REC'2016

Actas das

XII Jornadas sobre Sistemas Reconfiguráveis

20 e 21 de junho de 2016

Departamento de Engenharias
Escola de Ciências e Tecnologia
Universidade de Trás-os-Montes e Alto Douro

Editores:
José Carlos Cardoso
João Agostinho Pavão
Arnaldo S. R. Oliveira
Jorge Casal Santos

José Carlos Cardoso, João Agostinho Pavão
Universidade de Trás-os-Montes e Alto Douro – Comissão Organizadora da REC'2016
Arnaldo S. R. Oliveira, Jorge Casal Santos
Universidade de Aveiro – DETI / Instituto de Telecomunicações

ISBN: 978-989-704-110-5

Conteúdo

Prefácio

Este volume contém as comunicações apresentadas nas XII Jornadas sobre Sistemas Reconfiguráveis, REC 2016, que decorreram no Departamento de Engenharias da Escola de Ciência e Tecnologias da Universidade de Trás-os-Montes e Alto Douro em 20 e 21 de junho de 2016. As Jornadas sobre Sistemas Reconfiguráveis são o evento nacional para promover a interatividade e cooperação entre a comunidade científica de língua portuguesa com atividade de investigação e desenvolvimento na área dos sistemas eletrónicos reconfiguráveis.

O programa das jornadas REC 2016 contou com 12 publicações, cada uma revista por um mínimo de 2 elementos do respetivo Comité Cientifico. Incluiu também uma comunicação convidada proferida por Markus Weinhardt, da *Osnabrück University of Applied Sciences*, que gentilmente aceitou o convite para estar presente e enriquecer o programa das jornadas com uma apresentação intitulada "*Reconfigurable Computing - Architectures and High-Level Programming*".

O programa foi dividido em 5 sessões temáticas nos domínios da segurança, processamento de imagem, controlo, aplicações de processamento de sinal e multiprocessamento.

A organização agradece a todos os autores a disponibilidade demonstrada na submissão e revisão dos seus trabalhos, assim como na apreciação e partilha de conhecimentos com todos os participantes da REC 2016.

A organização agradece também aos membros do Comité Científico pela disponibilidade em rever os trabalhos submetidos e desta forma contribuir para o seu melhoramento.

A organização destas Jornadas contou com o apoio da Universidade de Trás-os-Montes e Alto Douro, através da sua secção GForm para o tratamento das inscrições e disponibilização de salas, e da Presidência da ECT na motivação para esta realização.

A organização quer também prestar um agradecimento especial à organização das XI edição das jornadas – REC 2015 – no ISEP, na pessoa do Prof. Doutor Manuel Gericota por ter fornecido uma base de informação, que muito facilitou o nosso trabalho.

Finalmente esperamos que esta edição das Jornadas tenha ido ao encontro das espectativas dos investigadores que nelas participaram, quer pela troca de ideias e experiências, quer pelo convívio.

José Carlos Cardoso, Universidade de Trás-os-Montes e Alto Douro
Arnaldo S. R. Oliveira, Universidade de Aveiro – DETI / Instituto de Telecomunicações
Junho 2016

Comissão Organizadora

José Carlos Cardoso
Universidade de Trás-os-Montes e Alto Douro – ECT

João Agostinho Pavão
Universidade de Trás-os-Montes e Alto Douro – ECT

Arnaldo S. R. Oliveira
Universidade de Aveiro – DETI
Instituto de Telecomunicações

Contacto Geral

Organização da REC'2016
Escola de Ciências e Tecnologias
Universidade de Trás-os-Montes e Alto Douro
Quinta de Prados
5000-801 Vila Real
Portugal
Tel.: +351 259 350 000
Fax: +351 259 350 480
E-mail: rec2016@utad.pt
URL: http://rec2016.utad.pt/

Comité Científico

Coordenação

José Carlos Cardoso	Universidade de Trás-os-Montes e Alto Douro
João Agostinho Pavão	Universidade de Trás-os-Montes e Alto Douro
Arnaldo S. R. Oliveira	Universidade de Aveiro / IT

Comité de Programa

Adriano Tavares	Universidade do Minho
Ana Antunes	Instituto Politécnico de Setúbal
André Fidalgo	Instituto Superior de Engenharia do Porto
Aniko Costa	Universidade Nova de Lisboa / UNINOVA
António Esteves	Universidade do Minho
António Ferrari	Universidade de Aveiro / IEETA
Fernando Gonçalves	Instituto Superior Técnico / INESC-ID
Gabriel Falcão	Instituto de Telecomunicações
Helena Sarmento	Instituto Superior Técnico / INESC-ID
Horácio Neto	Instituto Superior Técnico / INESC-ID
Iouliia Skliarova	Universidade de Aveiro / IEETA
João Bispo	Instituto Superior Técnico / INESC-ID
João Canas Ferreira	Fac. de Engenharia da Univ. do Porto / INESC Porto
João M. P. Cardoso	Fac. de Engenharia da Univ. do Porto / INESC Porto
João Lima	Universidade do Algarve
João Paulo Teixeira	Instituto Superior Técnico / INESC-ID
Jorge Lobo	Universidade de Coimbra / ISR
José Augusto	Fac. de Ciências da Univ. de Lisboa / INESC-ID
José Carlos Alves	Fac. de Engenharia da Univ. do Porto / INESC Porto
José C. Metrôlho	Instuto Politécnico de Castelo Branco
Leonel Sousa	Instituto Superior Técnico / INESC-ID
Luís Cruz	Universidade de Coimbra / DEEC
Luís Gomes	Universidade Nova de Lisboa / UNINOVA
Luís Nero Alves	Universidade de Aveiro / IT
Manuel Gericota	Instituto Superior de Engenharia do Porto
Marco Gomes	Universidade de Coimbra / DEEC
Mário Véstias	Instituto Superior de Engenharia de Lisboa / INESC-ID
Mário Zenha-Rela	Universidade de Coimbra
Morgado Dias	Universidade da Madeira
Nuno Roma	Instituto Superior Técnico / INESC-ID
Orlando Moreira	ST-Ericsson
Paulo Flores	Instituto Superior Técnico / INESC-ID

Pedro Diniz Instituto Superior Técnico / INESC-ID
Ricardo Chaves Instituto Superior Técnico / INESC-ID

Comunicação Convidada

Moderação: Arnaldo S. R. Oliveira
Universidade de Aveiro – DETI / Instituto de Telecomunicações

Reconfigurable Computing - Architectures and High-Level Programming

Markus Weinhardt
Osnabrück University of Applied Sciences
mweinhardt@computer.org

Abstract

This talk gives an overview of Reconfigurable Computing with an emphasis on high-level programming methods.

First, different options for implementing algorithms are discussed. While hardware implementions can outperform software, the cost of hardware (i.e., design and/or production cost) is often prohibitive. Fine-grain reconfigurable (FPGA) hardware reduces the production cost, but designing circuits with hardware-description languages on the RT-level remains time-consuming and error-prone.

Therefore, high-level synthesis (HLS) and system-level synthesis tools are required. They claim to enable software programmers to use FPGAs. We present advances in this area and the remaining challenges in building "hardware compilers" for high-level (software) languages. Our own research investigated possible solutions to further improve HLS tools, and scalable and portable implementations of streaming applications on heterogeneous architectures.

The last part of the talk covers the role of many-core processors and Coarse-Grain Reconfigurable Arrays (CGRAs) - as FPGA overlays or as ASICs – for improving performance and designer productivity. Common architectures, compilers, and future developments are presented.

Author Short Bio

Markus Weinhardt is a professor for hardware/software systems at Osnabrück University of Applied Sciences, Germany. He received his diploma and Dr.-Ing. degrees in Informatics from the University of Karlsruhe (KIT), Germany, in 1992 and 1997, respectively. After graduation, he spent three years as postdoctoral researcher at the Department of Computing, Imperial College, London. Before joining Osnabrück UAS, he was the Chief Compiler Architect at PACT XPP Technologies AG in Munich, Germany.

His research interests include reconfigurable and parallel computing (focusing on FPGAs and Coarse-Grain Reconfigurable Arrays), high-level design methods, image processing and compiler construction. Markus Weinhardt has (co-)authored more than 30 conference papers, journal publications and book chapters and holds several patents. He serves on the program committees of several international conferences in the area of reconfigurable computing.

Markus Weinhardt spends the summer semester 2016 as a visiting researcher at INESC-id/IST, Lisbon, Portugal.

Sessão Regular I

Segurança

Moderação: João Canas Ferreira

Fac. de Engenharia da Univ. do Porto / INESC Porto

Efficient Hardware Implementation of the SHA-3 Hash Function

Magnus Sundal and Ricardo Chaves
INESC-ID, Instituto Superior Técnico, Universidade Lisboa
mvsundal@outlook.com, Ricardo.Chaves@inesc-id.pt

Abstract

This paper explores the existing hardware designs for the four sub-versions of the SHA-3 hash algorithm, with the aim of discovering potential improvements towards pushing the performance and efficiency further. Herein we consider FPGA implementations, but several of the studied techniques can also be considered for ASIC designs, with the exception of the utilization of dedicated FPGA resources. From this analysis, a combination of the individual approaches of the state-of-the-art can be used to increase the throughput and reduce the area requirements. In order to better evaluate and analyze the existing state-of-the-art, two designs were also implemented, namely a simple unfolded and a folded design. Based on this, a clear view is given of the limiting factors of the various existing implementations and where future work should focus.

1. Introduction

Hash functions are an essential part of modern cryptography in integrity and authentication applications. In 2007, The National Institute of Security and Technology (NIST) concluded that it was in due time to determine a successor to the SHA-2 hash function standard [1]. This decision was based on general life-expectancy as well as recently published papers proving a reduction in its strength. A public competition which was initiated in 2007 and ended in 2012 after multiple elimination rounds, determined that a subset of the Keccak sponge function family was the optimal candidate for the new standard titled SHA-3.

Efficiency is key in the implementation of cryptographic algorithms and since 2007, various implementations in both software [2] and hardware [3] have emerged in a range of performances for the SHA-3 hash function, exceeding its predecessors. Hardware co-processors such as FPGAs and ASICs are advantageous for non-general tasks such as in cryptography and offer high parallel processing power compared to general purpose CPUs. Another advantage of hardware implementations when comparing with software is decreased accessibility, which is a great asset concerning security. While software has a short development path it usually runs in shared memory space on top of an operating system, ensuing much room for vulnerabilities.

The scope of this paper covers efficient hardware implementations of SHA-3 with a main focus on FPGAs. It is imperative with a thorough understanding of the existing literature to be able to advance in the technical field. An argument for the choice of technology has been that the majority of the state-of-the-art falls within the area of FPGA implementations. For comparison reasons, ASIC implementations can be less convenient because of the many variables caused by a diversity in technologies and approaches. The challenge of FPGA implementations is to achieve a high frequency. As the size of a design increases, the delay caused by routing can force the system clock to operate at a much lower frequency than what is supported by the FPGA model. A modern FPGA consists mainly of slices containing Configurable Logic Blocks (CLBs) as well as additional Digital Signal Processing (DSP) slices, Block RAM (BRAM), various clock resources and input and output ports (IOs). Slices are conventionally the main unit for measuring the consumption of area on the FPGA, but the utilization of the additional resources should also be included. As with most other relevant literature, the performance objective is optimal efficiency which is a measure of throughput over area.

Based on thorough analysis of the existing implementations, the goal of this project is to develop a design which is compatible with all four sub-versions of SHA-3 and to improve the performance and efficiency of the state-of-the-art. A makeshift design has already been implemented, to explore the validity and divergence of the performance of existing implementations. This analysis is herein presented towards identifying the approach and design techniques that may lead to an improved SHA-3 implementation, particularly on FPGAs.

The paper is organized as follows. In Section 2, the SHA-3 hash function and the underlying Keccak algorithm will be briefly presented [4]. Subsequently in Section 3, the state-of-the-art is presented and analyzed with regard to the most relevant implementations. In Section 4, the work accomplished so far is displayed. Results and evaluation are given in Section 5 and Section 6 concludes the work so far and presents the planned future work for this project.

2. The SHA-3 Algorithm

The algorithm is a family of sponge functions called Keccak which in turn is based on the sponge construction, as depicted in Figure 1. The sponge construction provides a generalized security proof and involves the iteration of an underlying sponge function along with injecting a padded input message with XORs and truncation of the output digest. The data block in which the sponge function acts is

 978-989-704-110-5

called the state and is divided into an outer state - where data is both injected and extracted after processing - and an inner state which is reset to zeros at each new message. The functionality of the sponge therefore depends on the length of the input message. The iteration takes place in the two phases of the sponge, the absorbing and the squeezing phase respectively - if the input message and the output digest is larger than the outer state. Otherwise, the underlying sponge function is only processed once.

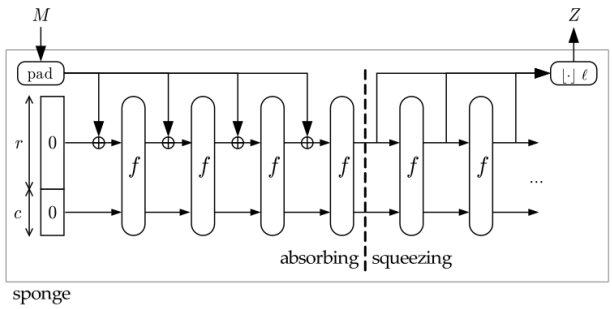


Figure 1: The sponge construction. / CC BY 3.0 @noekeon.org

The state is presented as a 3-dimensional block for the benefit of easy apprehension of the sponge function operation, as illustrated in Figure 2. Words constituting the padded input message fills the state lane-wise starting at the center. The inner state is therefore located at the highest coordinates, i.e. from 4,4 and downwards. The size

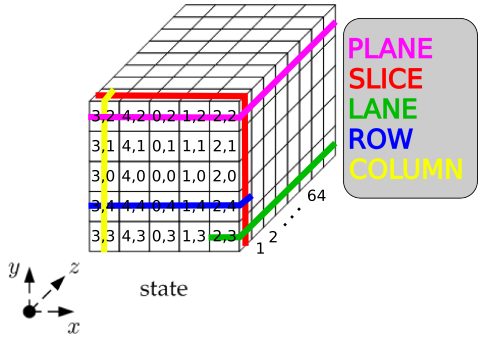


Figure 2: The state.

of the outer state where the message is injected is conventionally referred to as the block size. The size of the inner state is the main parameter influencing the security proof of the sub-versions of SHA-3 and the digest is roughly half this size. There are currently four sub-versions of SHA-3 supported by NIST, as seen in Table 1. They differ in the size-ratio between the inner state and block size, but the total state is always 1600 bits (5x5x64).

Version	Block size	Inner state	Digest
SHA3-224	1152	448	224
SHA3-256	1088	512	256
SHA3-384	832	768	384
SHA3-512	576	1024	512

Table 1: The four sub-versions of SHA-3.

The underlying function consists of 24 rounds of a five-step sequence of transformations and permutations called the round function. These are largely based on XORs, rotations as well as a few NOT and AND operators:

θ-Theta (A)

```
B[x,z] = A[x,0,z] ⊕ A[x,1,z] ⊕ ...
      ⊕ A[x,4,z]
C[x,z] = B[x−1,z] ⊕ B[x+1,z−1]
D[x,y,z] = A[x,y,z] ⊕ C[x,z]
```

ρ-Rho (D,r)

```
E[x,y,z+r(x,y)] = D[x,y,z]
```

π-Pi (E)

```
F[y,2x+3y,z] = E[x,y,z]
```

χ-Chi (F)

```
G[x,y,z] = F[x,y,z] ⊕
      ((NOT F[x+1,y,z]) AND F[x+2,y,z])
```

ι-Iota (G,RC)

```
H[0,0,z] = G[0,0,z] ⊕ RC[z]
```

Code 1: Pseudo-code of the five steps of the round function.

Theta provides diffusion on to two adjacent columns. Rho permutates each lane internally by a rotation offset given by a 5x5 matrix r. Pi permutates the lanes with respect to each other in the x and y positions. Chi provides non-linearity, acting on each row and Iota differentiates each round by XORing the center lane with round constants. The round constants can either be generated by a 7-bit linear-feedback-shift-registers (LFSRs) or be pre-generated and stored as an array.

The padding of messages is such that a '1' bit is appended after the LSB of the last byte of the message and finally a 0x80 is appended to the last byte of the block. The NIST API specifies the byte-ordering as little-endian and bit-ordering as big-endian so that the MSB of each byte is located at the lower address.

3. State of the art

The existing hardware implementations can be grouped into two classes: high-speed and compact designs. The former contains simple unfolded architectures with the highest performance related with efficiency. These designs have maximum internal data path with registers usually based on flip flops. The latter contains the more complex designs with minimal footprint. Block or distributed RAM is usually utilized in these implementations and the width of the internal data path is minimized. The compact designs are usually more complex as they inhibit elegant instruction cycles and control units in the pursuit of maximum concurrency. Still, they tend to be too conservative in their area utilization to obtain a decent throughput and

are for that reason inferior in efficiency compared to the high-speed designs. A brief chronological introduction of these designs are given below, starting with the high-speed implementations.

Strömbergson [5] provided valuable feedback in 2008 during the early stages of the SHA-3 competition, reviewing the proposed VHDL code for FPGA implementations by the Keccak team (Bertoni *et al.*) [6]. Specifically, he discovered early problems and potential improvements in the early design which have since been corrected. Baldwin *et al.* [7] compared in 2010 the round-two candidates with a general purpose wrapper and included all four subversions of Keccak. The wrapper is the only component which separates this implementation from the previous literature and contains a hardware padding unit. A limited 32-bit input port is the bottleneck of all but the SHA3-512 version. Homsirikamol *et al.* [3] presented at the same time a similar comparison of the 256 bit versions of the same candidates and the following year, the 512 bit versions with and without pipelining. Additionally, they explored the potential for unrolled and unfolded implementations and concludes that this is not relevant for Keccak. They obtain a higher frequency than previous implementations and for that also a higher throughput with the non-pipelined design. The pipelined implementations do not perform well as the area increases more than the frequency. The most critical path remains through the round function as it does not contain any pipeline registers. Input and output buffers are implemented with FIFOs in BRAM, thus saving slices. The late 2010 paper by Akin *et al.* [8] explores internal pipelining and with that obtains close to maximum frequency on a Virtex-4 FPGA and record high throughput. However, the area consumed is too high to achieve a good efficiency, even with optimal FPGA frequency. Another comparison was performed by Jararweh *et al.* [9] in 2012 before the end of the competition, but without any radical new achievements. Athanasiou *et al.* [10] introduced in 2014 a general SHA-3 design supporting all sub-versions and a pipeline register in the round function. They obtain a high frequency and a relatively low consumption of slices. Ayuzawa *et al.* [11] explores the utilization of DSP slices, specifically on the pipelined design by Akin *et al* as well as variations of this. They find that certain pipelined designs benefit from utilizing the DSP slices as no delay is added to the most critical path. The following year, Ioannou *et al.* [12] presents an implementation without additional pipeline registers, a smaller area than Athanasiou *et al.* but with a comparable frequency. Additionally, they present a pipelined 2-factor unrolled design which performs even better with respect to efficiency than the straight-forward design.

As with the High-speed design, the initial compact design was first suggested by Bertoni *et al.* with the low-area coprocessor. Instead of processing the full width of the state, this design has a lane-wise architecture so that the internal data path is 64 bits. A very high latency results in a poor throughput for this design. Each round contains 55 bubbles which are cycles where the state is not accessed. Kerckhof *et al.* [13] compares the compact design of the five SHA-3 finalists in 2012. Their implementation of Keccak is an improvement of the low-area coprocessor by Bertoni *et al.*. The internal data path is the same, but the state as well as intermediate values are stored in BRAM where two lanes can be read or written each cycle. The area is reduced by more than 50 % compared to its predecessor. Jungk & Apfelbeck [14] presents a slice-wise architecture where the state is split into 8 pieces of 8 slices, resulting in an internal data path of 200 bits and latency of 200 cycles. The state is stored in distributed RAM, which in the Virtex-5 FPGA can be read asynchronously. The round function in this design is re-scheduled so that the Rho and Pi steps are the last steps in all but the first and last round. This results in 3 different rounds, but the dependency problem of the permutation steps is avoided. San & At [15] improves on the lane-wise architecture by introducing a fine-tuned instruction sequence which allows for high concurrency along a serialized round function. The latency is further reduced by 50 % and along with an optimal frequency this implementation yields a high throughput and efficiency compared to the other compact, folded designs.

4. Proposed implementation

None of the existing designs utilize any distinct techniques for minimizing the critical path or the area, such as have been suggested in the Keccak Implementation Overview [6], e.g. with bit-interleaving and lane-complementing to save NOT operations. Area is simply saved by utilizing alternative resources to slices such as BRAM, distributed RAM and DSP slices and additionally decreasing the internal data path. The latter naturally causes an increase in latency and the area is seldom proportionally decreased. On the other hand, the downside of utilizing BRAM is the limited width of the memory blocks. Cascading the deep memory blocks will quickly consume a large portion of what is available on the FPGA and will further defeat the purpose of compact cascade-able designs.

The initial task has been to explore the unfolded high-speed designs and to examine the feasibility of matching or surpassing the state of the art. Additionally, the reported performances of the existing designs are diverging more than expected for such similar implementations. It is uncertain how much effort has been put into the existing designs with respect to manually placing components on the FPGA. This is a time consuming task, but one that can yield good results in frequency if done properly. San & At mentions careful placing and routing of components and reports close to optimal frequency of their design. It is up for discussion whether it is a reasonable approach to rely on manual methods instead of the standard vendor-provided tools. The basis for the reported performance of the individual designs are not equal. While area consumption remains mostly the same, synthesis results of clock frequency are vastly different than those which are based on Place And Route (PAR). It has therefore been interesting to personally experience which performances that can be reached by replicating the literature.

A straight-forward design was conceived with the main

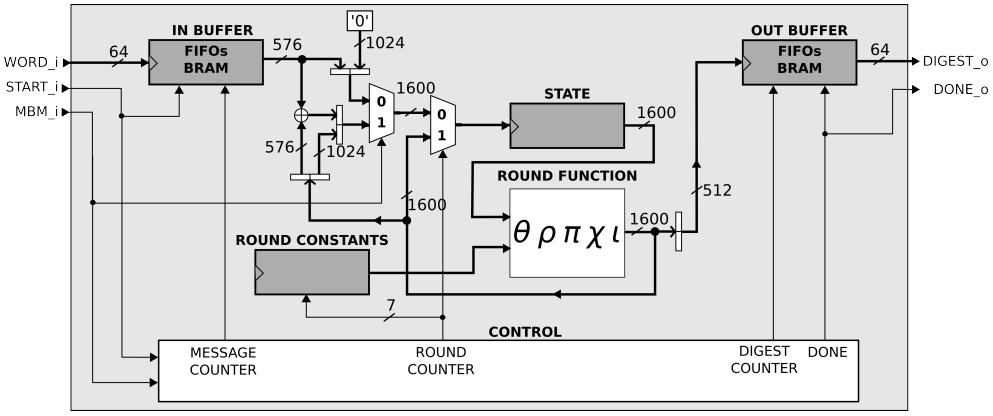


Figure 3: Unrolled high-speed design for SHA3-512.

purpose of studying the achievable critical path of the round function without internal pipeline registers. A schematic of this design is depicted in Figure 3 and is specifically for SHA3-512 with 9 input lanes and 8 output lanes. BRAM FIFOs with first-word-fall-through are used for the IO buffers and the state register is implemented in flip-flops, thus consuming slices. The latency is kept at the bare minimum of 24 clock cycles. The design is dependent on a padding component signaling the start of a message and whether or not it is a multi-block message. In that case there is an iterated absorbing phase of the sponge construction so that the input message must be XOR'ed with the previous outer state and the inner state remains unchanged.

Additionally, a folded design was developed based on the design by Jungk & Apfelbeck [14] with round function rescheduling. The internal data path is 200 bits and both the state and the IO buffers are implemented in BRAM. The shortening of the internal data path saves routing delay and resources. Apart from this, everything else is similar to the unfolded design.

5. Results and evaluation

A summary of the relevant existing implementations including our unfolded and folded architectures are listed in Table 2. Where the original literature has presented results for other versions than the SHA3-512, an estimated scaling has been made based on the version difference. The latency (L) is the number of clock cycles required to finish the processing of a complete message block for 24 rounds.

The literature differ in the expectation of messages. A realistic scenario for hash functions is the hashing of Ethernet packages and so messages should be expected to have sizes larger than the block size, i.e. multi-block-messages (MBM). For this reason, as is also pointed out by Ioannou *et al.* [12], splitting up the round function with pipeline registers should be avoided if this is to be considered. The succeeding message will not be ready for injection before the previous has been processed 24 rounds. The eventual result of this is that the most critical path of the design will be through the round function. As can be seen in Table 2, the designs which have considered an MBM scenario are marked with a YES and the designs with a pipelined round function are not. Considering small messages, we implemented an additional pipelined version of our unfolded design with improved frequency. It contains two additional registers in the round function.

Based on block size, latency and PAR results of maximum frequency and slice consumption, the efficiency is calculated as follows:

$$\textit{Throughput per slice} = \frac{\textit{Blocksize} \text{ x } \textit{Frequency}}{\textit{Latency} \text{ x } \textit{Slices}} \quad (1)$$

In non-MBM scenarios where the designs are pipelined, the latency is naturally larger. However, the effective latency remains the same as multiple small messages are processed simultaneously. The efficiency formula makes it convenient to pin-point the weak factor of a design. The unfolded high-speed designs tend to have a lower working frequency because of high routing delay which is caused by the large data being accessed simultaneously. Still, their low latency results in high throughput and efficiency. The following components are required for the unfolded high-speed designs:

- Msg injection - 576-1152 XORs
- Theta - 3200 XORs (1280+320+1600)
- Rho+Pi - addressing/wiring
- Chi - 1600 XORs, 1600 NOTs and 1600 ANDs
- Iota - 7 XORs

The factors which differentiate the high-speed designs are the utilization of pipeline registers, whether I/O buffers are included and also the FPGA family. It is therefore not a straight forward task to directly compare them. Considering an MBM-scenario where pipeline registers should be avoided, Ioannou *et al.* obtains the best performance. For small messages, a compromise between Athanasiou *et al.* and Akin *et al.* should yield a decent performance as two or three well-balanced pipeline registers should be sufficient to achieve optimal frequency. Homsirikamol *et al.* lacks frequency, but utilizing the available BRAM and First-word-Fall-Through FIFOs to save slices are an obvious advantage for optimizing the total resources on the

Paper	**FPGA** (fam.)	**MBM**	***f*** (MHz)	***L*** (Clk cycles)	***A*** (Slices)	**BRAM** (36 Kb)	***T*** (Gbps)	***T/A*** (Gbps/slice)
Ioannou [12]	V-5	YES	382	24	1581	0	9.2	5.79
Our design unfolded	V-5	-	373	72	1590	16/60	8.9	5.63
Athanasiou [10]*	V-5	-	389	48	1702	0	9.3	5.48
Homsirikamol [3]	V-5	YES	275	24	1220	16/60	6.6	5.37
Baldwin [7]	V-5	YES	189	25	1117	0	8.5	4.32
Our design unfolded	V-5	YES	201	24	1247	16/60	4.8	3.85
Akin [8]*	V-4	-	509	121	4356	0	11.7	2.67
Jararweh [9]*	V-5	YES	271	24	2828	0	6.5	2.29
Strömbergson [5]*	V-5	YES	118	25	1483	0	3.35	2.26
Akin [8]*	V-4	YES	143	24	2024	0	3.29	1.62
Our design folded**	V-5	YES	253	200	291	41/60	0.7	2.50
San & At [15]	V-5	YES	520	1062	151	3/48	0.3	1.66
Jungk & Apfelbeck [14]	V-5	YES	159	200	393	0	0.5	1.17
Kerckhof [13]	V-6	YES	250	2164	144	2	0.07	0.47
Bertoni [6]*	V-5	YES	265	5160	448	0	0.052	0.12

Table 2: Relevant unfolded (top) and folded (bottom) SHA3-512 implementations sorted by efficiency. *Not SHA3-512, adjusted for comparability. **Preliminary results.

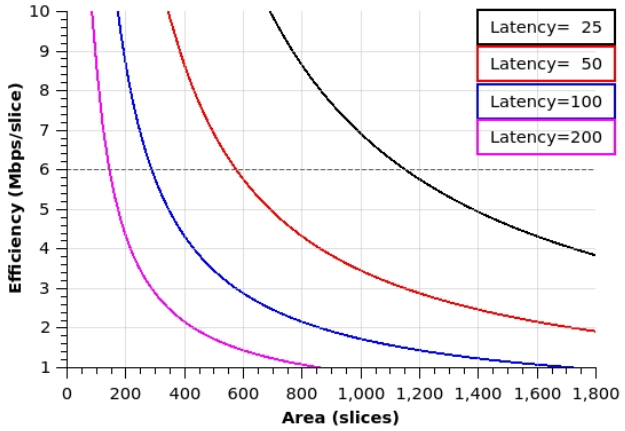


Figure 4: Efficiency formula plot with block size=576 bits, f=300 MHz.

FPGA. The compact designs economize in slice consumption, but obtain a high latency. Limiting the internal data path too much comes with a great cost. San & Al's and Jungk & Apfelbeck's designs are comparable to the worst of the high-speed designs. With the high level of folding in the architecture of San & Al, it seems hard to reach any higher performance than what they report. Jungk & Apfelbeck's frequency does seem to have a potential for improvement.

To illustrate the impact of increased latency, the efficiency formula has been plotted in an area versus efficiency graph for a set of latency values in Figure 4. The frequency has been fixed to 300 MHz, as this is usually achievable even for unfolded high-speed design. The dotted line represents roughly the top of the state-of-the-art performance and is therefore the threshold of interest. As can be seen, a doubling of the latency adds a tight restriction to slice consumption for a desired efficiency.

Returning to Table 2, our non-pipelined unfolded design is closest in resemblance with that of Homsirikamol *et al.*. A majority of the slices are consumed by the state register and XOR operators. No effort has been put into manual placement of components and there has been minimal tweaking of synthesis and MAP options. The obvious weak point is the frequency and over 80 % of the delay in the most critical path is caused by routing through 3 levels of logic in the round function. The folded design obtains an expected higher frequency than the unfolded design, but it is still lower than the performance reported by Ioannou and Homsirikamol.

The choice of FPGA family clearly impacts the performance, both in frequency and slice utilization. Smaller transistor size improves routing delay and the content of a slice differs such as how many inputs each LUT contains. Much of the literature [12, 3, 9, 10] have compared the varying performances between both older and more modern FPGA families. For the Virtex families, when migrating from V-4 to V-5 the frequency tend to increase between 20-40 % and the area decreases between 30-50 %. From V-5 to V-6, the frequency increases between 10-20 % and the area decreases between 3-30 %. The tendencies are similar for V-7. Our designs have been implemented on the Virtex-5 xc5vlx50t which is an average costly device in the V-5 model-range. Table 3 presents the utilization of the relevant resources on the FPGA for the folded and unfolded design. The unfolded design is clearly best balanced of the two and storing both the state and IO buffers in BRAM is not a scalable solution. The folded implementation consumes too much BRAM and too little slices. An approach which has not been much considered since the vanilla design by Bertoni *et al.* is to combine the input and output buffer. Resources can then be saved by scheduling the reception of input words after transmitting the digest. The block size and digest length of all sub-versions of SHA-3 avoids overlapping. DSP slices are not utilized in any of the two designs, but should be considered in the

Design	Type	Util.	Avail.	Ratio
Unfolded	Slices	1229	7200	17 %
	BRAM	16	60	26 %
	DSP	0	48	0 %
Folded	Slices	291	7200	4 %
	BRAM	41	60	68 %
	DSP	0	48	0 %

Table 3: Resource utilization for the unfolded and folded design on a xc5vlx50t FPGA model.

case of a non-MBM scenario where the round function is pipelined so that the most critical path is not affected. This is further covered by Ayuzawa *et al.* [11]. Alternatively, distributed RAM can be used such as in the implementation by Jungk & Apfelbeck where they utilized 25 cascaded 8x8 distributed RAM blocks. For the Virtex-5 FPGA family, the ratio of slices which supports distributed RAM, called SLICEMs, and regular SLICEL slices are between 50-60%. The downside of distributed RAM is the limited width of the output of the LUT. With the unfolded design, the whole block must be accessed concurrently and so only one LUT is used for each bit. In this case the technique provides no benefit in slice consumption as only four bits are stored in each slice.

6. Conclusions and future work

What seem to be missing in the state-of-the-art is an implementation with a proportional utilization of the available resources that is scalable. This means that the ratios of utilized versus available slices, BRAM and DSP slices on the FPGA are balanced. It is imperative for the design to obtain optimal working frequency after routing and that any folding and increased latency is justified. For optimal efficiency, high-factor folding is not found to be an effective solution and should only be considered where minimal area is of priority. This is clearly illustrated in Figure 4. Apart from this, the approaches done by most of what constitutes the state of the art and is covered here should be considered, but only to a relative extent.

The performance of our unfolded design is inferior to the existing designs by Ioannou [12] and Homsirikamol [3]. This is caused by high routing delay and is assumed to be solved by manual placement of components. Our folded design is not well balanced with respect to utilized resources on the FPGA and is therefore not fully scalable.

The results presented here are by no means the final work and serves only as a progress report. The future work will involve the development of a fully balanced design which also is version-flexible so that all four sub-versions of SHA-3 are supported. As the unfolded designs are prone to obtain a high critical path caused by routing, general techniques for reducing this will be explored.

Acknowledgments

This work was supported by the ARTEMIS Joint Undertaking under grant agreement n. 621429 and Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

[1] National Institute of Standards and Technology. FIPS PUB 180-4 - Secure Hash Standard (SHS), 2015. http://dx.doi.org/10.6028/NIST.FIPS.180-4.

[2] D. J. Bernstein & T. Lange. "eBACS: ECRYPT Benchmarking of Cryptographic Systems", 2012. https://bench.cr.yp.to/results-sha3.html.

[3] M. Rogawski E. Homsirikamol and K. Gaj. Comparing Hardware Performance of Round 3 SHA-3 Candidates using Multiple Hardware Architectures in Xilinx and Altera FPGAs. *Researchgate*, 2011.

[4] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche and R. V. Keer. "Keccak sponge function family main document ", 2008. http://keccak.noekeon.org/Keccak-main-1.0.pdf.

[5] J. Stromgbergson. Implementation of the Keccak Hash Function in FPGA Devices. December 2008.

[6] G. Bertoni, J. Daemen, M. Peeters, G. V. Assche and R. V. Keer. "Keccak implementation overview Version 3.2", 2012. http://keccak.noekeon.org/.

[7] B. Baldwin, N. Hanley, M. Hamilton, L. Lu, A. Byrne, M. O'Neill and W. P. Marnane. "FPGA Implementations of the Round Two SHA-3 Candidates". 2010. http://csrc.nist.gov/groups/ST/hash/sha-3/Round2/Aug2010/documents/papers/BALDWIN_FPGA_SHA3.pdf.

[8] O. C. Ulusel A. Akin, A. Aysu and E. Savas. Efficient Hardware Implementations of High Throughput SHA-3 Candidates Keccak, Luffa, Blue Midnight WIsh for Single- and Multi-Message Hashing. *SINCONF, Taganrog, Russia*, pages 168–177, September 2010.

[9] H. Tawalbeh Y. Jararweh, L. Tawalbeh and A. Moh'd. Hardware Performance Evaluation of SHA-3 Candidate Algorithms. *Journal of Information Security*, pages 69–76, April 2012.

[10] G. P. Makkas G. S. Athanasiou and G. Theodoridis. High Throughput Pipelined FPGA Implementation of the New SHA-3 Cryptographic Hash Algorithm. *IEEE*, 2014.

[11] Y. Ayuzawa, N. Fujieda, and S. Ichikawa. "Design Tradeoffs in SHA-3 Multi-Message Hashing on FPGAs", 2014.

[12] H. E. Michail L. Ioannou and A. G. Voyiatzis. High Performance Pipelined FPGA Implementation of the SHA-3 Hash Algorithm. *MECO*, pages 1–4, 2015.

[13] N. Veyrat-Charvillon F. Regazzoni G. M. de Dormale S. Kerckhof, F. Durvaux and F. X. Standaert. Compact FPGA Implementations of the Five SHA-3 Finalists. *Researchgate*, January 2011.

[14] B. Jungk and J. Apfelbeck. Area-efficient FPGA Implementations of the SHA-3 Finalists. *IEEE*, 2011.

[15] I. San and N. At. Compact Keccak Hardware Architecture for Data Integrity and Authenticaion on FPGAs. *Information Security Journal: A Global Perspective, 21*, pages 231–242, August 2012.

[16] NIST. "Approved hashing algorithms". 2015. http://csrc.nist.gov/groups/ST/toolkit/secure_hashing.html.

Secure external memory on embedded devices

Diogo Prata, Ricardo Chaves, and Aleksandar Ilic
INESC-ID, Instituto Superior Técnico, Universidade Lisboa
diogokmrprata@hotmail.com, Ricardo.Chaves@inesc-id.pt,
ilic@sips.inesc-id.pt

Abstract

Security always has been a main concern in the FPGA-based systems, especially when these systems manipulate sensitive applications and data. Since Systems-on-Chip (SoC) are connected to an external memory, the content of this memory can easily be retrieved or tampered by physically probing the memory bus. While the existing state of the art already proposes several solutions, these are not completely secure and impose a significant impact to the systems performance. This paper proposes the design of a reliable system to properly secure the external memory of the device. The goal is to achieve a security system with low performance impact and low on-chip memory overhead. The prototype is based on a Xillinx Zynq architecture.

1. Introduction

MODERN embedded systems carry more and more sensitive information as the range of provided services increases. During the course of time, concerns regarding system performance and power consumption have been taken in consideration, however due to the increasing need of data protection in modern systems, security has become a leading issue. A typical embedded system integrates a processing system and programmable logic in a single device and an external memory. An adversary can perform a board level attack probing the bus between the System-on-Chip (SoC) and external memory, observing data transfers and may even modify or manipulate data without detection [1].

In order to tackle this problem there are several solutions in the literature [5] [7] which implement a security core inside the FPGA to guarantees data confidentiality and integrity. However, the existing state-of-the-art implementation of such hardware security modules impose an increased latency and cause a significant performance loss [2], causing in some cases high on-chip memory overheads [5] [7] [8].

In this paper we propose a simpler and more direct approach to encryption and protection of the data stored on the external DDR memory. The proposed hardware security core is placed inside the FPGA, which is in the same SoC-area as the CPU. By deploying high performance dedicated cryptographic cores low performance impact is expected. The security system is further improved with the use of a victim cache, also implemented on the FPGA fabric, in order to mitigate some possible performance overheads. The memory security system will be implemented and tested on a Zynq-7000 SoC architecture.
The paper is organized as follows: Section 2 presents the considered threat model. The related works are reviewed in Section III. Section IV describes the proposed memory security system. Section V concludes the paper.

2. Threat model

In this work we consider board-level attacks, such as bus probing and memory tampering, while the SoC is considered to be safe. This way an adversary cannot access secure keys or registers internal to the chip, by either physical or logical attacks. The bus that interconnects the SoC to the external memory is a fragile points of an embedded system, since if not protected the system becomes vulnerable to the following attacks:

- *Spoofing* attacks: where the adversary intends to alter the program behaviour and place fake data onto the memory;
- *Splicing or relocation* attacks: the attacker replaces a memory block of one address by the memory block of another address;
- *Replay* attacks: where the adversary records the data flowing in the processor-memory bus and reuses this data on the same address, replacing new information with old information.

3. Related Work

Execute-Only Memory (XOM) [2] proposes a security solution that aims for a secure execution environment. XOM architecture separates different programs in several compartments. A compartment is a logic container that prevents information from circulating into or out of it. This way the programs cannot tamper with each other. The basic approach to implement a compartment is to tag all data with a XOM identifier [3]. These identifiers are stored on-chip. To guarantee data confidentiality, XOM encrypts all instructions and data. AES 256 is the symmetric deciphering algorithm used. In order to prevent data tampering, the integrity mechanism generates a message authentication code that is concatenated with the data before the ciphering process [4], thus avoiding spoofing attacks. The write address is also concatenated in order detect splicing attacks [4]. XOM [2] guarantees a high-level security, but performance overheads can reach 50% [5]. Regarding cache management, information added to cache presents a real overhead [4]. High latency is another consequence of this architecture due to the data confidentiality and data integrity mechanisms. These two mechanisms are not done in parallel, thus decreasing performance.

The AEGIS Project [6] proposes a physical secure platform, incorporating platform and software authentication mechanisms and data privacy and integrity protection. In order to authenticate the processor Physical Random Functions or Physical Unclonable Functions (PUF) are used [6]. Using PUF, each processor has a unique key. This key can be used to authenticate the processor to the users. AEGIS processors offer four different secure modes to off-chip memory [6], controlled by the operating system. Data confidentiality is achieved by the one-time-pad (OTP) encryption scheme. AEGIS uses a hash tree to provide an integrity checking mechanism, where the root hash is stored on-chip. This way on-chip memory overhead is reduced. This memory security solution offers a high-level security [5]. In order to increase efficiency, some hash tree nodes are stored in cache, making the system dependent on cache memory size and presenting performance losses when a cache miss occurs.

