWM algorithm is applied to identify the best edges. Thus operation ops_1 is assigned to opr_1 thanks to the edge weight 3 in Figure 6(b). Edges connected to opr_1 node are then removed (see Figure 6(c)). In other word, connection between opr_3 (FU implementing the ops_1 predecessor) and opr_1 is maximized. Thereby, the creation of multiplexers is avoided, and thus the final architecture has been optimized.

3.6. Operator sizing

In this design step, the operators have to be sized according to the operations which have been assigned on. In order to get correct computing results, the width of the operator inputs/outputs have to be greater or equal to the width of the operation variables.

In the available literature, the input's width of an operator is used to be the maximum of all its inputs (see [14, 18], e.g.). This computing method increases considerably the final area (see Figures 3, 7, and [16]). However, an operator can have different input width. Thus operator sizing task

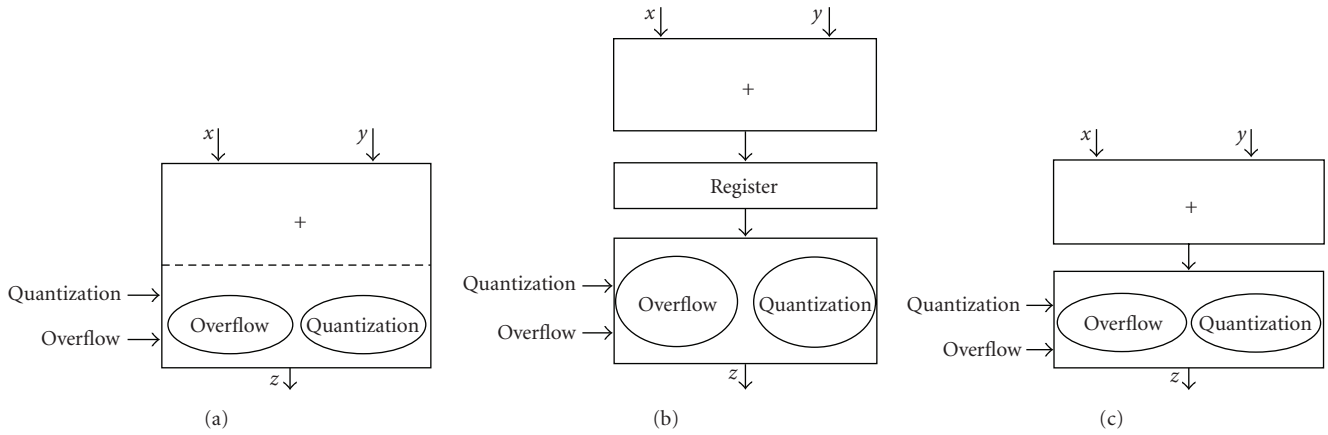


Figure 5: (a) Monolithic fixed-point operator, (b) "unchained" fixed-point operator, and (c) chained fixed-point operator.

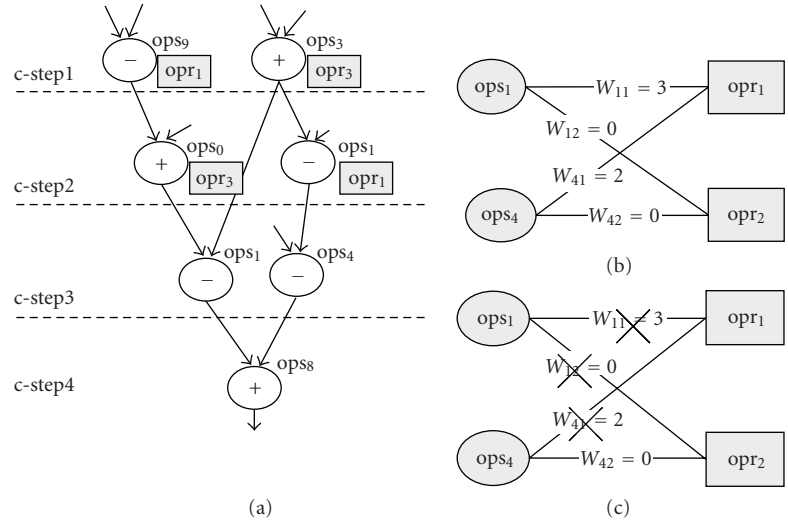


Figure 6: (a) DFG example, (b) bipartite weighted graph, and (c) maximal-weighted edge matching.

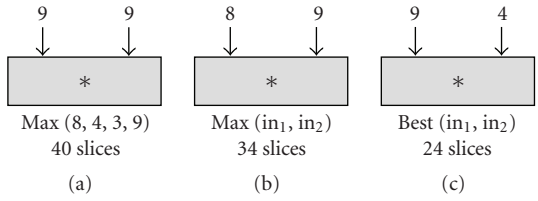


Figure 7: Sizing results.

can optimize the final operator area by (1) computing the maximum width for each input, respectively, (Figure 7(b)) or (2) computing the optimal size for each input by considering commutativity (Figure 7(c)). However, swapping inputs can infer steering logic.

Let us consider a multiplier that executes two operations ops_1 and ops_2. Their respective input widths are ($in_1 = 8$, $in_2 = 4$) and ($in_1 = 3$, $in_2 = 9$) and output width is 12.

Figure 7 shows, respectively, for each approach the synthesis results obtained by using a Xilinx Virtex2 xc2v8000 FPGA device and the ISE 8.2 logic synthesis tool. Considering different widths for each input can thus reduce considerably the operator area.

4. EXPERIMENTS

To show the effectiveness of our approach and its associated high-level synthesis tool [20], several experiments with some well-known DSP applications have been made. All the experiments have been realized on Xilinx Virtex2 xc2v8000-4 FPGA device using ISE 8.2 logic synthesis tool. All the algorithmic specifications have been done using the C language and the Algorithmic C class library from Mentor Graphics [21]. All the syntheses have been realized using a 10 nanoseconds clock period constraint. Experiments have been done using a 2.59 GHz AMD Opteron Processor 252, with 3G RAM and running Windows XP. The execution time of each synthesis using our tool was less than 2 minutes for the considered examples. The RTL descriptions have been simulated using Modeltech's ModelSim6.1b simulator and

the functional validation has been completed by comparing RTL results to the C/Algorithmic C model ones.

These experiments objective is to present the interest of the full bit-aware HLS flow presented in this paper. First, we show the effectiveness of our bit-width aware synthesis flow with only our resizing step. For this purpose, comparisons have been made against bit-width unaware flows. Second, we highlight the interest of our proposed "bit-width aware" binding and scheduling algorithms, as well as the benefit of the combination of all the synthesis steps. Finally, the last experiment outlines the usefulness of arithmetic operators sharing between bit-accurate and/or fixed-point operations.

4.1. The proposed operator resizing approach

In this first experiment, we compare the three following methodologies:

(i) a pure C ANSI specification with a *classical* HLS flow. All the variables and the constants have thus been defined as integers, that is, their size depends only on the compiler and on the technological library. Standard integer widths have thus been considered, that is, 16 and 32 bits;

(ii) a C/C++ specification using bit-accurate integers (through the Algorithmic C^{TM} class library) with a *uniform* HLS approach that does not consider word-length information. The *uniform synthesis* approach generates thus an oversized architecture where all the components (logical and arithmetic operators, registers, multiplexers, etc.) have the same width, that is, the width of the largest data of the specification;

(iii) a C/C++ specification using bit-accurate integers (through the Algorithmic C^{TM} class library) with the *proposed bit-width aware* HLS approach applying only the resizing step (i.e., the proposed bit-with aware scheduling and binding steps have not been used in this first experiment).

In the *classical* approach, the specification was written by using ANSI C, where the data were of type *int*. Each data has thus been represented with 16 bits in the DFG. In the *uniform* synthesis, we set (by using the Algorithmic C data types) the width of all the variables equal to the width of the largest data of the input specification. For this purpose, we used the bit-width analysis tool presented in Section 3.1 to infer output data widths by propagating the input data bit-widths.

Figure 8 presents the results provided by the three approaches previously described through three well known DSP applications. The first application is a low-pass video line filter. The filter has five nonzero taps with symmetric coefficients, 96 pixels and the image is padded with the boundary pixels. The dataflow graph provided by the compilation step contains 1354 nodes (725 data and 629 operations) with a width varying from 4 to 16 bits. From the figure, standard 16 bits architecture obtained with *classical* flow and *uniform* one (set to 16 bits to respect the computing dynamic of the example) have obviously the same area cost. In contrast, *bit-accurate* architecture is almost two-thirds and thus achieves 27% area reduction (see Figure 8(a)).

The second application is an elliptic filter with 25 operation nodes and 35 data nodes. Data bit-width varies between 4 and 15. *Uniform* architecture is slightly smaller than *classical* 16 bits one, saving only 4.5% amount of area whereas our *bit-accurate* architecture exploits data word-length to reach 40% area saving (see Figure 8(b)).

For the third application (Figure 8(c)), we chose a four taps infinite impulse response IIR filter containing 23 data nodes and 8 operation nodes shared between addition and multiplication operations. Data width varies from 2 up to 14 bits. This experiment shows that *uniform* approach is sometimes unprofitable and leads to an architecture larger than the 16 bits *classical* one. This is due to the fact that the logic synthesis tool uses, for standard widths, optimized hardwired resources [24]. Using our bit-accurate approach reduces noticeably the total area which is two times smaller than the *classical* 16 bits architecture (i.e., 50% area saving).

When using a compiler that considers the integers as 32 bits data, the *classical* approach generates obviously a 32 bit-width architecture. In this case, area waste is considerable. The final area obtained for the line filter and elliptical filter examples is almost three times lager than bit-accurate area (area reduction ratio is around 70%). The most noticeable area reduction occurs for the IIR filter where area is reduced to the fifth reaching 80% area saving. However, when at least one data requires more than 16 bits, the 32-bits compiler has to be used with the *classical* approach.

We synthesized a 16-taps finite impulse response FIR filter where 28-bits data were needed. Figure 9 shows, respectively, 82% and 80% of area reduction obtained by the proposed *bit-accurate* approach compared to the 32-bits *classical* and *uniform* (28 bits) approach.

This first experiment set has shown the interest of the resizing algorithm, we proposed in this paper, which provides alone a large area reduction. Further optimizations can be obtained by using scheduling and binding bit-width aware algorithms as presented in Section 4.2.

4.2. Proposed scheduling and binding algorithms

The scheduling step is a list-scheduling algorithm with a priority criterion combining both mobility and operation overcost (see Section 3.4). A parameter α controls the impact of operation overcost in the operation selection. The binding step has to respond to the minimization of interconnections (steering logic) and to the minimization of the components size (operators and registers). As described in Section 3.5, our binding algorithm provides a parameter β to find a tradeoff between the computation and the routing area. The following experiments were performed with α set to 0.3, meaning that the mobility has greater impact than the operation overcost on the scheduling priority. β has been set to 0.6, meaning that routing area has a greater priority than the combinatorial area. These values were chosen out from a dozen of experiments and outcome, in average, the best area results.

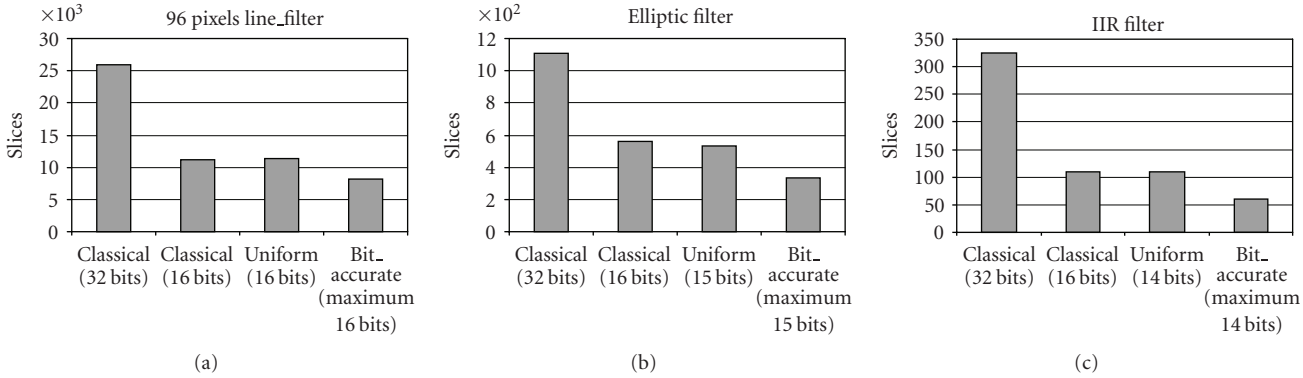


Figure 8: Uniform versus bit-accurate architecture area for (a) line filter, (b) elliptic filter, and (c) IIR filter.

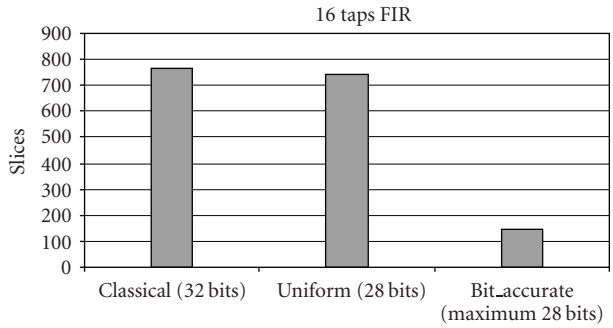


Figure 9: 16 taps FIR design.

This second experiment set shows the interest of considering bit-width information during synthesis steps through tree DSP applications; (1) 96 pixels line filter, described previously; (2) Volterra which consists in first order, nonlinear, differential equations; and (3) a 32 points fast Fourier transform FFT. Figure 10 shows the synthesis results of the presented applications at different throughput constraints.

The line filter design was kept in this set of experiments in order to highlight the interest of optimization algorithms. Actually, the reduction ratio obtained in the first experiment, that is, 70% (see Section 4.1) is improved. The proposed clustering algorithm performs an extra reduction by 6%. The most optimization ratio, 9% is obtained by combining all steps. Thus total area saving is 76.3%.

The Volterra design was synthesized under two different throughputs constraints; 50 Mb/s and 10 Mb/s which leads, respectively, to a 2-stage pipeline architecture and a non-pipeline one. When throughput is equal to 50 Mb/s, area reduction reaches 25% by combing the proposed clustering and binding algorithms. Otherwise, the binding algorithm alone improves area cost by 14% and the clustering algorithm provides an improvement about 18%. Under low throughput constraints, that is, 10 Mb/s, operators are hardly shared. Scheduling and binding algorithms have no effect on total area. Only clustering algorithm manages to reduce area by 10%.

From the 32 points FFT design, we notice that the classical clustering (which does not take into account the propagation time of operators) can lead to oversized area. Indeed, Figure 10(c) shows for abscise $X = (1), (2), (3)$, and (4) a circuit area with a throughput of 4 Mb/s greater or equal that a circuit area that satisfies a 6.66 Mb/s constraint. The proposed time clustering algorithm alone has reduced the area, respectively, for throughputs 4 Mb/s and 6.66 Mb/s, by 6% and 1.6%. Moreover, coupled with the binding algorithm, the gain is raised, respectively, to 6.7% and 3.4%.

The results point out that making each step of the HLS flow "bit-width aware" allows to optimize the circuit area. Actually, the area obtained with an HLS flow which only resizes the operators is reduced utmost by 25% for the Volterra application. Furthermore, a mean reduction of 13% is observed.

4.3. Operator reusing between bit-accurate integer and fixed-point operations

Decoupling the arithmetic part and the quantization/overflow functionality (see Section 3.1) for fixed point operators allows the resources sharing as described in Section 3.4. Performance optimizations are addressed through the operator chaining. In this subsection, efficiency of operator reusing is shown through the synthesis of a 32 taps least mean squares (LMSs) filter. This application consists of two distinct computations (1) adaptive coefficient computation which is described with fixed-point data types (ac_fixed) and (2) filtered sample computation specified with bit-accurate integer data types (ac_int).

Figure 11 compares total area of designs produced by the traditionalapproaches and the proposed one. The traditionalapproaches consider fixed-point operator as monolithic. Thus they generate an architecture composed by one integer and one fixed-point subpart which cannot share the operators. Otherwise, the *proposed* approach leads to a unified circuit where operators are shared between bit-accurate integer and fixed-point operations. The proposed approach hence provides a reduction ratio of 10%.

5. CONCLUSION

We have presented in this paper a high-level synthesis flow, which allows the design of multiple word-length

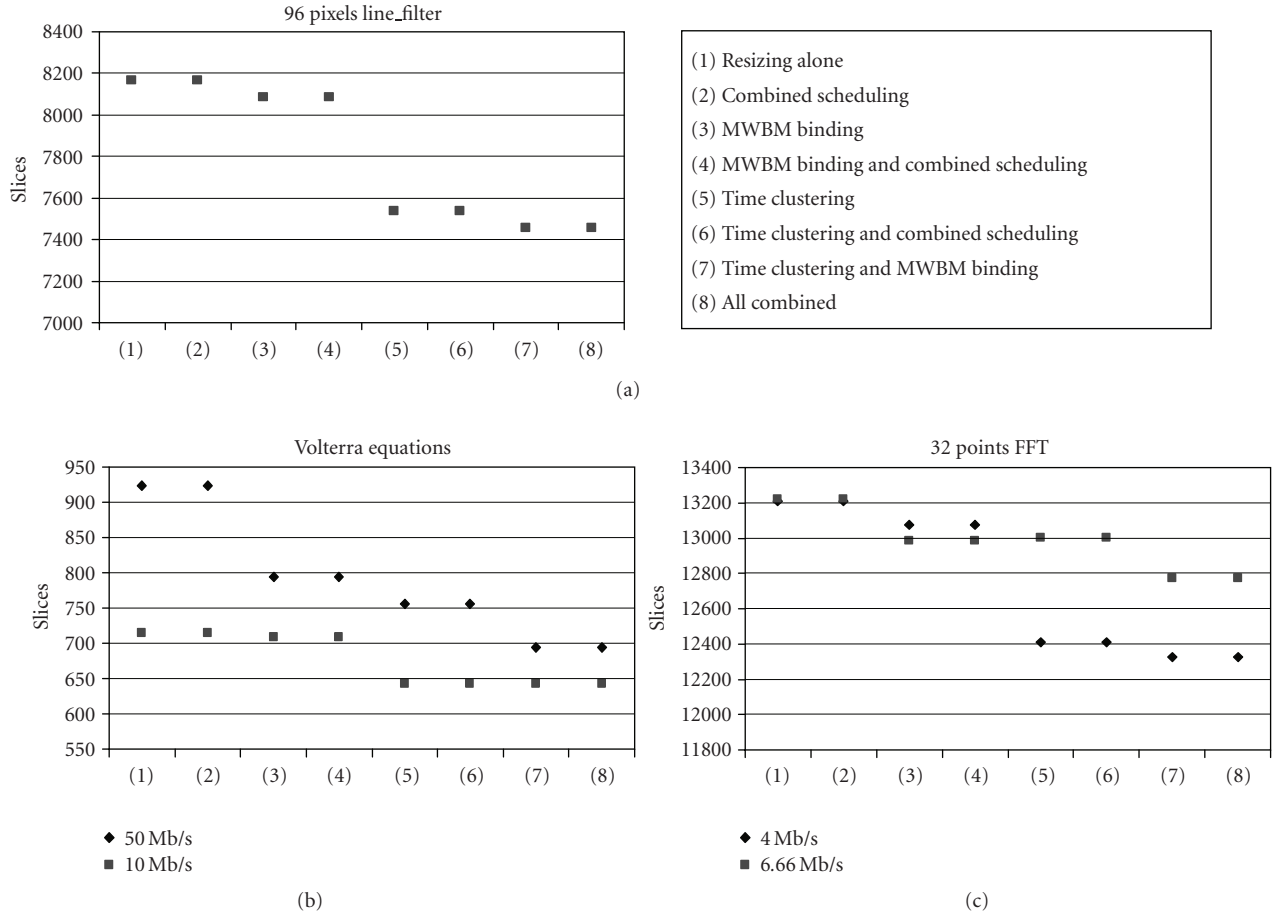


Figure 10: Impact of the bit-width aware design steps for (a) line filter, (b)Volterra, and (c) 32-points FFT.