Parallelize data Encryption and Integrity Checking engine (PE-ICE) [7] proposes full parallelization of encryption and integrity checking. In this architecture, data confidentiality and integrity mechanisms come from a single encryption algorithm. This approach has the objective of latency reduction, introduced by the cryptographic functions, and hardware cost reduction. Data privacy is guarantee by the AES algorithm. Tamper resistance is reached by the association of a tag to the plaintext. PE-ICE uses different tags for Read Only data (RO) and Read/Write data (RW). PE-ICE has a high on-chip memory overhead since for every block of data it is necessary to store a tag on-chip. This architecture also presents a high off-chip memory overhead since it is necessary to concatenate a tag to every block of data to be encrypted.

Tamper-Evident Counter Tree (TEC-Tree) proposes fully-parallelized security architecture with low on-chip memory requirements [8]. The main focus of this work is on protecting the integrity of data transferred between the SoC and external memory. As in [7], TEC-Tree concatenates tag with the plaintext, adding integrity checking capability. This architecture relies on a modified PE-ICE principle in order to minimize the on-chip memory overhead introduced by this architecture. Being a tree structure, TEC-Tree [4] is composed of leafs and nodes. A leaf consists of the ciphered data and its tag, while a tree node is composed of several concatenated tags. The goal of this architecture is to store the least possible information on the on-chip memory while still guaranteeing integrity checking. For this reason the tree root is the only node that is stored on-chip, while the rest of the tree is securely saved in the off-chip memory.

Romain Vaslin et al. in [5] propose an hardware-based security core. This architecture is able to manage tree security levels: i) no security, ii) confidentially only, and iii) confidentiality and integrity. The hardware-security core contains a security memory map (SMM), which stores the security levels of memory segments and configures the correct datapath inside the core according to the received base address. This component is independent of the processor and operating system. In order to achieve confidentiality, a keystream is generated, using an AES encryption algorithm, and xored with the data to generate the cipher text. The security core has a component called Integrity Checking that evaluates data integrity using one AES round. Integrity Checking stores a tag composed of bits from the AES output on on-chip memory. This component guarantees data protection against replay and relocation attacks. This hardware-security core has a high on-chip memory overhead. This design was tested using an Altera NIOS II soft processor on a Stratix II based prototyping system.

J. Crenne et al. [9] presents a lightweight memory security approach, which is an extension of the work proposed in [5]. In [9], the authenticated encryption is now based on AES-GCM, which can

be pipelined and parallelized. The time stamp and integrity checking tag are still stored on an on-chip memory, which results in high on-chip memory overhead.

SeCBus [10] approach presents a secure software/hardware architecture that guarantees confidentiality and integrity to external memory. The cryptographic operations are implemented in a Hardware Security Module (HSM) in the programmable logic part of the SoC system. This architecture allows different security levels. A Software Security Manager (SSM) is responsible for the dynamic management of Security Policies (SP). The SSM is integrated with the Operating System kernel and an application can access the SeCBus security features through an Application Programming Interface. When a memory access is issued, the HSM fetches the security policy associated with the request's address and guarantees defined cryptographic operations.

Zynq SoC architectures have been used to develop cryptographic co-processors. Systems like [11] [12] offer to the user a service that is secured by a hardware module implemented in the programmable logic (PL) part of the SoC chip. The processor produces data and sends it to the PL. The PL performs the cryptographic operations and returns the ciphered data to the processor. The PL works as a dedicated support processor to the processing system (PS).

Zynq has also been used in another type of systems like memory mapping systems and firewall systems [13] [14] [15]. In [13], a memory tracing tool is implemented on a Zynq board. This implementation disconnects the CPU from memory. When the CPU performs a read or write operation, the request is forwarded to a general purpose (GP) port that connects the processing system (PS) to the programmable logic (PL). The PL accesses the external memory, executes the read/write operation and in case of a read, returns the data through the GP port. The communication between the PL and the external memory is made by a high performance port (HP).

In [14] and [15], a Network-On-Chip Firewall mechanism was prototyped using the Zynq architecture. Like the previous system the CPU has no direct access to external memory. The Firewall module is implemented in the PL and is connected to memory through an HP port. The PS communicates with the PL through a GP port.

In summary, the XOM architecture offers a high-security to external memory but has a big performance loss, a high memory overhead and high latency [16]. AEGIS reduced system latency with the use of trees but still has a big performance loss when a cache miss occurs. PE-ICE has a high on-chip memory overhead due to the fact that it stores all tags on-chip. TEC-Tree presents high latency introduced by node calculation and increased off-chip data transfers. Romain Vaslin et al. store integrity checking tags and the time stamps on chip which causes a high on-chip memory overhead.

4. Proposed Implementation

The proposed solution strives to create a memory security system for external memory with low performance impact and low on-chip memory overhead. The system will be implemented on a Zynq architecture using a ZedBoard platform. The overall design is depicted in Figure 1.

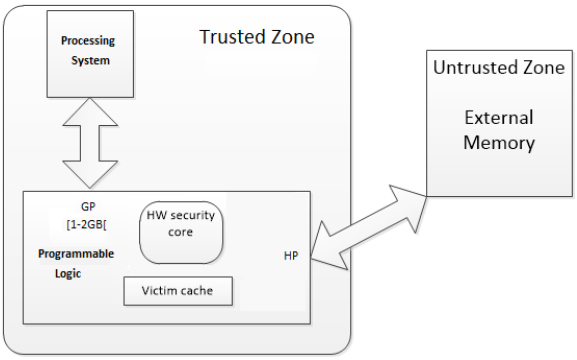


Figure 1: Memory security system

As depicted in Fig.1, the security core is implemented inside the PL. Data produced or requested by the PS passes through the PL to be encrypted or decrypted. The goal is deflect the direct flow of data between the CPU and external memory. The PL works as a security bridge between the PS and external memory. In our proposal, the PS has no knowledge of the existence of this bridge, thus allowing all PS components like the memory management unit, snoop control unit and caches to be fully operational.

When the CPU issues a read or write request to the external memory, the indirect addressing is performed via the PL, instead of directly addressing the memory. A hardware security core, inside the PL, guarantees the cryptographic mechanisms in order to attain two main objectives, i.e., data confidentiality and data integrity. Data confidentiality is guaranteed for all data inside the external memory. Hence, when a write operation occurs, the PS data is passed to the PL where it is encrypted and further stored in the external memory. For a read request, the PL is responsible for retrieving the requested data from memory, which is

decrypted and forwarded to the PS. The communication between the CPU and FPGA is performed through a GP port, while the FPGA is directly connected to memory through an HP port.

In normal conditions the CPU directly accesses external memory through the DDR controller in the [0-1GB[address range. In the proposed architecture the CPU will issue read/write requests through a GP port. The Zynq architecture has two master GP ports associated with the [1-2GB[and [2-3GB[address range. In order to redirect the memory access back to the external memory address range, the security module implemented inside the PL needs to clear the 2 most significant bits of the incoming memory address before accessing external memory. The PL accesses external memory via HP ports.

To experimentally evaluate the feasibility of the proposed approach, a direct memory access mechanism has already been implemented using the PL, within the Zynq architecture. With this mechanism, the PL produces and stores data on the external memory, in an address supplied by the CPU. The validity of these accesses was tested and confirmed using the CPU, which directly accessed the memory to confirm the written values.

Future work will consist in the development and implementation of the complete system and its evaluation, both in terms of performance but also in terms of security. Additionally, second level victim cache implemented on the PL fabric will also be considered, using the existing BRAMs. This will allow reduction of some cache misses thus reducing the overall performance impact of the added security components.

5. Conclusions

In this paper an initial design of a memory security system was proposed, which is to be developed and deployed on a Zynq architecture. In the proposed solution, the CPU is not aware of the existence of a memory security system, which is implemented on the FPGA fabric. The main goal of this work is to achieve the necessary level of security with a low performance overhead and reduced on-chip memory overhead.

Acknowledgements

This work was supported by the ARTEMIS Joint Undertaking under grant agreement n. 621429 and Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

[1] M. G. Kuhn, “Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP,” in IEEE Trans. On Comp., Oct. 1998, pp. 1153-1157.

[2] D.Lie, C. Thekkatah, and M. Horowitz, “Implementing an untrusted operating system on trusted hardware,” in Proc. 19th ACM Symp. Oper. Syst. Princip., 2003, pp. 178-192.

[3] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell and M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," in Architectural Support for Programming Languages and Operating Systems, 2000, pp. 168-177.

[4] R. Vaslin, “Hardware Core for Off-Chip Memory Security Management in Embedded Systems,” Ph.D thesis, Univ. South Brittany, Morbihan, France, 2008.

[5] R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, D. Unnikrishnan, and K. Gaj, "Memory security management for reconfigurable embedded systems," in Proceedings: Int. Conf on Field-Programmable Technology, 2008, pp. 153-160.

[6] G.E. Suh, C. O’Donnell, and S. Devadas, “AEGIS: A Single-Chip Secure Processor,” in IEEE Design & Test of Computers 24(6), 2007, pp. 570–580.

[7] R. Elbaz, L. Torres, G. Sassatelli, "Added Redundancy Explicit Authentication at the Block Level for Parallelized Encryption and Integrity Checking on Processor-Memory Buses", Sep 2007.

[8] R. Elbaz, D. Champagne, R.B. Lee, L. Torres, G. Sassatelli, and P. Guillemin, “Tec-Tree allow: A low cost and parallelized tree for efficient defense against memory replay attacks,” in Proc. Cryptogr. Hardware and Embedded Syst. CHES, 2007, pp. 289-302.

[9] J. Crenne, R. Vaslin, G. Gogniat, J.-P. Diguet, R. Tessier, and D.Unnikrishnan, “Configurable Memory Security in Embedded Systems,”, ACM Trans. Embedded Comput. Syst., vol. 12, no. 71, Sep 2011.

[10] J. Brunel, S. Ouaarab, R. Pacalet, and G. Duc, “SeCBus, a software/hardware architecture for securing external memories,” in IEEE Int. Conf on MOB Cloud Comp. Serv, and Eng, 2014, pp. 277-282.

[11] P. Svasta, A. Marghescu, T. Neacsa, “Cryptographic Coprocessor for Data Integrity Algorithms,” in Proceedings: Int. Spring Seminar on ELEC Tech., May 2014, pp. 91-94.

[12] A. Marghescu, P. Svasta, “Secure Communication Protocol using Embedded Devices based on FPGA,” in Electronics System-Integration Technology Conference, 2014, pp. 1-4.

[13] L. W. Li, G. Duc, R. Pacalet, “Hardware-assisted Memory Tracing on New Socs Embedding FPGA Fabrics,” in Proceeding: Computer Security Applications Conference, 2015, pp. 461-470.

[14] G. Kornaros, I. Christoforakis, and O. Tomoutzoglou, "Hardware Support for Cost-Effective System-level Protection in Multi-Core SoCs", in Digital System Design, Aug 2015, pp. 41-48.

[15] M. D. Grammatikakis, P. Petrakis, A. Papagrigoriou, G. Kornaros, and M. Coppola, “High-Level Security Services based on a Hardware NoC Firewall Module,” in Intel. Sol. in EMBD SYS, Oct 2015, pp. 73-78.

[16] Z. Liu, Q. Zhu, D. Li, and X. Zou, “Off-Chip Memory Ecryption and Integrity Protection Based on AES-GCM in Embedded Systems,” in IEEE Design & Test, Dez 2013, pp. 54-62.

Sessão Regular II

Processamento de Imagem

Moderação: Horácio Neto

Instituto Superior Técnico / INESC-ID

Uma Abordagem Multi-softcore Baseada em FPGA para o Algoritmo HOG

José Arnaldo Mascagni de Holanda[†], João Manuel Paiva Cardoso[‡], Eduardo Marques[§]
[†]*Instituto Federal de São Paulo,* [‡]*Universidade do Porto,* [§]*Universidade de São Paulo,*
arnaldomh@ifsp.edu.br, jmpc@fe.up.pt, emarques@icmc.usp.br

Resumo

O algoritmo Histograma da Gradientes Orientados (HOG) é um dos mais utilizados atualmente para detecção de objetos em imagens. Seu uso para detectar eficientemente diferentes classes de objetos requer variações e adaptações facilmente providas por soluções de software. Algoritmos desse tipo têm sido parte de sistemas embarcados inteligentes de tempo real, necessitando de aplicações eficientes que forneçam alto desempenho, baixo consumo de energia e programabilidade que permita uma maior flexibilidade no desenvolvimento. Nesse contexto, mostramos nosso trabalho para mapear o algoritmo HOG em sistemas embarcados, tendo em mente problemas de alto desempenho e programabilidade. Mais especificamente, considera-se a execução do algoritmo em um sistema baseado em FPGA contendo múltiplos processadores softcore Nios II e também em um processador embarcado ARM. Mostra-se como reduzir o tempo de execução do algoritmo por transformações source-to-source e eliminação de processamento redundante. Mostra-se como a utilização de pipeline de processamento com dois processadores Nios II obtém speedup de $47,2\times$.

1. Introdução

O algoritmo Histograma de Gradientes Orientados (HOG) [1] é atualmente uma das abordagens mais utilizadas para a detecção de diversas classes de objetos, como de pedestres, veículos e sinais de trânsito. No contexto de detecção de pedestres, [2] apresentam um *ranking* com 44 métodos de detecção de objetos, dentre os quais 75% utilizam *features* HOG, incluindo 6 dos 10 métodos melhores classificados. Estas estatísticas mostram a popularidade e a efetividade deste algoritmo. Ainda, a diversidade de trabalhos baseados em HOG mostra que variações deste algoritmo têm sido exploradas.

Diversos estudos têm como objetivo implementações embarcadas do algoritmo HOG. Exemplos de trabalhos recentes envolvendo implementações em FPGA incluem [3, 4]. A maior parte dos trabalhos baseados em FPGA para implementação do algoritmo HOG é baseada em implementações fixas em hardware. Apesar de fornecer bom desempenho, tais implementações não oferecem a flexibilidade necessária, e.g., para explorar diferentes variações algorítmicas. Outros trabalhos, tais como [5] e [6], focam no uso de processadores soft-core próprios, capazes de acelerarem o algoritmo. Estes, contudo, fornecem pouco suporte de ferramentas que permitam explorar suas arquiteturas com software já desenvolvido.

Neste contexto, o propósito deste artigo é mostrar alguns aspectos da implementação do algoritmo HOG em sistemas embarcados, representados por um sistema multisofcore baseado em FPGA e um processador multi-core ARM. Busca-se também compreender o desempenho possível atingido por um ou mais núcleos de processamento, considerando otimizações por software, sem o uso de hardware específico. Acreditamos que este seja um passo importante para estabelecer limitantes superiores de desempenho e explorar formas de se conseguir resultados mais flexíveis. Mais especificamente, este artigo mostra: (i) os esforços necessários para melhorar as implementações em software do algoritmo HOG, de forma a atingir melhor desempenho em plataformas embarcadas; e (ii) uma arquitetura multi-core em *pipeline* que pode ser utilizada e customizada para aproveitar o paralelismo de pipelining no algoritmo HOG.

2. Descrição do Algoritmo

O método HOG [1] utiliza janelas de detecção que deslizam sobre a imagem de entrada extraindo descritores baseados em histogramas de orientações de gradientes. Um classificador SVM é então utilizado para determinar se o descritor contém uma instância do objeto buscado. Cada janela de detecção é subdividida em blocos de pixels sobrepostos, dos quais são extraídas *features* que compõe o descritor HOG. Um bloco, por sua vez, é composto por regiões uniformemente espaçadas chamadas de células, as quais contribuem com histogramas 1-D de orientações dos gradientes de seus pixels. O agrupamento dos pixels em regiões, como células e blocos, possibilita a aplicação de filtros e normalizações locais que aumentam a eficiência da detecção. A Figura 1 mostra os estágios utilizados para a detecção objetos com *features* HOG.

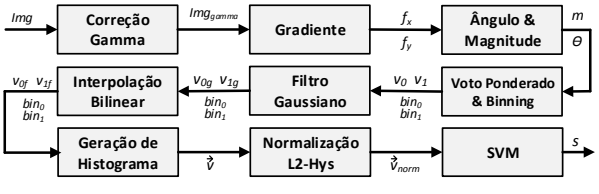


Figura 1. Cadeia de extração de features HOG e detecção de objetos.

 978-989-704-110-5

No início na cadeia de processamento, os pixels da imagem de entrada *Img* são filtrados com correção gamma para reduzir a influência de mudanças na intensidade da luz. Em seguida, aplica-se a Img_{gamma} a máscara 1-D $[-1\,0\,1]$ em ambas as direções (x e y) de forma a obter as derivativas espaciais f_x e f_y para cada pixel. Das derivativas, são obtidos a magnitude m e o ângulo θ de cada pixel. Para cada célula, os valores de m e θ são acumulados em histogramas de orientações compostos por 9 bins, que cobrem orientações de 0^o a 360^o em intervalos de 20^o. Para reduzir o efeito de *aliasing*, o valor de m é ponderado entre dois bins vizinhos, bin_0 e bin_1 (*binning*), na forma de dois votos, v_0 e v_1.

Para cada bloco, um filtro gaussiano é aplicado a fim de reduzir a influência dos votos dos pixels que se encontram nas bordas. Ainda dentro dos blocos, os votos de cada pixel são interpolados bilinearmente entre suas células. Os histogramas das células são então concatenados e o histograma de bloco resultante ($\vec{v}$) é submetido a uma normalização L2-Hys. Após este estágio, os histogramas de bloco normalizados ($(\vec{v}_{norm})$) da janela de detecção são concatenados em um descritor, o qual é submetido ao classificador SVM. Este, por sua vez, retorna o valor de confiança da classificação, o qual é comparado a um limiar. Se o valor for maior que o limiar, isso significa que foi detectada um instância do objeto procurado.

A cadeia de detecção pode ser repetida para diferentes escalas da imagem de entrada e múltiplas detecções da mesma instância de um objeto podem ser evitadas aplicando-se um algoritmo de supressão não máxima nos resultados. Esses últimos estágios não são abordados nesse trabalho.

3. *Setup* do Sistema

Como ponto de partida, foi desenvolvida uma versão de referência do algoritmo HOG baseada na implementação das funções HOG da biblioteca OpenCV [7]. Buscou-se manter as otimizações originais, como o uso de tabelas de consulta para coeficientes de filtros e normalizações, utilizando, contudo, somente bibliotecas padrão da linguagem C++, de modo a facilitar a portabilidade do código.

Os parâmetros do algoritmo para a aplicação de referência seguiram, na maior parte, o trabalho original de [1]. Utiliza-se uma imagem de entrada com resolução VGA (640×480 pixels) em níveis de cinza, selecionada a partir da base de dados de pedestres do INRIA [1]. Uma abordagem de única escala é considerada. Utilizam-se janelas de detecção de 64×128 pixels e blocos de 16×16 pixels contendo 2×2 células de 8×8 pixels. Os blocos se sobrepõem em 8 pixels, as orientações dos gradientes são acumuladas em histogramas de 9 bins, e a janela desliza na direção de ambos os eixos com um deslocamento de 8 pixels, totalizando 3.285 janelas por imagem.

Para avaliar essa versão da aplicação são consideradas duas plataformas embarcadas. A primeira é um placa ODROID XU3, com um processador ARM Cortex-A7 quad core, com frequência de 1.2Ghz, 512KB Cache e 2GB RAM. A segunda plataforma utilizada é um kit de desenvolvimento DE2i-150 [8], contendo um FPGA Altera Cyclone IV EP4CGX150DF31 GX com 149.760 elementos lógicos, 720 blocos de memória M9K e 6.480 Kbits de memória embarcada e 128MB de SDRAM externa. No FPGA, inicialmente, foi instanciado um sistema *single-core* baseado no processador Nios II [9]. O sistema é composto por um núcleo Nios II/f com 5 estágios de pipeline, 4 KB de cache de instruções, 2 KB de cache de dados, acesso à memória RAM on-chip e externa, ROM on-chip com o vetor de classificação SVM e multiplicadores e divisores em hardware habilitados. Como o algoritmo HOG faz uso extensivo de operações de ponto flutuante com precisões simples e dupla, o conjunto de instruções é estendido com o hardware de ponto flutuante da Altera [9].

4. Avaliação e Aceleração do Código

A Tabela 1 apresenta os resultados do *profiling* da aplicação de referência ao ser executada em um único processador nas duas plataformas. Por simplicidade, algumas funções HOG foram agrupadas de acordo com seu contexto na cadeia de extração de *features* e classificação. As tarefas relacionadas ao processamento de blocos são responsáveis por 92,9% e 78,6% do tempo de execução da aplicação para os processadores ARM e Nios II, respectivamente. Este grupo é composto pelas funções de filtro gaussiano, pela interpolação bilinear e pela geração de histogramas (veja Fig. 1), as quais são todas aplicadas sequencialmente ao mesmo bloco de votos e bins. Como ocupam a maior parte do tempo de execução, serão foco neste trabalho dos esforços para otimização.

Funções	**ARM@1.2GHz**		**Nios II@150MHz**	
	%	**# Ciclos**	**%**	**# Ciclos**
Proc. de bloco	92,9	1,48G	78,6	11,0G
L2-Hys	3,7	59,0M	7,7	1,1G
SVM	2,8	44,7M	11,1	1,6G
Grad. e Binning	0,4	6,38M	2,2	0,32G
Outros	0,2	3,19M	0,4	57,0M
Total	1.59G ciclos		14,0G ciclos	

Tabela 1. Resultado do *profiling* da aplicação de referência nos processadores ARM e Nios II

Apesar do uso de tabelas de consulta nas funções de processamento de bloco já permitir uma execução mais eficiente, essas funções ainda executam muitas vezes um conjunto de multiplicações e somas em ponto flutuante. Mais precisamente, as funções de processamento de bloco são executadas 105 (7×15 blocos) vezes por janela de detecção, uma vez por cada bloco dentro de uma janela. Para os parâmetros especificados, a aplicação de referência processa 3.285 janelas de detecção sobrepostas e, consequentemente, as funções de processamento de bloco são executadas 344.925 vezes. Contudo, como as normalizações e os filtros são aplicados no máximo em nível de blocos, não há necessidade de processar um bloco mais de uma vez, não importando a que janela ele pertença. Ao considerar o reuso de dados, é possível obter uma diminuição de duas ordens de magnitude na execução de blocos, reduzindo o

número de chamadas a funções de processamento de bloco para 4.661 (número de blocos sobrepostos dentro de uma imagem).

Para eliminar o processamento redundante de blocos, foi implementado um *buffer* de blocos que armazena dados de blocos já processados em janelas de detecção anteriores. Este *buffer* age como um apoio para a janela de detecção deslizante, permitindo-a reutilizar blocos antigos e processar somente os novos. De forma similar à solução adotada em hardware por [3], cada elemento do buffer consiste no histograma de bloco normalizado de tamanho 1×36. Para os parâmetros do algoritmo HOG mencionados, o *buffer* de blocos requer 58 KB de memória para armazenar 7×59 histogramas de blocos utilizando números em ponto flutuante com precisão simples.

Alguns melhoramentos menores também são feitos sobre a versão de referência, como a substituição da representação de números de ponto flutuante com precisão dupla por precisão simples, o reuso de resultados intermediários para evitar acessos à memória, alinhameno de arranjos de estruturas e a implementação eficiente de funções (e.g. função *floor*). Denomina-se, a partir daqui, *otimizada* a versão com as melhorias apresentadas.

5. Arquitetura Multicore

A partir da aplicação otimizada, procura-se aproveitar a flexibilidade oferecida pelo FPGA e utilizar processamento paralelo para obter ganhos em desempenho. Para isso, é feita uma distribuição das funções do HOG entre um arranjo de processadores em pipeline. Neste trabalho, considera-se a implementação de um *pipeline* com dois estágios de processadores Nios II, conforme ilustrado na Figura 2.

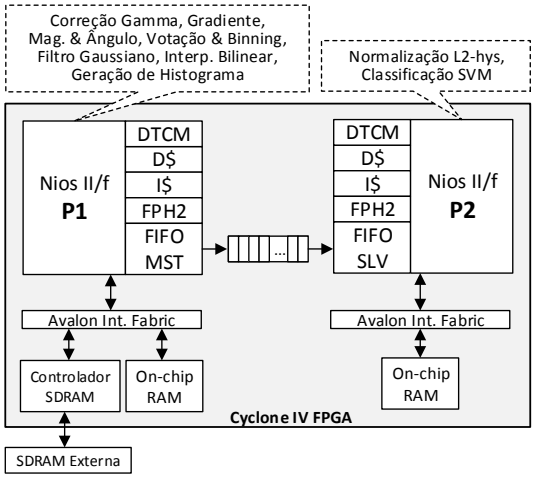


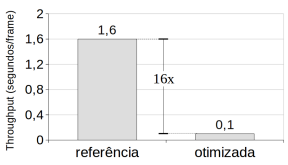
Figura 2. Arquitetura multicore em *pipeline* com dois estágios.

Cada instância do Nios II (P1 e P2) contém caches de instrução (I$) e de dados (D$), hardware de ponto flutuante (FPH2) e memórias on-chip locais. Os processadores intercomunicam-se por meio de um módulo FIFO. Somente P1 acessa a memória SDRAM externa, de onde é obtida a imagem de entrada. Além disso, os processadores utilizam memórias fortemente acopladas (DTCM) para armazenar dados frequentemente utilizados, tais como tabelas de consulta e pesos do vetor de classificação SVM. Como essas memórias não compartilham o barramento com outros módulos, acessos são realizados em um único ciclo de relógio. Os processadores interagem com a FIFO utilizando interfaces de instrução customizada (FIFO MST e FIFO SLV), evitando acessos ao barramento do sistema. Do ponto de vista do software, acessos ao FIFO são realizados em dois ciclos de relógio, por meio de instruções bloqueantes simples do tipo *put()* e *get()*.

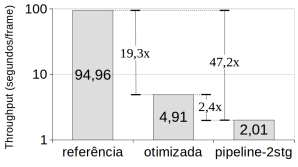
A aplicação é balanceada entre P1 e P2 considerando os dados de profiling da Tabela 1. Os melhores resultados são obtidos mapeando-se a normalização L2-*Hys* e a classificação SVM em P2. P1 realiza os estágios de processamento desde a correção gamma até a interpolação bilinear, enviando os histograma de blocos parciais para o FIFO (com capacidade de 4KB). As aplicações em P1 e P2 implementam o buffer de blocos para reuso dos dados do histograma de blocos. A aplicação sendo executada no sistema multicore é denominada *pipeline-2stg*.

6. Resultados Experimentais

As versões de referência e otimizada são avaliadas nas duas plataformas embarcadas, considerando sistemas single-core. Os resultados são apresentados na Figura 3. Percebe-se que o uso do *buffer* de blocos e das demais melhorias resultam em *speedups* de $16\times$ e $19,3\times$ para os processadores ARM e Nios II, respectivamente. Além disso, a influência das funções de processamento de bloco no tempo total de execução é reduzida a 41% no ARM e a 20% no Nios II. O tempo de execução no ARM é reduzido para $0,1s$ por frame, possibilitando o processamento de 10 fps.



(a) Execução no processador ARM (single-core)



(b) Execução nos sistemas single-core e multicore NiosII

Figura 3. Throughputs em termos de seg./frame para os sistemas baseados em (a) ARM e (b) Nios II.

Avalia-se também a execução da versão (*pipeline-2stg*) no sistema multicore da Seção 5. A distribuição da aplicação no *pipeline* de processadores superou em $2.4\times$ a versão otimizada e em $47,2\times$ a versão original. Observa-se que o tempo de execução por frame é reduzido de $4,91s$ para $2,1s$, mais que dobrando o desempenho. Códigos menores divididos entre processadores e menor carga de dados por núcleo de processamento permitem melhor eficiência no uso das memórias locais. Ainda, a utilização de memórias fortemente acopladas também possibilitam acesso mais rápido a dados frequentemente utilizados e melhora o desempenho.

7. Trabalhos Relacionados

Desde a publicação original por [1], vários autores desenvolveram implementações em FPGA do algoritmo HOG (veja, e.g., os trabalhos em [3, 4, 10, 11]). Embora essas arquiteturas alcancem desempenho em tempo real e níveis de consumo de energia adequados para sistemas embarcados, o uso de hardware fixo torna difícil a adaptação do algoritmo a diferentes situações, aplicações, combinações com outras técnicas e exploração do espaço de projeto.

Alguns autores propuseram soluções mais flexíveis, como a arquitetura EFFEX de [5], desenvolvida para acelerar aplicações de extração de *features* como o HOG. A arquitetura utiliza um núcleo de processamento central mais complexo, capaz de processar tarefas de alto nível, rodeado por vários núcleos periféricos mais simples para operações em vizinhanças de pixels. Os resultados são obtidos mediante simulação, atingindo 10 fps com resolução 1024×768. O trabalho em [6] utiliza processadores soft-core específicos para processamento de imagem baseados em FPGA, chamados de IPPro. Nos experimentos apresentados para os primeiros estágios do algoritmo HOG e um único núcleo IPPro, atinge-se 3 fps na geração de histogramas de células para um *frame* de 1920×1280 pixels. Como comparação, para os mesmos estágios, nossa arquitetura atinge 2 fps utilizando, contudo, um frame VGA.

Neste trabalho, utiliza-se um processador soft-core já bem estabelecido e que permite acelerar o algoritmo HOG, explorando diferentes características arquiteturais. Esse fato representa uma vantagem, considerando-se o conjunto de ferramentas disponíveis ao desenvolvedores. Além disso, a flexibilidade dessa proposta possibilita a exploração de diferentes configurações e variações do algoritmo HOG, assim como sua combinação com outras *features*. A comparação com as implementações customizadas em hardware destacam a lacuna de desempenho e a necessidade de pesquisas adicionais em aceleradores customizados em hardware para melhorar o desempenho, mas preservando os objetivos em relação à solução flexível.

8. Conclusões

Neste artigo, o desempenho do algoritmo HOG foi analisado em duas plaformas embarcadas, uma baseada em FPGA e uma baseada em um processador embarcado de propósito geral. A partir da análise do algoritmo por meio de *profiling*, mostramos algumas transformações de código e exploração de paralelismo para obtenção de maior desempenho. As transformações de código possibilitaram melhorar em $16\times$ vezes o tempo de execução no processador ARM e em $19,3\times$ no processador Nios II, em comparação à implementação de referência. No caso da execução no sistema do processador Nios II, buscou-se ainda explorar a execução do algoritmo em um sistema multicore com pipeline de processadores. A exploração do paralelismo temporal do algoritmo proporcionou um speedup de $47,2\times$ vezes em relação à execução sequencial do algoritmo de referência em um sistema Nios II de um núcleo.

Apesar dos speedups significativos alcançados, o tempo de execução ainda é bastante longo para muitos sistemas que requerem detecção de objetos em tempo real. O trabalho em desenvolvimento está focado no uso de aceleradores em hardware customizados para melhorar o desempenho, na avaliação de escalabilidade e no uso de mais núcleos de processamento no pipeline do sistema multicore. Ainda, planeja-se avaliar outros esquemas de comunicação entre processadores para aumentar o *throughput*.

Agradecimento

Este trabalho foi realizado com o auxílio do Programa de Cooperação Internacional CAPES/PDSE na Universidade do Porto. Processo # BEX 3625/15-0.

Referências

[1] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 886–893, San Diego, CA, USA, June 2005.

[2] Rodrigo Benenson, Mohamed Omran, Jan Hendrik Hosang, and Bernt Schiele. Ten years of pedestrian detection, what have we learned? In *Computer Vision - ECCV 2014 Workshops, Proceedings, Part II*, volume 8926 of *Lecture Notes in Computer Science*, pages 613–627, Zurich, Switzerland, September 2014. Springer.

[3] Xiaoyin Ma, W.A. Najjar, and A.K. Roy-Chowdhury. Evaluation and acceleration of high-throughput fixed-point object detection on fpgas. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(6):1051–1062, June 2015.

[4] S. Advani, Y. Tanabe, K. Irick, J. Sampson, and V. Narayanan. A scalable architecture for multi-class visual object detection. In *25th International Conference on Field Programmable Logic and Applications*, pages 1–8, London, Sept 2015. IEEE.

[5] J. Clemons, A. Jones, R. Perricone, S. Savarese, and T. Austin. Effex: An embedded processor for computer vision based feature extraction. In *48th ACM/EDAC/IEEE Design Automation Conference*, pages 1020–1025, New York, June 2011. IEEE.

[6] C. Kelly, F.M. Siddiqui, B. Bardak, and R. Woods. Histogram of oriented gradients front end processing: An fpga based processor approach. In *IEEE Workshop on Signal Processing Systems*, pages 1–6, Belfast, Oct 2014. IEEE.

[7] Adrian Bradski. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. O'Reilly Media, 1. ed. edition, 2008. Gary Bradski and Adrian Kaehler.

[8] Terasic. *DE2i-150 FPGA System User Manual*. Terasic Technologies Inc., 2013.

[9] Altera Corporation. Nios ii classic processor reference guide, April 2015.

[10] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll. Fpga-based real-time pedestrian detection on high-resolution images. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 629–635, Portland, June 2013. IEEE.

[11] M. Hemmati, M. Biglari-Abhari, S. Berber, and S. Niar. Hog feature extractor hardware accelerator for real-time pedestrian detection. In *17th Euromicro Conference on Digital System Design*, pages 543–550, Verona, Aug 2014. IEEE.

A hardware/software codesign framework for vision-based ADAS

Leandro A. Martinez, José A. M. de Holanda, Eduardo Marques
Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo
Av. Trabalhador São-Carlense, 400 – CEP 13566-590 São Carlos, SP, Brasil
lmartinez@usp.br, arnaldo@icmc.usp.br, emarques@usp.br

***Abstract*— This paper presents a framework for hardware and software co-design to building systems designed to driver assistant using computer vision. This work is part of a doctoral research project nearing completion. To validate the model, a modular pedestrian detection is implemented by comparing the results obtained with other design.**

Keywords—ADAS; Hardware Software Co-design; FPGA; Reconfigurable Architectures;

I. INTRODUCTION

Avoid traffic accidents are still one of the biggest problems facing the world. With the new technologies of embedded systems emerge several proposals that need to be thoroughly validated and tested. However, simulate and test new projects in the various possible environments combining software algorithms with reprogrammable hardware makes the problem complex and multidisciplinary in computing. Thus, to contribute to this area of research, this article presents a framework for hardware and software co-design to build systems designed to support drivers using computer vision.

All processes and components used to build ADAS (Advanced Driver Assistance Systems) must be safe, secure, and reliable. Validation and extensive tests is an enormous undertaking and the most challenging aspect of ADAS development, especially when it happens to the vision system. To test all scenarios and to produce 100% accuracy and zero false positives, under all possible conditions, thousands of hours of video clips must be gathered and run in regression test database. [1,2]

II. VISION FRAMEWORKS

There are many frameworks for video and image processing used in automotive vision systems and in constantly evolving, however, many of these projects have either been closed, some are unsuitable for a real-time view, some lacked a user-friendly, or not adopted a modular design.

The Baselabs [3] provides the infrastructure for the development of rapid prototyping ADAS to develop ADAS allowing collection, recording and reproduction of sensor data and the fusion of complex data sensors; the tool includes a graphical user interface with several components.

The Intempora RTMaps [4] is a framework for prototyping algorithms with a modular environment for testing and evaluating functions based on different sets of sensors with different settings. It also has a component library prepared to develop ADAS. It is extensible allowing the customization of user modules.

The Comemso ADTF [5] is also a framework for image processing systems in the automobile industry. Allows creation of image filters and hardware interfaces connections for embedded systems. It provides tools for testing and simulation.

MontiVision [6] provides a set of tools that allows the development of image processing applications in a customized manner. The development kit includes several image processing modules. The image filters can be defined and customized by the user through a visual interface. The kit also includes a set of sample applications to demonstrate use.

ImprovCV [7] offers an open source modular system for vision data flow processing in software. It enables rapid interactivity with the user for the development of ADAS vision applications.

The NI LabVIEW [8] is a software base of National Instruments design platform for the development of measurement or control systems. Integrating several tools and a development environment aimed at problem solving, accelerated productivity and continuous innovation.

MATLAB HDL Coder and HDL Verifier [9] provides an integrated project environment that accelerates the development of FPGAs and ASIC designs, integrating design tools and IP for FPGA from Xilinx and Altera. This environment provides an extensible programming system as well as a dataflow architecture.

Unfortunately, most these systems are not open source, and a few have hardware/software integrated simulation.

III. SYSTEM DESIGN

Through an open source platform, the Vision-ADAS Framework presents the user with a simple data stream interface (see Figure 1). Using composite nodes could make several combinations of filters in video streams or images.

A. Developing Environment

The platform offers many features for image processing. There is an own library of blocks that can be synthesized directly by the tool for use in FPGA. Even if not all nodes of filters are

available for hardware and embedded software versions, it can be used in simulations and data entry for blocks already available.

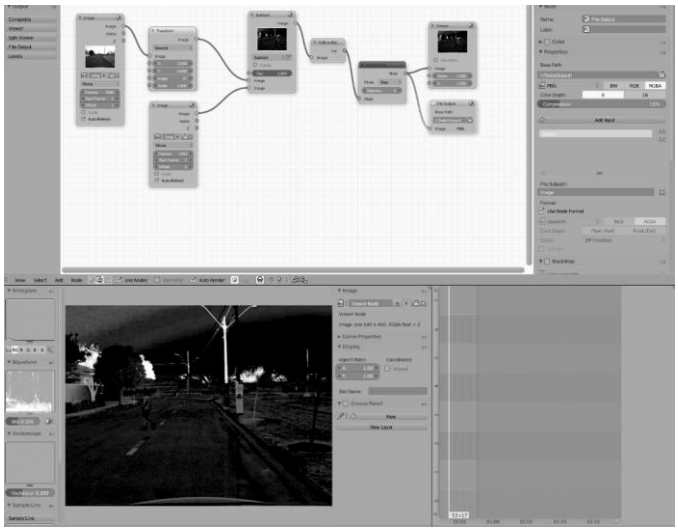


Fig. 1. Main Screen Design Tool

Through IP blocks (Intellectual Property) the user can drag and drop image processing filters for the GUI and view its effect directly on the selected video. Each filter has parameters that can be adjusted in real time to provide immediate feedback. Allowing complete interactive testing of vision systems and simplifies parameter tuning.

B. Simulation

The video input node can be connected allowing direct comparisons of various computer vision algorithms. The application is an open source software and can be extended or adapted to interoperate with other systems.

There are three features that the IP nodes may have:

1) Nodes only for simulation: The simulations can use all kind of nodes, but some are not able to be embedded directly into FPGA. Usually coded for tests and converted later. An example is the OpenCV [10] libraries that can be used. The great advantage of this kind of block is the possibility of using other C-based applications to validate expected results.

2) Nodes ready for embedded processor (SoC): The nodes can have pre-programmed codes to be used in an embedded processor. During the compilation, control subroutines are included with the code to be used by embedded processor.

3) Nodes ready to be synthesized in hardware: With prepared blocks to hardware generation, the system uses a pre-compiled HDL software (Hardware Description Language) to simulate the application. In the project compilation, a script code is created for generation hardware containing the blocks and execution systems controls.

In this environment, the IP creation can first be tested by a software team then converted into a hardware version or even an embedded software. Thus, a software block or a hardware block can use its inputs and outputs to contribute to the construction and validation of its complement. This method allows better cooperation between development teams.

C. Overall System Architecture

The ultimate goal is the design deployment on a hardware architecture and is needed to follow several steps after validation in functional simulation. The Figure 2 shows a full flow using the framework for a hardware validation.

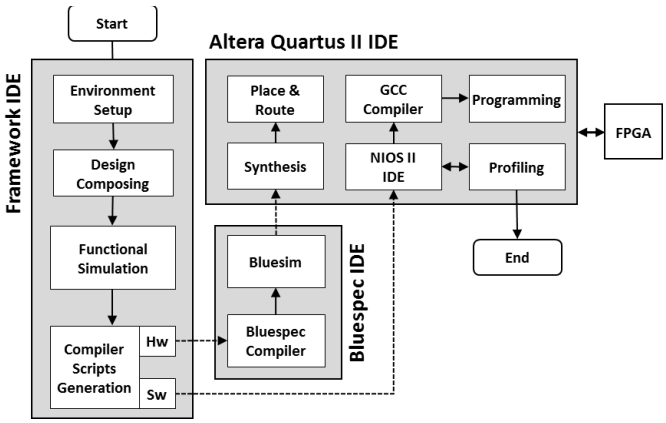


Fig. 2. Full flow using the framework

Environment Configuration: The first step is to select the FPGA board with intended design. With the user design specifications, a template model is selected with a preassembled system and can be changed later. The templates are validated designs containing examples of the use of the tool for various ADAS.