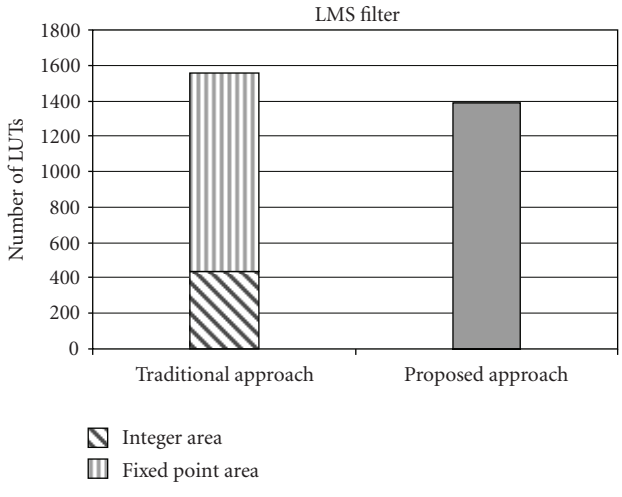


Figure 11: Operator sharing approach versus monolithic approach.

architectures. The approach is based on dedicated compilation, scheduling, binding, and sizing steps that allow optimizing the final architecture area. The experiments have shown promising results for overall area reduction through well known DSP applications on a Xilinx Virtex-2 FPGA.

In the future work, the effects on power consumption will be investigated and the results will be compared with traditional HLS design flows.

REFERENCES

[1] Mathworks, http://www.mathworks.com/.

[2] L. B. Jackson, "Roundoff-noise analysis for fixed-point digital filters realized in cascade or parallel form," *IEEE Transactions on Audio and Electroacoustics*, vol. 18, no. 2, pp. 107–122, 1970.

[3] L. B. Jackson, "Roundoff-noise bounds derived from coefficient sensitivities for digital filters," *IEEE Transactions on Circuits and Systems*, vol. 23, no. 5, pp. 481–485, 1976.

[4] C. T. Mullis and R. A. Roberts, "Synthesis of minimum roundoff noise fixed point digital filters," *IEEE Transactions on Circuits and Systems*, vol. 23, no. 9, pp. 551–562, 1976.

[5] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE '98)*, pp. 429–435, Paris, France, February 1998.

[6] L. Wanhammar, *DSP Integrated Circuits*, Academic Press, New York, NY, USA, 1999.

[7] M. Budiu, S. Goldstein, K. Walker, and M. Sakr, "Bitvalue inference: detecting and exploiting narrow bitwidth computations," in *Proceedings of the 6th International Euro-Par*

Conference on Parallel Processing, A. Bode, T. Ludwig, W. Karl, and R. Wism"uller, Eds., vol. 1900 of *Lecture Notes in Computer Science*, pp. 969–979, Springer, Munich, Germany, August-September 2000.

[8] M. Stephenson, J. Babb, and S. Amarasinghe, "Bitwidth analysis with application to silicon compilation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '00)*, pp. 108–120, Vancouver, Canada, June 2000.

[9] S. A. Wadekar and A. C. Parker, "Accuracy sensitive word-length selection for algorithm optimization," in *Proceedings of IEEE International Conference on Computer Design (ICCD '98)*, pp. 54–61, Austin, Tex, USA, October 1998.

[10] K.-I. Kum and W. Sung, "Combined word-length optimization and high-level synthesis of digital signal processing systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 8, pp. 921–930, 2001.

[11] D. D. Gajksi, N. Dutt, S. Y.-L. Lin, and A. Wu, *High Level Synthesis: Introduction to Chip and System Design*, Springer, Berlin, Germany, 1992.

[12] G. De Micheli and D. ku, *High Level Synthesis of ASICs under Timing and Synchronization Constraints*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1992.

[13] P. Coussy and A. Morawiec, Eds., *High-Level Synthesis: From Algorithm to Digital Circuit*, Springer, Berlin, Germany, 2008.

[14] S. Mahlke, R. Ravindran, M. Schlansker, R. Schreiber, and T. Sherwood, "Bitwidth cognizant architecture synthesis of custom hardware accelerators," *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, vol. 20, no. 11, pp. 1355–1371, 2001.

[15] J.-L. Choi, H.-S. Jun, and S.-Y. Hwang, "Efficient hardware optimisation algorithm for fixed point digital signal processing ASIC design," *Electronics Letters*, vol. 32, no. 11, pp. 992–994, 1996.

[16] N. Hervé, D. Ménard, and O. Sentieys, "Data wordlength optimization for FPGA synthesis," in *Proceedings of IEEE Workshop on Signal Processing Systems: Design and Implementation (SiPS '05)*, pp. 623–628, Athens, Greece, November 2005.

[17] G. A. Constantinides, P. Y. K. Cheung, and W. Luk, "Optimal datapath allocation for multiple-wordlength systems," *Electronics Letters*, vol. 36, no. 17, pp. 1508–1509, 2000.

[18] J. Cong, Y. Fan, G. Han, et al., "Bitwidth-aware scheduling and binding in high-level synthesis," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, vol. 2, pp. 856–861, Shanghai, China, January 2005.

[19] B. Le Gal, C. Andriamisaina, and E. Casseaut, "Bit-width aware high-level synthesis for digital signal processing systems," in *Proceedings of IEEE International Systems-on-Chip Conference (SOCC '06)*, pp. 175–178, Austin, Tex, USA, September 2006.

[20] http://web.univ-ubs.fr/lester/www-gaut.

[21] www.mentor.com/.

[22] http://www.systemc.org.

[23] http://gcc.gnu.org/gcc-4.2.

[24] http://www.xilinx.com/ise.

[25] http://www.altera.com.

[26] Z. Galil, "Efficient algorithms for finding maximum matching in graphs," *ACM Computing Surveys*, vol. 18, no. 1, pp. 23–38, 1986.

Hindawi Publishing Corporation
EURASIP Journal on Embedded Systems
Volume 2008, Article ID 242584, 12 pages
doi:10.1155/2008/242584

Research Article

Accuracy Constraint Determination in Fixed-Point System Design

D. Menard,[1] R. Serizel,[2] R. Rocher,[1] and O. Sentieys[1]

[1] *IRISA/INRIA, University of Rennes, 6 Rue de Kerampont, 22300 Lannion, France*
[2] *K.U.Leuven, ESAT/SISTA, Kasteelpark Arenberg 10, 3001 Leuven-Heverlee, Belgium*

Correspondence should be addressed to D. Menard, menard@enssat.fr

Received 12 March 2008; Revised 4 July 2008; Accepted 2 September 2008

Recommended by Markus Rupp

Most of digital signal processing applications are specified and designed with floatingpoint arithmetic but are finally implemented using fixed-point architectures. Thus, the design flow requires a floating-point to fixed-point conversion stage which optimizes the implementation cost under execution time and accuracy constraints. This accuracy constraint is linked to the application performances and the determination of this constraint is one of the key issues of the conversion process. In this paper, a method is proposed to determine the accuracy constraint from the application performance. The fixed-point system is modeled with an infinite precision version of the system and a single noise source located at the system output. Then, an iterative approach for optimizing the fixed-point specification under the application performance constraint is defined and detailed. Finally the efficiency of our approach is demonstrated by experiments on an MP3 encoder.

1. INTRODUCTION

In digital image and signal processing domains, computing oriented applications are widespread in embedded systems. To satisfy cost and power consumption challenges, fixed-point arithmetic is favored compared to floating-point arithmetic. In fixed-point architectures, memory and bus widths are smaller, leading to a definitively lower cost and power consumption. Moreover, floating-point operators are more complex, having to deal with the exponent and the mantissa, and hence, their area and latency are greater than those of fixed-point operators. Nevertheless, digital signal processing (DSP) algorithms are usually specified and designed with floating-point data types. Therefore, prior to the implementation, a fixed-point conversion is required.

Finite precision computation modifies the application functionalities and degrades the desired performances. Fixed-point conversion must, however, maintain a sufficient level of accuracy. The unavoidable error due to fixed-point arithmetic can be evaluated through analytical- or simulation-based approaches. In our case, the analytical approach has been favored to obtain reasonable optimization times for fixed-point design space exploration. In an analytical approach, the performance degradations are not analyzed directly in the conversion process. An intermediate metric is used to measure the computational accuracy. Thus, the global conversion method is split into two main steps. Firstly, a computational accuracy constraint is determined according to the application performances, and secondly the fixed-point conversion is carried out. The implementation cost is minimized under this accuracy constraint during the fixed-point conversion process. The determination of the computational accuracy constraint is a difficult and open problem and this value cannot be defined directly. This accuracy constraint has to be linked to the quality evaluation and to the performance of the application.

A fixed-point conversion method has been developed for software implementation in [1] and for hardware implementation in [2]. This method is based on an analytical approach to evaluate the fixed-point accuracy. The implementation cost is optimized under an accuracy constraint. In this paper, an approach to determine this accuracy constraint from the application performance requirements is proposed. This module for accuracy constraint determination allows the achieving of a complete fixed-point design flow for

which the user only specifies the application performance requirements and not an intermediate metric. The approach proposed in this paper is based on an iterative process to adjust the accuracy constraint. The first value of the accuracy constraint is determined through simulations and depends on the application performance requirements. The fixed-point system behavior is modeled with an infinite precision version of the system and a single noise source located at the system output. The accuracy constraint is thus determined as the maximal value of the noise source power which maintains the desired application quality. Our noise model is valid for rounding quantization law and for systems based on arithmetic operations (addition, subtraction, multiplication, division). This includes LTI and non-LTI systems with or without feedbacks. In summary, the contributions of this paper are (i) a technique to determine the accuracy constraint according to the application performance requirements, (ii) a noise model to estimate the application performances according to the quantization noise level, (iii) an iterative process to adjust the accuracy constraint.

The paper is organized as follows. After the description of the problem and the related works in Section 2, our proposed fixed-point design flow is presented in Section 3. The noise model used to determine the fixed-point accuracy is detailed in Section 4. The case study of an MP3 coder is presented in Section 5. Different experiments and simulations have been conducted to illustrate our approach ability to model quantization effect and to predict performance degradations due to fixed-point arithmetic.

2. PROBLEM DESCRIPTION AND RELATED WORKS

The aim of fixed-point design is to optimize the fixed-point specification by minimizing the implementation cost. Nevertheless, fixed-point arithmetic introduces an unalterable quantization error which modifies the application functionalities and degrades the desired performance. A minimum computational accuracy must be guaranteed to maintain the application performance. Thus, in the fixed-point conversion process, the fixed-point specification is optimized. The implementation cost is minimized as long as the application performances are fulfilled. In the case of software implementations, the cost corresponds to the execution time, the memory size, or the energy consumption. In the case of hardware implementations, the cost corresponds to the chip area, the critical path delay, or the power consumption.

One of the most critical parts of the conversion process is the evaluation of the degradation of the application performance due to fixed-point arithmetic. This degradation can be evaluated with two kinds of methods corresponding to analytical- and simulation-based approaches. In the simulation-based method, fixed-point simulations are carried out to analyze the application performances [3]. This simulation can be done with system-level design tools such as CoCentric (Synopsys) [4] or Matlab-Simulink (Mathworks) [5]. Also, C++ classes to emulate the fixed-point mechanisms have been developed as in *SystemC* [4] or *Algorithmic C data types* [6]. These techniques suffer from a major drawback which is the time required for the simulation [7]. It becomes a severe limitation when these methods are used in the fixed-point specification optimization process where multiple simulations are needed. This optimization process needs to explore the design-space of different data word-lengths. A new fixed-point simulation is required when a fixed-point format is modified. The simulations are made on floating-point machines and the extra-code used to emulate fixed-point mechanisms increases the execution time to between one and two orders of magnitude compared to traditional simulations with floating-point data types [8]. Different techniques [7, 9, 10] have been investigated to reduce this emulation extra-cost. To obtain an accurate estimation of the application performance, a great number of samples must be taken for the simulation. For example, in the digital communication domain, to measure a bit error rate of 10^{-a}, at least 10^{2+a} samples are required. This large number of samples combined with the fixed-point mechanism emulation leads to very long simulation times. For example, in our case, one fixed-point C code simulation of an MP3 coder required 480 seconds. Thus, fixed-point optimization based on simulation leads to too long execution times.

In the case of analytical approaches, a mathematical expression of a metric is determined. Determining an expression of the performance for every kind of application is generally an issue. Thus, the performance degradations are not analyzed directly in the conversion process and an intermediate metric which measures the fixed-point accuracy must be used. This computational accuracy metric can be the quantization error bounds [11], the mean square error [12], or the quantization noise power [10, 13]. In the conversion process, the implementation cost is minimized as long as the fixed-point accuracy metric is greater than the accuracy constraint. The analytical expression of the fixed-point accuracy metric is first determined. Then, in the optimization process, this mathematical expression is evaluated to obtain the accuracy value for a given fixed-point specification. This evaluation is much more rapid than in the case of a simulation-based approach. The determination of the accuracy constraint is a difficult problem and this value cannot be defined directly. This accuracy constraint has to be linked to the quality evaluation and performances of the application.

Most of the existing fixed-point conversion methods based on an analytical approach [1, 11, 13–15] evaluate the output noise level, but they do not predict the application performance degradations due to fixed-point arithmetic. In [12], an analytical expression is proposed to link the bit error rate and the mean square error. Nevertheless, to our knowledge, no general method was proposed to link computational accuracy constraint with any application performance metric. In this paper, a global fixed-point design flow is presented to optimize the fixed-point specification under application performance requirements. A technique to determine the fixed-point accuracy constraint is proposed and the associated noise model is detailed.

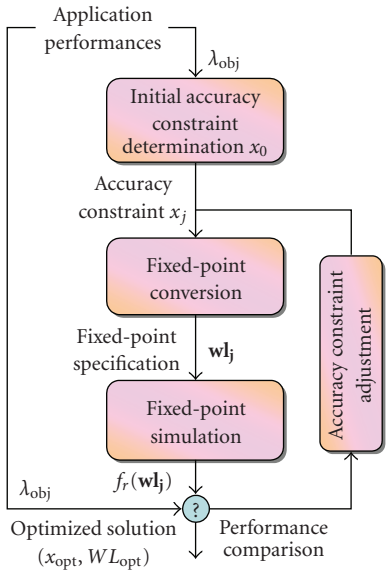


Figure 1: Global fixed-point design process. This design flow is made-up of three stages. An iterative process is used to adjust the accuracy constraint x_j for the fixed-point conversion.

3. PROPOSED FIXED-POINT DESIGN PROCESS

3.1. Global process

A fixed-point datum a of wl_a bits is made up of an integer part and a fractional part. The number of bits associated with each part does not change during the processing leading to a fixed binary position. Let iwl_a and fwl_a be the binary-point position referenced, respectively, from the most significant bit (MSB) and the least significant bit (LSB). The terms iwl_a and fwl_a correspond, respectively, to the integer and fractional part word-length. The word-length wl_a is equal to the sum of iwl_a and fwl_a. The aim of the fixed-point conversion is to determine the number of bits for each part and for each datum.

The global process proposed for designing fixed-point systems under application performance constraints is presented in Figure 1 and detailed in the next sections. The metric used to evaluate the fixed-point accuracy is the output quantization noise power. Let b_y be the system output quantization noise. The noise power is also called in this paper noise level and is represented by the term $x = E(b_y^2)$. The accuracy constraint x_j used for the fixed-point conversion process corresponds to the maximal noise level under which the application performance is maintained. The challenge is to establish a link between the accuracy constraint x_j and the desired application performances λ_{obj}. These application performances must be predicted according to the noise level x. The global fixed-point design flow is made-up of three stages and an iterative process is used to adjust the accuracy constraint used for the fixed-point conversion. These three stages are detailed in the following sections.

3.2. Accuracy constraint determination

The first step corresponds to the initial accuracy constraint determination x_0 which is the maximal value of the noise level satisfying the performance objective λ_{obj}. For example, in a digital communication receiver, the maximal quantization noise level is determined according to the desired bit error rate.

First, a prediction of the application performance is performed with the technique presented below. Let $f_p(x)$ be the function representing the predicted performances according to the noise level x. To determine the initial accuracy constraint value (x_0), equation $f_p(x) = \lambda_{obj}$ is solved graphically, and x_0 is the solution of this equation.

3.2.1. Performance prediction

To define the initial value of the accuracy constraint (x_0), the application performance is predicted according to the noise level x. The fixed-point system is modeled by the infinite precision version of the system and a single noise source b_p located at the system output as shown in Figure 2. This noise source models all the quantization noise sources inside the fixed-point system. The system floating-point version is used and the noise b_p is added to the output. The noise model used for b_p is presented in Section 4. Different noise levels x for the noise source b_p are tested to measure the application performance and to obtain the predicted performance f_p function according to the noise level x. The accuracy constraint x_0 corresponds to the maximal value of the noise level which allows the maintenance of the desired application performance.

Most of the time, the floating-point simulation has already been developed during the application design step, and the application output samples can be used directly. Therefore, the time required for exploring the noise power values is significantly reduced and becomes negligible with regard to the global implementation flow. Nevertheless, this technique cannot be applied for systems where the decision on the output is used inside the system like, for example, decision-feedback equalization. In this case, a new floating-point simulation is required for each noise level which is tested.

3.3. Fixed-point conversion process

The second step corresponds to the fixed-point conversion. The goal is to optimize the application fixed-point specification under the accuracy constraint x_j. The approaches presented in [1] for software implementation and in [2] for hardware implementation are used. This fixed-point conversion can be divided into two main modules. The flow diagram used for this conversion is shown in Figure 3.

The first part corresponds to the determination of the integer part word-length of each datum. The number of bits iwl_i for this integer part must allow the representation of all the values taken by the data and is obtained from the data bound values. Thus, firstly the dynamic range is evaluated for each datum. Then, these results are used to determine,

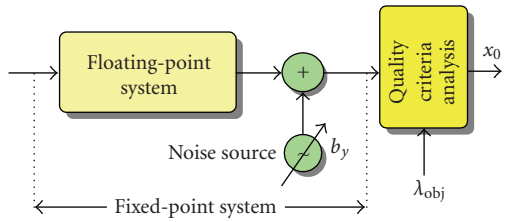


Figure 2: Accuracy constraint determination. The fixed-point system is modeled by the infinite precision version of the system and a single noise source b_y located at the system output.

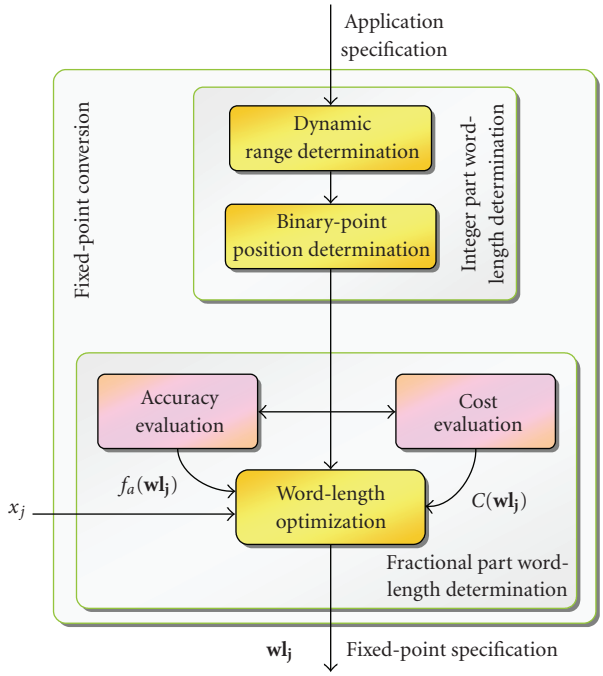


Figure 3: Fixed-point conversion process. For each datum, the number of bits for the integer part is determined and the fractional part word-length is optimized.

for each data, the binary-point position which minimizes the integer part word-length and which avoids overflow. Moreover, scaling operations are inserted in the application to adapt the fixed-point format of a datum to its dynamic range or to align the binary-point of the addition inputs.