Design Composing: The system provides an IP library where the user can drag and drop to the workspace and link them to other blocks. The IP can be hardware or software, depending on the user's choice and availability in the library. Each IP has a set of parameters that can be modified during the project maintenance.

Functional Simulation: During the drafting of the project, you can do a functional simulation of the design. The system uses pre-compiled routines IPs to make the processed images preview. The user can also switch between sequential images bases for comparison of results.

Framework Compiler: In the compilation, the system checks if all used blocks in the design allow embedded use. In this case, for hardware IPs, the system generates source code scripts in Bluespec language [11]. For the software IPs, the compiler creates source code scripts for NIOS II. These scripts have a standard format for communicating with the internal block execution control.

Send to Hardware: For a whole generation of hardware, the user must compile the project generated by the framework in Bluespec development IDE. If required, other hardware simulations using BlueSim tool can be made. The Bluespec generates files in Verilog format into a standard design. The project needs to be synthesized by the user in Quartus II IDE [12] and sent the FPGA board. Full implementation depends on

the compilation of the project generated within the NIOS II IDE [13] and sends to the embedded processor in the FPGA.

D. IP Construction

To extend the tool with new IP synthesized, developing embedded code structures is required.

For IP software validation, in Figure3, a pre-established pattern of construction must be used. The main file *IP.c* contains the proposed algorithm and should be compatible with the libraries used by NIOS II. For testing, the script *TestBench.c must run*, allowing read the entries of data from a source file *iStream.bin* to be used in the algorithm test and save the output to a verification file: *oStream.out*. Finally, with the system running in the embedded environment, the profiling can be done to extract various features of the proposed algorithm.

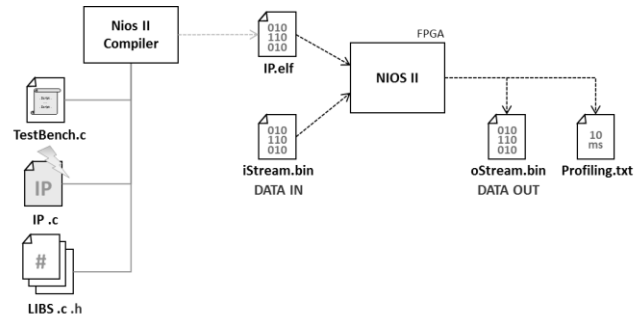


Fig. 3. Embedded software validation of IP

For IP hardware validation in Figure 4, the main file *IP.bsv* contains the proposed algorithm in Bluespec language. If necessary, the language allows using verilog structures using wrapping to this purpose. Compilation generates a verilog file *IP.v* that can be added to a validation template in Quartus II. By profiling techniques, it is possible to extract various characteristics of the hardware generated.

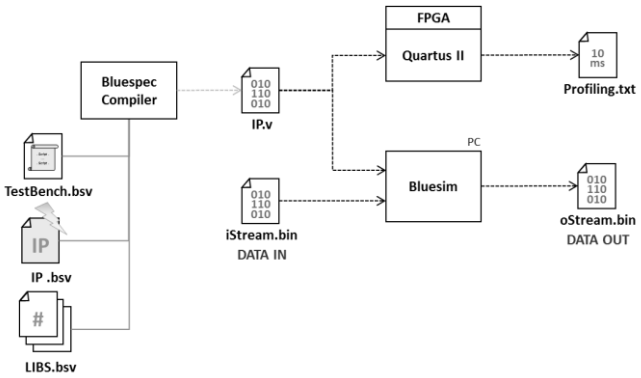


Fig. 4. Embedded hardware validation of IP

The BlueSim allows the validation of the generated hardware through a program written in Bluespec language at *TestBench.bsv*. Just as the blocks are validated software, the data inputs can be read from a source file *iStream.bin* and generates an output file *oStream.out*.

In both cases, we compare the *fStream.bin* files with the output PC software implementation.

IV. Pedestrian Detection

The proposal for the tools validation is a use of a system of pedestrian detection as in [14] in Figure 5, comparing the final results with original work. Then modify the image stabilization system with an another author propose. In the first detection work, the stabilizer is made with a cumulative vector of values pixel and the compares with the same vector of the previous frame. In the second proposal, there is an implementation of optic flow in hardware.

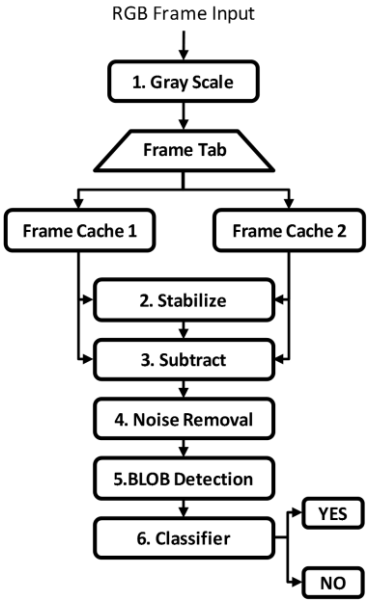


Fig. 5. Proposed model for pedestrian detection

A. Vector Image Stabilization

Differentiation of frames in a video sequence requires that there be alignment between images, so this IP is necessary for the proper functioning of the system. The detection of moving objects in a real environment is a complex task because the background image is constantly changing. Therefore, this module uses a vector size of the image height for storing the sum of each pixel row. In the module to stabilize these vectors are compared and bring results in displacement of the axis Y.

By using only one vector, this implementation becomes very simple to build in a hardware system. There are many approaches to image stabilization. However, many of them depend on many logic elements for its construction.

$$v(h) = \sum_{i=0}^{w} p \qquad (1)$$

In the equation (1), h is each line of the vector; *w* is the frame width, and *p* is a pixel intensity. The vector size has the height of the frame.

Fig. 6. Image stabilization vector

In the construction of this resource, the vector movement was determined by the algorithm histogram of each image line of the current frame compared to the histogram of each row in the table above. This approach proved to be efficient, yet has only been developed to identify vertical displacements, so the system is not prepared to work with the vehicle cornering, only allows the vehicle to move forward.

B. Optical Flow

A proposed solution to solve the alignment problem between frames is to create motion vectors of the image. Thus, to minimize the effect of camera shake the image is divided equally into four regions and calculated on the four vectors representing the motion vector. The main disadvantage of this approach is the computational cost when compared to vector stabilization.

In order to evaluate the image stabilization system, was used as a base, the work of [15] which was based on the [16] architecture, which is built an optical flow system.

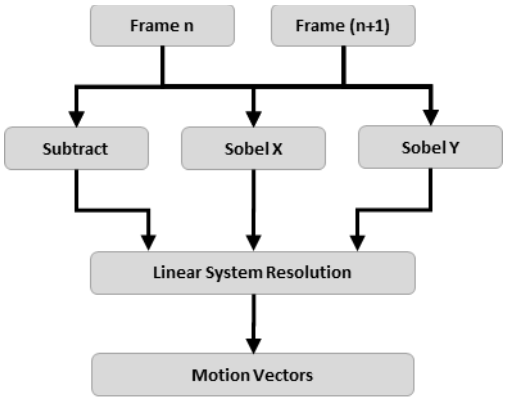


Fig. 7. Proposed Optical Flow

Optical flow algorithms are used to detect the direction and relative magnitude of motion of a scene relative to an observer. In this project, the optical flow will be used for image stabilization, creating a correction azimuth to be used in the subtraction of frames.

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_x \end{bmatrix} \begin{bmatrix} u1 \\ u2 \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \quad (2)$$

The logic for calculating the optical flow estimation is given by a discrete spatial and temporal derivatives. Using convolution with Sobel filter, spatial derivatives are calculated. The temporal derivative is the result of the difference between the two frames. With these data, the system equation (2) is solved using multiple products per pixel and the result is summed over the neighborhood of pixels (3 x 3). As a result generates the optical flow field.

C. Motion Detection Object

A process widely used to detect moving objects is the differentiation of image frames. The algorithm used for this has low complexity. However, as seen in the previous section, this algorithm does not work correctly when the camera is in motion and the alignment of frames from getting too far apart. With the vector representing the vertical camera movement (offset), is possible to compensate the shift for remaining frame alignment.

The equation for frame differencing:

$$d(i,j) = |f_n(i+x, j+y) - f_{n-1}(i,j)| \quad (3)$$

Where x and y are vectors representing movement of the camera. Figure 8 shows the image with the difference in intensity between two consecutive frames.

D. Noise Removal

After the differentiation of subsequent image frames is possible to detect objects that moved in the transition, however, the irregular movement of the camera can generate noise when there is differentiation. To solve this problem, a preprocessing step is necessary to minimize noise regions. Thus, several algorithms for noise reduction can be used, for example, the nonlinear median filter, which smooths the image without lowering the resolution.

Fig. 8. Results of the difference of two frames

The original technique proposed by [17], consists of calculating image histograms, combining them to form the feature vector. First, the image is normalized. Then the

histograms are calculated horizontal and vertical histograms well known and used for recognition.

E. Connected-component labeling

In figure 9, the connected-component labeling algorithm used for the detection of scene objects. This algorithm is used in computer vision to detect connected regions in binary digital images.

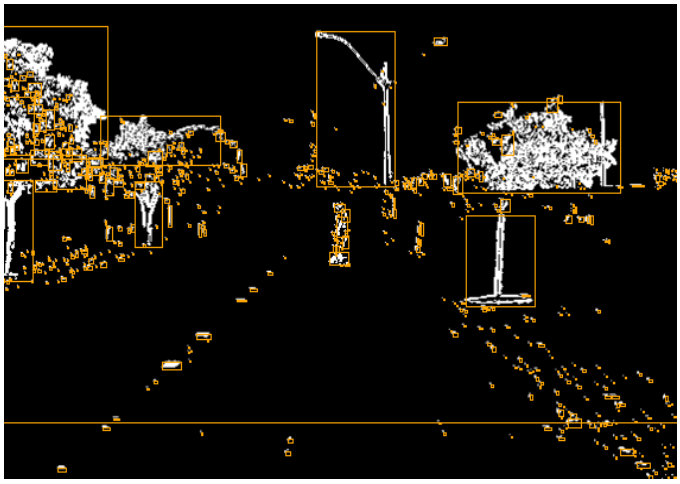


Fig. 9. Qualification of the Region of Interest

F. Classifier

As shown in Figure 10, there is considerable variation in the size of the pedestrian's point of view of the camera. Thus, it becomes necessary to use simple algorithms to extract distances, therefore, to avoid a collision, the pedestrian recognition of a task should be performed in real time [18].

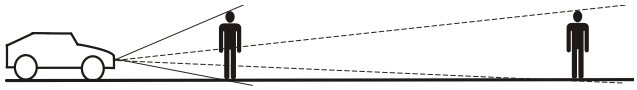


Fig. 10. Size range of pedestrian [14]

With the extraction of features and applying a comparison distance and size of the second generator makes the most accurate information.

G. Detection Results

Some objects have to be manually implemented because not yet available in the tool, such as an SD card reader used by the original design for image reading by NIOS II.

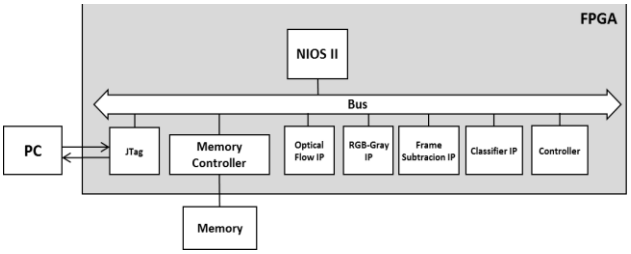


Fig. 11. Schematic model of the generated hardware

Using the same source of image data and the same algorithms, the result of detections were identical to the results of the core work.

During the synthesis, it was necessary to make some timing adjustments.

How expected, the optical flow for the image stabilization has good vantage; Table 1 shows the comparison between image stabilizers:

TABLE I. DETECTION WITH DIFFERENT STABILIZATIONS

Stabilization	Hits	False Positive	False Negative
N/A	40%	20%	40%
Vector	83%	9%	6%
Optical Flow	91%	2%	7%

[a.] Images without any stabilization.

H. Hardware and Software Development Platform

The platform for the generated hardware validation is the Terasic FPGA board DE2i-150 [19]. The specifications are:

- Cyclone IV EP4CGX150DF31 device
- 149,760 LEs
- 720 M9K memory blocks
- 6,480 Kbits embedded memory
- 8 PLLs
- 128MB (32Mx32bit) SDRAM
- 4MB (1Mx32) SSRAM
- 64MB (4Mx16) Flash with 16-bit mode
- Three 50MHz oscillator clock inputs

The main window of the application is based on Blender Software [20].

V. FUTURE WORKS

There is still much work to do, the tool still needs many resources to be easily used. Such as:

- Construction of an IP validation system: it facilitates the creation of new blocks.
- Tests with different FPGA boards, exploring its resources.
- Construction of various IPs, mainly by converting the existing algorithms for embedded use.
- Construction of several templates to facilitate learning with the tool.
- Create a repository for the results compared with other works.
- Create components that can be tested by the tool by communicating directly with the embedded hardware.

VI. Conclusion

We have presented a vision-ADAS Framework for hardware/software co-design, an open source component-based automotive vision application. The main application advantages have been demonstrated through a use case for the detection of pedestrians. Although still not be a complete tool, the framework presents promising features through validations of some IPs and running some algorithms tests. From the data presented in Table 1, it was proved that in the proposed detection scheme, the image stabilization using the optical flow is critical. It may also be noted that the vector for stabilization algorithm, despite its simplicity, gives good results compared to results without image stabilization. In this work, using components from other projects and with the same data source samples were obtained the same original results, thus showing, that is possible to migrate external projects to the tool without creating interference in the results and allowing design space exploration.

Acknowledgment

The authors would like to thank CAPES and ICMC-USP for the financial support given to this research project. Additionally, the authors are also grateful to the reviewers for their significant contribution to improving the paper clarity and for future work suggestions.

References

[1] Kisačanin, B., Gelautz, M. (eds), Advances in Embedded Computer Vision, Springer, 2014.

[2] K. Bengler, K. Dietmayer, B. Färber, M. Maurer, C. Stiller, and H. Winner, “Three Decades of Driver Assistance Systems: Review and Future Perspectives,” IEEE Intell. Transp. Syst. Mag., vol. 6, no. 4, pp. 6 – 22, 2014

[3] Baselabs Flexible ADAS Development web site. [Online] Accessed April 2016. http://baselabs.de/flexible-adas-development-kopie.html

[4] Intempora web site. [Online] Accessed April 2016. http://www.intempora.com

[5] Comemso web site. [Online] Accessed April 2016. http://www.comemso.com/index.php/software

[6] MontiVision Development Kit (SDK) web site. [Online] Accessed April 2016. http://www.montivision.com

[7] Boeing, A.; Braunl, T. "ImprovCV: Open component based automotive vision," Intelligent Vehicles Symposium, 2008 IEEE , vol., no., p.297,302,4-6 Jun 2008

[8] MATLAB LabVIEW web site. [Online] Accessed April, 2016. http://www.ni.com/labview

[9] MATLAB HDL Coder web site. [Online] Accessed April, 2016. http://www.mathworks.com/fpga-design/hardware-software-codesign.html

[10] OpenCV web site. [Online] Accessed April 2016. http://www.opencv.org

[11] Bluespec SystemVerilog User Guide, web site. [Online] Accessed April 2016. http://wiki.bluespec.com/Home/BSV-Documentation

[12] Introduction to the Quartus II Software. web site. [Online] Accessed April 2016. http://www.altera.com/literature/manual/intro_to_quartus2.pdf

[13] Altera NIOS II Hardware Tutorial web site. [Online] Accessed April 2016. http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf

[14] Martinez, L. A.. Projeto de um sistema embarcado de predição de colisão a pedestres baseado em computação reconfigurável, 2011. Master degree work. [Online] Accessed April 2016. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-27022012-110356/pt-br.php

[15] Lobo, T. M.. Co-projeto hardware/software para cálculo de fluxo ótico, 2009. Master degree work. [Online] Accessed April 2016. http://www.teses.usp.br/teses/disponiveis/55/55134/tde-28082013-094816/en.php

[16] MIT 6.375 web site. [Online] Accessed April 2016, http://csg.csail.mit.edu/6.375/6375_2011.

[17] Jalalian, A. , Fathy, M. Pedestrian Detection from a Moving Camera with an Advanced Camera-Motion Estimator. Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, 2008.

[18] Gavrila D., Munder S. “Multi-cue Pedestrian Detection and Tracking from a Moving Vehicle” International Journal of Computer Vision Springer Science, 2006.

[19] Terasic DE2i-150 FGA Development Kit. web site. [Online] Accessed April 2016. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=11&No=529&PartNo=1

[20] Blender website. [Online]. Acessed April, 2016. https://www.blender.org/

Image Fusion in FPGA Using Xilinx Design Tools

João P. Malés
Computational Intelligence Group of CTS/UNINOVA
Campus FCT-UNL, Caparica, 2829-516, Portugal
jpmp@ca3-uninova.org

António J. Falcão
Computational Intelligence Group of CTS/UNINOVA
Campus FCT-UNL, Caparica 2829-516, Portugal
ajf@uninova.pt

Tiago M. A. Santos
Computational Intelligence Group of CTS/UNINOVA
Campus FCT-UNL, Caparica, 2829-516, Portugal
tms@ca3-uninova.org

Rita A. Ribeiro
Computational Intelligence Group of CTS/UNINOVA
Campus FCT-UNL, Caparica, 2829-516, Portugal
rar@uninova.pt

***Abstract* — Field-Programmable Gate Arrays (FPGAs) have been used in many areas of research. As described in this paper, they are used to accelerate an image fusion algorithm, previously coded in ANSI C. The chosen option to convert the algorithm's code into a Hardware Description Language (HDL) is Vivado High-Level Synthesis (HLS), a semi-automatic tool from Xilinx vendor. The adopted hardware was a ZedBoard, which comprises a dual-core ARM Cortex-A9 processor and a Xilinx FPGA equivalent to an Artix-7.**

The first objective of this paper is to analyze the Xilinx bundle and to discuss in detail the tools' workflow in order to provide support to prospective developers.

The second objective is to discuss the obtained results, regarding the area and latency of the FPGA design, and also by comparing them with results of the algorithm's modules compiled in two other architectures.

Keywords — Field-Programmable Gate Array (FPGA); Hardware Description Language (HDL); Image Manipulation and Fusion Algorithm; Vivado High-Level Synthesis (HLS).

I. INTRODUCTION

FPGAs [1][2] are powerful reprogrammable integrated circuits containing grids of logic blocks, where the wires connecting them can be programmed, again and again, changing each time the envisioned function for the set of blocks. The logic blocks can be configured in diverse ways so as to capitalize on parallel processing to increase processing performance. This parallelization characteristic makes FPGAs the perfect candidate for algorithms' optimization and thus they have been widely used in different areas of research e.g. image processing [2], image and video compression [3], neural networks [4], biomolecular simulations [5], etc.

One of the areas where it is natural to implement software into FPGAs is image manipulation and fusion – image fusion is the process of aggregating data from different sources, such as multi-sensor' images, to obtain an aggregated global picture where information on the different inputs is encapsulated [6]; before the fusion of the data, handling and manipulating each inputs' data is necessary. Within these algorithms there is, generally, a repetition of the tasks done to each pixel, thus having routines ideal for possible parallelization. Therefore, the first proposed objective for this paper is to accelerate the processing time of an information fusion algorithm [7][8] by resorting to an FPGA. For this purpose, the hardware used is a ZedBoard [9], which includes two main components: a Processing System (PS) that includes a dual-core ARM Cortex-A9 processor and the Programmable Logic (PL) side that is equivalent to a Xilinx's Artix-7 FPGA. The selected hardware was chosen because of its availability at UNINOVA and because our aim was to demonstrate that FPGA implementations could significantly reduce the processing time of the intrinsic huge amounts of data to be handled by any image fusion algorithm.

For the transformation of the algorithm's modules C code into FPGA, it is necessary to convert the software language to Hardware Description Language (HDL), which can be uploaded into the FPGA. There are two ways to perform this transformation: (a) learn a HDL language and hand-craft the HDL design; (b) utilize one of the available semi-automatic tools to convert C into HDL following some user guidelines [10]. Since the first solution is too demanding, especially for software engineers [2][11] we opted for a semi-automatic tool from Xilinx, the Vivado High-Level Synthesis (HLS) [12]. This constitutes the second proposed objective for the paper: study the existing tools – in more detail Vivado HLS which was the chosen one – in order to check their maturity level and if they already constitute valid alternatives to hand-crafted solutions.

The ZedBoard's design [9] – PS and PL sides included – was created with Xilinx's Design Tools: Vivado HLS, Vivado Design Suite and Software Development Kit (SDK). Each tool of the bundle is analyzed in this article and the work

performed in each of them hereby demonstrated and discussed.

Further, results regarding performance and FPGA's utilized resources are discussed. Results of the implemented FPGA design are compared with the same modules implemented in two different hardware architectures: an Intel Core™ 2 Duo 3.33 GHz and the ARM Cortex-A9 processor of the ZedBoard, working at 667 MHz.

II. Image Fusion Algorithms

In this work we used modules of a general fusion algorithm, called Fuzzy Information Fusion (FIF) [13] – Figure 1 –, specifically the data preparation and data manipulation modules (in FIF the manipulation module is separated into rating process and aggregation process [13]). The implemented code and datasets used for demonstration of the manipulation and fusion of images come from the Intelligent Planetary SIte Selection (IPSIS) algorithm [7][8]. The IPSIS goal is to select, in real-time, an adequate spacecraft landing site, meeting mission, safety and reachability requirements [7][8].

Figure 1 –FIF algorithm schematic.

The objective of our demonstrative modules (data preparation and data manipulation modules of IPSIS [7][13]) is to compute an aggregated single *N*x*M* global hazard map (fusion of images), where the information derives from the input maps (and hence parameters) previously rated. The input is a set of *n* maps (images representing evaluation criteria) with data on different parameters - e.g. slope, shadow, visibility, roughness, etc – and all of them have the same *N*x*M* pixels' size. These *n* maps are collected in different units hence, the data in those maps needs to be converted and prepared in order to be numerically comparable, manipulated and aggregated.

The Data Preparation module comprises [13]: a) normalization taking into account fuzzy membership functions for each input parameter [14]; b) filtering of uncertainty related with the confidence and accuracy of each parameter; c) weighting functions to express the relative importance given by the decision maker to the different parameters. In turn, the Aggregation module [13] is responsible for fusing the normalized, filtered and weighted values of every parameter for each pixel.

Fundamental mathematical operations for each procedure consist of: *sums/subtractions*, *multiplications* and *divisions* for both Data Preparation and Aggregation modules; e*xponentials* only in Data Preparation module; *arc tangents* used within Aggregation module.

The workflow designed for the two modules can be observed in Figure 2, where on top one can see a more detailed flow of Data Preparation and below the Aggregation process.

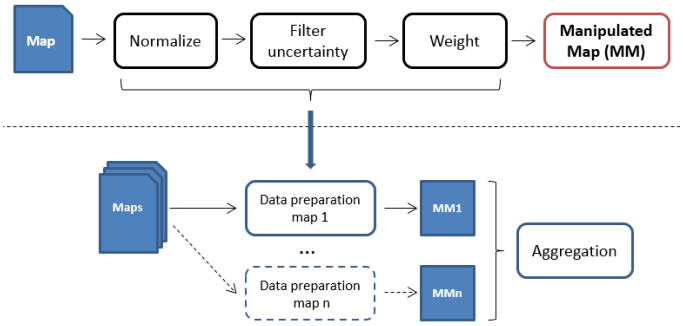


Figure 2 – Workflow for the Data Preparation and Aggregation modules.

In terms of coding structure, it is presented hereafter a high-level representation in pseudocode of how the modules in question are called.

```
CYCLE1: for each value in dimension_M of images do
        CYCLE2: for each value in dimension_N of images do
                CYCLE3: for each map in n input maps do
                        Normalize (current pixel of current map)
                        Filter_uncertainty (normalized value)
                        Weight (filtered value)
                end CYCLE3
                final_result = Aggregate (weighted values)
        end CYCLE2
end CYCLE1
```

Looking at the given pseudocode, one can easily see that the Data Preparation section runs *N*x*M*x*n* times while the aggregation part runs *N*x*M* times. Thus, the image manipulation and fusion processes in question have significant room for parallelization and the transition to FPGAs seemed promising, since their programmable logic is inherently parallel.

III. Hardware Overview and Methodology

A. Hardware

The chosen equipment was a ZedBoard, a board developed by DIGILENT [9] which comprises a dual-core ARM Cortex-A9 processor with a Xilinx FPGA equivalent to the Artix-7 FPGA family and several other components such as OLEDs, 512 MB of DDR3 memory, USB and Ethernet ports, 4 GB SD card slot, etc. As mentioned above, the choice of hardware was due to its availability at UNINOVA and because our focus is on demonstrating that a FPGA implementation of image fusion processes – always a problem when dealing with

huge quantities of data – highly contributes to reduce their processing time.

From this point on, the ARM A9 processor will be referred to as the PS side of the system, while the FPGA components will be referred to as the PL side. In table 1 one can observe the main specifications and components of each side on the ZedBoard [9].

Processing System (PS)		
	Processor Core	Dual ARM Cortex-A9
	Maximum Frequency	Up to 667 MHz
	On-Chip Memory	512 MB DDR3
Programmable Logic (PL)		
	Xilinx 7 Series PL Equivalent	Artix-7 FPGA
	PL Cells	85K Logic Cells[1]
	Look-Up Tables (LUT)	53,200
	Memory LUT	17,400
	Flip-Flops (FF)	106,400
	Extensible RAM (BRAM)	140
	Digital Signal Processing (DSP) Slices	220
	Clock Buffers (BUFG)	32

Table 1 - ZedBoard's components *[9]*.

B. Method

The devised strategy to convert our image manipulation and fusion modules, previously coded in ANSI C language [7][8], to HDL capable of being synthesized[2] into the ZedBoard's PL side, used a semi-automatic tool, the Vivado HLS [12]. The emphasis on "semi" is due to the fact that these tools are incapable of producing solutions that can be exported into FPGAs without user control and specific directives, let alone achieving solutions efficient enough to rival with manually produced HDL solutions.

The reasons for choosing a semi-automatic tool instead of manually generating HDL code were: 1) Even with the associated learning curve characteristic of every Xilinx's software, developing a HLS based solution promises significant shortening of the design cycle compared to a manual HDL model [10][16][17]. 2) When learning how to manually generate HDL code, one has to learn a new language, and not any common programming language such as C. A HDL language paradigm is completely different and software engineers must change their mindset: higher-level computer languages are sequential in nature; HDL languages are not. 3) It was a more natural first step for us, as software engineers, and at the same time a way of checking the current state of the semi-automatic HDL converters, which constitute strong candidates of becoming the future (or maybe the present already) of the HDL programming.

In order to make our input hazard maps available to the PL side of the ZedBoard, routines were developed to load the maps from an SD card into the DDR3 RAM. Also, another routine was created to dump the manipulated maps, as well as the output map, into DDR and later into the SD card. These routines, alongside the high-level ones created for controlling the PL side, via the PS side, are never accounted for during the timings or area estimations and are just hereby referred so that the reader has some knowledge on how the data load and dump to and from the FPGA was achieved.

C. Software

The Xilinx design tools bundle constitute the software used within this article. The bundle's components consist in:

- *Vivado HLS* compiler that enables conversion of C, C++ and System C algorithms into a HDL language, specifically into either Very High Speed Integrated Circuits HDL (VHDL) or Verilog. It is intended for accelerating Intellectual Property (IP) creation and allows the user some level of abstraction while doing so. So as to generate a good and efficient solution capable of competing with a manual HDL design, the developer needs to set various directives for the interfaces, loops and other elements of the algorithm being converted and at the same time he may need to restructure some parts of that same algorithm in conversion. Even so, and taking into account the steep learning curve related with the comprehension of the wide range of options and specifications, the program conversion delivers a solution in much less time than the manual option [18].
  It was with the use of this software that our C coded algorithm's modules were converted into VHDL.
- *Vivado Design Suite* is a program devised for the 7 series FPGAs devices based on the formerly and discontinued Integrated Synthesis Environment (ISE). It groups a wide list of capabilities from which we highlight the ones used in this article: a) design of the model's structure with the necessary IPs and its connections. It was in this phase that the communication between the PS and PL sides was implemented; b) simulator for the designed model; c) synthesizer to generate the bitstream for the PL side; d) integrated logic analyzer to debug the design by monitoring the internal signals; e) area and timing estimation tools; f) built-in strategies for route and place optimizations.
- *SDK* is the Xilinx integrated design environment for creating and debugging embedded applications on any of

[1] The 85K Logic Cells are equivalent to roughly 1.3M ASIC Gates (using the conversion given by Xilinx: 1 Logic Cell = ~15 ASIC Gates) [15].

[2] Synthesis is the process by which a HDL language is translated into a design implementation in terms of logic gates, so that the initial model can be uploaded into and understood by an FPGA. Within the Xilinx FPGAs, this corresponds to converting a HDL design into configuration data, commonly called bitstream, to be loaded into the FPGA.

Xilinx' microprocessors. Its Graphical User Interface (GUI) is based on Eclipse and it includes drivers and libraries for all supported Xilinx hardware IPs. It was within this software that the routines for loading and dumping of the maps from and to the SD card respectively as well as the high-level routines on the PS side were programmed. Also, it allowed the programming of the Vivado's generated bitstream into the FPGA and the connection via a terminal of the ZedBoard and a computer in order to send/receive messages and controls between each other.

The version used for all the referred programs was the 2014.2 [19], whose license is sold alongside the ZedBoard equipment.

IV. Implementation Details

The implementation of the algorithm's modules inside Vivado HLS, required some preliminary work, which can be summarized as follows:

- *Change in structure*: A restructuring of some modules' sub-functions had to be made so that they could be easily optimized.
- *Interfaces*: The modules' interfaces were modified in order to comply with the Advanced eXtensible Interface[3] (AXI4) protocol.
- *Integer sizes*: To save FPGA resources, modifications were made to the size of integer variables in the C modules using the "*ap_cint.h*" library of Vivado HLS for C programming language. This allows the definition of arbitrary precision integer data types. E.g. if there is a variable always positive that occupies a maximum of three bits, it would be wise to define a type "*uint3*" (unsigned integer of three bits) for that variable.
- *Loops' optimization*: In nested loops, the approach that generally obtains the best performance results is to apply directives only to the innermost loops [10]. For that reason only two directives were applied to the modules cycles: directive UNROLL was applied to CYCLE3 – see pseudocode on section II - with a factor equal to *n*, the number of input maps, which basically corresponds to eliminating this loop and instead having *n* simultaneous Data Preparations; directive PIPELINE was applied to CYCLE2 – see pseudocode on section II – with the minimum Initiation Interval[4] (II) possible which guarantees that a next loop iteration can start before the current one is finished.
- *Sub-maps division*: Due to the FPGA's resources limitations, the size of the matrices (images) to be processed needs to be reduced. For example, if we had input hazard maps with size 1024x1024 it would not be possible to process all the pixels at the same time. The decision here was to process the initial maps in blocks of sub-maps (or sub-matrices). Several sub-matrix sizes in the range 32x32 to 1024x1024 were tested and the 256x256 was the one achieving the best trade-off between performance and utilized area. Thus, our inputs maps as well as the output fused and normalized image were processed by the modified modules as chunks of 256x256 pixels.

After applying the referred steps, the modules were converted into IPs of the automatic generated VHDL code to be included in the next Xilinx tool in the workflow, Vivado Design Suite.

Within Vivado, the connections between the different IPs and the PS one were made. Here, it is important to highlight how the transfer of maps' data from/to DDR3 to/from modules' newly generated IPs was made. Notice that those transfer processes already count for the timings' measurement, i.e. in software they correspond to giving arguments to a function as thus the interest is in having transfers' velocities as high as possible.

The IP used to accomplish these transfers was AXI Direct Memory Access (DMA) core [15]. This core is highly customizable and the throughput obtained is dependent of that – following the nomenclature on Xilinx's reference guide, the process by which the maps are passed from DDR3 memory to the IPs is called Memory-Mapped to Stream (MM2S), while the other direction of passing from IP into DDR3 memory is called Stream to Memory-Mapped (S2MM). The best transfer speeds that could be achieved within this process are:

- MM2S: 409 MB/s (Megabytes per second)
- S2MM: 387 MB/s (see note: [5])

Another aspect within Vivado worth noting is that in the PS core, the Acceleration Coherency Port (ACP) was enabled for PS-PL connection. This port is enabled so that the hardware manages the cache coherency. If not, "flushes" and "invalidations" of cache ranges would be necessary after each pointer to memory is assigned in Xilinx SDK, which is runtime consuming.

In the last tool of the workflow, Xilinx SDK, the performed tasks did not have any particularity relevant enough to be mentioned.

V. Results

The round of tests was carried out by taking as input five 512x512 floating-point pixel maps, each occupying 1 MB when loaded into the DDR3 RAM. The first design model was done by analyzing only the Data Preparation process as a kind

[3] AXI is a protocol adopted by Xilinx for its IP cores to standardize the interfaces. AXI4 is the second version of the protocol released in 2010 by Advanced Microcontroller Bus Architecture (AMBA).

[4] Initiation Interval (II) is the number of clock cycles between the start times of consecutive loop iterations.

[5] The explanation for why the S2MM velocity is lower than the MM2S one is not linear and falls outside the scope of this article. For that reason, let us just state that it is related with the Interrupt on Complete (IOC) bit that is set differently on the MM2S and S2MM transfers.

of first step towards the objective, while the second model took into account both the Data Preparation and Aggregation modules (details in Figure 2). Table 2 shows the percentage of utilized PL resources in each round of tests.

	Total available	**Occupied (%)**	
		Data Preparation	*Data Preparation + Aggregation*
FF	106,400	24	34
LUT	53,200	46	68
Memory LUT	17,400	9	10
BRAM	140	4	14
DSP48	220	52	63
BUFG	32	3	3

Table 2 – Percentage of utilized PL resources.

As observed, the percentage of utilized resources for the second model increases in all fields except for the clock buffers (BUFG), which should in fact remain constant since the same clocks were used in both models.

Also, notice the 68% of used LUT and 63% of used DSP48 for the second model. These values are the result of dividing the main input maps into sub-matrices, as referred in section IV. Using the original input size of 512x512 and not the 256x256 chunks, this value would be well over 100%. As explained before, the size of concurrent Data Preparations plus Aggregations instances running in parallel greatly influences the percentage of used LUTs and DSP48.

Table 3 depicts the running times for both models in three different proposed architectures: our FPGA design implemented on the ZedBoard's PL side, an Intel Core™ 2 Duo with 3.33 GHz, and running Linux Mint 17.1 and one core of the ARM Cortex-A9 processors of the ZedBoard's PS side.

For the FPGA design, the timing was calculated using a 100 MHz clock. In the other two architectures the runtime was calculated using the function "*clock_get_time()*" given by the C library "*time.h*" and using the clock *CPU_THREAD_CPUTIME_ID*. This guarantees measuring CPU time, consumed only by the thread in question and not the relative CPU time i.e. it is not subject to any slowness caused by other possible concurrent processor's tasks.

	Running time (milliseconds)	
	Data Preparation	*Data Preparation + Aggregation*
ARM Cortex-A9	514	725
Intel Core™ 2 Duo 3.33 GHz	68	103
FPGA	13	27

Table 3 – Running time for the different architectures.

Note that the runtimes shown in Table 3 are the average for multiple runs of the same instance on each architecture and that in both the ARM Cortex-A9 and the Intel Core™ 2 Duo the modules were compiled using GNU Compiler Collection (GCC) with –O3 optimization[6].

Looking at Table 3, it is clear that the timings for the FPGA design were significantly better than those of other architectures. For the first model, there was a decrease of 80.9% and 97.5% when comparing to the Intel Core™ 2 Duo and to the ARM Cortex-A9 designs, respectively. In turn, the second model showed that the FPGA design had a runtime decrease of 73.8% when comparing to the Intel Core™ 2 Duo results and of 96.3% if compared to the Cortex-A9. Further, the second model FPGA design ran 3.81 and 26.85 times faster than the design in the Intel Core™ 2 Duo and in the Cortex-A9 respectively.

In addition, considering that the ARM Cortex-A9 is running the modules at a frequency of approximately 667 MHz, almost seven times the frequency used in our FPGA design (100 MHz), and its performance is much slower, the benefits of parallelization for performance are obvious.

VI. CONCLUSIONS

Mastering a semi-automatic tool such as Vivado HLS to the point where one generates HDL design solutions for FPGAs capable of rivaling with hand-crafted ones is definitely not an easy job. There is a steep learning curve in order to be proficient in the tool and the tools are still somewhat buggy. However, even with delays caused by the encountered bugs, for a software engineer wanting to venture himself in the FPGA world, the safe and natural first step is the semi-automatic tool.

Based on our experience on Vivado HLS and on our knowledge about other existing tools – complemented by the available research on today's high-level synthesis tools' overviews [21][22] – our opinion, as well as the general consensus, on semi-automatic tools for converting C into HDL is that they still have room for improvement but they already constitute a valid option for software engineers who want to see the benefits of converting some piece of software code into hardware without needing to learn a new HDL language.

This paper demonstrated that even non-experts in FPGAs can, with some work, improve the performance of an image manipulation and fusion algorithm with room for parallelization, when aided by a semi-automatic tool like the Vivado HLS.

The constructed designs resulted in a solution that, in the worst case scenario, displayed 3.81 and 26.85 times better performance for the FPGA than for the Intel Core™ 2 Duo and the ARM Cortex-A9 respectively. The objectives proposed for this paper were therefore met.

[6] GCC has a series of optimizations where –O3 is the most effective one in terms of timing. It inlines some sub-functions and tries to parallelize loops [20].

VII. Future Work

The planned next step for this work is to compare the differences between converting C into HDL with a semi-automatic tool or manually. This will be done using the percentage of utilized FPGA's resources, timing and hours spent on the making of each project. There are already some articles and studies on this subject [18][23] but there is not yet a general consensus on whether the results are always better with the manual approach. Also, as the quality and completeness of semi-automatic tools keep improving, new studies on new tools or new tools' versions may soon appear.

Some effort has already been put into this subject with the construction of the Aggregator module in VHDL. To develop the HDL design, fixed-point operations and CORDIC core [24][25] were used, to obtain trigonometric functions using only additions, shifts and lookup tables. The overall results were promising: the percentage of FPGA's utilized resources was considerably lower on the hand-crafted VHDL solution versus the semi-automatic one. There was a decrease on the percentage of utilized FFs from 6% to 4%, on the LUTs from 14% to 8% and on the DSP48s from 11% to 0% (11% represents only 24 DSP48s); in turn, the performance of the hand-crafted model was 1.19 times better.

Another planned future work would be optimizing even more the designs made with the help of Vivado HLS. This can be achieved since Vivado HLS is a tool with so many features and configurations that one can almost state: augmenting the design time by a certain *k* factor is proportional to having a solution *k*-times better optimized.

Acknowledgments

We want to thank Professor Luís Gomes and the Department of Electrical and Computer Engineering for providing the hardware for this work.

This work was partially funded by FCT Strategic Program UID/EEA/00066/203 of UNINOVA, CTS.

References

[1] J. Hwang, B. Milne, N. Shirazi, and J. D. Stroomer, 'System Level Tools for DSP in FPGAs', in *Field-Programmable Logic and Applications*, G. Brebner and R. Woods, Eds. Springer Berlin Heidelberg, 2001, pp. 534–543.

[2] B. A. Draper, J. R. Beveridge, A. P. W. Bohm, C. Ross, and M. Chawathe, 'Accelerated image processing on FPGAs', *IEEE Trans. Image Process.*, vol. 12, no. 12, pp. 1543–1551, Dec. 2003.

[3] R. W. Hartenstein, J. Becker, R. Kress, H. Reinig, and K. Schmidt, 'Reconfigurable machine for applications in image and video compression', *Proc. SPIE - Int. Soc. Opt. Eng.*, Feb. 1995.

[4] J. G. Eldredge and B. L. Hutchings, 'RRANN: a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs', in *, 1994 IEEE International Conference on Neural Networks, 1994. IEEE World Congress on Computational Intelligence*, 1994, vol. 4, pp. 2097–2102 vol.4.

[5] S. R. Alam, P. K. Agarwal, M. C. Smith, J. S. Vetter, and D. Caliga, 'Using FPGA Devices to Accelerate Biomolecular Simulations', *Computer*, vol. 40, no. 3, pp. 66–73, Mar. 2007.

[6] B. Khaleghi, A. Khamis, F. O. Karray, and S. N. Razavi, 'Multisensor data fusion: A review of the state-of-the-art', *Inf. Fusion*, vol. 14, no. 1, pp. 28–44, Jan. 2013.

[7] C. Boudarias, P. Da-Cunha, R. Drai, L. F. Simões, and R. A. Ribeiro, 'Optimized and flexible multi-criteria decision making for hazard avoidance', *Proc. 33rd Annu. AAS Rocky Mt. Guid. Control Conf. Colo. USA Am. Astronaut. Soc.*, 2010.