The second part corresponds to the determination of the fractional part word-length. The number of bits fwl_i for this fractional part defines the computational accuracy. Thus, the data word-lengths are optimized. The implementation cost is minimized under the accuracy constraint. Let $\mathbf{wl} = \{wl_0, wl_1, \ldots, wl_i, \ldots, wl_{N-1},\}$ be an N-size vector including the word-length of the N application data. Let $C(\mathbf{wl})$ be the implementation cost and let $f_a(\mathbf{wl})$ be the computational accuracy obtained for the word-length vector $\mathbf{wl}$. The implementation cost $C(\mathbf{wl})$ is minimized under the accuracy constraint x:

$$\min C(\mathbf{wl}_j) \quad \text{such as } f_a(\mathbf{wl}_j) \geq x_j. \tag{1}$$

The vector $\mathbf{wl}_j$ is the optimized fixed-point specification obtained for the constraint value x_j at iteration j of the process.

The data word-length determination corresponds to an optimization problem where the implementation cost and the application accuracy must be evaluated. The major challenge is to evaluate the fixed-point accuracy. To obtain reasonable optimization times, analytical approaches to evaluate the accuracy have been favored. The computational accuracy is evaluated using the quantization noise power. The mathematical expression of this noise power is computed for systems based on arithmetic operations with the technique presented in [16]. This mathematical expression is determined only once and is used for the different iterations of the fixed-point conversion process and for the different iterations of the global design flow.

3.4. Performance evaluation and accuracy constraint adjustment

The third step corresponds to the evaluation of the real application performance. The optimized fixed-point specification $\mathbf{wl}_j$ is simulated and the application performance is measured. The measured performances $f_r(x_j)$ and the objective value λ_{obj} are compared and, if (2) is not satisfied, the accuracy constraint is adjusted and a new iteration is performed:

$$\lambda_{obj} - \varepsilon < f_r(x_j) < \lambda_{obj} + \varepsilon, \tag{2}$$

where the term ε is the tolerance on the objective value.

To modify the accuracy constraint value, two measurements are used. Nevertheless, in the first iteration, only the point $(x_0, f_r(x_0))$ is available. To obtain a second point, all the data word-lengths are increased (or decreased) by p bits. In this case, as demonstrated in appendix the noise level is increased (or decreased) by $6 \cdot p$ dB. The number of bits p is chosen as the minimal value respecting the following inequality:

$$\frac{f_r(x_0) - \lambda_{obj}}{|f_r(x_0) - \lambda_{obj}|} \neq \frac{f_p(x_0 \pm 6 \cdot p) - \lambda_{obj}}{|f_p(x_0 \pm 6 \cdot p) - \lambda_{obj}|}. \tag{3}$$

The choice to increment or decrement depends on the slope sign of f_p and the sign of the difference between $f_r(x_0)$ and λ_{obj}. For the next iterations, two or more measured points are available. The two consecutive points of abscissa x_a and x_b such as $f_r(x_a) < \lambda_{obj} < f_r(x_b)$ are selected and let f_{ab} be the linear equation linking the two points $(x_a, f_r(x_a))$ and $(x_b, f_r(x_b))$. The adjusted accuracy constraint used for the next iteration k is x_k defined such as $f_{ab}(x_k) = \lambda_{obj}$. The adjustment process is illustrated in Section 5.3 through an example.

4. NOISE MODEL

4.1. Noise model description

4.1.1. Quantization noise model

The use of fixed-point arithmetic introduces an unavoidable quantization error when a signal is quantified. A common

model for signal quantization has been proposed by Widrow in [17] and refined in [18]. The quantization of a signal is modeled by the sum of this signal and a random variable b, which represents the quantization noise. This additive noise b is a uniformly distributed white noise that is uncorrelated with the signal, and independent from the other quantization noises. In this study, the round-off method is used rather than truncation. For convergent rounding, the quantization leads to an error with a zero mean. For classical rounding, the mean can be assumed to be null as soon as several bits (more than 3 bits) are eliminated in the quantization process. The expression of the statistical parameters of the noise sources can be found in [16]. If q is the quantization step (accuracy), the noise values are in the interval $[-q/2; q/2]$.

4.1.2. Noise model for fixed-point system

The noise model for fixed-point systems presented in [16] is used. The output quantization noise b_y is the contribution of the different noise sources. Each noise source b_i is due to the elimination of some bits during a cast operation. From the propagation model presented in [16], for each arithmetic operation it can be shown that the operation output noise is a weighted sum of the input noises associated with each operation input. The weights of the sum do not include noise terms, because the product of the noise terms can be neglected. Thus, in the case of systems based on arithmetic operations, the expression of the output quantization noise b_y is as follows:

$$b_y(n) = \sum_{i=1}^{N_e} b_i'(n) = \sum_{i=1}^{N_e} \sum_{k=0}^{n} h_i(k) b_i(n-k), \tag{4}$$

where the term h represents the impulse response of the system having b_y as output and b_i as input. In the case of linear time invariant (LTI) systems, the different terms $h(i)$ are constant. In the case of non-LTI systems the terms $h(i)$ are time varying. In this context, two extreme cases can be distinguished. In the first case, a quantization noise b_i' predominates in terms of variance compared to the other noise sources. A typical example is an extensive reduction of the number of bits at the system output compared to the other fixed-point formats. In this case, the level of this output quantization noise exceeds the other noise source levels. Thus, the probability density function of the output quantization noise is very close to that of the predominant noise source and can be assimilated to a uniform distribution. In the second case, an important number of independent noise sources have similar statistical parameters and no noise source predominates. All the noise sources are uniformly distributed and independent of each other. By using the central limit theorem, the sum of the different noise sources can be modeled by a centered normally distributed noise.

From these two extreme cases, an intuitive way to model the output quantization noises of a complex system is to use a noise b_p which is the weighted sum of a Gaussian and a uniform noise. Let f_b be the probability density function of the noise b. Let b_n be a normally distributed noise with a mean and variance equal, respectively, to 0 and 1. Let b_u be a uniformly distributed noise in the interval $[-1; 1]$. The noise b_p is defined with the following expression:

$$b_p = v(\beta \times b_u + (1-\beta) \times b_n). \tag{5}$$

The weight β is set in the interval $[0; 1]$ and allows the representation of the different intermediate cases between the two extreme cases presented above. The weight v fixes the global noise variance.

4.1.3. Choice of noise model parameters

The noise b_p is assumed to be white noise. Nevertheless, the spectral density function of the real quantization noise depends on the system and most of the time is not white. If the application performance is sensitive to the noise spectral characteristic, this assumption will degrade the performance prediction. Nevertheless, the imperfections of the noise model are compensated by the iteration process which adapts the accuracy constraint. The effects of the noise model imperfections increase in the number of iterations required to converge to the optimized solution.

To take account of the noise spectral characteristics, the initial accuracy constraint x_0 can be adjusted and determined in a two-step process. The accuracy constraint x_0 is determined firstly assuming that the noise b_p is white. Then, the fixed-point conversion is carried out and the fixed-point specification $\mathbf{wl}_0$ is simulated. The spectral characteristics of the real output quantization noise $b_r(\mathbf{wl}_0)$ are measured. Afterwards, the accuracy constraint x_0 is adjusted and determined a second time assuming that the noise b_p has the same spectral characteristics as the real quantization noise $b_r(\mathbf{wl}_0)$.

Like for the spectral characteristics, the weight β is set to an arbitrary value depending of the kind of implementation. Then, after the first iteration, the β value is adjusted by using the measured β value obtained from the real output quantization noise $b_r(\mathbf{wl}_0)$.

In most of the processors the architecture is based on a double precision computation. Inside the processing unit, most of the computations are carried out without loss of information and truncation occurs when the data are stored in memory. This approach tends to obtain a predominant noise source at the system output. Thus, for software implementation the weight β is fixed to 1. For hardware implementation the optimization of the operator word-length leads to a fixed-point system where no noise source is predominant. The optimization distributes the noise to each operation. Thus, for hardware implementation the weight β is fixed to 0.

4.2. Validation of the proposed model

4.2.1. Validation methodology

The aim of this section is to analyze the accuracy of our model with real quantization noises. The real noises are obtained through simulations. The output quantization noise is the difference between the system outputs obtained with a fixed-point and a floating-point simulation. The

floating-point simulation which uses double-precision types is considered to be the reference. Indeed, in this case, the error due to the floating-point arithmetic is definitely less than the error due to the fixed-point arithmetic. Thus, the floating-point arithmetic errors can be neglected.

Our model is valid if a balance weight β can be found to model the real noise probability density function with (5). The accuracy of our model with real noises is analyzed with the χ^2 goodness-of-fit test. This test is a statistical tool which can be used to determine if a real quantization noise b_r follows a chosen probability density function f_{b_p} [19]. Let H_X be the hypothesis that b_r follows the probability density function f_{b_p}. The test is based on the distance between the two probability density functions. If y_s is the observed frequency for interval s, E_s is the expected frequency for s and k the number of interval s, the statistical test is:

$$\chi^2 = \sum_{s=1}^{k} \frac{(y_s - E_s)^2}{E_s}. \tag{6}$$

This statistical test follows the χ^2 distribution with $k-1$ degrees of freedom. Therefore, if the distance is higher than the threshold χ_{th}, then the hypothesis H_X (b_r follows the probability density function f_{b_p}) is rejected. The significance level of the test is the probability of rejecting H_X when the hypothesis is true. Choosing a certain value for this level will set the threshold distance for the test. According to [20], the significance level α should be in [0.001 0.05].

Concerning the observed noise, there is no a priori knowledge of the balance coefficient β. Thus, the χ^2 test has to be used collectively with a searching algorithm. This algorithm finds the β weight for which the f_b fits the best to the noise. Let β_{optim} be the optimized value which minimizes the term χ^2 from (6), then:

$$\chi^2(\beta_{\text{optim}}) = \min_{\beta\in[0;1]} (\chi^2(\beta)). \tag{7}$$

The real quantization noise can be modeled with (5) if the optimized value β_{optim} is lower than the threshold χ_{th}.