[8] L. F. Simões, B. Clément, and R. A. Ribeiro, 'Real-Time Planetary Landing Site Selection – A Non-Exhaustive Approach', *Acta Futura*, vol. 5, pp. 39–52, 2012.

[9] 'ZedBoard'. [Online]. Available: http://zedboard.org/. [Accessed: 15-Jan-2016].

[10] Xilinx, 'Vivado Design Suite User Guide: High-Level Synthesis (UG902- v2014.1)'. 2014.

[11] R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin, C. Kitchen, C. W. Ad, R. Wain, I. Bush, M. Guest, M. Deegan, I. Kozin, and C. Kitchen, 'An overview of FPGAs and FPGA programming; Initial experiences at Daresbury.', 2006.

[12] 'Vivado High-Level Synthesis'. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html. [Accessed: 29-Jan-2016].

[13] R. A. Ribeiro, A. Falcão, A. Mora, and J. M. Fonseca, 'FIF: A fuzzy information fusion algorithm based on multi-criteria decision making', *Knowl.-Based Syst.*, vol. 58, pp. 23–32, 2014.

[14] T. J. Ross, *Fuzzy Logic with Engineering Applications*, 2nd Edition. Wiley, 2004.

[15] Xilinx, 'AXI DMA v7.1: LogiCORE IP Product Guide (PG021)'. 2015.

[16] F. Winterstein, S. Bayliss, and G. A. Constantinides, 'High-level synthesis of dynamic data structures: A case study using Vivado HLS', 2013, pp. 362–365.

[17] D. Navarro, O. Lucia, L. A. Barragan, I. Urriza, and O. Jimenez, 'High-Level Synthesis for Accelerating the FPGA Implementation of Computationally Demanding Control Algorithms for Power Converters', *IEEE Trans. Ind. Inform.*, vol. 9, no. 3, pp. 1371–1379, Aug. 2013.

[18] J. Xu, N. Subramanian, A. Alessio, and S. Hauck, 'Impulse C vs. VHDL for Accelerating Tomographic Reconstruction', in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 171–174.

[19] 'Vivado and SDK Standalone Web Install Client, 2014.2'. [Online]. Available: http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2014-2.html. [Accessed: 29-Jan-2016].

[20] ‘Optimize Options - Using the GNU Compiler Collection (GCC)’. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. [Accessed: 15-Jan-2016].
[21] W. Meeus, K. Van Beeck, T. Goedemé, J. Meel, and D. Stroobandt, ‘An overview of today’s high-level synthesis tools’, *Des. Autom. Embed. Syst.*, vol. 16, no. 3, pp. 31–51, Sep. 2012.
[22] S. Sarkar, S. Dabral, P. K. Tiwari, and R. S. Mitra, ‘Lessons and Experiences with High-Level Synthesis’, *IEEE Des. Test Comput.*, vol. 26, no. 4, pp. 34–45, Jul. 2009.
[23] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers, ‘Optimized Generation of Data-Path from C Codes for FPGAs’, in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, Washington, DC, USA, 2005, pp. 112–117.
[24] R. Andraka, ‘A Survey of CORDIC Algorithms for FPGA Based Computers’, in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, 1998, pp. 191–200.
[25] ‘CORDIC core’. [Online]. Available: http://opencores.org/project,cordic. [Accessed: 15-Jan-2016].

Sessão Regular III

Controlo

Moderação: José Carlos Cardoso
Universidade de Trás-os-Montes e Alto Douro

Implementation and Tuning of PID Controllers Using FPAAs

Paulo J. R. Fonseca[#], Ramiro S. Barbosa[*]
Dept. of Electrical Engineering
GECAD – Knowledge Engineering and Decision
Support Research Center
Institute of Engineering of Porto
Porto, Portugal
[#]*1090038@isep.ipp.pt*, [*]*rsb@isep.ipp.pt*

Abstract

This paper describes the implementation and tuning of PID controllers using FPAA/dpASP devices. The PID controllers are tuned using the well-known Ziegler-Nichols heuristic rules. The effectiveness of the proposed reconfigurable technology is demonstrated by the good accordance between the simulation results and the FPAA/dpASP circuit responses.

1. Introduction

PID control represents about 90% of all industrial control loops. The PID has been thoroughly researched and is a well understood algorithm. We can mention numerous analysis tools, tuning techniques, and simulation tools available to analyze and design PID controllers [1],[2],[3]. So, there are good reasons to look for better design methods or alternative controllers because of the widespread use of these algorithms.

The emergence of dynamically reconfigurable programmable analog circuits has been added to the options available to control engineers. Field Programmable Analog Arrays (FPAAs) and dynamically programmed Analog Signal Processors (dpASPs) can be used for that purpose. These devices can be viewed as the analog equivalent of the well-established FPGAs (Field Programmable Gate Arrays), digital programmable devices. FPAA/dpASP technologies provide an easy way to implement analog circuits that can be reconfigurable by programming tools from manufactures, and in case of dpASPs are able to be dynamically reconfigurable "on the fly". Their use increases the analog design integration and productivity, reducing the development time and facilitating future hardware reconfigurations reducing the costs. These technologies are very recent and are in rapid development to achieve a level of flexibility and integration to penetrate more easily the market. The applications of this technology are wide and include signal conditioning, filtering, data acquisition, and closed-loop control [4],[5],[6],[7],[8],[9],[10],[11], [12],[13].

In this paper we design PID controllers using FPAAs/dpASPs to control a benchmark typical high-order plant transfer function. The plant model is also implemented in a dpASP device. The tuning of the PID controller is made by using the Ziegler-Nichols (Z-N) heuristic rules. The performance of the PID controlled system is assessed both in simulations with MATLAB and with the simulator of the development software tool of the FPAA/dpASP devices.

The paper is organized as follows. Section 2 gives the fundamentals of reconfigurable analog hardware FPAA/dpASP devices. Section 3 presents the PID controller and the Z-N tuning rules. Section 4 shows one application example and comparative results assessing the performance of the FPAA/dpASP technology. Finally, section 5 gives the main conclusions.

2. FPAA/DPASP Technology

In this work we use the Anadigm QuadApex development board from Anadigm manufacturer [14]. It is a platform to get started for implementing and testing analog designs on the AnadigmApex AN231E04 dpASP devices. Furthermore, with its 32 bit PIC32 microcontroller and four dpASP devices, it provides a powerful platform to develop programmable analog designs [15],[16].

The AnadigmApex represents the third generation of FPAA/dpASP devices from Anadigm. Two members of the AnadigmApex family are AN131E04 and AN231E04 (Fig. 1). Both of these devices provide seven analog I/O Cells and four

Configuration Analog Blocks (CABs) (Fig. 2). These structures are constructed from a combination of conventional switched-capacitor (SC) circuit elements and are programmed from off-chip non-volatile memory or by a host processor. Most of analog signal processing occurs within the CABs and is done with fully differential SC circuitry [17],[18],[19]. The device processes analog signals in their I/O Cells and mainly on the CABs that share access to a single Look Up Table (LUT) which offers a method of adjusting any programmable element within the device in response to a signal or time base. The LUT can also be used to implement arbitrary input-to-output transfer functions such as sensor linearization, generate arbitrary signals and construct voltage dependent filtering. Analog signals are routed in and out of the device core via the available I/O cells. The SRAM based dpASP AN23x devices are dynamically reconfigurable and their behavior can be modified partially or completely while operating. Hosted configuration is available with either AN13x or AN23x devices. The real potential of programmable analog however is best leveraged when the host processor is also used to generate and download new configuration data sets on-the-fly as analog signal processing requirements change. This dynamic reconfiguration is only available on AN23x devices. The configuration interface presents itself as either a serial data master or serial data slave. As a serial data master, the FPAA/dpASP can automatically retrieve its configuration data set from any industry standard SPI PROM attached. As a serial data slave, the FPAA/dpASP is compatible with SPI signaling from a host processor and can accept its configuration data from that host [14],[15],[16],[20].

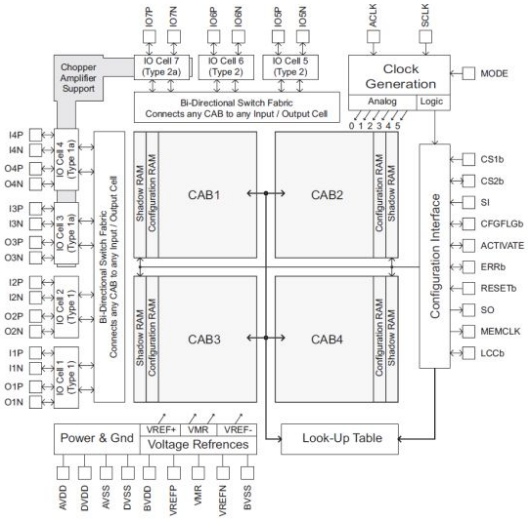


Fig. 1. Schematic of FPAA AN13x and dpASP AN23x.

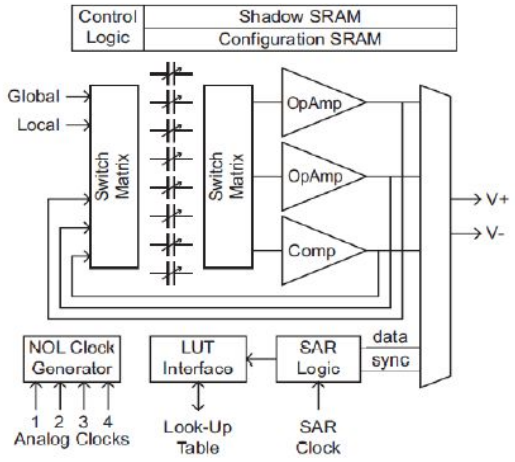


Fig. 2. Block diagram of a CAB.

The software development tool is the AnadigmDesigner®2 [21] (also provided by Anadigm), which makes it possible to design the desired circuit by using pre-defined blocks named Configurable Analog Modules (CAMs) [22]. One of the most important CAMs in our experiments is the bilinear filter CAM. Its transfer function is:

$$\frac{Vout(s)}{Vin(s)} = \pm\frac{2\pi f_0 G}{s+2\pi f_0} \qquad (1)$$

where f_0 is the corner frequency and G the pass-band gain. Figs. 3 and 4 show the circuit of bilinear filter CAM and his configuration window from AnadigmDesigner®2 software, respectively.

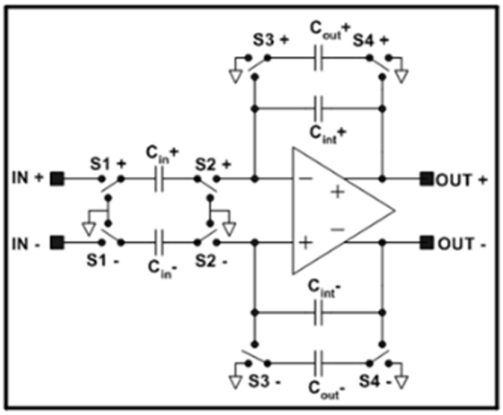


Fig. 3. Scheme of a bilinear filter CAM.

3. PID Controller and Tuning

3.1. dpASP Based PID Controllers

The block diagram of a typical PID controlled system is illustrated in Fig. 5. The transfer function of the PID controller is given by [1],[2]:

$$G_c(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s+1} \qquad (2)$$

where K_p, K_i and K_d are correspondingly the proportional, integral and derivative gains to be tuned. T_f is the time constant of the first-order low-pass filter to limit the action of the differentiator.

As already mentioned, the PID controller is realized on third generation FPAAs/dpASPs using CAM modules available on AnadigmDesigner®2 CAM library. The proportional, integral and differential actions of PID controller are implemented by using the inverting gain, the integrator and the differentiator CAMs, respectively. The derivative bilinear filter CAM uses a corner frequency of 50 Hz while the corner frequency of the bilinear filters CAMs for the error and controller output is 2 kHz. The complete PID controller CAM scheme is reported in Fig. 6. Note that it was used two dpASPs (FPAA1-PID and FPAA2-PID) due to limited resources of a single device to accommodate all necessary CAMs of the PID circuit.

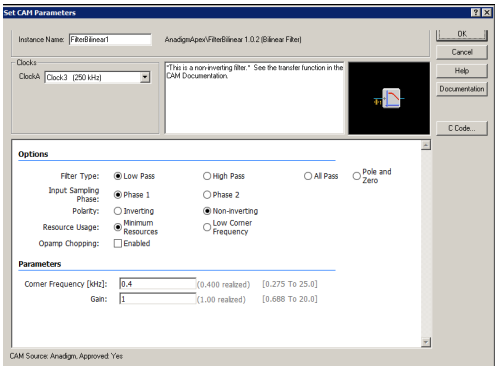


Fig. 4. CAM configuration window.

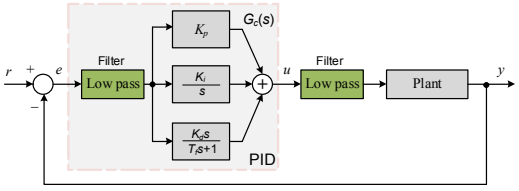


Fig. 5. Feedback control system with PID controller.

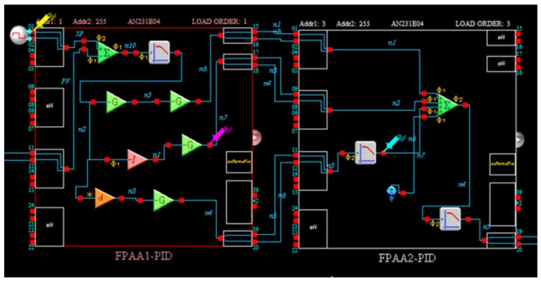


Fig. 6. PID controller implementation using two dpASPs AN231E04.

Furthermore, different applications usually require different ranges to select K_p, K_i and K_d parameters of PID controller. However, the differential time constant from differential CAM and integration time constant from integration CAM is limited to a range selected on AnadigmDesigner®2. The implementation of gain stages on separate P, I and D blocks gives more flexibility on the available settings for the PID gains. Some cautions in terms of noise and harmonics filtering are made, using bilinear CAMs, especially due to noise on output of differentiator and also harmonics on the sum/difference output CAMs. The output of the proportional and integrator blocks should be monitored using probes during simulation as saturation can occur easily after increase of K_p or K_i. The half sum/difference CAM on second dpASP (FPAA2-PID) has the function to sum the proportional, integral and differential actions and a small DC voltage that can be programmed in conjunction in order to obtain very small DC output value that in some circumstances can improve the response of the PID controller. In the third generation of Anadigm FPAA/dpASP, the saturation is internally achieved at amplitude of approximately 3 V and should be considered that VMR level or internal ground signal reference is 1.5 V. These reference voltage levels should be considered during simulation to avoid response errors during implementation of the PID controllers.

3.2. Ziegler-Nichols Tuning

For tuning PID controllers, Ziegler and Nichols suggested rules or heuristic methods based on experimental dynamic responses of the process. Ziegler-Nichols (Z-N) methods are also useful when mathematical models of plants are not known. These methods can, of course, also be applied to the design of systems with known mathematical models. The methods suggest a set of values of K_p, K_i and K_d, that will give a stable operation of the system. However, the resulting system may exhibit a large maximum overshoot in the step response, which may be unacceptable. In such a case it is necessary to perform a series of fine tunings until an acceptable result is obtained. In fact, the Z-N tuning methods provide initial parameter values and it is a starting point for fine tuning, rather than giving the final settings for K_p, K_i and K_d of a PID controller [1],[2],[3].

In the first method, the response of a plant to a unit-step input is obtained experimentally or by simulation. If the plant does not involve integrator(s) or dominant complex-conjugate poles, then the unit-step response curve may look *S*-shaped.

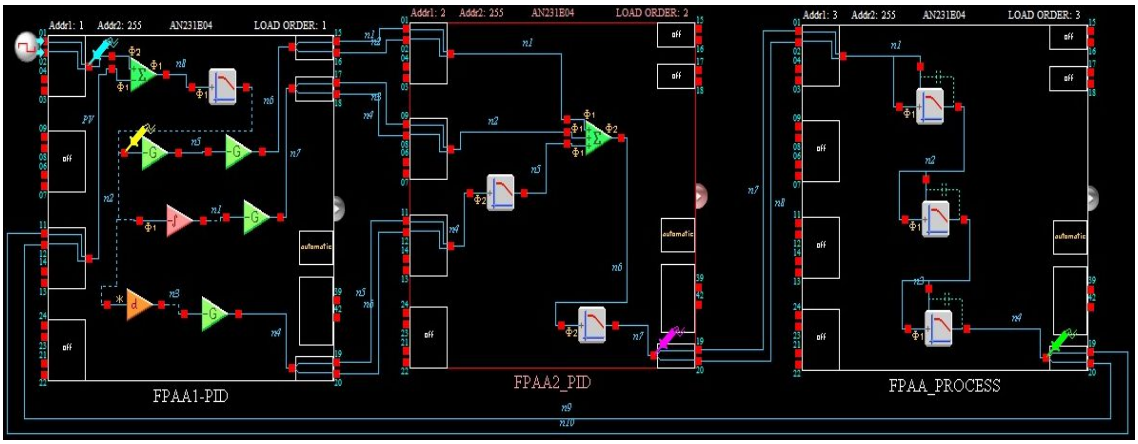


Fig. 7. Closed-loop PID controlled system with third-order plant transfer function.

In this way, the dynamics of the plant can be described by a first-order transfer function with a time delay, $G_P(s) = Ke^{-sL}/(Ts + 1)$, where K is the gain, L the time delay and T the time constant. Ziegler and Nichols suggested to set the values of K_p, K_i and K_d according the controller type and plant parameters (K, L, T) using tabulated formulas. The controller parameters are designed to result in a closed-loop step response transient with approximately a quarter decay ratio [2],[3].

On second method, the ultimate sensitivity method, the criteria for adjusting the parameters are based on evaluating the amplitude and frequency of the oscillations of the system at the limit of stability. To use the method, the proportional gain is increased until the system becomes marginally stable and continuous oscillations just begin, with amplitude limited by the saturation of the actuator. The corresponding gain is defined as K_u (called the ultimate gain) and the period of oscillation is P_u (called the ultimate period). P_u should be measured when the amplitude of oscillation is as small as possible. Then, the tuning parameters K_p, K_i and K_d are selected according controller type and plant parameters (K_u,P_u) using tabulated formulas [2],[3].

4. Application Example

In this section we design PID controllers using FPAA/dsASP devices to control one typical industrial plant transfer function [23]. For that, the adopted procedure is as follows:

- Model a dynamic system by a benchmark plant transfer function on MATLAB and AnadigmDesigner®2. Compare the results of the open-loop step responses with both methods.
- Test the PID controller in closed-loop with plant transfer function on MATLAB and AnadigmDesigner®2 using the Z-N tuning methods.
- Compare the results of the closed-loop step responses using the PID parameters from Z-N tuning rules.

Consider the plant model given by the third-order transfer function:

$$G(s) = \frac{1}{(s+1)^3} \quad (3)$$

Fig. 7 shows the entire PID control circuit using dpASPs. The first two dpASPs (FPAA1_PID and FPAA2_PID) are used to implement the PID controller in parallel form. The plant process $G(s)$ is implemented by the third dpASP (FPAA_PROCESS) using in cascade three bilinear filter CAMs.

Fig. 8 depicts the open-loop step response of $G(s)$ on MATLAB and Anadigm Simulator when applied a step input of amplitude of 1 V. The process parameters are determined as $(K,\ L,\ T) \equiv (1,\ 0.805,\ 3.69)$. Then, the PID gains obtained from Z-N process reaction curve method are $K_p = 1.2T/KL$ =5.50, $K_i = K_p/2L$ =3.42 and K_d=0.5$K_pL = 2.22$. Fig. 9 illustrates the closed-loop step responses on MATLAB and Anadigm Simulator when applied a step input of amplitude of 25 mV. As can be seen, the decay of oscillation is near a quarter amplitude which is in accordance to the Z-N rules. Also, both responses are very similar in terms of steady-state and transient responses. These results validate the plant transfer function model implemented on dpASP.

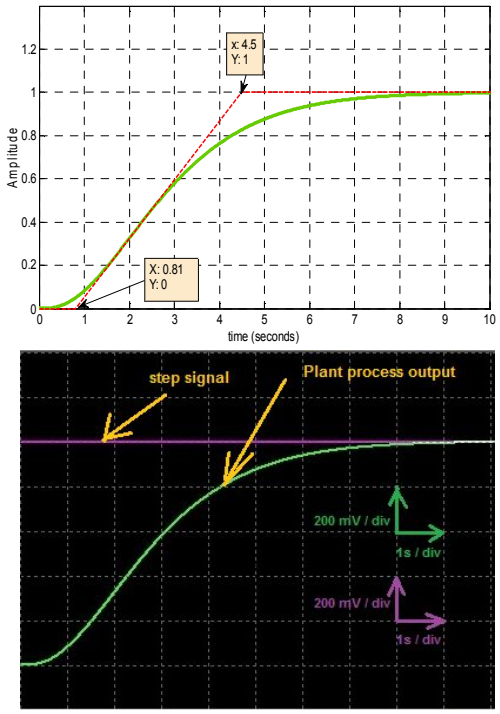


Fig. 8. Open-loop step response of $G(s)$ with MATLAB (up) and Anadigm Simulator (down).

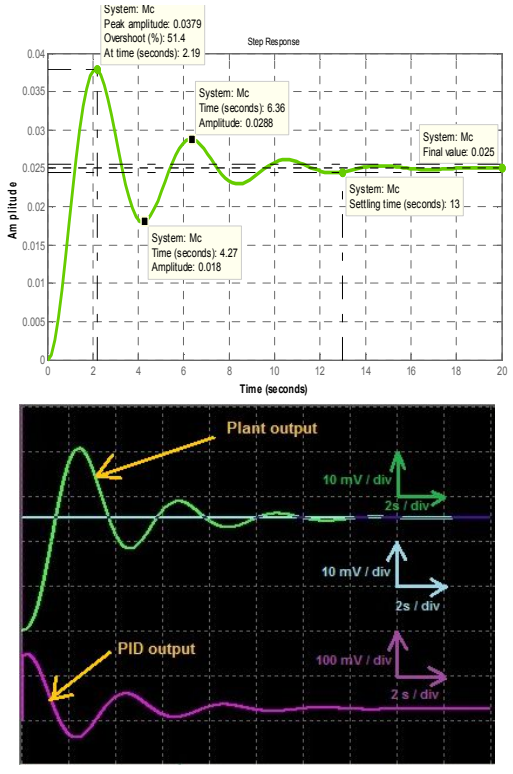


Fig. 9. Closed-loop step response with PID controller tuned according Z-N process reaction curve method in MATLAB (up) and Anadigm Simulator (down).

Next, we apply the Z-N ultimate sensitivity method (or Z-N closed-loop method) as shown in Fig. 10. The corresponding ultimate gain is $K_u = 8$ and the period of oscillation is $P_u = 3.61$ s. Then, the calculated PID gains are K_P=0.6K_u =4.8, $K_i = 2K_p/P_u = 2.66$ and $K_d = K_pP_u/8 = 2.17$. Figs. 11 and 12 illustrate the closed-loop step responses correspondingly on MATLAB and Anadigm Simulator when applied a step input of amplitude of 25 mV. We observe again that the decay ratio of oscillation is near a quarter amplitude which is in accordance with the Z-N rules. Note that the step responses on Anadigm Simulator and MATLAB are very similar assessing the performance of the controller and plant implementations in analog reconfigurable hardware.

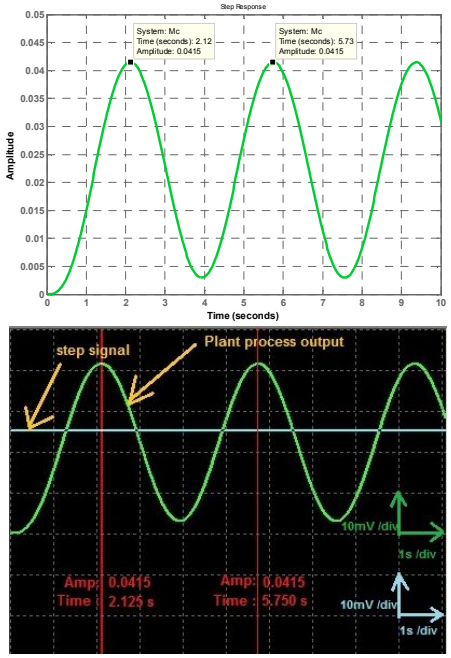


Fig. 10. Ultimate gain and period of oscillation on MATLAB (up) and Anadigm Simulator (down).

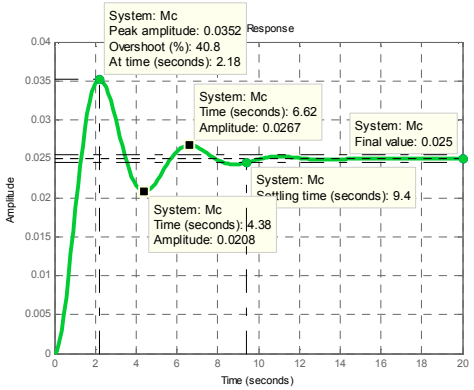


Fig. 11. Closed-loop step response with PID parameters tuned according to Z-N ultimate sensitivity method in MATLAB.

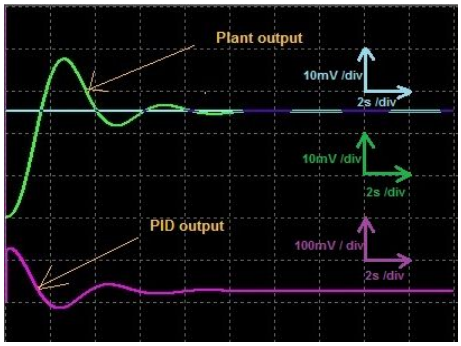


Fig. 12. Closed-loop step response with PID parameters tuned according to Z-N ultimate sensitivity method in Anadigm Simulator.

5. Conclusions

The FPAA is a nice platform to design and implement singled-loop PID controllers. The design of PID controller can be done at block level, simulate and test each circuit in few minutes without need to do complex mathematical calculations, choosing discrete components or focus on analog circuit details. However, cautions in terms of filtering should be considered during design, especially due to noise on the output of differentiator block and also harmonics on the sum/difference output CAMs. Also, the output of proportional and integrator blocks should be monitored using probes during simulation as saturation can occur easily after increase of proportional and integral gains.

It was described one application example of PID tuning controller using the Z-N heuristic rules. The obtained step responses using both methods show the quarter decay amplitude between the first and second oscillation which is in accordance with the Z-N rules. The obtained results on Anadigm simulator are very similar to those of MATLAB, revealing the good performance of FPAA/dpASP devices in implementing PID controlled systems.

References

[1] K.J. Åström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*, Instrument Society of America, North Carolina, 1995.

[2] G.F. Franklin, J.D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, Prentice-Hall, New Jersey, 5th Edition, 2006.

[3] K. Ogata, *Modern Control Engineering*, Prentice-Hall, 4th Edition, 2002.

[4] C. Schene, "Implementing process control with field programmable analog arrays," in *Embed Systems Conference*, San Francisco, 2004.

[5] P. Falkowski and A Malcher, "Dynamically programmable analog arrays in acoustic frequency range signal processing," *Metrology and Measurements Systems*, vol. XVIII, pp. 77-90, 2011.

[6] S. Mahji, V. Kotwal, and U. Mehta, "FPAA-based PI controller for servo position control system," in *IFAC Conference on Advances in PID Control*, Brescia, Italy, March 28-30, 2012.

[7] A. Malcher and P. Falkowski, "Analog reconfigurable circuits," *Intl. Journal of Electronics and Telecommunications,* vol. 60, pp. 15-26, 2014.

[8] W. Zhang and Y. Li, "A self-adapted PID system based on intrinsic evolvable hardware," in *Proc. of the Intl. MultiConf. on Engineers and Computers Scientists*, Hong Kong, March 19-21, 2008.

[9] P. Dong, G.L. Bilbro, and M-.Y Chow, "Implementation of artificial neural network for real time applications using field programmable analog arrays," in *2006 Intl. Joint Conference on Neural Networks*, Vancouver, BC, Canada, July 16-21, 2006.

[10] C.R.. Schlottmann and P.E. Hasler, "A high dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol 1, pp. 403-411, 2011.

[11] C.R. Schlottmann, C. Petre, and P.E. Hasler, "A high-level simulink-based tool for FPAA configuration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, pp. 10-18, 2012.

[12] D.A. Visan, I. Lita, and I.B. Cioc, "Temperature control system based on adaptive PID algorithm implemented in FPAA," in *Proc. of the 2011 34th Intl. Spring Seminar on Electronics Technology (ISSE)*, Tratanska Lomnica, 2011, pp. 501-504.

[13] I. Lita, D.A. Visan, and I.B. Cioc, "FPAA based PID controller with applications in the nuclear domain," *2009 32nd International Spring Seminar on Electronics Technology*, Brno, 2009, pp. 1-4.

[14] Anadigm QuadApex Development Board, UM231004-K001, Anadigm Inc. [Online]. Available: http://www.anadigm.com/

[15] AnadigmApex dpASP Family User Manual, AN13x series, AN23x series, UM000231-U001e, Anadigm Inc. [Online]. Available: http://www.anadigm.com/

[16] AN231E04 Datasheet Rev 1.2, Anadigm Inc. [Online]. Available: http://www.anadigm.com/an231e04.asp

[17] S. Franco, *Design With Operational Amplifiers And Analog Integrated Circuits*, McGraw-Hill Series in Electrical and Computer Engineering, 1988.

[18] L.P. Huelsman, *Active and Passive Analog Filter Design: An Introduction*, McGraw-Hill, 1993.

[19] Theory - Introduction to Switched Capacitor - Anadigm tech. Switched Capacitor: Sampled Data Systems, Anadigm Inc. [Online]. Available: http://www.anadigm.com/_apps/BasicSC-tech.pdf

[20] App Note – 310 Application Note, "Addressing Multiple FPAAs Using a SPI Interface," Anadigm Inc. [Online]. Available: http://www.anadigm.com.

[21] AnadigmDesigner®2 User Manual, Anadigm Inc. [Online]. Availabe: http://www.anadigm.com.

[22] CAM documentation of AnadigmDesigner®2 development tool, Anadigm Inc. [Online]. Available: http://www.anadigm.com.

[23] K.J. Åström and T. Hägglund, "Benchmark systems for PID control," in *Proc. of IFAC Workshop on Digital Control*, Terrassa, Spain, 2000, pp. 165-166.

Control of a Temperature Peltier System with FPAAs

Paulo J. R. Fonseca[#], Ramiro S. Barbosa[*]
Dept. of Electrical Engineering
GECAD – Knowledge Engineering and Decision
Support Research Center
Institute of Engineering of Porto
Porto, Portugal
[#]1090038@isep.ipp.pt, [*]rsb@isep.ipp.pt

Abstract

This paper describes a practical implementation of a PID controlled system using FPAA/dpASP technologies to control the temperature of a closed chamber. The chamber has a cooling element mounted on his top using a Peltier assembly module and on each lateral face are installed silicon flexible heaters for heating. Experimental results of the cooling and heating processes of system are shown to illustrate the effectiveness of the FPAA/dpASP devices in temperature control applications.

1. Introduction

The emergence of dynamically reconfigurable programmable analog circuits has been added to the options available to control designers. Field Programmable Analog Arrays (FPAAs) and dynamically programmed Analog Signal Processors (dpASPs) can be used for that purpose. These devices can be viewed as the analog equivalent of the well-established FPGAs (Field Programmable Gate Arrays), digital programmable devices. FPAAs/dpASPs technologies provide an easy way to implement analog circuits that can be reconfigurable by programming tools from manufactures, and in case of dpASPs are able to be dynamically reconfigurable "on the fly". The use of these devices increases the analog design integration and productivity, reducing the development time and facilitating future hardware reconfigurations reducing the costs. These technologies are very recent and are in rapid development to achieve a level of flexibility and integration to penetrate more easily the market. The applications of this technology are wide and include signal conditioning, filtering, data acquisition, and closed-loop control [1],[2],[3],[4],[5],[6],[7]. Implementation of PID controllers using FPAAs with application in temperature control can be found in related works [8],[9].

In this paper we describe the implementation of a PID controller with dpASPs in a temperature control application. The PID controller is responsible to provide the control voltage to PWM generator, and then the output PWM signal is applied to power drivers of cooling process using an air to air Peltier Thermoelectric Assembly and also for the heating process using two flexible silicone heaters. The PID controller is implemented using two dpASP devices. Another dpASP generates the PWM control signals to the power drivers of thermoelectric assembly module as well to the power driver of silicone flexible heaters. The whole system is based on using analog reconfigurable hardware to control the chamber temperature.

The paper is organized as follows. Section 2 presents the fundamentals of reconfigurable analog hardware (FPAA/dpASP) devices. Section 3 outlines the PID controlled temperature system while section 4 gives the details of the Peltier assembly module. Section 5 describes the hardware used in the control system. The experimental results are given in section 6. Finally, section 7 draws the main conclusions.

2. FPAA/dpASP Technology

The AnadigmApex represents the third generation of FPAA/dpASP devices from Anadigm [10],[11],[12]. Two members of the AnadigmApex family are AN131E04 and AN231E04 (Fig. 1). Both of these devices provide seven analog I/O Cells and four Configurable Analog Blocks (CABs) (Fig. 2). These structures are constructed from a combination of conventional switched-capacitor (SC) circuit elements and are programmed from off-chip non-volatile memory or by a host processor. Most of analog signal processing occurs within the CABs and is done with fully differential SC circuitry

 978-989-704-110-5

[13],[14]. The device processes analog signals in their I/O Cells and mainly on the CABs that share access to a single Look Up Table (LUT) which offers a method of adjusting any programmable element within the device in response to a signal or time base. The LUT can also be used to implement arbitrary input-to-output transfer functions such as sensor linearization, generate arbitrary signals and construct voltage dependent filtering. Analog signals are routed in and out of the device core via the available I/O cells. The SRAM based dpASP AN23x devices are dynamically reconfigurable and their behavior can be modified partially or completely while operating. The FPAA AN13x family devices are also SRAM based and can be reprogrammed as many times as desired, however the device must always first be reset before get a new configuration data set. Hosted configuration is available with either AN13x or AN23x devices. The real potential of programmable analog however is best leveraged when the host processor is also used to generate and download new configuration data sets on-the-fly as analog signal processing requirements change. This dynamic reconfiguration is only available on AN23x devices. The configuration interface presents itself as either a serial data master or serial data slave. As a serial data master, the FPAA/dpASP can automatically retrieve its configuration data set from any industry standard SPI PROM attached. As a serial data slave, the FPAA/dpASP is compatible with SPI signaling from a host processor and can accept its configuration data from that host [11], [12], [15].

In this work we use the Anadigm QuadApex development board [10]. It is a platform to get started for implementing and testing analog designs on the AnadigmApex AN231E04 dpASP devices. Furthermore, with its 32 bit PIC32 microcontroller and four dpASP devices, it provides a powerful platform to develop programmable analog designs [11],[12].

The Anadigm software tool AnadigmDesigner®2 [16] makes it possible to design the desired circuit by using pre-defined blocks named Configurable Analog Modules (CAMs) [17], each of which can be used to implement a range of analog functions. One of the most important CAMs in our experiments is the bilinear filter CAM that creates a single pole, in a low-pass configuration. Its transfer function is:

$$\frac{Vout(s)}{Vin(s)} = \pm\frac{2\pi f_0 G}{s+2\pi f_0} \qquad (1)$$

where f_0 is the corner frequency and G the pass-band gain. Fig. 3 shows the circuit of a bilinear filter CAM.

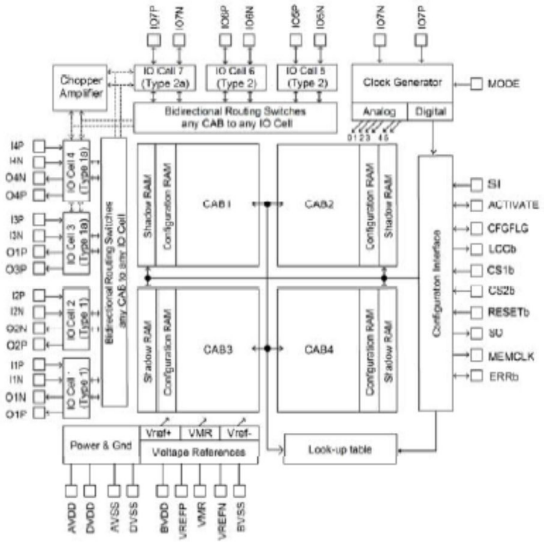


Fig. 1. Schematic of FPAA AN13x and dpASP AN23x.

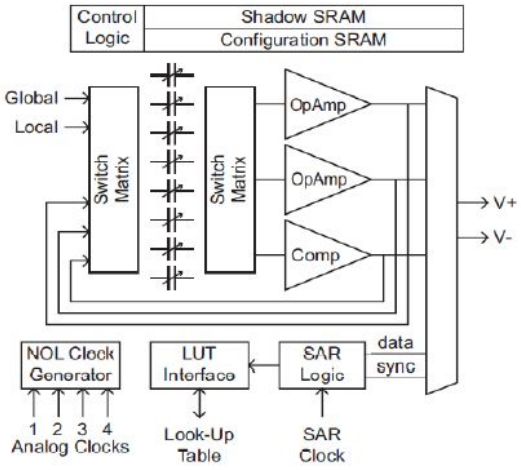


Fig. 2. Block diagram of a CAB.

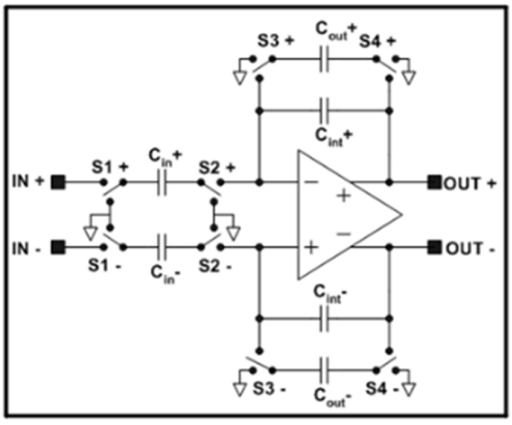


Fig. 3. Scheme of a bilinear filter CAM.

3. PID Controller on FPAA/dpASP Devices

The block diagram of the PID controlled system used in this study is illustrated in Fig. 4. The transfer function of a practical PID controller is given by [18],[19]:

$$G_c(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s+1} \qquad (2)$$

where K_p, K_i and K_d are correspondingly the proportional, integral and derivative gains to be tuned. T_f is the filter time constant of the derivative term.

The PID controller is implemented on third generation of FPAA/dpASP technology using CAM modules available on AnadigmDesigner®2 CAM library. The proportional, integral and differential actions of the PID controller are implemented by using the inverting gain, the integrator and the differentiator CAMs, respectively. The derivative bilinear filter CAM uses a corner frequency of 50 Hz while the corner frequency of the bilinear filters CAMs for the error and controller output is 2 kHz. The complete PID controller CAM scheme is reported in Fig. 5. Note that it was used two dpASPs (FPAA1-PID and FPAA2-PID) due to limited resources of a single device to accommodate all necessary CAMs of the PID analog circuit.

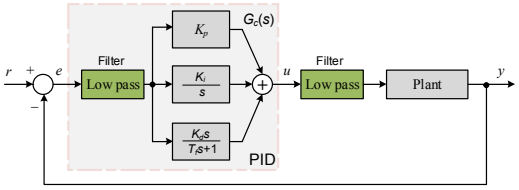


Fig. 4. Feedback control system with PID controller.

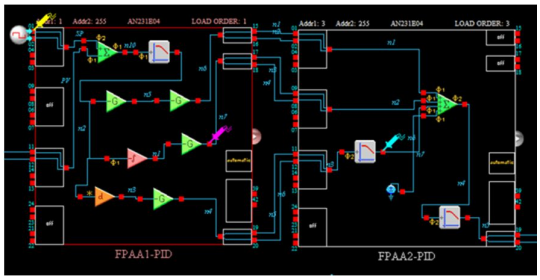


Fig. 5. PID controller implementation using two dpASPs AN231E04.

Different applications usually require different ranges to select the K_p, K_i and K_d parameters. However, the differential time constant from differential CAM and integration time constant from integration CAM is limited to a range automatically selected by AnadigmDesigner®2 based on chosen system clock. The implementation of the gain stages on separate P, I and D blocks gives more flexibility on the available settings for the PID gain parameters. Some cautions in terms of harmonics and noise filtering are made, using bilinear CAMs, especially due to noise on the output of differentiator and harmonics on the sum/difference output CAMs. The output of the proportional and integrator blocks should be monitored using probes during simulation as saturation can occur easily after increase of K_p or K_i. The half sum/difference CAM on second dpASP (FPAA2-PID) has the function to sum the proportional, integral and differential actions as well a small DC voltage that in some circumstances can improve the response of the PID controller.