4.2.2. FIR filter example

The first system on which the before-mentioned test has been performed is a 32-tap FIR filter. The filter output $y(n)$ is obtained from the following expression:

$$y(n) = \sum_{i=0}^{N-1} c_i \cdot e(n-i), \tag{8}$$

where e is the filter input and c_i the filter coefficients.

The signal flow graph of one cell i is presented in Figure 4. To simplify the presentation, the integer part word-length for the multiplication output and the input and output of the addition are set to be equal. Thus, no scaling operation is necessary to align the binary point positions at the adder input. This simplification has no influence on the generality of the results.

The word-lengths of the input signal (wl_e) and of the coefficient (wl_{c_i}) are equal to 16 bits. If no bit is eliminated during the multiplication, the multiplier output word-length wl_{mult} is equal to 32 bits. The adder input and output word-length are equal to wl_{add}. At the filter output, the data is stored in memory with a word-length wl_y equal to 16 bits.

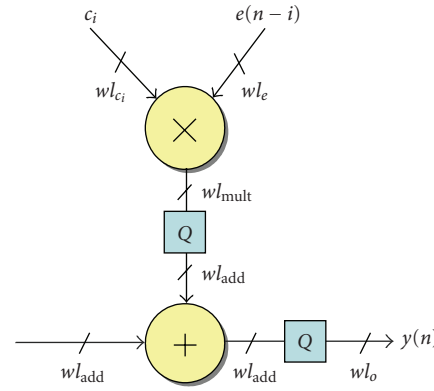


Figure 4: Signal flow graph for one FIR filter tap.

Two kinds of quantization noise sources can be located in the filter. A noise source b_{m_i} can be located at each multiplier output if bits are eliminated between the multiplication and the addition. The number of eliminated bits k_{m_i} is obtained with the following expression:

$$k_m = wl_{\text{mult}} - wl_{\text{add}} = wl_e + wl_h - wl_{\text{add}}. \tag{9}$$

A noise source b_y is located at the filter output if bits are eliminated when the addition output is stored in memory. The number of eliminated bits k_y is obtained with the following expression:

$$k_y = wl_{\text{add}} - wl_y. \tag{10}$$

The adder word-length wl_{add} is varying between 16 and 32 bits, while the output of the system is always quantized on 16 bits.

The probability density function of the filter output quantization noise is shown in Figure 5 for different values of wl_{add}. The noise is uniform when one source is prevailing (the adder is on 32 bits). As long as the influence of the sources at the output of the multiplier is increasing (the length is decreasing), the distribution of the output noise tends to become Gaussian. These simple visual observations can be confirmed using the β-searching algorithm. The change of the optimized value β_{optim} for different adder word-lengths, varying from 16 to 32 bits, is shown in Figure 6. When the output of the multiplier is 16 or 17 bits, $\beta = 0$, the sources are numerous. Their influence on the system output leads to a Gaussian noise. As the length of the multiplier increases, β also grows and eventually tends to 1. When wl_{add} is greater than 26 bits, the variance of the noise sources b_{m_i} located at each multiplier output is insignificant compared to the variance of the noise source b_y located at the filter output. Thus, this latter is prevailing and its influence on the output signal is a uniform white noise. In this case, the boundary values of the noise are $[-q/2; q/2]$.

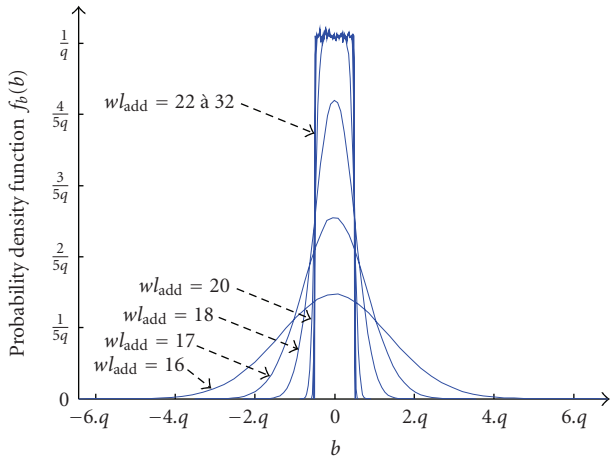


Figure 5: Probability density function of the 32-tap FIR filter output quantization noise measured for different adder word-lengths wl_{add}.

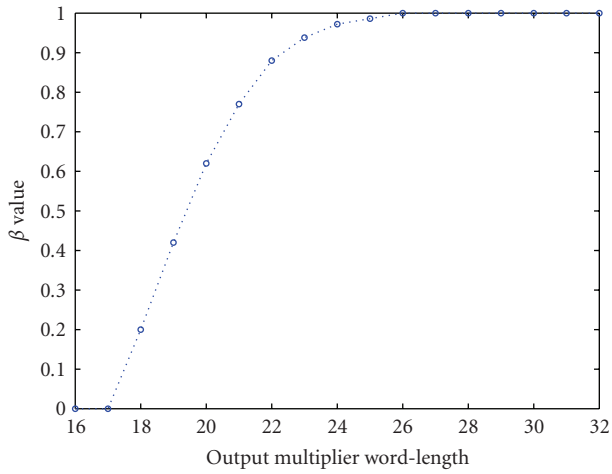


Figure 6: Balance weight β found for different adder word-lengths wl_{add}. This weight is obtained with the β-search algorithm presented in Section 4.2.1.

4.2.3. Benchmarks

To validate our noise model, different DSP application benchmarks have been tested and the adequacy between our model and real noises has been measured. For each application, different output noises have been obtained by evaluating several fixed-point specifications and different application parameters. The number of output noises analyzed for one application is defined through the term N_t. For these different applications based on arithmetic operations, the input and the output word-length are fixed to 16 bits. The different fixed-point specifications are obtained by modifying the adder input and output word-lengths. Eight values are tested for the adder: (16, 17, 18, 19, 20, 22, 24, 32).

The results are shown in Table 1 for two significance levels α corresponding to the boundary values (0.05 and 0.001). For each application, the number N_s of real noise which can be modeled with our noise model is measured.

Table 1: Adequacy between our model and real noises for different DSP applications.

Applications	Test number N_t	Significance level $\alpha = 0.05$	$\alpha = 0.001$
FFT (16 and 32 samples)	16	100%	100%
IIR 8 direct form I	192	98%	99%
IIR 8 direct form II	192	100%	100%
IIR 8 transposed form	192	97%	99%
Adaptive APA filter	8	87%	100%
Volterra filter	8	100%	100%
WCDMA receiver	16	100%	100%
MP3	28800	78%	87%

The adequacy between our model and real noises is measured with the metric Γ defined with the following expression:

$$\Gamma = \frac{N_s}{N_t}. \tag{11}$$

This metric corresponds to the ratio of output noises for which a weight β can be found to model the noise probability density function with (5).

The different applications used to test our approach are presented in this paragraph. A fast Fourier transform (FFT) has been performed on vectors made-up of 16 or 32 samples. Linear time-invariant (LTI) recursive systems have been tested through an eight-order infinite impulse filter (IIR). This filter is implemented with a cascaded form based on four second order cells. For this cascaded eight-order IIR filter, 24 permutations of the second-order cells can be tested leading to very different output noise characteristics [21]. Three forms have been tested corresponding to *Direct-form I*, *Direct-Form II* and *Transposed-Form*. An adaptive filter based on the affine projection algorithm (APA) structure [22] has been tested. This filter is made-up of eight taps and the observation vector length is equal to five. A nonlinear nonrecursive filter has been tested using a second-order Volterra filter. Our benchmarks do not include non linear systems with memory and thus do not validate this specific class of algorithms.

More complex applications have been studied through a WCDMA receiver and an MP3 coder. The MP3 coder is presented in Section 5.1. For the third generation mobile communication systems based on the WCDMA technique, the receiver is mainly made up of an FIR receiving filter and a rake receiver including synchronization mechanisms [23]. The rake receiver is made-up of three parts corresponding to the transmission channel estimation, synchronization and symbol decoding. The synchronization of the code and the received signal is realized with a delay-locked loop (DLL). The noises are observed at the output of the symbol decoding part.

The results from Table 1 show that our noise model can be applied to most of the real noises obtained for different applications. For some applications, like FFT, FIR, WCDMA receiver and the Volterra filter, a balance coefficient β can

always be found. These four applications are nonrecursive and the FFT, FIR, WCDMA receiver are LTI systems.

For the eight-order infinite impulse filter, almost all the noises (97%–100%) can be modeled with our approach. For these filters, 90% of the output quantization noise are modeled with a balance coefficient β equal to 0. Thus, the output noise is a purely normally distributed noise. In LTI system, the output noise b'_g due to the noise b_g corresponds to the convolution of the noise b'_g with h_g. This term h_g is the impulse response of the transfer function between the noise source and the output. Thus, the output noise is the weighted sum of the delayed version of the noise b_g. The noise b_g is a uniformly distributed white noise, thus the delayed versions of the noise b_g are uncorrelated. The samples are uncorrelated but are not independent and thus the central limit theorem cannot be applied directly. Even if only one noise source is located in the filter, the output noise is a sum of noncorrelated noises and this output noise tends to have a Gaussian distribution. For the MP3 coder, when the level is 0.001 the test is successful about 87% of the time (78% when α is 0.05).

For the different applications, the metric Γ is close to 100%. These results show that our model is suitable to model the output quantization noise of fixed-point systems.

5. CASE STUDY: MP3 CODER

The application used to illustrate our approach and to underline its efficiency comes from audio compression and corresponds to an MP3 coder. First, the application and the associated quality criteria are briefly described. Then, the ability of our noise model to predict application performance is evaluated. Finally, a case study to obtain an optimized fixed-point specification which ensures the desired performances is detailed.