4. Thermoelectric Peltier Module

As already mentioned the plant is a hermetic chamber with cooling element mounted on his top using a Peltier effect assembly module and on each internal lateral face a silicone flexible heating element. Due to the fact that the use of Peltier elements has advantage over compressed gas refrigeration when cooling small applications, we will put focus on this device technology [20].

This module is made of semiconductor-based electronic components that functions as a small heat pump. By applying a low voltage DC power source to a thermoelectric cooler (TEC) module, heat will be moved through the module from one side to the other. One module face, therefore, will be cooled while the opposite face simultaneously is heated. It is important to note that this phenomenon may be reversed when polarity of applied DC voltage is changed [21].

On this application the module is used only for cooling as the mechanical and thermoelectric structures are designed only for that purpose. Fig. 6 illustrates a Peltier TEC with heat and cold side and the Peltier assembly module used in this application [22],[23].

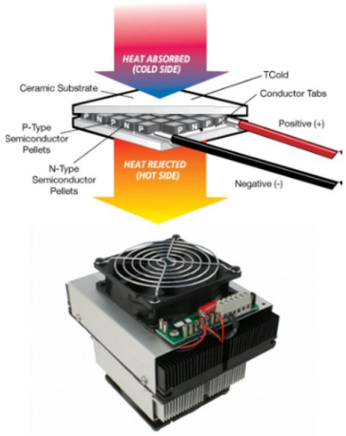


Fig. 6. Peltier TEC with heat and cold side (up) and the Peltier assembly module used in this application (down).

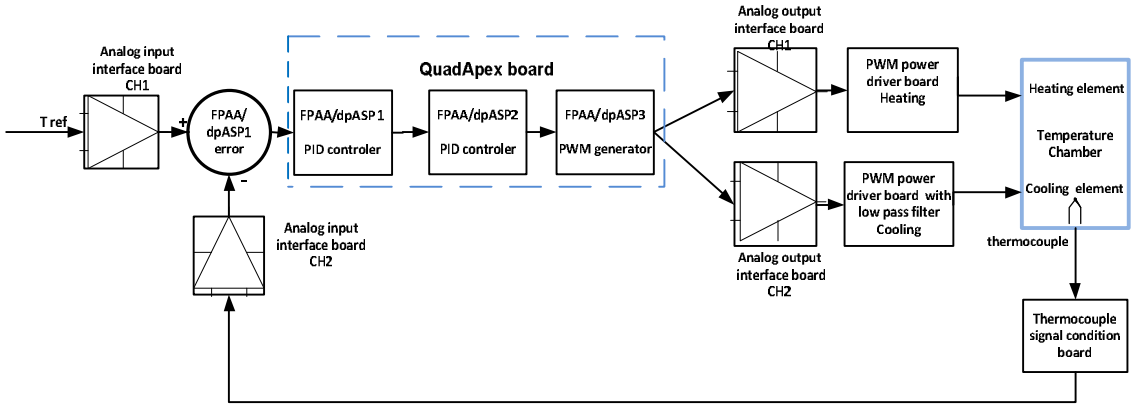


Fig. 7. Block diagram of the closed-loop temperature control system implemented on dpASP hardware.

5. FPAA/dpASP Temperature Control Application

This section describes the hardware of a practical control application that uses a PID controller on dpASP devices to control the temperature of a closed chamber with a thermoelectric Peltier module.

5.1. General Description

Fig. 7 presents the general block diagram of the closed-loop control system implemented on hardware. The temperature chamber is hermetic isolated with all faces constructed in acrylic. Mounted on top, it is installed a thermoelectric assembly element for cooling and on right and left sides are installed silicon flexible heaters for heating. On front side is inserted a thermocouple to measure the actual value of temperature. Setting the reference temperature, the error is the difference between the measured value of temperature chamber and the reference temperature. This error is applied to the PID controller which output voltage control signal to the PWM generator. These parts are implemented on three FPAA/dpASPs. The output PWM generator drives the PWM power driver for the heaters or in case of cooling, drives the PWM generator for the Peltier thermoelectric assembly through a LC biquadratic low pass filter. The chamber temperature is measured by a thermocouple type K that is connected to a specialized signal conditioning board. The analog interface for the FPAAs/dpASPs is performed by analog input interface board and output interface board with optical transceiver isolation.

5.2. Temperature Signal Conditioning Board

A thermocouple is positioned inside temperature chamber to measure temperature. This thermocouple is connected to an analog output K-Type Thermocouple Amplifier board that uses IC AD8495 for the signal conditioning. This circuit is shown in Fig. 8. The output analog voltage from thermocouple amplifier is then connected to channel two of the input interface board as illustrated in Fig. 9. This block gives a linear output voltage as function of the measured temperature, $Vout = 0.05T(°C) + 1.25$.

The output from input interface board is a differential signal with appropriated amplitude to connect to input channel of dpASP/FPAA. However, the thermocouple board output voltage signal is half attenuated on input interface board to avoid saturation on highest temperatures and added VMR of 1.5 V that corresponds to the internal reference voltage of dpASP. The desired reference temperature is selected by a precision potentiometer connected to channel one of the analog input interface board (Fig. 9).

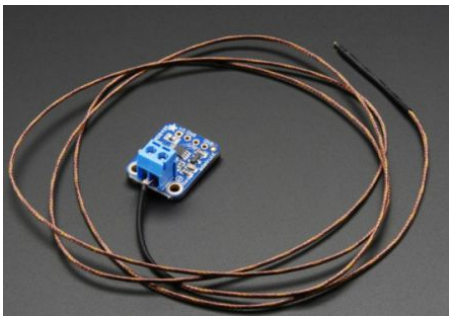


Fig. 8. Analog output K-Type Thermocouple Amplifier board with thermocouple [24].

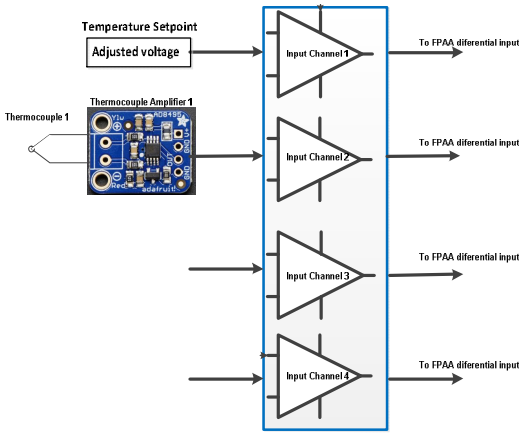


Fig. 9. Temperature sensing block diagram.

5.3. FPAA/dpASP Control Circuit and PWM Power Drivers

The PID with PWM generator is implemented on Anadigm QuadApex board which has four AN231E04 dpASPs. The PID controller is the main component of the control system and is based on design according to Fig. 4. It is implemented a PWM module on a third FPAA connected in cascade that receives the voltage output signal from PID controller and generates the correspondent PWM signal. The duty cycle is proportional to the input voltage level received from PID controller. Then, PWM generator will send the signal for each PWM power driver board which has a H-bridge power circuit able to provide PWM power signal to the silicon flexible heaters [25] in case of heating process or in case of cooling process give the PWM power signal to the LC power filter of the Peltier thermoelectric assembly module.

The LC filter converts the square wave power signal coming from the PWM driver into a low ripple DC (zero frequency) signal. As the square wave is composed of an infinite sum of harmonics (*i.e.*, its Fourier series), then the implementation of LC filters will remove all the harmonics except the DC component, being the amplitude of the DC component directly proportional to the duty cycle of the PWM signal [26]. These filters are required, because the thermal stress generated in the Peltier material when applying fast transients can shorten its life. The manufacturer recommends that Peltier element should not have more than 10% of ripple. Figs. 10 and 11 show the block diagrams of the Peltier and heater power drives, respectively. The POLOLU-HP Motor driver (18v15) 15 A [27] is used for both Peltier and heater power drivers.

The complete circuits of the PID and PWM generator on dpASPs for the heating and cooling processes are shown in Fig. 12 and Fig. 13, respectively. Note that the connections of the error amplifier are inverted in both designs.

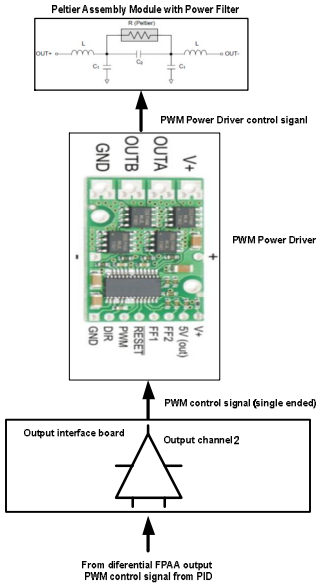


Fig. 10. Thermoelectric assembly power driver block diagram.

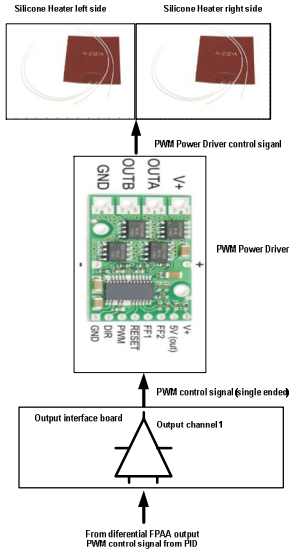


Fig. 11. Heater power driver block diagram.

6. Experimental Results

The dynamic responses for the heating and cooling temperature processes are obtained by PC connected to a multi-meter with temperature data-logger. It is applied a step of 24 V in both cases. Figs. 14 and 15 show the setups and experimental results of both experiments. As can be seen, the process responses are very slow showing nonlinearities, as expected in thermal processes.

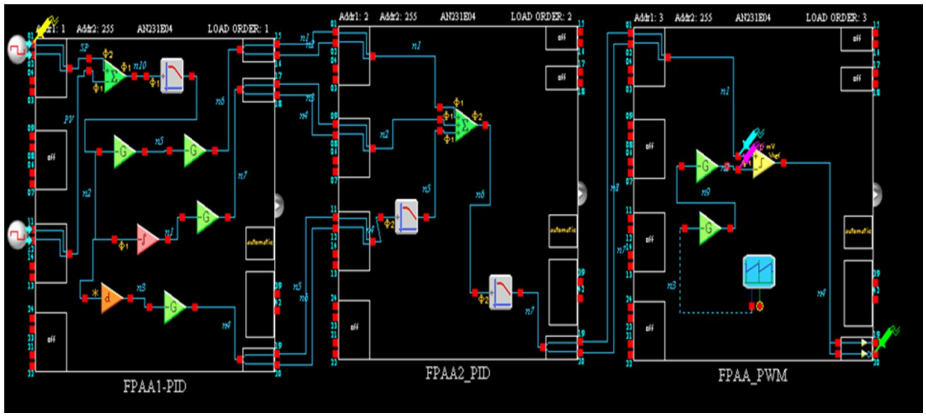


Fig. 12. PID controller with PWM generator for the heating process.

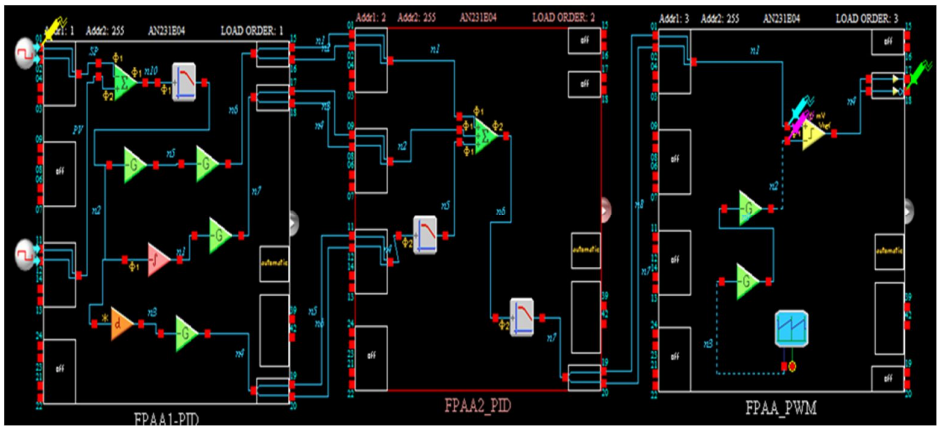


Fig. 13. PID controller with PWM generator for the cooling process.

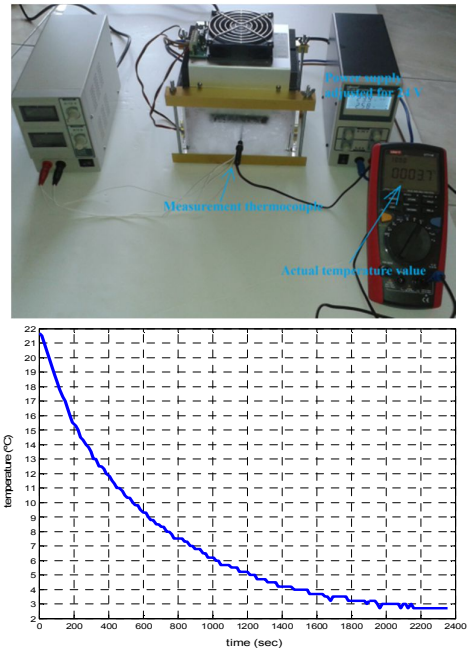


Fig. 14. Setup (up) and experimental response (down) of the cooling process.

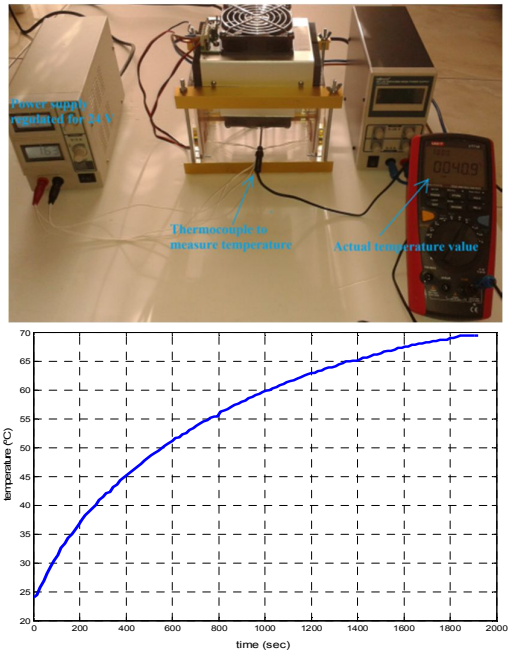


Fig. 15. Setup (up) and experimental response (down) of the heating process.

For the control of the system it was used a PI controller, since the heating and cooling processes are very slow in addition to the existence of nonlinearities and noise. So, the derivative action is omitted from the control. It was applied a simple manual tuning method commonly used in the industry. It can be described in two steps as follows:

- Set and increase the proportional gain until get a small steady-state error;
- Set and increase, if necessary, the integral gain to reduce or eliminate the steady-state error.

Fig. 16 presents the system responses with PI tuning method for different values of K_p and K_i parameters during the cooling process. The PI parameters that give the best response (in terms of rise time and steady-state error) are $K_p = 70$ and $K_i = 20$. Fig. 17 shows the process response for several temperature setpoints during cooling process. Fig. 18 illustrates the system responses with PI tuning method for different values of K_p and K_i parameters during the heating process. The PI parameters that give the best response (in terms of rise time and steady-state error) are $K_p = 30$ and $K_i = 20$. Fig. 19 shows the process response for several temperature setpoints during heating process. We can conclude that the PI controller design used for both heating and cooling processes fulfill the control requirements of stabilizing temperature and transient response behavior [28].

7. Conclusions

The PI controller implemented on dpASPs to control the temperature chamber was tested with success as proved by the response dynamic curves obtained by the data-logger for different temperature setpoints. It was applied a simple manual tuning method but other methods could be also used like heuristic techniques (*e.g.*, Ziegler-Nichols or Cohen-Coon rules) or optimal tuning.

The capabilities of dpASP used in the control application shown flexibility being a fast method to implement controllers. This is due to fact that it is possible to do several simulations with different settings and download "on the fly" new configurations in few milliseconds without making reset or modify the hardware which normally is time consuming. On the other hand, it is still necessary to implement external hardware to make the input and output signal interface to the I/O pins which shown some problems and concerns to avoid noise injection, particularly on the input pins that affects the response of the implemented models.

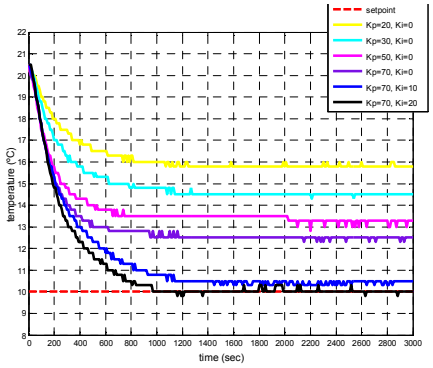


Fig. 16. Dynamic responses of system during tuning method for the cooling process.

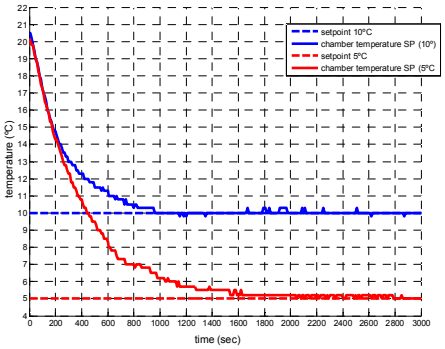


Fig. 17. System responses for different temperature setpoints during cooling process.

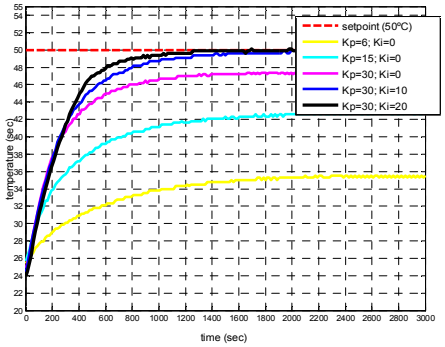


Fig. 18. Dynamic responses of system during tuning method for the heating process.

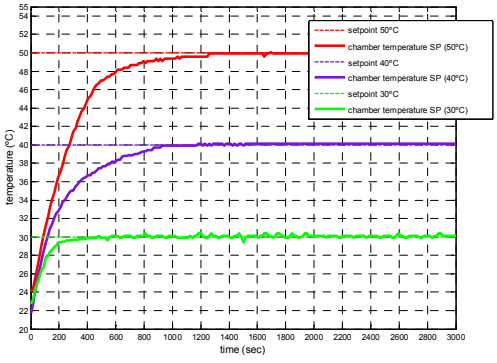


Fig. 19. System responses for different temperature setpoints during heating process.

It is important to note, that it was necessary to do always a previous model analysis using Anadigm simulator to check the possible saturation on the CAMs, especially on the proportional and integral gain blocks, or to check the noise appearance on the differential blocks and harmonics on output of the half cycle sum/diff CAMS. To avoid this noise it was necessary to filter the signals using bilinear transfer functions configured with proper corner frequencies. Also, it was necessary to use small amplitude input signals to avoid saturation as third generation AnadigmApex device supply voltage is 3 V and internal reference ground VMR is 1.5 V.

Future work developments could be done using mixed technology of FPAA and FPGA interconnected by SPI bus or ADC/DAC converters. FPGA can implement the logic I/O control system and the FPAA/dpASP can implement analog signal system, signal processing and signal conditioning. This mixed technology will implement the equivalent system of a reconfigurable SOC. Usage of dpASPs permits the easy replication of controllers on same hardware which is a great advantage where the design of fault tolerant control systems is mandatory due to critical applications, as well in conjunction with FPGAs to generate logic control.

References

[1] C. Schene, "Implementing process control with field programmable analog arrays," in *Embed Systems Conference*, San Francisco, 2004.

[2] P. Falkowski and A Malcher, "Dynamically programmable analog arrays in acoustic frequency range signal processing," *Metrology and Measurements Systems*, vol. XVIII, pp. 77-90, 2011.

[3] S. Mahji, V. Kotwal, and U. Mehta, "FPAA-based PI controller for servo position control system," in *IFAC Conference on Advances in PID Control*, Brescia, Italy, March 28-30, 2012.

[4] A. Malcher and P. Falkowski, "Analog reconfigurable circuits," *Intl. Journal of Electronics and Telecommunications,* vol. 60, pp. 15-26, 2014.

[5] W. Zhang and Y. Li, "A self-adapted PID system based on intrinsic evolvable hardware," in *Proc. of the Intl. MultiConf. on Engineers and Computers Scientists*, Hong Kong, March 19-21, 2008.

[6] P. Dong, G.L. Bilbro, and M-.Y Chow, "Implementation of artificial neural network for real time applications using field programmable analog arrays," in *2006 Intl. Joint Conference on Neural Networks*, Vancouver, BC, Canada, July 16-21, 2006.

[7] C.R.. Schlottmann, and P.E. Hasler, "A high dense, low power, programmable analog vector-matrix multiplier: The FPAA implementation," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol 1, pp. 403-411, 2011.

[8] D.A. Visan, I. Lita, and I.B. Cioc, "Temperature control system based on adaptive PID algorithm implemented in FPAA," in *Proc. of the 2011 34th Intl. Spring Seminar on Electronics Technology (ISSE)*, Tratanska Lomnica, 2011, pp. 501-504.

[9] I. Lita, D.A. Visan, and I.B. Cioc, "FPAA based PID controller with applications in the nuclear domain," *2009 32nd International Spring Seminar on Electronics Technology*, Brno, 2009, pp. 1-4.

[10] Anadigm QuadApex Development Board, UM231004-K001, Anadigm Inc. [Online]. Available: http://www.anadigm.com/

[11] AnadigmApex dpASP Family User Manual, AN13x series, AN23x series, UM000231-U001e, Anadigm Inc. [Online]. Available: http://www.anadigm.com/

[12] AN231E04 Datasheet Rev 1.2, Anadigm Inc. [Online]. Available: http://www.anadigm.com/an231e04.asp

[13] S. Franco, *Design With Operational Amplifiers And Analog Integrated Circuits*, McGraw-Hill Series in Electrical and Computer Engineering, 1988.

[14] L.P. Huelsman, *Active and Passive Analog Filter Design: An Introduction*, McGraw-Hill, 1993.

[15] App Note – 310 Application Note, "Addressing Multiple FPAAs Using a SPI Interface," Anadigm Inc. [Online]. Available: http://www.anadigm.com.

[16] AnadigmDesigner®2 User Manual, Anadigm Inc. [Online]. Availabe: http://www.anadigm.com.

[17] CAM documentation of AnadigmDesigner®2 development tool, Anadigm Inc. [Online]. Available: http://www.anadigm.com.

[18] K.J. Åström and T. Hägglund, *PID Controllers: Theory, Design, and Tuning*, Instrument Society of America, North Carolina, 1995.

[19] G.F. Franklin, J.D. Powell, and A. Emami-Naeini, Feedback Control of Dynamic Systems, Prentice-Hall, New Jersey, 5th Edition, 2006.

[20] Thermoelectric Reference Guide, Ferrotec. [Online]. Available: https://thermal.ferrotec.com/technology/thermoelectric-reference-guide/

[21] Tellurex manuals, Tellurex. [Online]. Available: http://tellurex.com/products/manuals

[22] Thermoelectric assemblies, Laird Technologies. [Online]. Available: http://www.lairdtech.com/product-categories/thermal-management/thermoelectric-assemblies

[23] AA PowerCool Series, AA-040-24-22, Thermoelectric Assembly, Laird Technologies. [Online]. Available: http://www.lairdtech.com/products/AA-024-24-22-00-00

[24] Analog Output K-Type Thermocouple Amplifier - AD8495 Breakout. [Online]. Available: http://www.adafruit.com/products/1778

[25] Etched Foil Silicone heaters. [Online]. Available: http://docs-europe.electrocomponents.com/webdocs/0ebe/0900766b80ebefff.pdf

[26] Texas Instruments (February 2003) Application Report SPRA873 - Thermoelectric Cooler Control Using a TMS320F2812 DSP and a DRV592 Power Amplifier. [Online]. Available: http://www.ti.com/lit/an/spra873/spra873.pdf

[27] Pololu High-Power Motor Driver 18v15. Pololu Electronics. [Online]. Available: https://www.pololu.com/product/755

[28] P.J.R. Fonseca, "Controller implementation using analog reconfigurable hardware (FPAA)," Master thesis, Instituto Superior de Engenharia do Porto (ISEP), Porto, Portugal, November 2015.

Sessão Regular IV

Aplicações de Processamento de Sinal

Moderação: João Agostinho Pavão
Universidade de Trás-os-Montes e Alto Douro

FPGA-Based Dynamic Partial Reconfiguration application in Cognitive Radio Baseband Processing Systems

Mário Lopes Ferreira[†], Amin Barahimi[‡], João Canas Ferreira[*]
[†‡*]*INESC TEC and* [†*]*Faculty of Engineering of the University of Porto*
Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal
mario.l.ferreira@inesctec.pt, amin.barahimi@inesctec.pt, jcf@fe.up.pt

Abstract

Cognitive Radio (CR) is envisioned as a solution for spectrum utilization concerns raised by the rapid development of wireless communications. In turn, the Physical layer of Cognitive Radio devices should be highly flexible and real-time adaptable to support multiple communication standards and provide spectral agility in changing communication environments. Due to this requirements, FPGAs are good candidate platforms to implement baseband processors for Cognitive Radio systems. In particular, this paper focus on the application of Dynamic Partial Reconfiguration (DPR) techniques in FPGA-based hardware infrastructures for CR baseband processing. First, an architecture for a reconfigurable NC-OFDM baseband processor is discussed and proposed. Then, the implementation of a dynamically reconfigurable Fast Fourier Transform (FFT) processor - one of the most important baseband operations in wireless systems - is presented. DPR latency and power consumption overhead were evaluated for the implemented design and the obtained results stimulate the exploitation of DPR techniques to implement flexible and adaptable hardware components for CR devices.

1. Introduction

The sharp growth of wireless communications led to an increasing service variety and high QoS user demands. To follow this trend, the electromagnetic spectrum has to be efficiently used and managed. However, large portions of licensed spectrum are underutilized and cannot be accessed by other users. Quite often, this situation turns spectrum access a more critical problem than spectrum scarcity [1]. Cognitive Radio (CR) [2] has been proposed as a solution to improve spectrum utilization and management by allowing licensed but unoccupied spectrum bands to be used by unlicensed users. Hence, CR is tightly coupled with Dynamic Spectrum Access (DSA) techniques and the concept of spectrum pooling [3] - a mechanism through which spectral resources from different owners are pooled and rented to secondary users during idle periods. During the operation of a CR transceiver, these spectral resources comprise variable frequency bands during variable idle periods. This requires a CR device to be aware of its surroundings and to adapt its internal operation accordingly. Apart from extremely surpass previous cellular communication generations in terms of data rates, network latency, energy consumption and system capacity requirements [4], 5G systems should be capable of supporting multiple wireless technologies and switching between them according to the communication environment, in order to maximize overall performance. Due to this, Badoi et al. [5] suggested that the CR terminal *"becomes the ideal 5G terminal candidate"*. The flexible and adaptable nature of CR carries interesting signal-processing challenges [6] and requires agile Physical layer (PHY) architectures and waveform techniques. Orthogonal Frequency Division Multiplexing (OFDM)-based waveforms are strong candidates for CR PHY implementation [7]. Particularly, a spectrally agile variant of OFDM called Non-Contiguous OFDM (NC-OFDM) shows promising features for opportunistic wireless access [8] and dynamic spectrum aggregation [9]. In the design of such baseband architectures, it would be convenient to bring together the flexibility and programmability of software with the real-time performance of hardware [10]. Offering a balanced trade-off between flexibility, throughput and power consumption, FPGAs are a good platform to implement baseband processing engines for CR devices. It is possible to handle the multiple baseband operation modes in CR devices by adopting a *Velcro* approach, in which different datapaths are implemented on the FPGA logic fabric and the input data is demultiplexed and driven to the desired path. Yet, this strategy makes an inefficient use of hardware resources, as only one datapath is used at a time. Alternatively, SRAM-based FPGAs allow the employment of Dynamic Partial Reconfiguration (DPR) - the reconfiguration of certain portions of the device while the rest of the system continues to operate normally. DPR techniques can provide superior system adaptability and computational specialization to the nearly instantaneous application demands [11], which are desirable features in CR systems. However, the impact of DPR in terms if reconfiguration time and power consumption overhead must be evaluated in the context of the target application.

This paper intends to discuss the application of FPGA-based DPR techniques in Cognitive Radio baseband processors, by presenting a proposal for a reconfigurable NC-OFDM baseband processor architecture and the implementation of a dynamically reconfigurable Fast Fourier Transform (FFT) processor. For the implemented system, re-

 978-989-704-110-5

configuration times and power overhead measurements are presented. The obtained results encourage the use of DPR techniques in hardware infrastructures for CR baseband processing.

The rest of the paper is organized in the following way: Section 2 reviews related work on Radio platforms using DPR techniques; Section 3 proposed an architecture for a reconfigurable NC-OFDM processor; Section 4 presents a dynamically reconfigurable FFT processor implementation and evaluates DPR impact regarding reconfiguration latency and power consumption; conclusions are drawn in Section 5

2. Related Work

The applicability of Dynamic Partial Reconfiguration in Cognitive Radio systems has been explored in several research works. However, most of them draw the attention to specific implementation aspects like reconfigurable synchronizers [12], FFT processors [13], coding or digital modulation schemes [14]. Moreover, little emphasis is put on he implementation dynamic spectrum aggregation techniques.

A Digital Front-End and baseband processor architecture for Software Defined Radio (SDR) transmitters was presented by He et al. [15]. This work identifies common features between several wireless standards and then aims at an efficient hardware resource utilization by applying Dynamic Reconfiguration techniques to change functionality (Digital Modulation scheme, FFT size and cyclic prefix length) and clock frequency at run-time.

Casado et al. [16] proposed an architecture for a reconfigurable cognitive radio providing run-time reconfiguration in the antenna, RF front-end and FPGA framework (IF frequency generation and baseband processing). Although this work focus more on the design of a reconfigurable antenna, it shows another application of DPR in CR applications. Regarding the portion of the system implemented on the the FPGA (Xilinx Virtex-6 board), it comprises baseband processing, a DSA algorithm and a MicroBlaze processor. The baseband processing tasks implemented are QPSK modulation/demodulation, encoding/decoding and Costas loop synchronization (in the receiver). The DSA module implements a Energy Detection algorithm for free frequencies detection and a Cyclostationary Features Detection algorithm to distinguish between primary user from CR signals. In turn, the MicroBlaze processor is responsible for DPR management and generation of control signals for both RF front-end and antenna. DPR is used to reconfigure the digital frequency oscillator that generates IF frequency, allowing small in-band frequency changes. So, the baseband processing itself is static, since it does not support the variation of operation modes and parameters (e.g. modulation scheme). Nevertheless, the authors mentioned that DPR could also be employed in baseband processing tasks. In the other hand, the baseband processing data path has limited functionalities, as it does not support several operations required by the most used wireless standards, such as FFT/IFFT and channel equalization.

Exploring a hybrid FPGA platform (Xilinx Zynq) capabilities, Shreejith et al. [17] presented a dynamic CR design which considers two conceptual planes: data plane and control plane. The data plane is in charge of baseband processing operations. As these tasks are usually quite demanding in terms of performance and computational complexity, they are implemented in the FPGA logic fabric. The control plane is responsible for *cognitive tasks*, that is, either sensing the medium or retrieving and interpreting external information, and triggering the baseband run-time reconfiguration accordingly. To do so, an *observe-decide-act* loop is implemented. Due to the need for high flexibility and ease programmability, the control plane is implemented in the ARM processor embedded on the Zynq board. This approach considers two ways to reconfigure the baseband processor: parametric and partial reconfiguration. Parametric reconfiguration consists of using control signals to vary some operation parameters (e.g. code rate) and is implemented through multiplex-based circuits. This technique is suitable for small changes in the baseband processing and do not physically modify the logic blocks currently in use. When more significant hardware changes are required, partial reconfiguration is employed. The baseband processing data path is implemented in a Reconfigurable Region, so it is possible to perform DPR and replace hardware within that region on-the-fly. To improve system's reactivity and mitigate reconfiguration times, the authors use a custom reconfiguration manager called *ZyCAP* [18]. The baseband reconfiguration delay is 786μs for a throughput of 380 MB/s. In spite of not covering relevant CR features as spectrum agility, this approach is interesting from a system architecture point of view and shows again the applicability of DPR in CR baseband processing. However, the reconfiguration is implemented with a very coarse granularity level, as the entire baseband processor is placed in a single reconfigurable region.

3. Reconfigurable NC-OFDM Baseband Processor Proposed Architecture

We have previously proposed an architecture for a reconfigurable FPGA-based NC-OFDM baseband processor [19]. The novelty of the proposed approach consists in the exploration of innovative NC-OFDM hardware processing architectures using partial run-time reconfiguration of the system and providing on-line capability for designing optimized waveforms employing custom modulations for each NC-OFDM sub-band. The goal is to produce a design meeting the requirements of next generation Cognitive Radio devices in terms of multi-carrier, multi-standard communications and spectral agility in changing environments. So, the flexible NC-OFDM transceiver should provide support for spectrum aggregation and run-time selection of modulation schemes and active sub-carriers. Such a baseband processor also has to respect requirements in terms of robustness, power consumption, throughput, flexibility and adaptability. FPGAs will be the implementation platform adopted, as they provide a hybrid hard-

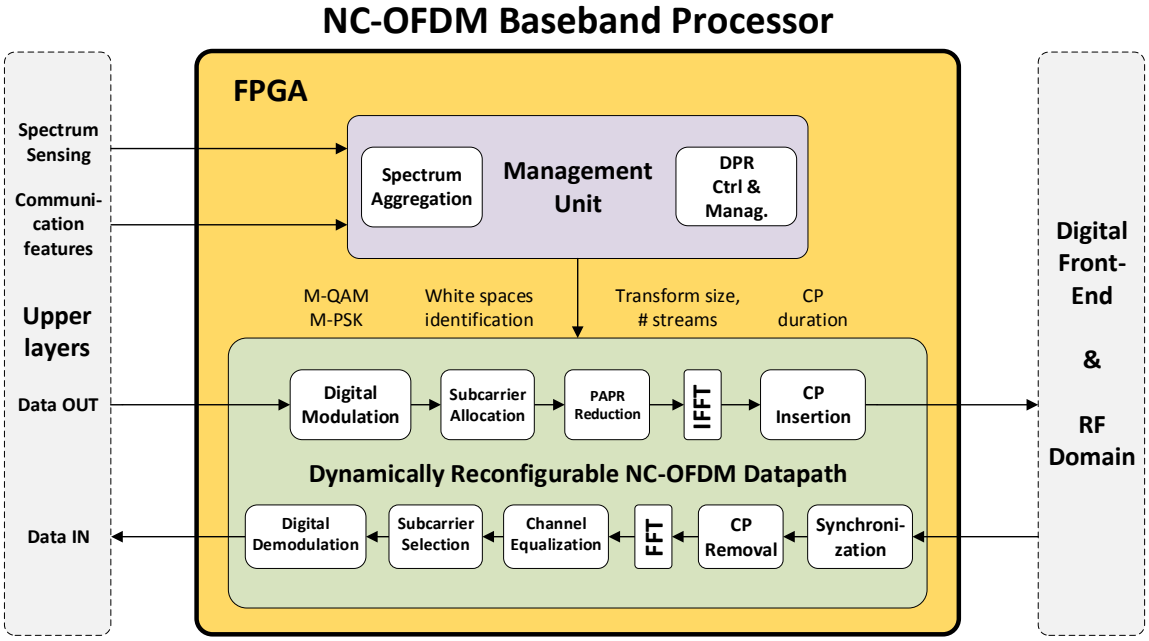


Figure 1. NC-OFDM baseband processor proposed architecture

ware/software framework based on a heterogeneous architecture that combines embedded general-purpose processors with custom parallel hardware accelerators. FPGAs also provide support for dynamic partial reconfiguration (DPR), the ability to dynamically modify portions of logic while the rest of the device continues to operate without interruption. This technique will be applied to the NC-OFDM baseband processor, with the expectation to benefit from its potential for hardware savings and execution overhead reduction, and to achieve higher levels of system adaptability, computation specialization and efficiency. However, employing DPR remains an architectural challenge, as the system needs to maintain operation integrity while being reconfigured. An additional challenge consists in developing approaches to effectively mitigate the latency and energy consumption introduced by the reconfiguration process.

The proposed high-level architecture for the NC-OFDM baseband processor (Figure 1) is composed of two domains: *Dynamically Reconfigurable NC-OFDM Datapath* and *Management Unit*. The Dynamically Reconfigurable NC-OFDM Datapath consists of a modular pipeline of computation blocks for NC-OFDM baseband processing, whose operation can be reconfigured in run-time depending on the protocol or operation mode in use. The choice of appropriate system partitioning and granularity level is crucial for the success of DPR application. Hence, in the context of the NC-OFDM baseband processor, the analysis of datapath modules as well as the algorithms and architectures to implement them is important. Then, based on this analysis, opportunities for DPR should be identified. For instance, *non-contiguous spectrum aggregation* requires a multidimensional PHY layer, even when data aggregation schemes are performed in the MAC layer [20]. A possible approach to implement this multidimensional PHY would be the implementation of a dynamically reconfigurable mosaic of PHY units that could be turned on/off based on the system needs. The resources allocated to unused PHY units could be reused for other system tasks. This strategy raises some relevant challenges related with FPGA resources and power budget management or parallel execution handling of several PHY units in real-time. Another possible DPR use case is the run-time customization of datapath modules architecture to adapt the module according to real-time performance requirements.

The Management Unit implements a reconfiguration framework to intelligently manage the real-time adaptability of the Dynamically Reconfigurable NC-OFDM Datapath, while exploring methods to mitigate the overhead introduced by DPR. This unit should be aware of the communication context by using information (e.g.: spectrum utilization and communication parameters) received from higher protocol layers. It should also keep information about the current hardware configuration and, when the need for reconfiguration is detected, use that information to figure out how different the next baseband configuration will be. Based on that, the processing engine should take a decision on which reconfiguration method is to be employed (partial reconfiguration, difference-based reconfiguration, etc.). Additionally, the Management Unit should be able to identify opportunities to optimize the reconfiguration process through techniques like scheduling, bitstream pre-fetching, compressing [21] and relocation [22].

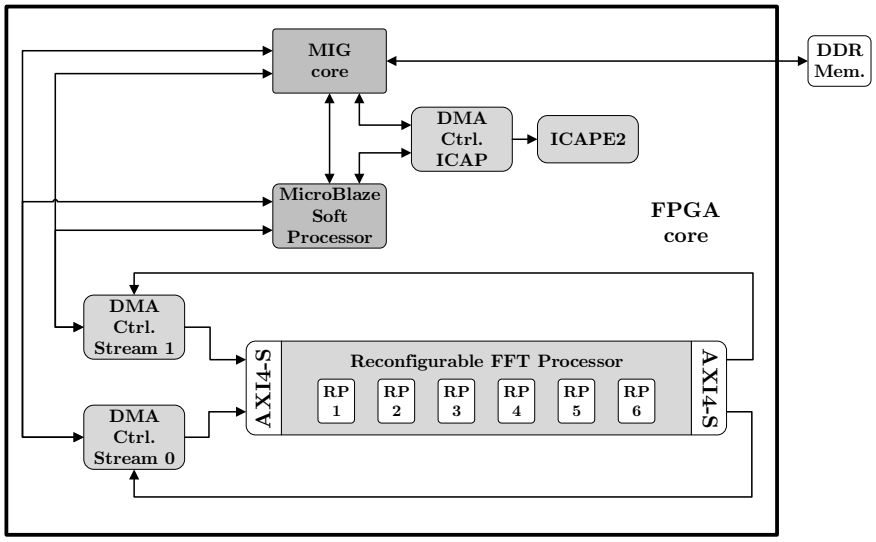


Figure 2. Reconfigurable FFT High-Level Architecture

4. Reconfigurable FFT Processor

One of the most important operations in wireless systems baseband processing is the Fast Fourier Transform (FFT). The FFT operation is present in most of the strongest waveform candidates for 5G, such as OFDM, GFDM [23] and FBMC [24], and in crucial CR operations as spectrum sensing [25]. For OFDM systems, such as NC-OFDM, signal modulation/demodulation is actually performed through IFFT/FFT operations. Analysing FFT requirements of widely used OFDM-based wireless systems, one can observe that the FFT size and number of streams vary for different standards and modes of operation within a certain standard. In a communication changing environment, support for multiple FFT sizes could be achieved by having an FFT processor to handle the largest of all needed sizes. Larger FFT sizes require more hardware resources and, as the largest FFT size would not be permanently needed, this approach shows a poor resource usage efficiency. Instead, an architecture providing specialized computation at run-time could improve resource usage efficiency and potentially lead to power consumption savings. From our perspective, this scenario provides a good opportunity to exploit and evaluate DPR application in the context of CR systems, in particular NC-OFDM baseband processors. From Figure 1 one realizes that there are other baseband operations than FFT/IFFT in an NC-OFDM system, but FFT is the most computationally complex. Thus, a flexible, resource-efficient, power-aware and dynamically reconfigurable FFT processor is a relevant contribution for hardware infrastructures intended for CR systems. Such an FFT processor was implemented [26] on a Xilinx Virtex-7 FPGA board (device: XC7VX485T-2FF1761C) running at 100 MHz and supports throughputs and FFT sizes required by most used wireless standards (such as IEEE 802.11, WiMAX or 3GPP-LTE): powers-of-two from 64 to 2048 and 1536. Double-stream 64-point FFT, which is required by some standards, is also supported. Both FFT input and output vectors appear in natural order and the arithmetic operations involve complex numbers whose real and imaginary parts are represented by 16-bit fixed-point values.

The Cooley-Tukey algorithm [27] was chosen to implement the FFT because it allows any factorization of the FFT size - N - with an acceptable computational complexity. As not all the supported FFT sizes are powers-of-two, the Mixed-Radix-2^2/2/3 variant of the Cooley Tukey algorithm was adopted. Mixed-Radix FFT algorithms are explained in detail in [28]. A pipeline FFT architecture was selected in this implementation. This category of FFT architectures takes advantage of parallel execution potential provided by hardware and shows a regular structure which is easy to scale for variable FFT sizes. Furthermore, pipelined architectures present simple control logic and allow for the continuous flow of data, which is an important feature in real-time applications like CR. Among the existing types of pipelined architectures, Single Delay Feedback (SDF) was selected due to its balance between memory requirements, throughput and implementation complexity. Cho et al. [29] provide details about processing elements for SDF FFT hardware implementations.

A schematic of the reconfigurable FFT high-level architecture is shown in Figure 2. The system architecture somehow resembles the proposed NC-OFDM baseband processor architecture (Figure 1), in which two domains were considered. In the implemented design, one can consider the *MicroBlaze Soft Processor* and the *Reconfigurable FFT Processor* as earlier versions of the Management Unit and the Dynamically Reconfigurable Datapath, respectively. The MicroBlaze is responsible for controlling the FFT dynamic reconfiguration by storing partial bitstreams in the DDR memory and later fetching and sending them to the FPGA configuration memory through the Xilinx Internal Configuration Port (ICAPE2) primitive. In turn, the Reconfigurable FFT Processor is where FFT computation actually occurs. It comprises six Reconfigurable Partitions (RPs) wrapped in an AXI4-Stream [30] IP core.

	RP1	RP2	RP3	RP4	RP5	RP6
Slice LUTs	400 (0.13%)	2800 (0.92%)	2800 (0.92%)	1600 (0.53%)	1600 (0.53%)	2800 (0.92%)
Slice Registers	800 (0.13%)	5600 (0.92%)	5600 (0.97%)	3200 (0.53%)	3200 (0.53%)	5600 (0.92%)
BRAM	10 (0.97%)	10 (0.97%)	10 (0.97%)	10 (0.97%)	10 (0.97%)	10 (0.97%)
DSP	0 (0%)	40 (1.43%)	40 (1.43%)	20 (0.71%)	20 (0.71%)	40 (1.43%)
# RM variants	7	3	3	5	4	4
Partial Bitstream Max. size (KB)	67	201	183	123	125	211

Table 1. Resources per RP and partial bitstream sizes

Through DPR, the functionality of each RP is defined by sending corresponding partial bitstreams to the FPGA port. The number of RPs used and their functionality are different from one FFT configuration to another. For example, 1536-FFT and 2048-FFT require the use of all six RPs, whereas 64-FFT only requires three RPs, leaving another 3 RPs unused (*blank*). In the double-stream 64-FFT configuration, those unused RPs are used to parallelly execute another 64-FFT.

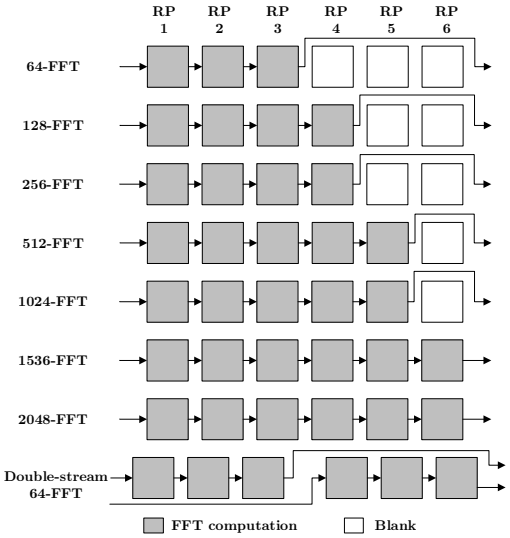


Figure 3. RPs role in each FFT configuration

Figure 3 shows, for every FFT configuration, how many RPs are used and how they are interconnected. These interconnection are included in the static part of the system and are controlled by multiplexing circuits. Table 1 presents values for resource utilization and partial bitstream sizes for each RP. Regarding the rest of the system (static part) the resources used are: 37054 (12.20% of available) slice LUTs, 29535 (4.86% of available) slice Registers, 168 (16.31% of available) BRAMs and 1 (0.04% of available) DSP block.

FFT input/output values are fetched from/stored in the DDR memory. To allow DDR access by the FFT core without MicroBlaze control and achieve higher data throughputs, Direct Memory Access (DMA) controllers for 32-bit data transactions are employed. Two DMA controllers are used by the FFT processor due to the double-stream 64-FFT configuration. In a similar way, reconfiguration throughput is improved by using a dedicated 32-bit data DMA controller to access the ICAPE2 primitive. All mentioned DMA controllers were implemented with Xilinx AXI DMA IP cores and were connected to a Xilinx MIG (Memory Interface Generator) block in charge of interfacing the FPGA device and the DDR memory.

4.1. Results and Discussion

The functional correctness of all FFT configurations was verified by comparing the FFT processor output results with MATLAB simulation results. For a continuous flow of received data, the FFT processor presents a throughput of at least 88 Msamples/s, in pipeline steady-state operation. This throughput is smaller than the theoretical limit of 100 Msamples/s imposed by SDF architectures operating at 100 MHz. The discrepancy between this limit and the observed throughput is mainly due to the overhead caused by DDR read/write accesses. However, the throughput of the implemented FFT processor is large enough to cover the requirements of standards like 3GPP-LTE, WiMAX and IEEE 802.11.

DPR's potential for improved resource usage efficiency is demonstrated by the double-stream 64-FFT configuration, where blank RP resources are reused to increment system's overall computation capacity. In the context of an NC-OFDM transceiver, those resources could be used to perform other baseband operations (e.g.: cyclic prefix insertion, digital modulation/demodulation).

The impact of DPR regarding reconfiguration latency was measured and the worst-case reconfiguration tme registered was 2.3 ms. The reconfiguration throughput observed was about 377 MiB/s (94.25% of the the ICAPE2 limit for 32-bit data transfer at 100 MHz). To evaluate the feasibility of DPR in CR systems, the observed latency was compared with reactivity time requirements defined in IEEE 802.22 [31] - a Wireless Regional Area Network (WRAN) wireless standard intended for communication in CR scenarios. According to this standard, radio devices should be able to perform operations such as communication establishment, closing or primary user detection within 2 s. Although a baseband processor may need to perform reconfiguration procedures that affect other modules other than FFT processor, the complexity and resource demands of FFTs are higher than in most baseband modules. Thus, we can conclude that DPR application in hardware infrastructures for CR systems is feasible.

Power consumption was also evaluated. A Texas In-

FFT Config.	Idle regime avg. output power (W)	Processing regime avg. output power (W)	Power difference (W)
64-FFT	1.23	1.39	0.16
256-FFT	1.24	1.42	0.18
512-FFT	1.25	1.43	0.18
1024-FFT	1.26	1.48	0.22
1536-FFT	1.27	1.50	0.23
2048-FFT	1.28	1.52	0.24
64-FFT double-stream	1.25	1.55	0.30

Table 2. Average output power measured for each FFT configuration

struments (TI) USB Interface Adapter was connected to the PMBus port on the Virtex-7 board and power measurements were monitored through the TI Fusion Digital Power Design software. In particular, the output power of the power rail which feeds the XC7VX485T FPGA core with a 1 V operating voltage was monitored. In the first power experiment, two operation regimes were defined: *Idle* (no FFT processing) and *Processing* (the FFT processor fetches data from DDR, computes FFT and stores the results back in DDR). For each configuration the FFT processor was left 10 min in the *Idle* regime followed by 10 min in the *Processing* regime. FPGA temperature was kept at 34 °C during all experiments. Power measurements over time for 2048-FFT are plotted in Figure 4, where it is possible to observe states and map them to the operation regimes initially defined. The averages for the output power in both *Idle* and *Processing* regimes measured for each configuration are presented in Table 2. From these results, one observes that power consumption is proportional to the number of RPs used for FFT computation. This may indicate that better resource usage efficiency offered by DPR can also lead to power savings. To assess this, a second power experiment was done.

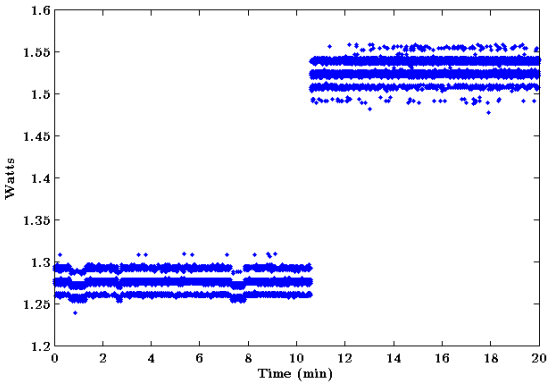


Figure 4. 2048-FFT measured power consumption

Two operation regimes were also considered: *Idle* regime (no FFT processing) and *Reconfiguration* regime, where the FFT processor is repeatedly reconfiguring between 64-FFT and 2048-FFT - this reconfiguration scenario produces the largest reconfiguration latency observed (2.3 ms). Power was measured considering the FFT processor 10 min in *Idle* regime and 10 min in *Reconfiguration* regime. Figure 5 shows the observed output power measurements over time and, like in the first experiment, the two operation regimes can be distinguished. No FFT processing is performed in the *Reconfiguration* regime. So, the increment in power is due to DPR activities. The average output power is 1.22 W for the *Idle* regime and 1.26 W for the *Reconfiguration* regime. Thus, the DPR power overhead is about 40 mW. While performing FFT computation, the difference between 2048-FFT and 64-FFT configurations is 130 mW (Table 2). Employing DPR, this amount of power can be saved at the cost of consuming 40 mW during 2.3 ms - reconfiguration latency. So, if the time interval between reconfigurations is significantly larger than the reconfigurations times, DPR can lead to power consumption reduction.

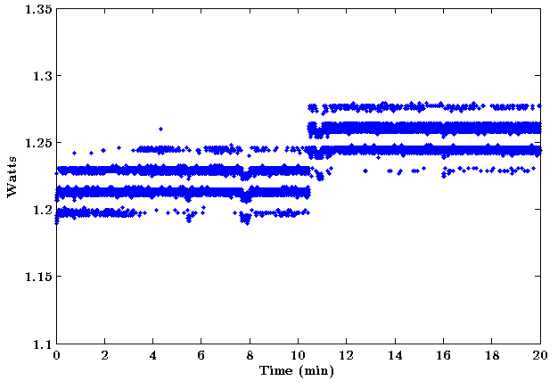


Figure 5. Measurements for reconfiguration power consumption

Considering IEEE 802.22 reactivity requirements and assuming that the operation mode of radio devices will remain unchanged for amounts of time much bigger than reconfiguration times, power efficiency can be improved through the application of DPR. This assumptions seem to be quite reasonable in wireless communication environments.

5. Conclusions

Flexible wireless communications require reconfigurable baseband processing engines able to adapt its internal operation according to communication demands. In this context, we proposed an architecture for an FPGA-based NC-OFDM processor composed of a dynamically reconfigurable baseband processing datapath and an intelligent management unit responsible for the control and run-time reconfiguration of the datapath, depending on communication requirements.

As the fisrt step towards the proposed NC-OFDM processor, a dynamically reconfigurable FFT processor supporting FFT sizes and throughputs required by most used 3G/4G standards was implemented on a Xilinx Virtex-7

FPGA. DPR techniques were exploited on this design, allowing a more efficient use of available resources. Reconfiguration times and power consumption were measures and their impact on CR environments was evaluated. The largest reconfiguration latency observed was about 2.3 ms, which is within a tolerable range for CR systems, and assesses the viability of DPR in this kind of applications. When compared with a worst-case and static hardware implementation, DPR-based implementations can improve power efficiency if radio device parameters, such as FFT size, remain unchanged for time intervals many times bigger than reconfiguration times.

References

[1] FCC. Spectrum Policy Task Force. Technical Report Rep. ET Docket no. 02-135, Federal Communications Commission, Nov. 2002.

[2] Joseph Mitola III. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. Doctor of technology, Royal Institute of Technology (KTH), Stockholm, Sweden, 2000.

[3] T. A. Weiss and F. K. Jondral. Spectrum pooling: an innovative strategy for the enhancement of spectrum efficiency. *IEEE Communications Magazine*, 42(3):S8–14, Mar 2004.

[4] J.G. Andrews, S. Buzzi, Wan Choi, S.V. Hanly, A. Lozano, A.C.K. Soong, and J.C. Zhang. What Will 5G Be? *IEEE Journal on Selected Areas in Communications*, 32(6):1065–1082, June 2014.

[5] Cornelia-Ionela Badoi, Neeli Prasad, Victor Croitoru, and Ramjee Prasad. 5G Based on Cognitive Radio. *Wireless Personal Communications*, 57(3):441–464, 2010.

[6] J. Ma, G. Y. Li, and B. H. Juang. Signal Processing in Cognitive Radio. *Proceedings of the IEEE*, 97(5):805–823, May 2009.

[7] Z. Kollar and P. Horvath. Physical Layer Considerations for Cognitive Radio: Modulation Techniques. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pages 1–5, May 2011.

[8] H. Bogucka, Alexander M. Wyglinski, S. Pagadarai, and A. Kliks. Spectrally agile multicarrier waveforms for opportunistic wireless access. *IEEE Communications Magazine*, 49(6):108–115, June 2011.

[9] H. Bogucka, P. Kryszkiewicz, and A. Kliks. Dynamic spectrum aggregation for future 5G communications. *IEEE Communications Magazine*, 53(5):35–43, May 2015.

[10] J. Chacko, C. Sahin, D. Nguyen, D. Pfeil, N. Kandasamy, and K. Dandekar. FPGA-based latency-insensitive OFDM pipeline for wireless research. In *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, pages 1–6, Sept 2014.

[11] R. Tessier, K. Pocek, and A. DeHon. Reconfigurable Computing Architectures. *Proceedings of the IEEE*, 103(3):332–354, March 2015.

[12] F. Shamani, R. Airoldi, T. Ahonen, and J. Nurmi. FPGA implementation of a flexible synchronizer for cognitive radio applications. In *2014 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8, Oct 2014.

[13] C. Vennila, G. Lakshminarayanan, and Seok-Bum Ko. Dynamic Partial Reconfigurable FFT for OFDM Based Communication Systems. *Circuits, Systems, and Signal Processing*, 31(3):1049–1066, 2012.

[14] C. Vennila, K. Suresh, R. Rathor, G. Lakshminarayanan, and Seok-Bum Ko. Dynamic partial reconfigurable adaptive transceiver for OFDM based cognitive radio. In *26th Annual IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), 2013*, pages 1–4, May 2013.

[15] Ke He, Louise Crockett, and Robert Stewart. Dynamic Reconfiguration Technologies Based on FPGA in Software Defined Radio System. *Journal of Signal Processing Systems*, 69(1):75–85, 2012.

[16] Félix Casado, Raúl Torrego, Pedro Rodríguez, Aitor Arriola, and Iñaki Val. Reconfigurable Antenna and Dynamic Spectrum Access Algorithm: Integration in a Cognitive Radio Platform for Reliable Communications. *Journal of Signal Processing Systems*, 78(3):267–274, 2015.

[17] S. Shreejith, B. Banarjee, K. Vipin, and S. A. Fahmy. Dynamic Cognitive Radios on the Xilinx Zynq Hybrid FPGA. In *Proceedings of the International Conference on Cognitive Radio Oriented Wireless Networks (CROWNCOM)*, April 2015.

[18] K. Vipin and S.A. Fahmy. ZyCAP: Efficient Partial Reconfiguration Management on the Xilinx Zynq. *IEEE Embedded Systems Letters*, 6(3):41–44, Sept 2014.

[19] Mario Lopes Ferreira and Joao Canas Ferreira. Reconfigurable NC-OFDM Processor for 5G Communications. In *Embedded and Ubiquitous Computing (EUC), 2015 IEEE 13th International Conference on*, pages 199–204, Oct 2015.

[20] Guangxiang Yuan, Xiang Zhang, Wenbo Wang, and Yang Yang. Carrier aggregation for LTE-advanced mobile communication systems. *IEEE Communications Magazine*, 48(2):88–93, February 2010.

[21] R. Bonamy, Hung-Manh Pham, Sebastien Pillement, and D. Chillet. UPaRC - Ultra-fast power-aware reconfiguration controller. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 1373–1378, March 2012.

[22] R. Oomen, Tuan Nguyen, A. Kumar, and H. Corporaal. An automated technique to generate relocatable partial bitstreams for Xilinx FPGAs. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–4, September 2015.

[23] G. Fettweis, M. Krondorf, and S. Bittner. GFDM - Generalized Frequency Division Multiplexing. In *IEEE 69th Vehicular Technology Conference, 2009. VTC Spring 2009.*, pages 1–4, April 2009.

[24] O. Font-Bach, N. Bartzoudis, X. Mestre, D. Lopez-Bueno, P. Mege, L. Martinod, V. Ringset, and T.A. Myrvoll. When SDR meets a 5G candidate waveform : Agile use of fragmented spectrum and interference protection in PMR networks. *IEEE Wireless Communications*, 22(6):56–66, December 2015.

[25] Ahmed Elsokary, Peter Lohmiller, Václav Valenta, and Hermann Schumacher. A Hardware Prototype of a Flexible Spectrum Sensing Node for Smart Sensing Networks. In Mark Weichold, Mounir Hamdi, Muhammad Zeeshan Shakir, Mohamed Abdallah, George K. Karagiannidis, and Muhammad Ismail, editors, *Cognitive Radio Oriented Wireless Networks*, number 156 in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, pages 391–404. Springer International Publishing, April 2015. DOI: 10.1007/978-3-319-24540-9_32.

[26] Mario Lopes Ferreira, Amin Barahimi, and Joao Canas Ferreira. Dynamic Reconfigurable FFT Processor for Flexible

OFDM Baseband Processing. accepted for publication in DTIS'16, April 12-14, 2016, Istanbul, Turkey.

[27] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[28] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Signals and communication technology. Springer, 2004.

[29] Inkeun Cho, T. Patyk, D. Guevorkian, J. Takala, and S. Bhattacharyya. Pipelined FFT for wireless communications supporting 128-2048 / 1536 -point transforms. In *2013 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1242–1245, December 2013.

[30] Xilinx Inc. *UG1037 - Vivado Design Suite: AXI Reference Guide*, June 2015.

[31] IEEE Std 802.22™-2011: Standard for Local and metropolitan area networks - Specific requirements - Part 22: Cognitive Wireless RAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Policies and procedures for operation in the TV Bands, 2011.

A Wireless Biosignal Measurement System using a Zynq SoC

Ricardo Joaquinito
Electronic System Design and Automation
INESC-ID/IST/ULisbon
Lisbon, Portugal
ricardo.joaquinito@tecnico.ulisboa.pt

Helena Sarmento
Electronic System Design and Automation
INESC-ID/IST/ULisbon
Lisbon, Portugal
helena.sarmento@tecnico.ulisboa.pt

***Abstract*—This paper presents a prototype for a wireless biosignal measurement system, which explores the use of a SoC FPGA device, the Zynq SoC, for the signal processing. In this system, the body temperature and heart rate are monitored using a steel-head thermistor and an electrocardiographic (ECG) signal acquisition module, respectively. Analog signals from the sensors are converted to digital format using an ADC component integrated in the FPGA fabric of the Zynq. A Real-Time Operating System runs on the Zynq's ARM processor. The algorithm that extracts the heart rate from the ECG signal is based on the Pan-Tompkins algorithm for QRS complex detection and is partially running on a core in the FPGA fabric, generated from C code using the Vivado High Level Synthesis software. The measurements are sent via Bluetooth Low Energy to a smartphone, where an application shows the body temperature and heart rate to the user.**

Keywords—biosignals; ECG; SoC FPGA; Zynq; Bluetooth Low Energy

I. INTRODUCTION

Human being's physiological parameters are important indicators of health. Recently, the measurement of these parameters has gained notoriety in the consumer electronics market, primarily through wearable devices like fitness bands and smart-watches. Medical-grade wearables that can be worn by patients at home are now in demand, but, in order to obtain the medical grade, such devices must go through a rigorous regulatory process which can take from 6 months to 2 years [1].

The ease of reconfiguration and versatility of Systems-on-Chip with Field Programmable Gate Array (SoC FPGA) have been noted by the medical industry's developers as a good solution for this kind of equipment [1]. They are ideal for developing embedded systems if processing with the aid of custom hardware and regular system upgrades are desired.

The electrocardiographic (ECG) signal is one of the physiological signals that can be monitored by wearable devices [2]. It can be used to keep track of the person's heart rate and detect cardiac problems and other health issues. A highly reliable algorithm for these tasks can be computationally demanding and has real-time constraints, so hardware solutions in FPGAs have been proposed [3][4].

Regarding wearable devices, consideration must be taken on their energy consumption. New low energy wireless standards have emerged in recent years aimed at this kind of devices. Bluetooth Low Energy (BLE), an energy efficient variation of the Bluetooth technology, is now the most widely used open standard in that category, with predictions stating that, by 2018, 96% of all smartphones will support BLE [5].

This work explores the use of a SoC FPGA device - the Zynq SoC – with a BLE connection for the development of a wireless biosignal measurement system, which can be the basis for a wearable health-monitoring device. The heart rate is calculated from the person's ECG signal and the body temperature is also measured. The heart rate detection algorithm is partially running on custom hardware implemented in the FPGA fabric.

II. BACKGROUND

This section briefly introduces the SoC FPGA device used in this project and the ECG signal, including the processing required for heart rate extraction.

A. Zynq SoC

The Zynq-7000 Extensible Processing Platform [6], or simply Zynq, is a family of SoC FPGA devices developed by Xilinx, Inc. and introduced in 2011, which combine a dual-core ARM Cortex-A9 processor and FPGA fabric. It is a well suited and powerful platform for the development of embedded systems, as it provides a flexible way to integrate hardware and software.

The general architecture of the Zynq comprises two parts: the Processing System (PS) and the Programmable Logic (PL). Simply put, the PS is related to the processor and software development and the PL is related to the FPGA fabric and hardware development. The two parts can be interfaced with each other and with peripherals.

Zynq projects can be entirely developed on Xilinx's Vivado Design Suite. It includes the Vivado Integrated Development Environment (IDE) and the Software Development Kit (SDK), which handle the hardware and the software design respectively, and also the Vivado High Level Synthesis (HLS). The HLS is able to generate hardware description code from functions implemented in C, C++ or SystemC programming language, which can then be exported as Intellectual Property (IP) cores and implemented on Zynq's PL.

B. The ECG signal

Electrocardiography is the process of recording the electrical activity of the heart using electrodes placed on a person's body. Although the most sophisticated 12-lead measurement systems use as many as 10 electrodes, a simple configuration of only 2 or 3 electrodes is sufficient to acquire a person's ECG signal.

The ECG signal is a quasi-periodic repetition of the P wave, the QRS complex and the T wave, as represented in Figure 1. The correct detection of the QRS complexes is an important step for heart rate measurement, as it is usually based on the intervals between the R-peaks.

The Pan-Tompkins algorithm [7] is one of the most notorious algorithms for QRS complex detection. Basically, its main processing steps are: (i) band-pass filtering, for noise removal; (ii) differentiation, for slope analysis; (iii) squaring, to intensify the slope response of the derivative; (iv) moving window integration, to get the slope and width of the QRS complex. The

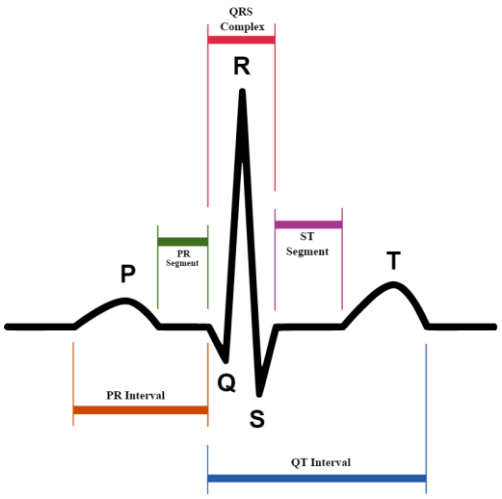


Fig.1. Typical waveform of the PQRST interval on the ECG signal.

big slope changes that happen during the QRS complex produce very high peaks in the derivative signal after squaring. The squaring process also helps to avoid false detections on the T wave, which in the original signal can reach an amplitude higher than the R peak, but with a less accentuated slope. Thresholds are used to detect the peaks and are adjusted periodically to adapt to changing characteristics of the signal.

III. System Overview

Since the proposed system can be achieved with different combinations of development boards and sensors in the market, the general architecture of this system is introduced first with a neutral view of the components, followed by a listing and short description of the components used in the implemented prototype.

A. General architecture

The general architecture of the implemented system is presented on Figure 2. Everything represented besides the smartphone composes the sensing device which is to be worn by a person.

The ECG signal is extracted from a 2 or 3-electrode configuration and is preprocessed before being converted to digital format. The conditioning circuit filters the signal in order to eliminate most of the noise that typically contaminates an acquired ECG signal, and must amplify it since the captured signal amplitude is in the order of microvolts (µV).

The temperature sensor is a steel-head thermistor and its conditioning circuit consists of a simple voltage divider that reflects the sensor's resistance changes with the temperature.

In the SoC FPGA, the analog signals must be converted to digital format by an Analog-to-Digital Converter (ADC). In Figure 2, the ADC is shown as part of the FPGA fabric as there is an ADC component included in this Zynq's FPGA fabric. In order to harness the hardware capabilities of this device, a part of the processing algorithm is implemented in the FPGA.

The BLE SoC handles the device's BLE connection and data transmission. It includes another less-powerful processor, with firmware that implements the necessary BLE protocols, and a radio transceiver. It receives the heart rate and temperature from the SoC FPGA through a one-way UART serial connection. Those values are received by a smartphone connected to this device and shown on an application which makes the user interface.

B. Prototype

The prototype developed for this project uses the following development boards and sensors:

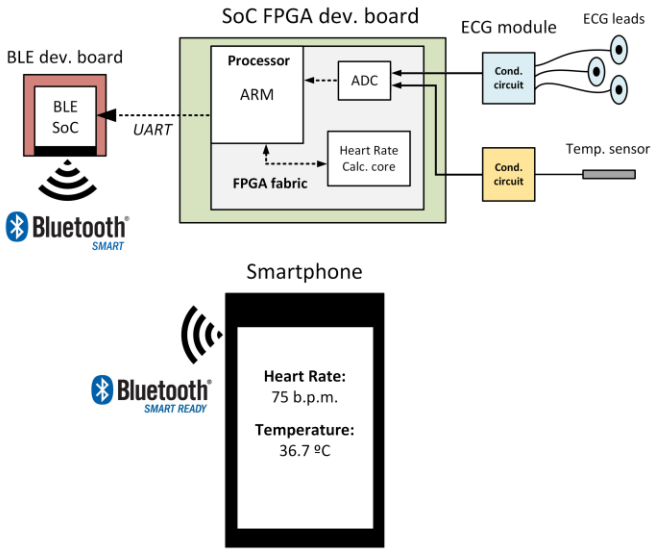


Fig. 2. General architecture of the proposed wireless biosignal measurement system.

- ZYBO: A low-cost Zynq Development Board which includes a Zynq-7010 SoC from Xilinx [6]. The Zynq-7010 includes an ARM Cortex-A9 dual-core processor and logic fabric based on the Artix-7 FPGA. The processor has a maximum clock frequency of 650 MHz.
- BITalino ECG module with 3-electrode configuration: A part of the BITalino development kit [7] that allows ECG signal acquisition and conditioning, outputting an analog signal.
- IM120628010: A steel-head NTC thermistor as the temperature sensor.
- BLE Nano: A very small-scale BLE development board featuring the Nordic nRF51822 SoC [8].

In the BITalino module, the signal captured by the ECG electrodes is subjected to an amplification gain of 1100 and filtering in the band 0.5 – 40 Hz, which raises the signal amplitude significantly and eliminates most of the artifacts and noise not associated with muscle.

IV. Hardware and Software Design

The most important aspects of the hardware used on the Zynq's PL, software running on the PS and the BLE connection are covered in this section.

A. XADC

The Xilinx Analog-to-Digital Converter (XADC) is a dual 12-bit ADC included as a hard component in the PL of the Zynq device, with a maximum sampling rate of a million samples per second. It has an input range of 1 V (0 – 1 V).

In this project, the XADC is included in the hardware system using the XADC Wizard core. The PS interfaces directly with the XADC through the PS-XADC interface. Two channels of the XADC are used to convert the analog signal from the ECG and temperature sensors to digital format.

B. Operating System and application

The ARM processor runs a light Real-Time Operating System (RTOS), the FreeRTOS [10]. This operating system makes it possible to divide the application code into multiple threads of execution, referred to as 'tasks', with an assigned priority and execution period. With its preemptive multitasking, it may put a task on hold to start the execution of a higher priority task with time constraints. Only one core of the processor is used.

In this project, three tasks are implemented:

- *ECGReadTask*: Period of 10 ms; priority level 2 (highest). Acquires the ECG signal data from the XADC, stores the ECG data in a buffer, stores the difference between the new ECG value and the previously acquired value in the ECG derivative buffer.
- *ECGProcessingTask*: Period of 100 ms; priority level 1. Executes the hardware part of the algorithm on the HeartRateCalculator core to obtain the heart rate.
- *TempRead&UARTTask*: Period of 1 s; priority level 1. Acquires the raw temperature data from the XADC, converts it to Celsius, sends the latest temperature and heart rate values through UART communication.

By making the *SensorReadTask* the highest priority task, it is assured that the acquisition of ECG values from the XADC is executed with precise intervals. This is important as this task defines the application sampling rate of the ECG signal, which is 100 Hz.

Figure 3 presents the flowchart of the application, divided by the three tasks. It shows specifically which steps are performed in the PS and the PL of the Zynq. Further detail on the ECG processing is provided on Section V.

C. Heart Rate Calculator core

This IP core was generated using the Vivado HLS, which converted the C code implementation to VHDL and exported it to the Vivado IDE. The core reads the ECG Derivative buffer and outputs an integer with the heart rate value. It interfaces with the PS using the AXI4-Lite protocol [11]. The software driver for this core was automatically generated by the Vivado SDK after its inclusion in the hardware project.

D. BLE Nano board and BLE communication

The nRF51822 SoC (referred to as nRF51 from here on) on the BLE Nano board receives data from the Zynq through a UART connection with baudrate 115200 bps. A string is sent every second from the Zynq to the nRF51 with the latest temperature and heart rate values. The nRF51 application will interpret the string and put those values in the appropriate data structures that will be transferred via BLE.

This BLE device is configured as a peripheral device, which means that it must periodically advertise its availability for connection to other central devices doing the scanning process. Only the central device (the smartphone) is able to initiate the connection. The advertising interval is set to 1 second.

The data transferred through BLE is encapsulated in services, which are basically containers for conceptually related information. The Bluetooth Special Interest Group defines standard BLE services [12] that developers can freely use. This device uses two of the standard services: the Heart Rate Service (UUID: 0x180D) and the Health Thermometer Service (UUID: 0x1809).

When the BLE connection with the smartphone is initiated, the relevant data values of the two services are updated and sent to the smartphone every time a new string is received from the Zynq. This means that the user will see the values updated once every second on the smartphone application.

V. ECG SIGNAL PROCESSING

The heart rate calculation relies on the detection of QRS complexes on the ECG signal. The algorithm implemented in this system is based on the Pan-Tompkins algorithm for QRS complex detection, without the moving window integration step.

The processing begins outside of the Zynq, in the BITalino ECG module, with the band-pass filtering in the band

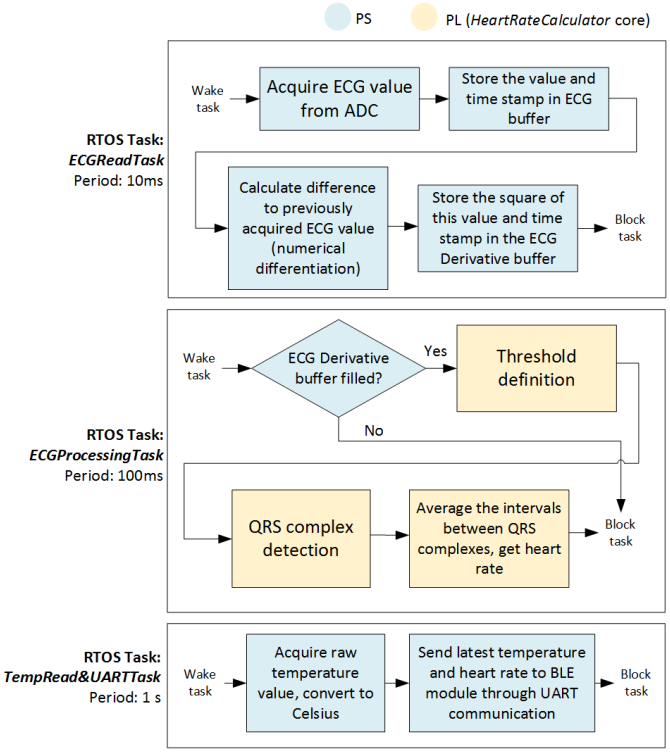


Fig.3 Task division of the application running on the Zynq's PS.

0.5 – 40 Hz. In the Zynq, the analog signal is first of all converted to a digital signal by the XADC. The sampling rate of the signal is 100 Hz. This sampling rate complies with the sampling theorem condition

$$f_s > 2B \tag{1}$$

in which f_s is the sampling rate and B is the bandwidth of the signal to be sampled. In this situation, B is equal to 40 Hz because of the filtering performed by the BITalino ECG module. It is also a sufficient sampling rate to obtain correct QRS complex detections.

For the real-time analysis of the ECG, two buffer arrays with a size of 600 elements (the equivalent to 6 seconds of signal data) are used to store elements of the type *ecg_t*, which includes an integer *value* with the signal amplitude and an unsigned integer *time*, a time stamp for the acquired sample. One of the arrays, *ECG_Buffer[]*, stores the acquired ECG signal, for results and debugging purposes. The other buffer, *ECG_Deriv_Buffer[]*, is the square of the ECG signal derivative (numerical differentiation), in which, for each element *k*,

$$ECG\_Deriv\_Buffer[k].value = (ECG\_Buffer[k+1].value - ECG\_Buffer[k].value)^2. \tag{2}$$

When full, the buffers shift their elements to make way for new samples.

QRS complexes are detected by comparing the derivative buffer with a threshold value, which is updated every 100 ms, each time the *ECGProcessingTask* is executed. This threshold is given by the formula

$$thre = \frac{\sum_{k=0}^{buffer\_size} ECG\_Deriv\_Buffer[k].value}{buffer\_size} \times F \tag{3}$$

in which F is a gain factor that the average of the ECG derivative buffer is multiplied by.

The values of the ECG derivative buffer are compared with the threshold, from the earliest to the latest. When a value is above the threshold, the corresponding time stamp is stored, and

the buffer analysis skips the next 30 elements. This is because some of the following buffer values, which are still responding to a QRS complex's fluctuations, will likely also be above the threshold and must not be stored. Lastly, the intervals between the QRS complexes detected in the buffer are averaged and converted to beats per minute.

VI. Tests and Results

The resource utilization in the FPGA fabric for this prototype is presented on Table 1. It includes the HeartRateCalculator core and the I/O connections for the peripherals (sensors input and UART output).

TABLE I. POST-IMPLEMENTATION RESOURCE UTILIZATION IN THE FPGA FABRIC.

Resource	*Utilization*	*Available*	*Utilization (%)*
LUT	1880	17600	10.38
LUTRAM	109	6000	1.82
FF	2232	35200	6.34
BRAM	2	60	3.33
DSP	6	80	7.50
IO	6	100	6.00
BUFG	1	32	3.12

Tests were conducted to check the operating range of the device, the temperature measurement and the ECG processing. The sensor tests were performed on a 24 year-old male.

A Sony Xperia V smartphone was used for the BLE connection tests. The maximum distance at which the smartphone could establish a connection with the device was roughly 10 m, with obstacles. During a connection, the heart rate and temperature values are successfully presented and updated every second on the Android app developed for demonstration.

Figure 4 shows a 3-second plot of the ECG signal acquired by the system, the corresponding squared derivative signal and the designated threshold. The *F* factor from equation (3) is set to 8 for the tests. The heart rate in this segment, as calculated by the system, is 68 b.p.m.. The signal analysis shows that the algorithm is able to avoid false QRS detections during the T wave when its amplitude is superior to the R peak. The success rate of QRS detection was 94.6% for multiple 1-minute tests.

The accuracy of the temperature sensor was tested by comparison with a digital thermometer with a precision of 0.1 ºC. Multiple tests to the armpit temperature after 2 minutes of skin contact resulted in a maximum error of ± 0.4 ºC in relation to the digital thermometer measurements, under the same conditions. The values measured by our sensor were in the range 36.4 ± 0.4 ºC.

VII. Conclusions

A wireless biosignal measurement system prototype for heart rate and body temperature monitoring, that makes use of the Zynq SoC and the Bluetooth Low Energy wireless communication standard, has been implemented successfully. The Zynq's capabilities, as a SoC FPGA device that can be used for hardware/software codesign, were explored with the use of a custom IP core that runs part of the algorithm for heart rate detection.

The tests show that the system is able to deliver reliable heart rate and body temperature data in real-time to a user, with a smartphone application. The algorithm that extracts the heart rate, using a QRS detection method mostly based on the Pan-Tompkins algorithm, delivers satisfactory results during the tests.

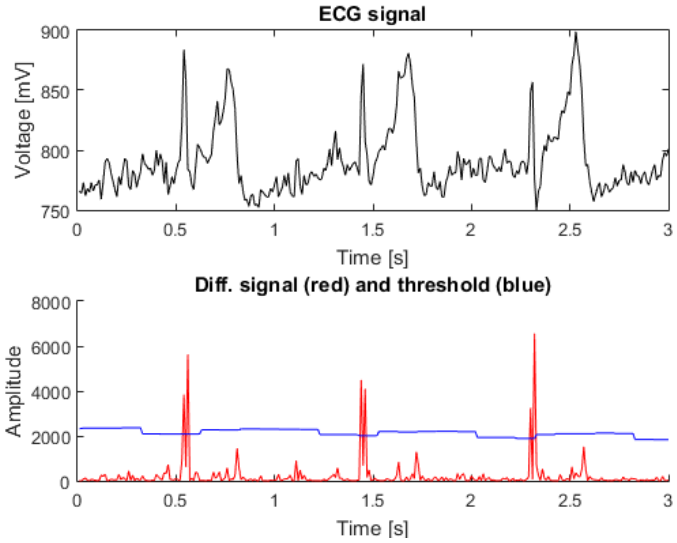


Fig.4 ECG signal acquired (top) and corresponding derivative signal and threshold (bottom).

Acknowledgments

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013. The authors also express their gratitude to Hugo Silva (PLUX - Wireless Biosignals S.A.) for kindly providing us with a BITalino toolkit.

References

[1] M. Santarini, "Xilinx Speeds Custom Medical Innovations to Market", in *XCell Journal*, no. 93, p. 9-13, 2015.

[2] M. Baig, H. Gholamhosseini and M. J. Connolly, "A comprehensive survey of wearable and wireless ECG monitoring systems for older adults", *Medical Biological Engineering Computing*, vol. 51, no. 5, May 2013 pp. 485-495, May 2013.

[3] D. Alhelal, K. A. I. Aboalayon, M. Daneshzand and M. Faezipour, "FPGA-based denoising and beat detection of the ECG signal," *Systems, Applications and Technology Conference (LISAT), 2015 IEEE Long Island*, Farmingdale, NY, pp. 1-5, 2015.

[4] C. C. Chou, W. C. Fang and H. C. Huang, "A novel wireless biomedical monitoring system with dedicated FPGA-based ECG processor," *Consumer Electronics (ISCE), 2012 IEEE 16th International Symposium on*, Harrisburg, PA, pp. 1-4, 2012.