5.1. Application presentation

5.1.1. MP3 coder description

The MP3 compression, corresponding to MPEG-1 Audio Layer 3, is a popular digital audio encoding which allows efficient audio data compression. It must maintain a reproduction quality close to the original uncompressed audio. This compression technique relies on a psychoacoustic model which analyzes the audio signal according to how human beings perceive a sound. This step allows the formation of some signal to noise masks which indicate where noise can be added without being heard. The MP3 encoder structure is shown in Figure 7. The signal to be compressed is analyzed using a polyphase filter which divides the signal into 32 equal-width frequency bands. The modified discrete cosine transform (MDCT) further decomposes the signal into 576 subbands to produce the signal which will actually be quantized. Then, the quantization loop allocates different accuracies to the frequency bands according to the signal to noise mask. The processed signal is coded using Huffman code [24]. The MP3 coder can be divided into two signal flows. The one composed from the polyphase filter and the MCDT and the one composed from the FFT and the psychoacoustic model. The fixed-point conversion of the second signal flow has low influence on compression quality and thus, in this paper, the results are presented only for the first signal flow.

The BLADE [25] coder has been used with a 192 Kbits/s constant bit rate. This coder leads to a good quality compression with floating-point data types. A sample group of audio data has been defined for the experiments. This group contains various kinds of sounds, where each can lead to different problems during encoding (harmonic purity, high or low dynamic range, ...). Ten different input tracks have been selected and tested.

5.1.2. Quality criteria

In the case of an MP3 coder, the output noise power metric cannot be used directly as a compression quality criterion. The compression is indeed based on adding quantization noises where it is imperceptible, or at least barely audible. The compression quality has been tested using EAQUAL [26] which stands for evaluation of audio quality. It is an objective measurement tool very similar to the ITU-R recommendation BS.1387 based on PEAQ technique. This has to be used because listening tests are impossible to formalize. In EAQUAL, the degradations due to compression are measured with the objective degradation grade (ODG) metric. This metric varies from 0 (no degradation) to -4 (inaudible). The level of -1 is the threshold beyond the degradation becomes annoying for ears. This ODG is used to measure the degradation due to fixed-point computation. Thus for the fixed-point design, the aim is to obtain the fixed-point specification of the coder which minimizes the implementation cost and maintains an ODG lower or equal to -1 for the different audio tracks of the sample group.

5.2. Performance prediction

The efficiency of our approach depends on the quality of the noise model used to determine the accuracy constraint. To validate this latter, its ability to model real quantization noises and its capability to predict the application performance according to the quantization noise level are analyzed through experiments.

Our model capability to predict application performance according to the quantization noise level has been analyzed. The real and the predicted performances are compared for different noise levels x. The application performance prediction is obtained with our model as described in Figure 2. The random signal modeling the output quantization noise with a noise level of x is added to the infinite precision system output. The real performances are measured with a fixed-point simulation of the application. This fixed-point specification is obtained with a fixed-point conversion having x as an accuracy constraint. For this audio compression application, the evolution of the predicted (f_p) and the real (f_r) objective degradation grade (ODG) is given in Figure 8(a). These performances have been measured for

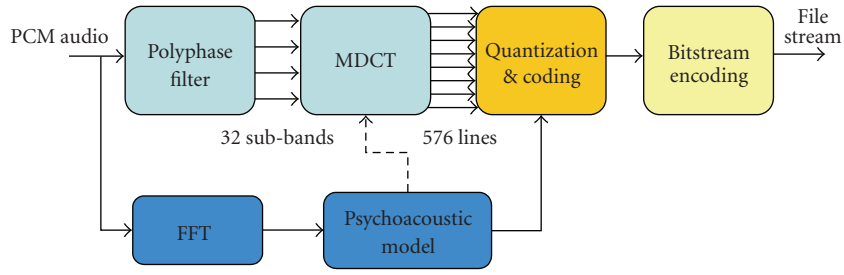


Figure 7: MP3 encoder structure.

different quantization noise levels x included in the range $[-108\,\text{dB}; -88\,\text{dB}]$.

The predicted and real performances are very close except for two noise levels equal to -98.5 dB and -88 dB. In these cases, the difference between the two functions f_p and f_r is, respectively, equal to 0.2 and 0.4. In the other cases, the difference is less than 0.1. It must be underlined that when the ODG is lower than -1.5, the ODG evolution slope is higher, and a slight difference in the noise level leads to a great difference on the ODG. The case where only the polyphase filter is considered to use fixed-point data type (the MDCT is computed with floating-point data-types) has been tested. The difference between the two functions f_p and f_r is less than 0.13. Thus, our approach allows the accurate prediction of the application performance.

5.3. *Fixed-point optimization under performance constraint*

In this section, the design process to obtain an optimized fixed-point specification which guarantees a given level of performances is detailed. The ODG objective λ_{obj} is fixed to -1 corresponding to the acceptable degradation limit.

First, to determine the initial value of the accuracy constraint, a prediction of the application performance is made. This initial value determination corresponds to the first stage of the design flow presented in Figure 1. The function f_p is determined for different noise levels with a balance coefficient varying from 0 to 1. Two results can be underlined. Firstly, the influence of the balance coefficient β has been tested and is relatively low. Between the extreme values of β, the ODG variation is on average less than 0.1. In these experiments, a hardware implementation is under consideration, thus the balance coefficient β is fixed to 0.

Secondly, the ODG change is strongly linked to the kind of input tracks used for the compression.

In these experiments, 10 different input tracks have been tested. To obtain an ODG equal to -1, a difference of 33 dB is obtained between the minimal and the maximal ODG. These results underline the necessity to have inputs which are representative of the different audio tracks encountered in the real world. In the rest of the study, the audio sample which leads to the minimal ODG is used as a reference. The results obtained in this case are shown in Figure 8(a) and a zoom is given in Figure 8(b).

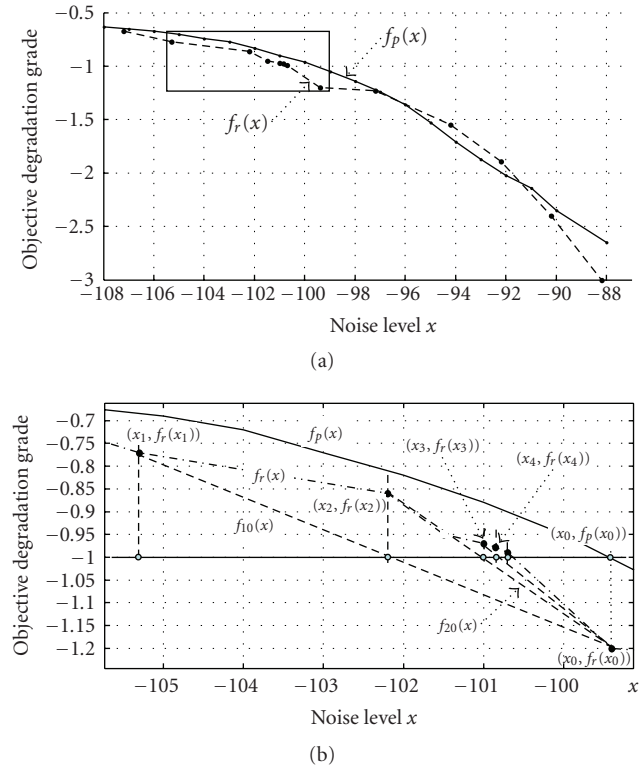


Figure 8: (a) Predicted (f_p) and the real (f_r) objective degradation grade (ODG) for the MP3 coder. (b) Zoom on the range $[-105\,\text{dB}, -99\,\text{dB}]$, the point $(x_k : f_r(x_k))$ involved in each iteration k are given.

To determine the initial value of the accuracy constraint, equation $f_p(x) = \lambda_{\text{obj}}$ is graphically solved. The noise level x_0 is equal to -99.35 dB. x_0 is used as an accuracy constraint for the fixed-point conversion. This conversion is the second stage of the design flow (see Figure 1). The aim is to find the fixed-point specification which minimizes the implementation cost and maintains a noise level lower than x_0. After the conversion, the fixed-point specification is simulated to measure the performance. It corresponds to the third stage of the design flow (see Figure 1). The measured performance $f_r(x_0)$ is equal to -1.2. To obtain the second point of x_1 abscissa, all the data word-lengths are increased by one bit and the measured performance $f_r(x_1)$ is equal

to −0.77. The accuracy constraint x_2 for the next iteration is obtained by solving $f_{10}(x) = -1$. The function $f_{10}(x)$ is the linear equation linking the points $(x_0 : f_r(x_0))$ and $(x_1 : f_r(x_1))$. The obtained value is equal to −102.2 dB. The process is iterated and the following accuracy constraints are equal to −101 dB and −100,7 dB. The values obtained for the different iterations are presented in Table 2. For this example, six iterations are needed to obtain an optimized fixed-point specification which leads to an ODG equal to −0.99. Only, three iterations are needed to obtain an ODG of −0.95.

5.3.1. Execution time

For our approach, the execution time of the iterative process has been measured. First, the initial accuracy constraint is determined. The floating-point simulations have already been carried out in the algorithm design process and this floating-point simulation time is not considered. For each tested noise level, the noise is added to the MDCT output and then the ODG is computed. The global execution time T_{IAC} of this first stage is equal to 420 seconds. This stage is carried out only once.

For the fixed-point conversion, the analytical model for noise level estimation is determined at the first iteration. This execution time T_{AED} of this process is equal to 120 seconds. It takes a small amount of time but it is done only once. Then, this model is used in the process of fixed-point optimization and the fixed-point accuracy is computed instantaneously by evaluating a mathematical expression. For this example, each fixed-point conversion T_{FpC} takes only 0.7 seconds due to the analytical approach.

In this fixed-point design process, most of the time is spent in the fixed-point simulation (stage 3). This simulation is carried out with C++ code with optimized fixed-point data types. For this example, the execution time T_{fpS} of each fixed-point simulation is equal to 480 seconds. But, only one fixed-point simulation is required by iteration and a small number of iterations are needed.