[5] M. Snow, "Developers wanted: Bluetooth Low Energy is the future of wearables", 2015. [Online]. Available: http://www.broadcom.com/blog/ces/developers-wanted-bluetooth-low-energy-is-the-future-of-wearables/ [Accessed: April 2016]

[6] Xilinx Inc., "Zynq-7000 All Programmable SoC Overview", v1.9, January 2016. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf [Accessed: April 2016].

[7] J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," in *IEEE Transactions on Biomedical Engineering*, vol. BME-32, no. 3, pp. 230-236, March 1985.

[8] H. Silva, J. Guerreiro, A. Lourenço, A. Fred and R. Martins, "BITalino: A novel hardware framework for physiological computing", *Proc International Conf. Physiological Computing Systems – PhyCS*, Lisbon, Portugal, pp. 246-253, January 2014.

[9] Nordic Semiconductor, nRF51822 product webpage. [Online]. Available: https://www.nordicsemi.com/eng/Products/Bluetooth-Smart-Bluetooth-low-energy/nRF51822 [Accessed: April 2016].

[10] Real Time Engineers Ltd., FreeRTOS webpage. [Online]. Available: http://www.freertos.org/ [Accessed: April 2016].

[11] Xilinx Inc., "Vivado Design Suite – AXI Reference Guide", v3.0, June 2015. Available: http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf [Accessed: April 2016].

[12] Bluetooth SIG, GATT Services webpage. [Online]. Available: https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx/ [Accessed: April 2016]

A real-time underwater acoustic direction finder in FPGA

José F. Valente
Faculty of Engineering, University of Porto
Porto, Portugal
Email: jfvalente@fe.up.pt

José C. Alves
Faculty of Engineering, University of Porto
INESC TEC
Porto, Portugal
Email: jca@fe.up.pt

***Abstract*—The automatic direction finding of an underwater sound source has been extensively used in underwater passive acoustic monitoring. The process consists in measuring the time difference of arrival (TDOA) of a sound wave to two or more hydrophones closely located and then extrapolating the relative direction of the acoustic source based on the time differences. Although the generalized cross-correlation between the received signals is a common technique for determining the TDOA, the underwater environment introduces several distortions in amplitude and phase of the sound waves due to reflections and the variation of the sound propagation speed with temperature, pressure and salinity. Particularly in confined spaces, the high level of reverberation makes the amplitude of the signals captured by the hydrophones very dissimilar and because of this the use of the cross-correlation is not effective to identify the time difference of arrival. In this work we propose an alternative method to calculate the TDOA consisting in the detection of the beginning of the signals by discovering a series of zero-crossing samples looking alike in each received signal, and then calculating the time difference between them. This process has been successfully implemented and tested in real-time in the programmable logic (PL) part of a Zynq device. In this paper we present the architecture of the system developed for calculating the TDOA, using as transmitter a $35\,\mathrm{kHz}$ underwater acoustic beacon and receiving the signal with two hydrophones sampled at approximately $2\,\mathrm{MHz}$.**

I. Introduction

Long endurance autonomous underwater vehicles (AUV), like electric underwater gliders capable of undertaking multi-month unassisted missions, suffer from the lack of real-time communications with the outside world, while submerged in remote ocean locations. In long range applications it may be desirable to have real-time communications for retrieving data or remotely changing the mission parameters. As acoustic communications is the only practical mean to transmit data underwater, this problem can be mitigated using a cooperative autonomous surface vehicle (ASV) capable of following at the surface the path of the submerged AUV and provide a relay to satellite data networks [1]. This may be done by attaching to the AUV a device transmitting a known acoustic signal and calculating the direction of the sound wave relative to an array of hydrophones, by measuring the time difference of arrival (TDOA) of the sound wave to the different hydrophones. Using only two hydrophones in a moving surface vehicle it is possible to determine the direction of the received sound wave in two dimensions and then extrapolate the relative position of the submerged acoustic source by combining different direction estimates along time.

In spite of being vulnerable to the uncertainty of the sea and weather conditions, autonomous sailing boats [2] are a type of surface vehicle with convenient characteristics for such application, because they are capable of performing oceanic operations during long periods of time. Having a better navigation performance than a long endurance AUV, in terms of speed and maneuverability, an autonomous sailing boat can be programmed to actively sail convenient routes around an uncertain position of an AUV for improving the estimation of its location.

The time difference of arrival (TDOA) method has been extensively used in passive acoustic source positioning by measuring the time difference between signals arriving in two or more hydrophones. In this work we propose an alternative and simple method to calculate the TDOA using two hydrophones. Our approach consists in detecting the beginning of the signals by discovering a series of zero-crossing samples looking alike in each received signal, and then calculating the time difference between them. This process requires a computing effort much lower than other methods usually used for this purpose, based on cross-correlation. We have built a proof-of-concept prototype in a field-programmable SoC (XILINX Zynq) that has been successfully validated with laboratory experiments in a test tank.

II. Related work

One of the methods used for estimating the time difference of arrival (TDOA) is the generalized cross-correlation (GCC) [3], [4]. Although being computationally intensive, this process is widely used because it can weaken the impact of the ambient noise on the accuracy of the lag estimation between the two signals. The process consists in detecting the peak time position of the cross-correlation function of two received signals, representing the delay between them. Usually, the signals are correlated in their significant whole duration, what can be difficult to do in real-time on long duration signals with high sampling frequencies and using low performance computing systems, due to the heavy computational effort needed. Besides, the propagation of acoustic signals through the underwater acoustic channel introduces spectral distortions and reverberation due to multi-path effects and variations in the sound speed with depth, temperature and salinity. This

 978-989-704-110-5

makes the signals received by two closely located hydrophones very different from each other, especially in their amplitude envelope when operating in confined regions, due to the high level of reverberation. Figure 1 shows the beginning of the two 35 kHz signals recorded in a test tank, where the amplitude envelope is very dissimilar and highly variable with small displacements of the acoustic source. Because of this, the cross-correlation method is not effective in such situations [5].

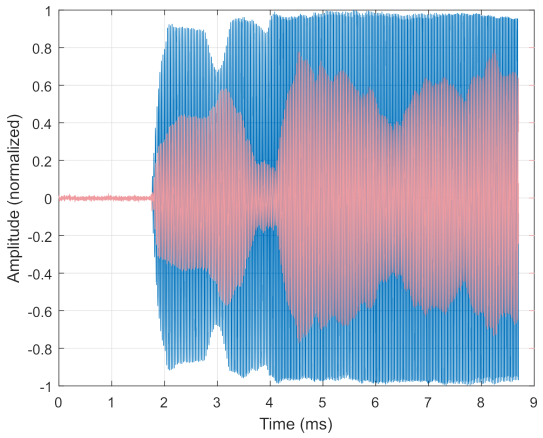


Fig. 1: The initial part of the two 35 kHz signals recorded in experimental tests.

Based on these observations, we propose a new method to estimate the TDOA of two received signals, not committed to a signal frequency and duration, thus making possible to use commercially available underwater acoustic beacons with long autonomy of operation (typically months to years).

III. Estimating the TDOA

To determine the instant of arrival of the two signals and then calculate the time difference between them, we focus in accurately detecting only the very beginning of each received signal, analyzing them in the time domain. We developed an algorithm that continuously detects the zero-crossings of the signal and measure the signal period as the time difference between two consecutive zero-crossing in the same direction (either rising or falling). A series of consecutive zeros occurring within a predefined time interval (the expected signal period), indicates the beginning of the received signal. After detecting the instant of arrival of both signals, we further analyze the signal samples closer to the first periods above a decision threshold and measure the TDOA as the time difference between them.

Figure 2 shows the initial part of the signals plotted in figure 1 (top plot) and the series of values obtained as the signals period, represented here as number of samples, by measuring the time difference between two consecutive zero-crossing samples occurring in the same direction (bottom plot). Around 0.3 ms the red signal (dash-dot) arrives and the measured period starts to increase to its expected nominal value (in this example equal to 56 samples). When a small number of periods of both signals (typically between 10 and 20) are detected above a threshold, the two signals are considered as arrived. Then the sequence of periods calculated before is analyzed to determine the instant of arrival of the two signals. This is done by selecting the periods of each signal below and above the threshold (two for each signal) and calculating the difference between the time of arrival of the most similar periods of each signal. Field experiments have shown that using threshold values between 75% and 95% of the nominal period, the TDOA results calculated match the expected time delay for different angles. To increase the robustness of the system, we have implemented four similar modules, two working with the rising edge and the other two with the falling edge, and applying different parameters to each one: the period threshold and the number of periods above the threshold required to consider the signal as received.

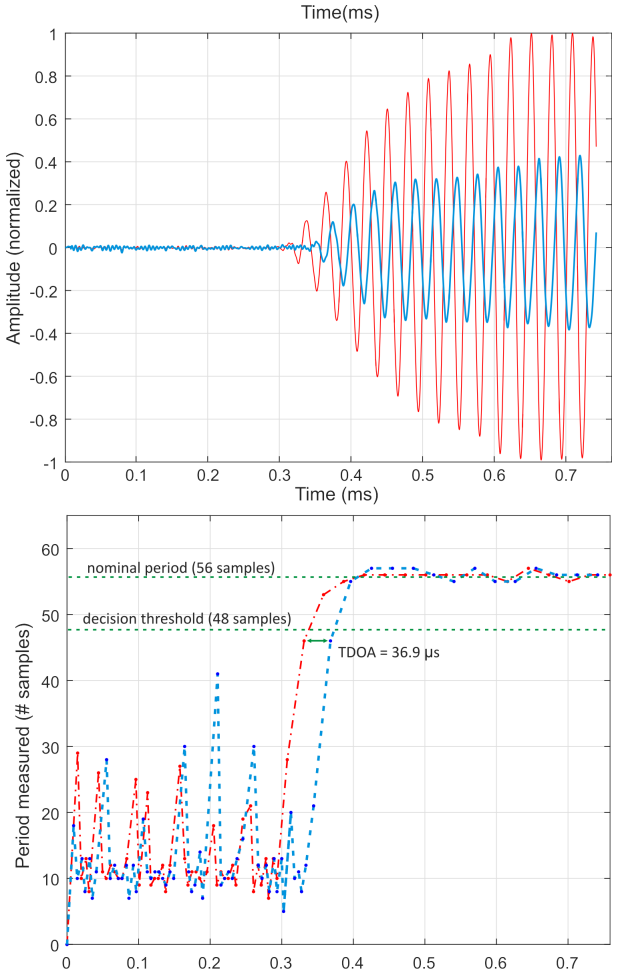


Fig. 2: Measuring the time difference of arrival of two signals.

Our system uses a 1.953 MHz sampling frequency (resulting from dividing the main 125 MHz clock signal by 64) because this is the highest sampling rate supported by the continuous recording process implemented in the RedPitaya system. For the 35 kHz frequency of the acoustic source used (approximately 4.24 cm underwater wavelength), the sampling period (512 ns) corresponds to a distance traveled by the underwater acoustic wave equal to 0.76 mm. If the process described above uses the time measurements as integer sampling periods,

that distance would translate to a theoretical minimum angle resolution below $1°$, within direction angles in the range of $\pm 43°$. However, calculating the time of zero-crossing with a linear interpolation using the samples below and above zero, will lead to values of the zero-crossing time as a fraction of the sampling period. Although this will theoretically improve the angle resolution with the number of fractional bits used to represent time, the preliminary experimental results have shown no relevant improvement in the accuracy of the angle calculations with the increase of the time resolution.

To remove the low frequency components and most of the background underwater noise due to ship activity, waves, wind and rain, the input signals must pass through a high-pass filter with a cut-off frequency adjusted to the frequency of the acoustic source being tracked. This filter is presently implemented in the digital platform as a 32-order FIR with a cutoff frequency equal to $30\,\mathrm{kHz}$, but could also be integrated in the analog front-end.

IV. Implementation

We have implemented and tested a prototype for this TDOA detector using as underwater acoustic source a commercial acoustic beacon emitting a series of $35\,\mathrm{kHz}$ pulses lasting $12\,\mathrm{ms}$ (EMT-01-3 from Sonotronics), two omnidirectional hydrophones (Aquarian H2a) separated by approximately $6cm$ ($1.5\,\mathrm{signalwavelength}$), a custom designed analog front-end with a variable gain amplifier and low-pass anti-aliasing filtering and a FPGA-based (XILINX Zynq) single-board embedded computer with a two channel high-speed ADC (RedPitaya [6]). The real-time TDOA processing block was built mainly in the programmable logic section and consists of four similar modules for analyzing independently the rising and the falling zero-crossings of the two input signals, using different configuration parameters. The four TDOA values are then read by the embedded ARM processor to implementing a final voting and filtering stage and provide the final TDOA estimate to the surface vehicle application layer for computing the source source direction.

A. Hardware platform

The digital platform used was the RedPitaya single board computer, based on the XILINX Zynq 7010 programmable SoC. Besides the usual on-board interfaces and devices required to run a Linux operating system, this board also includes a dual high-speed ADC and a dual high-speed DAC, both capable of operating at $125\,\mathrm{Msps}$. This platform is usually sold as a all-in-one laboratory instrument, providing the programmable logic (PL) hardware interfaces and web applications that implement an oscilloscope, an arbitrary function generator, a spectrum analyzer and also a general purpose PID controller. The hardware design for the PL section is provided as an open source project for the XILINX Vivado design tool. A variant of the initial hardware system also includes a DMA controller that allows the continuous acquisition and recording of the two analog input signals up to $2\,\mathrm{Msps}$.

To interface the hydrophones or high-impedance piezoelectric acoustic transducers to the RedPitaya analog inputs, we have developed an analog front-end daughter board implementing two variable and digitally controllable amplifier chains, followed by analog low-pass anti-aliasing filters with a cutoff frequency set to approximately $250\,\mathrm{kHz}$. This board also includes a high-speed analog switch, allowing to multiplex two additional analog signals by using a second amplifier and filtering board. The amplifier chain includes a first stage which gain was adjusted to $10\times$ and an additional stage which gain is programmed by I2C digital potentiometers from $0.1\times$ to $2500\times$. Figure 3 shows the RedPitaya embedded computer and the daughter board and figure 4 presents a simplified block diagram of one amplifier and filtering analog path.

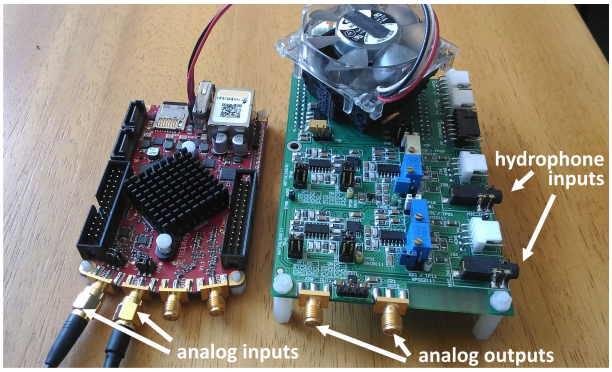


Fig. 3: The RedPitaya embedded computer (left) and the analog frontend amplifier board (right).

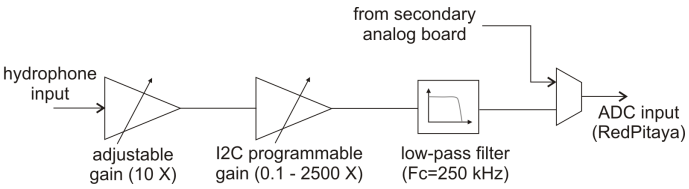


Fig. 4: A simplified block diagram of the analog programmable amplifier and filtering daughter board.

B. Digital implementation

The hardware system was based on the original design included in the RedPitaya distribution, maintaining the parts that implement the oscilloscope and the continuous signal recording, and removing all the other unused modules to free FPGA resources and facilitate the design optimization process to reach the $125\,\mathrm{MHz}$ main clock frequency. The input to the module implementing the oscilloscope is taken from a configurable decimator module that reduces the sampling frequency by factors equal to powers of two, which is used by the oscilloscope application to adjust the time-base. Maintaining this function is very convenient for the debugging process when performing field experiments, as analyzing the real signals being captured by the acoustic sensors. To further improve this feature, we have added a multiplexer at the input of the oscilloscope datapath to be able to use that same circuit and software application for observing other intermediate signals in different points of the TDOA calculator.

The TDOA calculator is implemented with four blocks shown in figure 5. Each block receives the signals of two

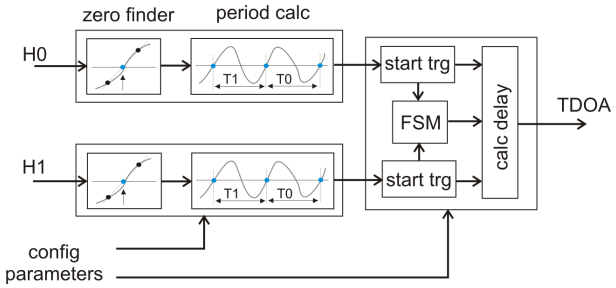


Fig. 5: Simplified block diagram of one TDOA calculator. Four modules are instantiated with different parameters.

hydrophones (H0 and H1), after passing through a digital high-pass filter with a cutoff frequency of 30 kHz to attenuate the low frequency components. Then, a zero-crossing comparator calculates continuously the time between two consecutive transitions by zero in the same direction (the rising edge in the block shown), representing the period in the input signal. The sequence of periods of each signal is analyzed to detect *Np* consecutive valid periods whose duration is above a threshold *Pth*. This is the trigger event that indicates the start of that signal. A finite state machine receives these trigger signals, analyzes the sequence of periods calculated and drives the delay calculator module that calculates the TDOA estimate applying the method described above.

Four of these modules are instantiated with different configuration parameters (*Np* and *Pth*) to produce 4 values for TDOA. Two modules measure the signal period using the rising edge and the other two use the falling edge. The four TDOA estimates are then read by an application running in the ARM embedded processor. This implements a filtering and voting process and calculates a final TDOA as the average of 2, 3 or 4 values read, after eliminating outlier results. This value is then used to estimate the direction of the sound source using the multilateration technique, which will be later used to define convenient routes of the surface vehicle for determining the location of the sound source.

The implementation in the Zynq 7010 programmable SoC, maintaining the oscilloscope and the DMA interface for real-time recording, occupies the FPGA resources shown in table I. The high utilization of the DSP slices is due to the pipelined implementation of the two high-pass FIR filters.

C. Experimental results

We have conducted a series of laboratory experiments in a test tank to validate the algorithm implemented and tune the parameters of the period calculator modules. The experiment consisted in obtaining 200 TDOA measurements for each of the 36 angles between $-90°$ and $-90°$ with a $5°$ step, using the 35 kHz acoustic pinger positioned at 2.5 m from the hydrophones. Figure 6 shows the results obtained as a cumulative percentage of the TDOA measurements considered correct (within a $\pm 10\%$ interval from the theoretical value), the values reported one period above and below the expected (labeled as "high" and "low", respectively) and the incorrect results. Besides most of the measurements being correct, the TDOA values differing one period above or below the correct one can be further fixed by subtracting or adding one period, taking into account the history of the previous readings.

TABLE I: FPGA resource usage (Zynq 7010).

Resource	Occupancy
LUT	33% (5759)
FF	17% (6001)
BRAM	27% (16)
DSP48	83% (66)

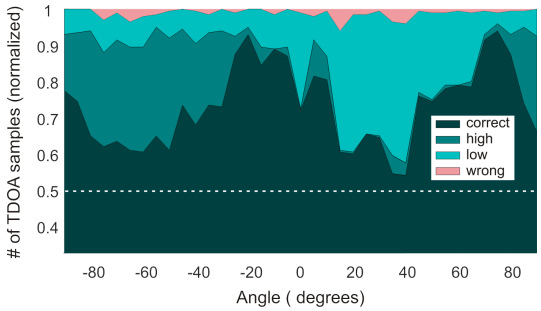


Fig. 6: Summary of the TDOA results obtained in laboratory experiments. The graphic shows the cumulative percentage of TDOA measurements for angles between $-90°$ to $+90°$.

V. Conclusions

In this paper we have presented the implementation of a calculator of the time difference of arrival (TDOA), as part of an underwater direction finder of an acoustic source. The system was implemented on a Zynq FPGA, receiving in two hydrophones the acoustic signal transmitted by an ultrasonic beacon. A simple mechanism based on the time domain analysis of the zero-crossings of the two input signals measure the relative time of arrival of the received signals. Preliminary experiments realized in a test tank have shown positive results, after tuning the configuration parameters used for the 4 instances of the TDOA calculator. Next, we will do a series of field experiments in different outdoor marine environments, to characterize the accuracy of the TDOA estimates, tune the configuration parameters and further refine the algorithm.

References

[1] G. Papadopoulos, M. F. Fallon, J. J. Leonard, and N. M. Patrikalakis, "Cooperative localization of marine vehicles using nonlinear state estimation," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 4874–4879.

[2] J. C. Alves and N. A. Cruz, "Fast-an autonomous sailing platform for oceanographic missions," in *OCEANS 2008*. IEEE, 2008, pp. 1–7.

[3] J. Chen, J. Benesty, and Y. Huang, "Time delay estimation in room acoustic environments: an overview," *EURASIP Journal on applied signal processing*, vol. 2006, pp. 170–170, 2006.

[4] S. Liu, C. Zhang, and Y. Huang, "Research on acoustic source localization using time difference of arrival measurements," in *Measurement, Information and Control (MIC), 2012 International Conference on*, vol. 1. IEEE, 2012, pp. 220–224.

[5] O. Le Bot, J. Mars, C. Gervaise, and Y. Simard, "Cross recurrence plot analysis based method for tdoa estimation of underwater acoustic signals," in *The sixth IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2015, pp. 325–328.

[6] "Red pitaya board," http://redpitaya.com, accessed: April 2016.

Sessão Regular V

Multiprocessamento

Moderação: José Carlos Alves
Fac. de Engenharia da Univ. do Porto / INESC Porto

An Implementation of MPI on FPGA for Distributed Memory Multiprocessing

Francisco Pires
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa, Portugal
francisco.pires@tecnico.ulisboa.pt

Mário Véstias
INESC-ID, ISEL
Instituto Politécnico de Lisboa
mvestias@deetc.isel.pt

Horácio Neto
INESC-ID, Instituto Superior Técnico,
Universidade de Lisboa, Portugal
hcn@inesc-id.pt

***Abstract*—In the context of distributed memory processing systems, this work presents an implementation of the Message Passing Interface (MPI) using FPGA soft-processors. This implementation consists not only in a C library but also in the configured FPGA hardware to support the communication between all the processors. Considering the limitations of the target devices, the low resource utilization is emphasized as well as the hardware scalability and the software reliability. Experimental results with several functions, including matrix-vector multiplication and backward substitution, on a 8-processor architecture validate the developed work and show that algorithms may be accelerated with good performance efficiencies.**

I. Introduction

Embedded computing applications have become very demanding over the years. In fact, a single core general-purpose microprocessor may not achieve the desired performance when running some specific algorithms, especially the ones with real-time constraints. Since the development of single core processors seems stagnated (the processor frequency has reached a limit due to power consumption and thermal reasons) but the number of transistors per chip increases every year, the multiprocessor approach is actually considered a viable solution to improve the performance of the most demanding embedded applications.

Multiprocessor Systems-on-Chip (MPSoCs) on Field Programmable Gate Arrays (FPGAs) exploit the parallelism of multi-processors in order to achieve better algorithmic performances for embedded systems.

Several studies and applications that solve these algorithmic problems using FPGA soft cores - or even heterogeneous systems with both soft and hard cores - have been frequently discussed. In these works, the proposed goals are frequently achieved but the application development cycle is never separated from the hardware implementation. This means that a software developer for these systems must also know the features of a hardware implementation, spending a considerable amount of his developing time writing low-level code to the application he wants to develop. Therefore, a lightweight version of the Message Passing Interface (MPI) [1] standard programming model that abstracts the communication between multiple soft processors on FPGA devices is proposed in this work.

The MPI is usually used for data exchange in a paradigm of distributed-memory high-processing computers. In fact, the MPI has been considered by diverse authors the de facto standard in this context for twenty years [2], [3]. The global levels of adoption of this programming interface are an obvious advantage since the embedded software developer does not need to learn a new library specification. Furthermore, a large amount of MPI applications, originally intended for clusters of workstations or supercomputers, may be ported to embedded systems using the implementation suggested here.

Since embedded systems are different from the usual cluster and supercomputer systems that MPI aims for, the implementation in this work approaches new questions regarding the small use of resources and system portability. Also, the implementation library must take into account the directives and the prototypes determined by the MPI-Forum in order to not confuse a software developer. Therefore, the main objectives of the work presented in this document consisted in studying and developing solutions that implement a low cost MPI interface in distributed memory soft-processors, considering scalability and resource utilization important constraints to be considered in this proposal.

Section II provides a description of the Message Passing Interface. The related work is described in section III. Section IV describes the proposed massage passing interface hardware and software implementation in FPGA. Section V describes and analyzes the results of the proposed MPI architecture in FPGA. Finally, section VI concludes the paper.

II. Message Passing Interface

The Message Passing Interface (MPI) was introduced in 1994 [1] to define a general library standard for parallel communication systems. The MPI protocol considers more than 100 functions though the great majority of the programs only use a small set of point-to-point and collective communication functions. In fact, a basic set of MPI functions provides the essential tools for the resolution of almost every parallelizable problem, namely:

- *MPI_Init* - the function where the entire MPI environment is started and important attributes, like the number of processors and the ranking of each processor, are set;
- *MPI_Comm_size* - return the number of processors to the user;
- *MPI_Comm_rank* - return the rank to the user;
- *MPI_Finalize* - shuts down the communication environment;

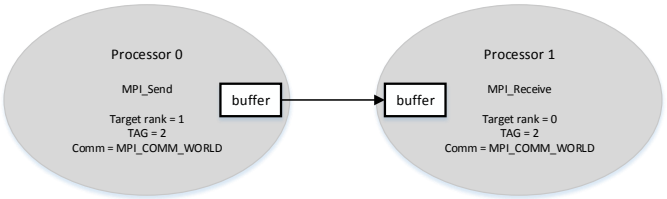


Fig. 1. Example of matched MPI message triples.

- *MPI_Send* and *MPI_Recv* - the core functions for the data transferring between multiple processors.

The implementation of the send and receive functions could adopt one of four different modes: the synchronous mode where both processors (the sender and the receiver) handshake and wait for each other to start the data transfer; the buffered mode, where the sender processor writes the data on the buffer and does not wait for the receiver to start the transfer; the standard mode, where it is up to the MPI implementation to determine whether the messages are buffered or not; and the ready mode where the sending operation only works if a receive request has been already posted.

Each MPI point-to-point function message is identified by a (target processor rank, tag, communicator) triple. The maximum value allowed for the rank is obviously related to the number of processors executing the application while the maximum tag value is defined by the implementation.

The communicator is a specific feature of the MPI standard that sets the communication context within or between groups of processors (intracommunicators and intercommunicators, respectively). The MPI message triple defines whether an *MPI_Send* message request matches or not an *MPI_Recv* request. Since the packet arrive order on some type of network is not deterministic, some message triples may arrive in a different order than originally expected, causing eventual matching problems. To solve this issue, the MPI standard suggests the use of queue buffers for the unexpected messages and pending receives (see figure 1).

Besides *MPI_Send* and *MPI_Recv* functions, collective functions are also defined by the MPIForum. The collective functions may be defined as facilities where multiple processors interact with each other using just a single function call. The main advantage brought by the collective functions is the less effort for the application developer to code certain problems. For example, with the basic set, if one processor wants to receive a data piece from every other processor and accumulate all the values received in a local variable, the application developer must call the *MPI_Send* function on the sender processors and implement a loop of *MPI_Recvs* on the root (receiver processor) where in each iteration the value received is accumulated in a local variable. With collective functions support, the application developer may simply call the *MPI_Reduce* function on every processor and all the reduction work is done internally (see figure 2).

Anther important function is the *MPI_Barrier*, used to synchronize all processors belonging to the same context of a communicator. Using just point-to-point MPI functions, the implementation of synchronization would require a significant

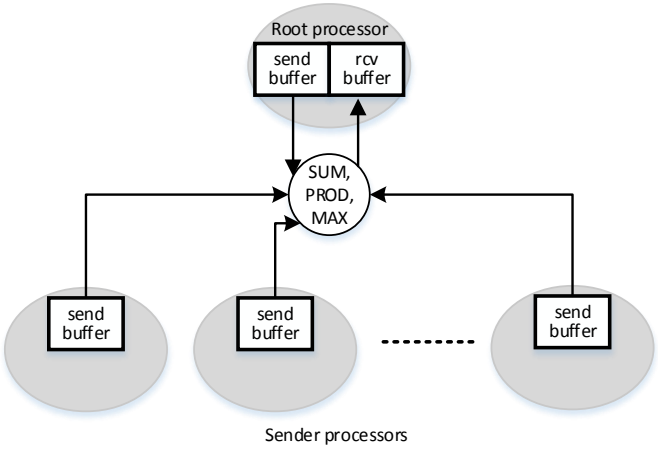


Fig. 2. *MPI_Reduce* function.

effort from the MPI user. With a single call to the *MPI_Barrier*, the complexity of synchronization is abstracted from the user.

Depending on the MPI implementation, some collective functions may have its internal code independent from the point-to-point functions, dealing directly with a lower layer of the software architecture. When these cases occur, the performance of those collective calls is usually optimized. Hence, the MPI user shall call, whenever possible, the collective functions instead of working over the point to point functions.

III. Related Work

Since the first MPI-1.0 reference document describing the MPI interface, several implementations and extensions have been proposed. While the majority of the implementations (like the FT-MPI [4], LAM/MPI, MPICH [5] or the OpenMPI [6]) aimed the state-of-the workstations and supercomputer systems running conventional operating systems, some work on MPI implementations for embedded systems has also been made, like the SCMP Multiprocessor [7].

Specifically on the FPGA field, some standard implementations for either a basic set of MPI functions or specific collective communication functions may be found. Initially, some implementations that intended to directly port the MPI libraries from the standard workstation versions to the FPGA systems (like the eMPI [8]) were presented. Their ports were considered too heavy, especially when in comparison with implementations designed specifically for the FPGA systems.

Other approaches did not port directly the MPI versions but relied on operating system facilities, which is a big disadvantage since many simple FPGA soft processors are not intended to run with operating systems. The works presented by Gao et al.[9], proposing specific hardware cores to accelerate the MPI Barrier function, and by Brightwell et al.[10], suggesting a new data structure to store and search more efficiently the MPI communication requests, are important developments on this topic.

Comparing with the implementation described in this paper, the works presented by Williams et al. [11] and Saldaña et al. [12] are the ones with the most similar objectives. In Williams, a reduced set of the MPI standard is implemented

for multiple Microblazes (Xilinx FPGA soft processors) connected to each other through Fast Simplex Links (FSLs). The number of FSL interfaces in a Microblaze processor is limited and, therefore, the system scalability is restricted to the maximum number of eight Microblaze FSL interfaces. Actually, the Microblaze soft-processors are also compatible with the AXI-Stream interfaces. Tailoring the system proposed by Williams to AXI-Stream would extend the maximum number of processors to 16. However, two memory elements (usually organized in a FIFO fashion) are required for each pair of linked Microblazes. Defining N_P as the number of Microblaze processors, the number of required memory elements for the MPI communication structure, M, is given by $N_P \times (N_P - 1)$. A design with 16 processors would, therefore, require 240 memory elements. This means that this architecture may be very memory demanding.

In [12] the problem of a limited number of processors is solved by using a custom network-on-chip (NoC). The interconnects (designated NetInterfaces by the authors) may receive an instruction from the receiver processor to select which sending processor is preferred in situations of contention. The eventual packets that arrive from other processors at the same time of the dealt transfer are retained in FIFO memories until the NetInferface is free. One memory element is needed per interface of each NetInterface. Two memory elements are also needed per Microblaze due to the FSL bidirectional interfaces used. Since there is one NetInterface per processor the total number of memory elements is $N_P \times (N_P - 1) + 2 \times N_P$, requiring more memory elements than in the [11].

Saldaña also proposes a version where a hardware accelerator engine is attached to each Microblaze in order to accelerate the tasks of receiving, analyzing and storing communication requests. This version requires three more memory elements per accelerator: two FIFOs to exchange data with the NetInterfaces and a queue memory to store unexpected MPI requests. The new value of M is therefore $N_P \times (N_P - 1) + 5 \times N_P$.

Looking at these equations and analyzing the architectures, it is clear that Saldaña's architecture is more scalable but it occupies much more memory and LUTs when compared to the architecture suggested by Williams. To benchmark his work, Saldaña implemented the Jacobi algorithm to solve the heat equation. Using a group of both Microblaze and PowerPC405 processors, maximum efficiency is achieved up to 10 processors while the maximum speedup obtained is around 27 using 40 processors. The authors justify the loss of performance with the limitations of resources of the FPGA device they used (Xilinx XC2VP100).

Over the years, Saldaña's work focused on heterogeneous systems where the FPGA soft-processors and specific hardware engines are able to communicate with x86 hard processors through MPI requests and data sent to a shared memory [13]. Dedicated hardware implementations for the collective *MPI_Bcast* and *MPI_Reduce* functions were also upgrades verified [14]. All these improvements made this MPI proposal in a complete and efficient solution, however, all the intellectual property (IP) cores implemented and all hardware used is too heavy for medium and low-cost FPGA systems.

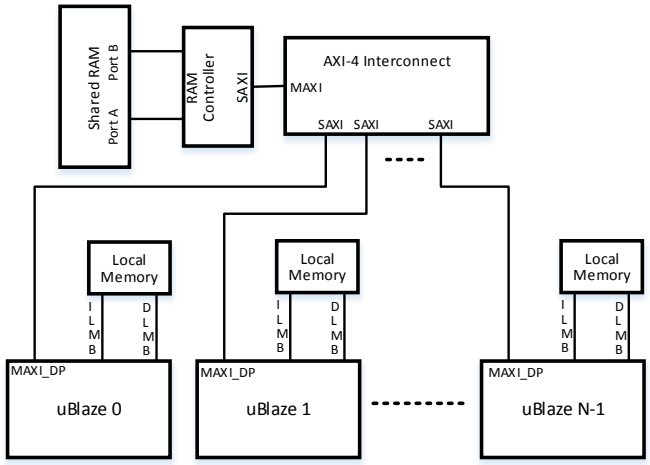


Fig. 3. Generic hardware architecture used to implement the MPI software.

IV. Hardware and Software Design of the Distributed Memory Multi-Processing System

In the following sections, we describe the proposed multi-processing architecture and the design of the software MPI routines over this architecture.

A. Hardware Design of the Multi-Processing Architecture

Since the target is a Xilinx FPGA, the Microblaze soft-processor was chosen as the processing core of the architecture. Microblaze is compatible with the standard C language libraries, do not take an excessive LUT area (allowing implementations on very low-cost FPGA devices) and provide interfaces compatible with the required protocols (AXI-4 and AXI-Stream) for efficient external communication.

The architecture defined to link all the processors consists in using an AXI-4 Interconnect that allows all the processors to access a shared RAM memory. It's important to note that this AXI RAM memory is only used for the MPI functions, each soft-processor remains with their local BRAM memories to store the application code, heap and stack (see figure 3).

Instead of using a shared memory, a system where all the processors were linked through AXI-Streams was also considered. In this case, the number of memory resources needed was considered too high, as can be seen in the related work section. The high resource requirements are even more evident when a custom NoC is implemented and an interconnect is needed for each processor.

Following with the shared RAM architecture, some hardware cores with the objective to accelerate the asynchronous point-to-point MPI communication functions were considered (the processors could execute some portion of computational work while these engines would send and receive data). After some experiments, it was concluded that these cores had an excessively complex computing work accessing and processing the BRAM memory. In addition, the resources to implement this structure would limit the scalability of the system (it would take a considerable LUT space and would require at least two additional FIFO memories per processor).

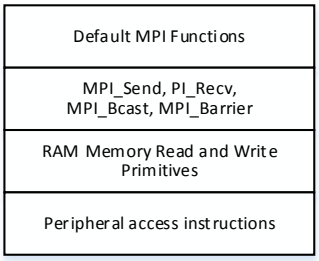


Fig. 4. Software stack of developed library.

The AXI-4 Interconnect, optionally, provides FIFO buffers to accelerate burst transfers. Since the burst mode is not compatible with the AXI DP interface used by the Microblazes, this shared memory architecture only needs one memory element (M = 1) to implement the communication structure of a fully working MPI library. An AXI Interconnect has a limit of 16 slave interfaces. Thus, when more than 16 processors are inserted, a second AXI-Interconnect level must be inserted. This fact does not have a big impact in the resource utilization since an AXI-Interconnect takes much fewer LUTs than a Microblaze processor.

B. Software Design of MPI Functions

Since we are targeting embedded system devices that may have very scarce resources, the software design had to take into account the space that the own code occupies focusing only on the essential MPI features. The final size of the developed MPI code was, therefore, around 20 kB (actual workstation implementations occupy about 100 MB). The developed library for the C programming language is organized according to a layered approach (see figure 4).

While the point-to-point *MPI_Send* and *MPI_Recv* functions and the optimized collective *MPI_Bcast* and *MPI_Barrier* functions were built directly over the BRAM memory access macros, the other MPI functions were developed on a higher level abstraction. Therefore, these functions call the lower level MPI functions instead of directly calling the BRAM memory read and write macros. This layer structure eases the system portability and future software improvements.

Another important point of this software implementation is the way how the data stored in the shared BRAM is organized and how that memory organization can be related to the MPI communication modes defined by the MPI-Forum. Considering that N_P processors are being used, the software logically divides the memory in $N_P \times (N_P - 1)$ blocks, each one consisting in the communication zone for a combination of two processors: a sender and a receiver (see figure 5).

The logical division by blocks eases the search process since multiple processors are trying to read and write in the same memory. The logical combinations depend on the role of the processors, e.g. processor 0 being a sender and processor 1 a receiver is a different combination of processor 1 being a sender and processor 0 a receiver. With this approach, the shared memory is both a link and a synchronous communication buffer. With the exception of the logical block receiver0-sender1 (the first in the memory), each block contains an area

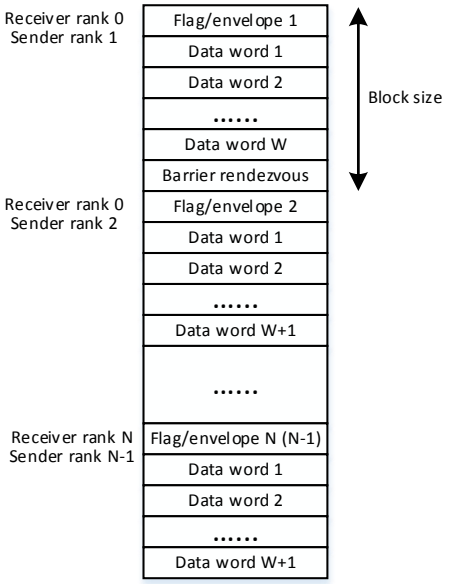


Fig. 5. Logical organization of the shared memory.

of one integer (4 bytes, 1 memory position), where the TAG envelope of the message is set by the sender, and an area (the remaining block size) reserved for data to be transferred. The first block (receiver0-sender1) has the same structure but an additional position in order to do a semaphore for the *MPI_Barrier* function.

The data transfer area not only allows all MPI datatypes that take 4 bytes but also the 8 bytes datatypes (*MPI_LONG_LONG* and *MPI_DOUBLE*). Each 8-byte value is halved in 2 memory data words. The communication mode may be considered the standard since, depending on the situation, a sender processor may or not send the data without waiting for the receiver processor. The conditions that hold back the sender are when the receiver did not read yet the latest data transmission and when the data to send is bigger than the block size.

The implementation of each MPI function developed is now described:

MPI_Init - In this function, every processor computes the size for each communication block of the memory, sets different environment variables (like the group size or the process rank) and finally synchronizes with the other processors by calling the *MPI_Barrier* function.

This initialization function relies on the Programmable Logic reset of the first launched processor to initialize the shared memory.