The expression of the global execution time T_{AA} for our approach based on analytical approach for fixed-point conversion is as follows:

$$\begin{aligned} T_{AA}(N_j) &= T_{IAC} + T_{AED} + N_j \times (T_{fpS} + T_{FpC}) \\ &= 540 + 480 \times N_j, \end{aligned} \tag{12}$$

where N_j represents the number of iterations required to obtain the optimized fixed-point specification for a given ODG constraint. In this example, six iterations are needed to obtain an optimized fixed-point specification which leads to an ODG equal to −0.99 and three iterations are needed to obtain an ODG of −0.95. Thus, the global execution time is equal to 49 minutes for an ODG of −0.99 and 33 minutes for an ODG of −0.95.

A classical fixed-point conversion approach based on simulations has also been tested to compare with our approach. The same code as in our proposed approach is used to simulate the application. For this approach most of the time is consumed by the fixed-point simulation used to evaluate the application performances. Thus, the expression of the global execution time T_{SA} for this approach based on simulation is as follows:

$$T_{AA}(N_s) \simeq N_s \times T_{fpS} = 480 \times N_s, \tag{13}$$

where N_s is the number of iterations of the optimization process based on simulation.

An optimization algorithm based on Min + b *bits procedure* [27] is used. This algorithm allows the limitation of the iteration number in the optimization process. The number of variables in the optimization process has been restricted to 9 to limit the fixed-point design search space. In this case, the number of iterations N_s is equal to 388. Given that each fixed-point simulation requires 480 seconds, the global execution becomes huge and is equal to 51 hours.

In our case, the optimization time is definitively lower. For this real application, a fixed-point simulation requires several minutes. For this example, the analytical approach reduces the execution time by a factor 63. Moreover, the fixed-point design space is very large and it cannot be explored with classical techniques based on fixed-point simulations.

Table 2: Description of the values obtained for the different iterations i. The term x_i corresponds to the accuracy constraint for the fixed-point conversion. The term $f_r(\mathbf{wl}_i)$ is the measured performance obtained after the fixed-point conversion.

Iteration i	Noise level x_i (dB)	Measured performances $f_r(\mathbf{wl}_i)$ (ODG)
0	−99.35	−1.2
1	−105.35	−0.77
2	−102,2	−0.86
3	−101	−0.97
4	−100,85	−0.98
5	−100,7	−0.99

6. CONCLUSION

In embedded systems, fixed-point arithmetic is favored but the application performances are reduced due to finite precision computation. During the fixed-point optimization process, the performance degradations are not analyzed directly during the conversion. An intermediate metric is used to measure the computational accuracy. In this paper, a technique to determine the accuracy constraint associated with a global noise model has been proposed. The probability density function of the noise model has been detailed and the choice of the parameters has been discussed. The different experiments show that our model predicts sufficiently accurately the application performances according to the noise level. The technique proposed to determine the accuracy constraint and the iterative process used to adjust this constraint allow the obtention of an optimized fixed-point specification guaranteeing minimum performance. The optimization time is definitively lower and has been divided by factor of 63 compared to the simulation based approach. Our future work is focused on the case of quantization by truncation.

APPENDIX

This section demonstrates the increase of $6 \cdot p$ dB of the noise power when all fixed-point data formats are increased of p bits. The output noise is given by the expression (4).

The output noise power P_b is presented in [16]. In our case, the rounding quantization mode is considered. Thus, noise means are equal to zero and the output noise power is given by the following relation:

$$P_b = \sum_{i=1}^{N_e} \sigma_{b_i}^2 L_i \tag{A.1}$$

with $\sigma_{b_i}^2$ the variance of the noise source $b_i(n)$. The terms L_i are *constants* computed with impulse response terms h_i and independent of the fixed-point formats [16]:

$$L_i = \sum_{k=0}^{\infty} E[h_i(k)^2] \tag{A.2}$$

with q_i the quantization step of the data after the quantization process which generates the noise $b_i(n)$. The term k is the number of bits that have been eliminated by the quantization process.

Let us consider that p bits are added to each fixed-point data format. In that case, the new quantization step q_i' is equal to:

$$q_i' = \frac{q_i}{2^p}. \tag{A.3}$$

Thus, the new noise variance $\sigma_{b_i}'^2$ is given by:

$$\sigma_{b_i}'^2 = \frac{q_i'^2}{12} = \frac{q_i^2}{12 * 2^{2p}} = \frac{\sigma_{b_i}^2}{2^{2p}}. \tag{A.4}$$

As terms L_i are constant and independent from fixed-point data formats, the new output noise power P_b' becomes:

$$P_b' = \frac{P_b}{2^{2p}}. \tag{A.5}$$

By expressing the noise power in dB, the following expression is obtained

$$\begin{aligned} P_{b_{\text{dB}}}' &= 10 \cdot \log_{10}(P_b') \\ &= 10 \cdot \log_{10}\left(\frac{P_b}{2^{2p}}\right) \\ &= 20 \cdot p \cdot \log_{10}(2) + \log_{10}(P_b) \\ &= 6 \cdot p + P_{b_{\text{dB}}}. \end{aligned} \tag{A.6}$$

This expression demonstrates the assumption that the increase of p bits of all fixed-point data formats increases the noise level of $6 \cdot p$ dB.

REFERENCES

[1] D. Menard, D. Chillet, and O. Sentieys, "Floating-to-fixed-point conversion for digital signal processors," *EURASIP Journal on Applied Signal Processing*, vol. 2006, Article ID 96421, 19 pages, 2006.

[2] N. Herve, D. Menard, and O. Sentieys, "Data wordlength optimization for FPGA synthesis," in *Proceedings of the IEEE Workshop on Signal Processing Systems (SIPS '05)*, pp. 623–628, Athens, Grece, November 2005.

[3] P. Belanovic and M. Rupp, "Automated floating-point to fixed-point conversion with the fixify environment," in *Proceedings of the 16th IEEE International Workshop on Rapid System Prototyping (RSP '05)*, pp. 172–178, Montreal, Canada, June 2005.

[4] F. Berens and N. Naser, "Algorithm to System-on-Chip Design Flow that Leverages System Studio and SystemC 2.0.1," Synopsys Inc., March 2004.

[5] Mathworks, "Fixed-Point Blockset User's Guide (ver. 2.0)," 2001.

[6] Mentor Graphics, "Algorithmic C Data Types," Mentor Graphics, version 1.2 edition, May 2007.

[7] L. De Coster, M. Adé, R. Lauwereins, and J. Peperstraete, "Code generation for compiled bit-true simulation of DSP applications," in *Proceedings of the 11th IEEE International Symposium on System Synthesis (ISSS '98)*, pp. 9–14, Hsinchu, Taiwan, December 1998.

[8] H. Keding, M. Willems, M. Coors, and H. Meyr, "FRIDGE: a fixed-point design and simulation environment," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '98)*, pp. 429–435, Paris, France, February 1998.

[9] H. Keding, M. Coors, O. Luthje, and H. Meyr, "Fast bit-true simulation," in *Proceedings of the 38th Design Automation Conference (DAC '01)*, pp. 708–713, Las Vegas, Nev, USA, June 2001.

[10] S. Kim, K.-I. Kum, and W. Sung, "Fixed-point optimization utility for C and C++ based digital signal processing programs," *IEEE Transactions on Circuits and Systems II*, vol. 45, no. 11, pp. 1455–1464, 1998.

[11] E. Özer, A. P. Nisbet, and D. Gregg, "Stochastic bit-width approximation using extreme value theory for customizable processors," Tech. Rep., Trinity College, Dublin, Ireland, October 2003.

[12] C. Shi and R. W. Brodersen, "Floating-point to fixed-point conversion with decision errors due to quantization," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, vol. 5, pp. 41–44, Montreal, Canada, May 2004.

[13] H. Keding, F. Hurtgen, M. Willems, and M. Coors, "Transformation of floating-point into fixed-point algorithms by interpolation applying a statistical approach," in *Proceeding of the 9th International Conference on Signal Processing Applications and Technology (ICSPAT '98)*, Toronto, Canada, September 1998.

[14] G. A. Constantinides, "Perturbation analysis for word-length optimization," in *Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '03)*, pp. 81–90, Napa, Calif, USA, April 2003.

[15] J. A. Lopez, G. Caffarena, C. Carreras, and O. Nieto-Taladriz, "Fast and accurate computation of the roundoff noise of linear time-invariant systems," *IET Circuits, Devices & Systems*, vol. 2, no. 4, pp. 393–408, 2008.

[16] R. Rocher, D. Menard, O. Sentieys, and P. Scalart, "Analytical accuracy evaluation of fixed-point systems," in *Proceedings of the 15th European Signal Processing Conference (EUSIPCO '07)*, Poznań, Poland, September 2007.

[17] B. Widrow, "Statistical analysis of amplitude quantized sampled-data systems," *Transactions of the American Institute of Electrical Engineers–Part II*, vol. 79, pp. 555–568, 1960.

[18] A. Sripad and D. Snyder, "A necessary and sufficient condition for quantization errors to be uniform and white," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 25, no. 5, pp. 442–448, 1977.

[19] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley Series in Computer Science and Information, Addison-Wesley, Boston, Mass, USA, 2nd edition, 1978.

[20] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Fla, USA, 1996.

[21] R. Rocher, D. Menard, N. Herve, and O. Sentieys, "Fixed-point configurable hardware components," *EURASIP Journal on Embedded Systems*, vol. 2006, Article ID 23197, 13 pages, 2006.

[22] K. Ozeki and T. Umeda, "An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties," *Electronics and Communications in Japan*, vol. 67, no. 5, pp. 19–27, 1984.

[23] T. Ojanperä and R. Prasad, *WCDMA: Towards IP Mobility and Mobile Internet*, Artech House Universal Personal Communications Series, Artech House, Boston, Mass, USA, 2002.

[24] D. Pan, "A tutorial on MPEG/audio compression," *IEEE MultiMedia*, vol. 2, no. 2, pp. 60–74, 1995.

[25] T. Jansson, "BladeEnc MP3 encoder," 2002.

[26] A. Lerch, "EAQUAL–Evaluation of Audio QUALity," Software repository: January 2002, http://sourceforge.net/projects/eaqual.

[27] M.-A. Cantin, Y. Savaria, and P. Lavoie, "A comparison of automatic word length optimization procedures," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '02)*, vol. 2, pp. 612–615, Scottsdale, Ariz, USA, May 2002.