MPI_Comm_size - This function just returns the value of the already set variable of the number of processors;

MPI_Comm_rank - Returns the value of the processor rank. This value is set by the Xilinx tools when the hardware is implemented;

MPI_Send and **MPI_Recv** - Since the memory is logically divided into blocks, when a processor wants to send a set of values, goes to the attributed block and checks in the

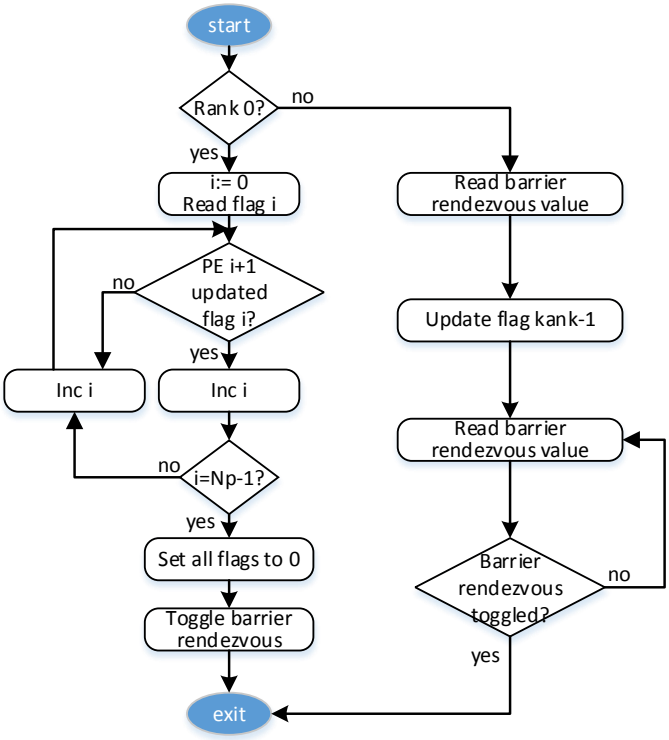


Fig. 6. Flowchart of the developed MPI Barrier function.

envelope position if the latest transmission was completed. If so, the *Send* function writes the data and changes the value of envelope position to the message tag. On the receiver side, the processor is reading in the appropriate block in the envelope position while it does not contain the tag value. When the value appears, the receiver reads the data and sets that envelope as free;

MPI_Barrier - The Barrier is implemented with the help of the envelope positions used for the point-to-point communications and a specific semaphore/rendezvous position (see figure 6). All the processors, except the master processor 0, set to a new value the envelope position of the block where these processors act as senders and the processor 0 act as receiver. The processor 0 reads those positions and, after noticing that all those positions were changed, it toggles the value of the dedicated barrier rendezvous position meaning that all the other processors are clear to leave the barrier;

MPI_Bcast - Though a broadcast function is easily implemented over the already developed point-to-point functions, a different and more efficient approach was taken in order to exploit the advantages of the hardware configuration of the system. In this implementation, all the processors synchronize and the root processor writes once in a block. The receiving processors store the data from that block and inform the root processor that the data has been read;

MPI_Reduce, **MPI_Gather**, **MPI_Scatter** - These collective communication functions were built over the point-to-point communication routines. Since it is not expected a significant performance improvement from implementations independent from those point-to-point functions (taking into account the defined architecture), this approach saves an important amount of code space, which is vital for an embedded implementation where the memory resources are very limited. The Reduce operations supported are the **MPI_SUM** and the **MPI_PROD**.

MPI_Finalize - This function sets the defined MPI global variables to the original state and calls the MPI Barrier where each process waits for the others to finish the MPI session.

TABLE I. FOOTPRINTS OF THE IMPLEMENTED MPI FUNCTION SETS IN THE MICROBLAZE.

Function set	Functions Implemented	Footprint
Basic set	MPI_Init, MPI_Comm_size, MPI_Comm_rank, MPI_Send, MPI_Recv, MPI_Finalize, MPI_Barrier	10.3 KB
Collective set	Basic set functions, MPI_Bcast, MPI_Reduce, MPI_Gather, MPI_Scatter,	25.4 KB

Table I shows the footprint of each set of functions.

There is a significant increase of the footprint when the set that also implements the collective functions is used. The main reason for this increase is the presence of the MPI Reduce function which is responsible for 60 % of this increase. In fact, allowing both the sum and product as reduce operations for 4-bytes and 8-bytes floating-point and non floating-point datatypes required a significant amount of assembly code. In this table, the MPI Barrier primitive was considered a function belonging to the Basic Set because the fundamental functions that start and finish the MPI session use the barrier to synchronize all the processors.

V. IMPLEMENTATION AND RESULTS

The system was implemeted and tested in a Xilinx Zynq-7020 All Programmable SoC FPGA. The Zynq-7020 provides a heterogeneous environment where FPGA programmable logic (Xilinx Artix-7 technology) is connected to an ARM Cortex-A9 based processing system. The software tools used for system prototyping and testing were Xilinx Vivado 2014.4 and Xilinx Software Development Kit (SDK).

The hardware testbed has eight non-cached Microblazes, each one with 32 kB of internal BRAM memory. All the internal data and instructions were stored on these memories. The shared memory was set with 8kB of BRAM memory and a clock frequency of 100 MHz was used for all the programmable logic. This configuration with eight Microblazes utilizes 66 BRAMs (47%) and 10541 LUTs (20%) available on the Zedboard.

To test the implementation of the library, several algorithms with different features were implemented and tested. Fo space reasons, we only present here two of them (1) matrix-vector multiplication and (2) backward substitution.

A. Matrix-Vector Multiplication

The matrix-vector multiplication algorithm was developed taking into account the scarce resources of the FPGA device used. This mathematical operation is simply formalized as Ax = y, where A is the input matrix, x is an input vector and y is the computed output vector.

Due to the limitations of the FPGA in terms of BRAMs, the Microblazes usually cannot store the complete matrix A. To overcome this problem, the processors only store a single row of A and the x vector to compute the corresponding y

TABLE II. EXECUTION TIMES FOR MATRIX-VECTOR MULTIPLICATION.

Multiple uBlaze execution times (ms)					
	Number of uBlazes				
Size of matrix	1	2	4	6	8
50 × 50	22	12	9	7	6
100 × 100	88	47	30	25	23
150 × 150	200	106	65	53	50
200 × 200	357	189	118	97	87
250 × 250	559	296	180	148	137
300 × 300	806	427	264	214	196

TABLE III. SPEEDUPS FOR MATRIX-VECTOR MULTIPLICATION.

Multiple uBlaze speedups				
	Number of uBlazes			
Size of matrix	2	4	6	8
50 × 50	1.8	3.1	4.0	4.0
100 × 100	1.9	3.3	3.9	4.4
150 × 150	1.9	3.3	3.9	4.4
200 × 200	1.9	3.1	4.0	4.5
250 × 250	1.9	3.1	4.0	4.4
300 × 300	1.9	3.2	4.0	4.5

element. At each iteration, the processors receive a new row of A and discard the last one. After all the processors compute their corresponding *y* elements, the entire *y* data is gathered to the master processor.

The execution times (see table II) and speedups (see table III) for single precision data were obtained for different sizes of the matrix A and number of processors.

We have also determined the performance efficiency (obtained performance/peak performance) of the architecture (see table IV)

With two processors the efficiency is above 90 % (the efficiency is only limited by the time spent in communication functions). With more than two processors, the efficiency starts decreasing. The cause of this new effect is the contention verified on the AXI-Interconnect (which uses a round-robin policy in these cases).

B. Backward Substitution

Considering a system of equations defined in the matrix form Ax = b, the backward substitution is the process of solving that system of equations when the matrix A is a triangular superior matrix. The algorithm to solve a problem of this kind is generically described by the C code in Listing 1.

Listing 1. Backward substitution example

```
for (i = n-1; i >= 0; i++)
  b[i] = b[i]/A[i][i];
  for (j = 0; j < i; j++)
    b[j] = b[j] - b[i] * a[j][i];
```

The parallelization of the back substitution is less efficient because of a poor scalability. This poor scalability is owing to the fact that only the internal loop of the algorithm is parallelizable. Therefore, every processor must run all the iterations of the external loop and compute/receive the b[i] value (pivot) of the external loop. The broadcast of this value is a critical point in the parallel algorithm and, because of this fact, the optimized MPI Bcast function had an ideal environment for testing in this algorithm. To complete the parallel algorithm, there is the usual data gathering of the vector *b* to the master processor after the computation. Tables V and VI shows the results obtained with this benchmark.

We have also determined the performance efficiency (obtained performance/peak performance) of the architecture (see table VII)

The results show that the best performance and maximum speedup was achieved with 6 processors followed by the 8 processors architecture. The high communication/ computation ratio, and consequent contention in the architecture interconnect, and a hard parallelization pattern limit the system scalability for backward substitution.

TABLE IV. SYSTEM EFFICIENCY FOR MATRIX-VECTOR MULTIPLICATION.

Number of uBlazes	Average Eff.	Maximum Eff.
2	0.94	0.94
4	0.73	0.78
6	0.60	0.63
8	0.50	0.51

TABLE V. EXECUTING TIMES FOR BACK SUBSTITUTION.

Multiple uBlaze execution times (ms)					
	Number of uBlazes				
Size of matrix	1	2	4	6	8
100 × 100	46	30	20	18	20
200 × 200	183	116	74	63	70
300 × 300	414	257	161	136	151
400 × 400	738	454	282	263	261
500 × 500	1155	707	437	364	403
600 × 600	1669	1018	627	520	575

TABLE VI. SPEEDUPS FOR FOR BACK SUBSTITUTION.

Multiple uBlaze speedups				
	Number of uBlazes			
Size of matrix	2	4	6	8
100 × 100	1.51	2.30	2.56	2.30
200 × 200	1.58	2.47	2.90	2.61
300 × 300	1.61	2.57	3.04	2.74
400 × 400	1.63	2.64	3.17	2.87
500 × 500	1.71	2.84	3.24	2.89
600 × 600	1.64	2.66	3.21	2.90

VI. CONCLUSIONS

The main objectives of the work presented in this document consisted in studying and developing solutions that implement the MPI interface in distributed memory soft-processors on FPGA.

In order to minimize the internal memory size of each processor, the implemented MPI library was intended to be as simple as possible (but compatible with every relevant datatype). The developed footprint of the MPI basic set is approximately 10 kB and the remaining implemented collective functions do not increase significantly that footprint value.

Good performance and efficiencies were achieved for two tested algorithms with distinct communication requirements.

TABLE VII. SYSTEM EFFICIENCY FOR BACKWARD-SUBSTITUTION.

Number of uBlazes	Average Eff.	Maximum Eff.
2	0.80	0.82
4	0.64	0.67
6	0.50	0.53
8	0.34	0.36

Future iterations of this work shall provide hardware improvements in order to minimize the effect of the interconnect contention, like using clusters of processors each with its shared RAM. Extend the compatible MPI functions, like *MPI_Allgather*, *MPI_ISend* and *MPI_IRecv*, to improve portability. Consider the implementation of some functions in hardware, like *MPI_Reduce*.

ACKNOWLEDGMENT

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references PTDC/EEA-ELC/122098/2010 and UID/CEC/50021/2013.

REFERENCES

[1] MPI Forum, http://icl.cs.utk.edu/ftmpi/ [Accessed in September, 2015]

[2] S. Lakshminarayana, S. Gosh, and N.Balakrishnan, "Implementation of MPI over HTTP", in 7th International Conference on High-Performance Computing and Networking, 1999, pp.1299-1302.

[3] E. Marques, F. Martins, V. Vasconcelos, N. Ng and N. Martins, "Towards deductive verification of MPI programs against session types", in Programming Language Approaches to Concurrency- and Communication-cEntric Software, 2013, pp.103-113.

[4] FT-MPI,http://icl.cs.utk.edu/ftmpi/ [Accessed in September, 2015]

[5] MPICH, https://www.mpich.org/ [Accessed in September, 2015]

[6] OpenMPI, https://www.open-mpi.org/ [Accessed in September, 2015]

[7] J. Poole, "Implementation of a Hardware-Optimized MPI Library for the SCMP Multiprocessor", MSc Thesis, 2004.

[8] T. P. McMahon and A. Skjellum, "eMPI/eMPICH: Embedding MPI," in MPI Developers Conference, 1996, pp.180-184.

[9] S. Gao, A. Schmidt, and R. Sass, "Hardware implementation of MPI Barrier on an FPGA cluster", in International Conference on Field Programmable Logic and Applications, 2009, pp.12-17.

[10] R. Brightwell, K. Hemmert, R. Murphy, A. Rodrigues and K. Underwood "A Hardware Acceleration Unit for MPI Queue Processing", in Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, 2005 pp.96-109.

[11] J. A. Williams, I. Syed, J. Wu, and N. W. Bergmann "A Reconfigurable Cluster-on- Chip Architecture with MPI Communication Layer", in Proceedings of 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2006, pp.350-352.

[12] M. Saldana and P. Chow, "TMD-MPI: An MPI Implementation for Multiple Processors Across Multiple FPGAs", in Proceedings of the 16th International Conference on Field-Programmable Logic and Applications, 2006, pp.1-6.

[13] M. Saldana, A. Patel, C. Madill, D. Nunes, D.Wang, H. Styles, A. Putnam, R. Wittig and P. Cho, "MPI as an Abstraction for Software-Hardware Interaction for HPRCs", in Second International Workshop on High-Performance Recofigurable Computing Technology and Applications, 2008, pp.1-10.

[14] Y. Peng, M. Saldana and P. Chow, "Hardware Support for Broadcast and Reduce in MPSoC", in International Conference on Field Programmable Logic and Applications, 2011, pp.144-150.

FPGA implementation of a Multi-Processor for Cluster Analysis

José Canilho
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
jose.canilho@tecnico.ulisboa.pt

Mário Véstias
INESC-ID, ISEL Instituto Politécnico de Lisboa
mvestias@deetc.isel.pt

Horácio Neto
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
hcn@inesc-id.pt

***Abstract*—In this paper, a hardware/software architecture is proposed to efficiently execute the widely known and commonly used K-means clustering algorithm. The architecture splits the algorithm's computational tasks by both hardware and software, with custom built hardware accelerators performing the most demanding computations. By doing so acceleration is achieved not only by performing the computationally demanding tasks faster but also by parallelizing different independent steps of the algorithm through both hardware and software domains. A prototype was designed and implemented on a Xilinx Zynq-7000 All Programmable SoC. The solution was evaluated using a set of relevant benchmarks and speed-ups over a software-only solution were measured. A maximum speed-up of 10.1 was observed using only 3 hardware processing elements. However, the system is fully scalable and therefore capable of achieving much higher speed-ups simply by increasing the number of processing elements.**

Keywords - Clustering, K-means, Hardware/Software Co-design, Hardware Acceleration, Systems on Chip, Custom Hardware Design

1. Introduction

Cluster analysis, or clustering, consists in grouping a set of objects in such a way that objects which are similar to one another according to some metric belong in the same group (called a cluster). It is one of the most useful and used task of exploratory data mining, and can be applied in a wide variety of fields. Being an unsupervised learning method, a previous system training with a training set is not required and classification can be performed directly from the unclassified dataset.

Clustering can be a rather lengthy process, when several iterations through the dataset are needed and when the dataset itself is large in both number of points and dimensionality. The increasing scientific interest in *Big Data* analysis (which is a broad term to define very large datasets) and real-time clustering emphasize the need of producing clustering results of large datasets while fulfilling several timing constraints. Even the most simple and straightforward algorithms may sometimes take too long to produce the results. This problem can be attenuated by the use of acceleration techniques, in order to speed up clustering algorithms.

The acceleration of algorithms has been a hot topic in the scientific community for quite some time and its importance in clustering algorithms is increasing dramatically with the equally increasing demand for classifying large datasets quickly. Several acceleration techniques have been applied to clustering algorithms, from parallel and distributed computing [8][16] to the use of GPUs [7][11][15] and hardware accelerators [10][5][12][6][14][9].

The remainder of the paper is organized as follows. Section 2 will explicitly detail the targeted clustering algorithm. Section 3 covers the previous related work. Section 4 presents the developed solution. Section 5 presents a theoretical analysis and experimental results conducted for the developed solution. Section 6 summarizes the paper.

2. The K-means clustering algorithm

The targeted clustering algorithm was the K-means algorithm. It is one of the most simple algorithms capable of performing the clustering task. Despite its simplicity, it is still one of the most widely used clustering algorithms, due to its easy implementation and fast execution time. The algorithm uses a centroid model. It separates the data into a set of clusters, each one represented by the mean vector of all the datapoints within the class. Each datapoint is classified into the cluster whose center is closest to it. The distance is usually judged using the euclidean distance as a metric, although some other types of metrics can be applied [2]. After an initial position is attributed to each center, the algorithm starts updating the position of each center in an iterative fashion. Each iteration is divided in two main steps:

1) **Assignment step:** each datapoint is assigned to the nearest center, given the chosen distance metric
2) **Update step:** after all the datapoints are assigned, the centers are re-calculated. The new positions correspond to the mean of all the datapoints within each cluster

The algorithm ends when the centers' positions stop changing between iterations.

Algorithm 1 presents the pseudo-code for the K-means algorithm. Lines 10-22 represent the assignment step and lines 23-25 represent the update step. The iteration is repeated until the centers stop changing. The overall algorithm

Algorithm 1 K-means pseudo-code.

```
1: centerInitialization();
2: repeat
3:    for each center k do
4:       classAccumulator[k] = 0;
5:       classCounter[k] = 0;
6:    end for
7:    for each datapoint d do
8:       classifications[d] = -1;
9:    end for
10:   for each datapoint d do
11:      minDistance = Infinity;
12:      for each center k do
13:         currentDistance = distance(d, k);
14:         if currentDistance < minDistance then
15:            minDistance = currentDistance;
16:            closestCenter = k;
17:         end if
18:      end for
19:      classifications[d] = closestCenter;
20:      classAccumulator[closestCenter] += d;
21:      classCounter[closestCenter]++;
22:   end for
23:   for each center k do
24:      newCenter[k] = classAccumulator[k] / class-
         Counter[k];
25:   end for
26: until (centers don't change)
```

presents a complexity of $O(nkdi)$, with n being the number of datapoints, k being the number of clusters, d being the dimensionality of each datapoint and i being the number of iterations needed for convergence.

3. Related Work

Given the importance of accelerating clustering algorithms, and specifically the K-means algorithm, multiple studies have been conducted targeting the algorithm's acceleration. Some of the covered studies focus on algorithmic transformations capable of speeding up the clustering process. One important possible modification to the standard algorithm lies on the distance metric used to evaluate the datapoints. Estlick, M. *et. al* [2] proposed the use of the L_1 norm (also known as the *Manhattan norm*), instead of the standard euclidean distance, in hardware based solutions. The Manhattan norm is characterized as the sum of the absolute value of the difference in all coordinates between the datapoint and the center. Its mathematical expression is presented in expression (1). By using the Manhattan norm instead of the euclidean norm, the need for multiplications is eliminated. This makes the Manhattan norm more suitable for custom hardware solutions. The authors tested the influence of the new norm in the classification results and a small and negligible impact on the results was observed. In comparison, the L_∞ norm, which also doesn't require multiplications, had a much more detrimental impact on the results, making it not as appropriate for the algorithmic transformation.

$$\sum_{i=1}^{D} |x_i - c_{l(i)}| \tag{1}$$

Several variants to the K-means algorithm can be found in the literature, which can vary very slightly to the original algorithm. Although they do not change the essence of how the classification is performed, some changes can prove to be beneficial in a small handful of applications [13]. One known variant which maintains the algorithm suitable for any *Big Data* application was suggested by V. Faber [3], called the *Continuous K-means*. In this variant, only a portion of the dataset is analysed per iteration. This modification is only suitable, however, if the dataset is large enough, making any subset chosen a viable representation of the entire dataset. By doing so, the time per iteration is decreased. Further testing showed that the Continuous K-means can converge up to 10 times faster than the standard algorithm, although no mention was made regarding the quality of the classification results.

3.1. GPU Computing

The proposed GPU implementations for the K-means algorithm take advantage of the several available threads blocks and parallelize the assignment step by having each thread compute the classification of one datapoint. In order for the whole algorithm to be completed, the host CPU is often used to take care of the remaining tasks.

In 2007, Che S. *et. al* [1] developed a GPU implementation adopting the CUDA programming model. The datapoints were transferred to the GPU and the assignment step was delegated to the GPU whilst the CPU gathered the results and performed the center updates. The implementation had large memory transfer overheads, specially due to the CUDA version used at the time, which only allowed synchronous DMA transfers (meaning that a transfer had to be completed in order for the computation to start). The results report only a comparison between the execution time of the distance calculation for all datapoints, which was accelerated up to 8 times, when compared to the CPU-only implementation. The results report datasets of different sizes, but no mentions are made to the used number of clusters and data dimensionality.

In 2008, Farivar, R. *et. al* [4] presented a GPU based solution, also designed using CUDA. The solution parallelizes the algorithm in a similar way as in [1]. Memory transfer overheads were improved and testing was performed on a dataset with 1 million 1-D datapoints and 4 clusters. A speed-up of 13 was observed, although the authors claim that a speed-up of up to 68 could be achieved, using the top of the line GPU at the time.

Following Farivar's work a year later, [15] reports the accelerating approach followed by Zechner, M. *et. al*, which also relies on the CUDA programming model. The same parallelization method was applied, this time on a more

powerful GPU. For large dataset sizes and dimensionalities, the maximum speed-up observed in the experiments was 14, as the time spent labelling the datapoints grows significantly. These results end up contradicting Farivar's expectations, however, for the use of more powerful GPU and instead emphasize the communication bottleneck.

3.2. Custom Hardware Designs

Most of the custom hardware solutions for the K-means algorithm rely on pure hardware implementations of the algorithm, where both steps of the K-means algorithm are mapped into distinguishable blocks of hardware, each one responsible for only one task. The recent introduction of SoC FPGAs allow the implementation of efficient hardware/software co-design architectures within a single chip. With embedded *hard-core* processors, among other useful components, and efficient bridging between hardware and software domains, high performances can be achieved.

In 2000, Lavenier, D. [10] presented an FPGA implementation of the algorithm, suited for the analysis of hyperspectral images, fed to the hardware via a stream of pixels. The assignment step was delegated to a systolic process array (SPA) with its size equal to the number of clusters. Each element computes the Manhattan distance to its center and passes the result along the array. At the end of the array, the closest center and its distance is obtained. In order for the complete algorithm to work, a host CPU is needed to perform the remaining tasks and transfer the datapoints to the FPGA when needed. Using DMA transfers between the host and the FPGA, the maximum observed speed-up was 336 when compared to a CPU-only implementation, using 256 clusters. However, no mentions are made regarding the type of dataset used nor the characteristics of the CPU-based implementation.

Later, in 2003, Gokhale, M. *et. al* [5] presented a hardware architecture which computed the distance calculation of the datapoints to the center and delegated the center updates to a processor. The design used several SPAs to compute the datapoint classifications in parallel and used 32-bit wide BRAMs in the transfers between processor and user logic. In the initial stages of development, both an ARM hard-core and a NIOS soft-core were considered, although testing was only performed with the ARM hard-core. The maximum speed-up observed was 11.8, when compared to a CPU-only implementation running at 1Ghz.

Two years later, Wei-Chuan Liu *et. al* [12] proposed a hardware architecture capable of performing the algorithm in its entirety. The framework was named *KACU*: K-means with hArdware Centroid Updating. The architecture and the overall work performed gave emphasis to the center updating step and the algorithm applied was the continuous K-means [3] instead of the standard algorithm, since it provides more update steps, given the number of the datapoints evaluated per iteration. The main architecture consists of an SPA, with the addition of the accumulators needed to perform the mean of each center and the divider block, along with some extra control blocks. The new centers only replaced the old ones after the control unit signalled so. The remaining tasks regarding control are handled by a host CPU. The architecture was designed in an Altera FPGA, using a NIOS *soft-core* processor running at 50 Mhz, for the control unit. Better speed-ups were achieved for small subsets, since the execution time of the update step becomes less negligible. A maximum of 5.6 was observed compared to the solution with software centroid updating.

Following the same line of thought, Wang, X. *et. al* developed another custom hardware solution, targeted for the standard K-means algorithm applied in the analysis of multi-spectral images [14]. This solution also delegated all the computational tasks to the custom hardware. The host processor is only needed for the initial center selection and to upload the datapoints and the centers to the memory within the FPGA. The Manhattan norm was the chosen distance metric. From all the custom hardware solutions covered so far, no mention was made regarding the numerical representation chosen. In Wang's implementation, fixed-point arithmetic was used, except in the center updates: floating-point converters were used in both inputs and outputs of the divider block and the new centers were computed in floating-point representation. The timing results obtained by the authors focused on complete execution time rather than the time taken per iteration of the algorithm. For 50 iterations, a speed-up of 11 was observed, when comparing to a full CPU implementation running on an Intel Pentium 4 operating at 3.2 Ghz. It is also claimed that a higher number of iterations will improve the speed-up results, although most K-means applications will converge in few iterations.

Hussain, H. et. al [6] proposed a specific hardware implementation of the K-means algorithm targeting bioinformatic applications. The architecture used fixed-point arithmetic and introduces a different way of parallelizing computation. Instead of splitting the dataset throughout several processing elements, the centers are split. Each element computed the distance of all datapoints to a subset of the centers and a final reduction block merged all results together, followed by a single divider used in the update step. An architecture capable of holding 8 centers was implemented in a Xilinx XC4VLX25-10SF363 FPGA, running at 126 Mhz. A speed-up of 10.3 was observed, when comparing to a CPU-only implementation running on an Intel Core2 Duo 3.0 GHz CPU. If the FPGA resources allow for the architecture to be replicated several times, then the speed-up results can improve further.

More recently, in 2013, Kutty, J. *et. al* [9] presented a high speed configurable FPGA architecture. Similarly to the previous approach, the architecture split the distance calculation of a single datapoint through several distance calculation blocks, one per each center. These blocks computed the Manhattan distance and delivered the results to a tree comparison block. The main difference to Wang's architecture is that a dividing block is used for each center. The dataset and the initial centers are stored *a priori* in the BRAMs within the FPGA. The architecture was implemented in a Xilinx Virtex-6 FPGA. A operating clock frequency of 400 Mhz was achieved for up to 9 clusters.

The clock frequency decreases with the increase of number of clusters, although a frequency of 300 Mhz can still be achieved for 32 clusters. The results focused on the occupied area of the architecture, rather than on the actual computation time, and no speed-ups were mentioned for the complete algorithm's execution, unfortunately. The only inference that can be made regarding the computation time is the link between the clock frequency and the architecture's throughput, which was claimed to be of 1 classification per clock cycle.

4. Architecture

The proposed architecture in this paper is a hardware/software solution, implemented in a Zynq-7020 All Programmable SoC, using 32-bit floating point as the numerical representation. The Zynq-7000 device family incorporates FPGA resources and software programmability through the use of two ARM Cortex-A9 hard-core CPUs. These SoC devices are appropriate for the implementation and deploy of hardware/software solutions as they already offer a clear separation between hardware and software domains. The description of the developed architecture starts with an overview of the target device, followed by a top-down description of the solution.

4.1. Target device

The Zynq-7020 device has two main distinguishable blocks: the Programmable Logic (PL) and the Processing System (PS). The PL holds the FPGA resources and the PS represents the software domain. The PS includes the two ARM CPUs, as well as the available caches: two 32KB L1 caches (one per CPU) and a 512KB shared L2 cache. Other important components used throughout the devised architecture are the On-Chip Memory (OCM) present in the PS's APU, the General Interrupt Controller (GIC) capable of receiving interrupt signals from the PL and the DDR controller. All transfers between the PS and the PL are accomplished using the AXI protocol. The Zynq-7020 PS/PL interface offers several AXI3 ports, ranging from one of three different types. The available ports consist of:

- 4 general-purpose (GP) 32-bit ports (2 masters and 2 slaves), suitable for control functions and peripheral access
- 4 high-performance (HP) 64-bit slave ports, suitable for large data transfers
- 1 64-bit accelerator coherency port (ACP), suitable for large transfers with cache coherency implications

4.2. Hardware/Software Architecture

The developed architecture utilizes both the PS and the PL. The ARM cores not only manage some control tasks needed to perform the algorithm but also perform computational tasks themselves. The PL implements various hardware components. All components are either supplied by the Xilinx LogiCORE IP catalog or custom designed. In particular, the PL will feature custom made hardware accelerators, responsible for the computation in the PL side.

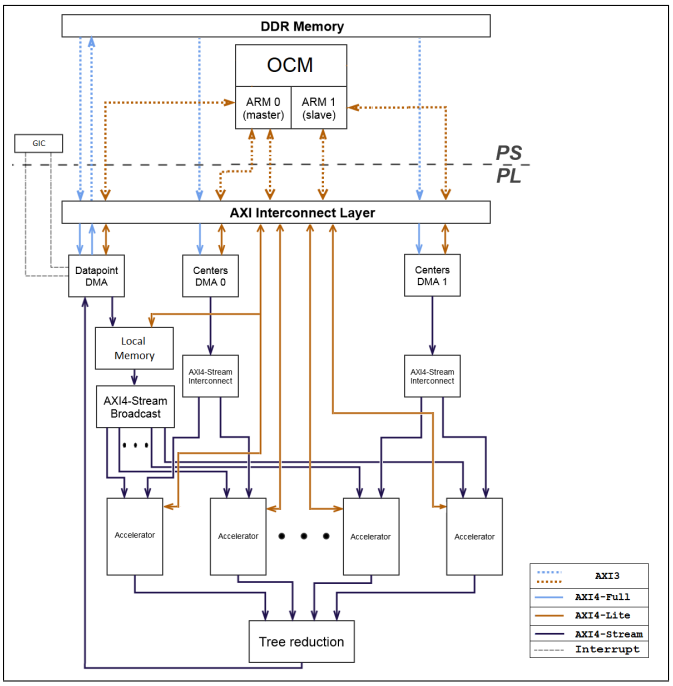


Figure 1. Top Level design architecture

As referred, the two main steps of the K-means algorithm are the assignment step and the update step. The assignment step classifies the datapoints and updates the accumulators and centers for each cluster. The update step re-calculates the centers of each cluster. Table 1 summarizes the tasks each computational element has to perform per iteration.

TABLE 1. TASK DELEGATION BETWEEN HARDWARE AND SOFTWARE DOMAINS

Task		Performed by
Assignment Step	Datapoint Classification	HW Components
	Accumulations/Countings	ARM
Update Step		ARM

The top level design is portrayed in figure 1. The design provides a first insight on what components feature in the architecture, in both the PS and the PL, and how these are interconnected. The DDR memory holds the dataset, as well as the centers and some intermediate results such as the classification results per iteration. Still on the PS side, the 2 ARM cores are featured, along with the OCM that holds the binary executables for both processors. Both the DDR memory and the ARM cores interact with several AXI3 ports. The blue dashed lines represent connections meant to transfer data and use the HP ports. The orange dashed lines represent connections meant for control functions and use the GP ports. Still in the PS side, the GIC receives two interrupt signals from one of the components in the PL.

On the PL side, the hardware architecture responsible for computing the datapoint classifications is featured. The first component is the AXI interconnect layer. This component represents several AXI interconnect blocks, used for both component connection purposes and protocol conversion. The interconnect layer converts the AXI3 protocol used in the device's AXI interfaces to either AXI4-Full or AXI4-Lite. The connections meant for data transfers are converted to AXI4-Full, which allows the use of burst transfers. The connections meant for control functions are converted to AXI4-Lite, which is a simplified implementation of the AXI4-Full protocol and has a smaller resource footprint.

After the routing and protocol conversion is performed, the connection needed with the several PL components can be made. The data connections from the DDR memory arrive to one of the three DMA available DMA components, which have the capability of converting incoming data via AXI4-Full to AXI4-Stream and vice-versa. The AXI4-Stream is much more suitable for sending and receiving large portions of data due to its unlimited bursting size. The way the data flows through the hardware architecture will be detailed in section 4.3.

The datapoints and the centers arrive to the accelerators after passing through several different components, whose job will be detailed in section 4.3. The accelerators are initially configured by the ARM cores with the multiple dataset parameters (such as number of datapoints, number of centers and data dimensionality) and each one is responsible for computing a partial datapoint classification. The results of each accelerator need to be merged together for the correct classification to be obtained. This is accomplished by a simple tree reduction block. The final result is fed back to one of the DMA components, which writes the result back in the DDR memory.

4.3. PL Dataflow

The hardware accelerators in the PL can be configured to handle any number of datapoints and centers. This flexibility adds some complexity on how the data flows through the PL architecture. Similarly to some of the previous devised solutions, the hardware architecture parallelizes the computation by splitting the centers through several accelerators. To accomplish that, the datapoints need to be broadcasted to all accelerators and the centers need to be delivered to the appropriate accelerators.

The datapoints are received by the DMA and are then sent to a local memory block. This block contains a dual-port BRAM and custom logic able to interact with the BRAM given the data received via AXI4-Stream. Since the accelerators can hold more than one center, it is possible that the same datapoint needs to be evaluated several times (once per center). By using a local memory in the PL, a datapoint can be accessed multiple times without the need of multiple DMA transactions of the same datapoint. However, it is not viable to hold an entire dataset in the local memory, due to its size. Instead, the local memory holds only two datapoints at a time: the one being processed, and the next one. Both ports of the BRAM are exploited, in order to read and write values independently.

As mentioned previously, the datapoints need to be broadcasted throughout the several accelerators. This is accomplished by a custom made AXI4-Stream Broadcast block, which relies only on combinatorial logic to perform the broadcast. By performing a broadcast directly in the PL side, the hardware solution becomes highly scalable, as adding more accelerators does not require any more data transfers and thus preserves the PS/PL bandwidth.

As for the centers, each DMA component is responsible for sending their half of the centers to half of the accelerators. In the presented diagram, an example using only 4 accelerators was used, and each interconnect block distributes its half of the centers through 2 accelerators.

After the accelerators' results are computed and merged in the tree reduction block, they are sent back to the Datapoint DMA core. Differently from the Centers DMA cores, which use a polling mechanism in their transfers, the Datapoint DMA core uses an interrupt mechanism, in both read and write channels. When sending the dataset to the PL, maximum transfer length may not be large enough to send the entire dataset in one single transfer. An interrupt mechanism will allow the ARM cores to start their side of the computation without actively polling the DMA core for the remainder of the dataset to be sent.

Using an interrupt mechanism for the incoming results is also highly beneficial. In order to have both ARMs and the accelerators working in the assignment step simultaneously, a result burst size is defined beforehand. Once a complete result burst arrives to the Datapoint DMA, an interrupt signal is issued and the ARMs are signalled to start their computation, using the newly received burst. This way, computational parallelism between hardware and software domains is achieved.

4.4. Hardware Accelerators

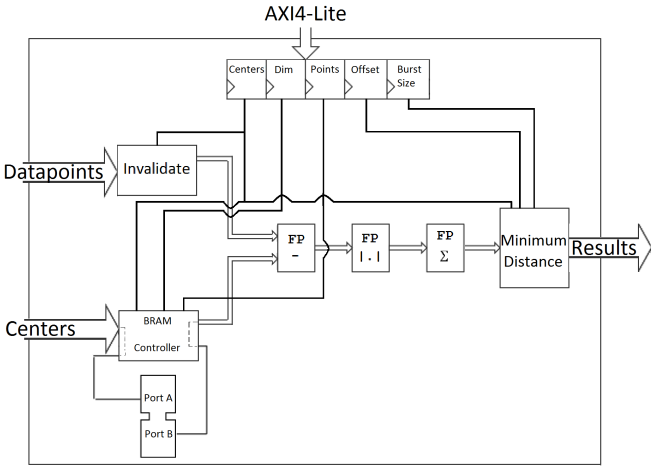


Figure 2. Block Diagram of the Hardware Accelerator

The hardware accelerators are the most important component in the PL side. Each accelerator computes the Manhattan distance between datapoints and a subset of the centers, in 32-bit floating-point representation, and needs to compute both the closest center found to the datapoint and the distance to said center.

Figure 2 illustrates the block diagram of one hardware accelerator. In the presented figure, the connections inside the accelerator represented with arrows stand for AXI4-Stream connections and the ones represented with straight lines stand for simple connections.

As evidenced in figure 1, each accelerator needs to be initially configured through its AXI4-Lite interface. In the accelerator's diagram, 5 different parameters are illustrated: number of centers per accelerator, number of dimensions, number of datapoints in the dataset, offset and result burst size. The number of centers is specified beforehand. The offset parameter is needed by the accelerator, in order to keep track of which cluster ID each center belongs to. The result burst size states how many classification results are in a single burst of results.

The incoming datapoints firstly go through a custom made component, named *Invalidate* block. This component was added as a fix to an issue that occurs when the accelerators have different number of centers assigned to them. As mentioned before, the same datapoint can be broadcasted to all accelerators more than once, depending on the number of centers per accelerator. If one accelerator has more centers to evaluate than others, and thus needs the same datapoint more times, the accelerators with fewer centers need to invalidate the incoming repeated broadcasts that are no longer needed.

The incoming centers are stored in a memory, local to the accelerator. The local memory is identical to the memory used for the datapoints in the top level design. Since the number of centers is usually much inferior to the number of datapoints, it is conceivable to hold all the datapoints in local memories in the PL side, specially if the multiple accelerators (and hence, multiple local memories) are available. In each BRAM, port A is used for storing incoming centers, and port B is used to read them from the local memory.

Each accelerator features 3 floating-point cores, responsible for computing the Manhattan distance. Recalling equation 1, the required operations are the subtraction, the "absolute value", and the accumulation. Each one of the floating-point cores performs one of the operations. After a complete Manhattan distance is computed, the result is handed to another custom made component, named *Minimum Distance* block. This component performs successive comparisons between the distances to each evaluated center and keeps the closest distance. With the aid of the offset parameter, it also outputs the ID of the cluster correspondent to the closest center. The burst size parameter is also used, in order to signal the end of a burst of results in the AXI4-Stream protocol.

5. Analysis and Experiments

In algorithm acceleration, the execution time results are the most meaningful extractable metric from the architecture, among several other important measurements such as area occupied, hardware throughput and hardware/software bandwidth. This section focuses on the timing analysis for the devised architecture, featuring an analytical model that can predict the expected timing results for any dataset and hardware configuration. The model is then validated with actual measurements, ranging from a wide number of algorithm executions and dataset configurations.

5.1. Analytical Model

The analytical model created allows one to predict the obtained speed-up in any given configuration. These predictions can be obtained by modelling the iteration times for both software-only and hardware/software implementations. Each major step of the algorithm was analysed separately and mathematical expressions were formulated for the following execution times:

1) T_classify_ARM - time the ARM processor takes to classify a datapoint

2) T_classify_Accel - time the accelerators take to classify a datapoint

3) T_accum - time the ARM takes to update the accumulator and the counter, given a single classification

4) T_update - time the ARM takes to update a single center

In a software-only implementation, all steps have to be performed sequentially. Consequently, the sequential iteration time is given by equation 2, with N being the number of datapoints and C the number of centers.

$$T_{iteration_{Seq}} = NT_{classify_{ARM}} + NT_{accum} + CT_{update} \quad (2)$$

5.2. Experimental Results

The conducted experiments show the impact of each parameter on the execution time. Each experiment consisted of several executions of the algorithm, whilst performing a sweep in one of the parameters. The main targeted parameters were the number of centers, dimensions, and datapoints. Also, each experiment was conducted with up to 3 accelerators, in order to also evidence how an increase in the number of accelerators used would influence the results. In all experiments, the ARM cores operated at 650 Mhz, while the accelerators and all PL components operated at 100 Mhz.

Figures 3 and 4 portray the results from the first two experiments, which consisted of a sweep in the number of

centers used. The experiments differ in the dataset used: the first experiment used a substantially smaller dataset. In both graphs, the dashed lines represent the estimates given by the analytical model and the full lines represent the actual measured values. A rough interpretation of both graphs indicates that the analytical model was more accurate in the 2nd experiment. In fact, the maximum relative error decreased from 13% to 10%. This is expected as the 2nd experiment deals with much larger execution times, that can be measured more accurately. The results also suggest that, depending on the number of accelerators used, there is an ideal number of centers per accelerator, able to provide the best speed-up. This value is around 2 centers per accelerator. When more centers are added, the accumulator/counter updates become the main bottleneck.

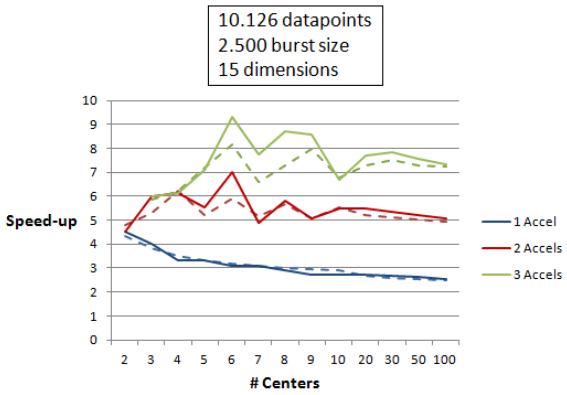


Figure 3. Speed-up results for the 1st center sweep

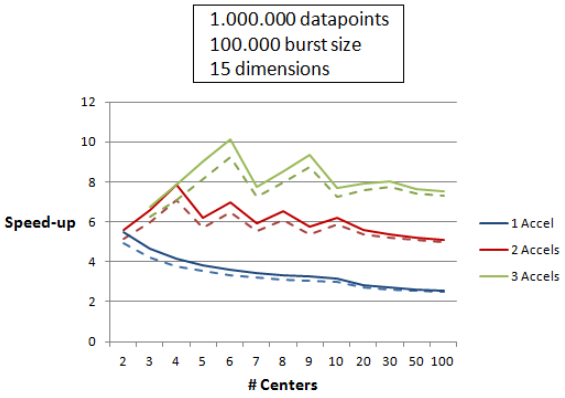


Figure 4. Speed-up results for the 2nd center sweep

The 3rd experiment focused on the data dimensionality (see figure 5). The size of the dataset from the 2nd experiment was kept and the number of centers was fixed. Studying the influence of the number of dimensions used on the execution time is important, as clustering problems in *Big Data* analysis not only focus on datasets with a large number of points but also on very high dimensionality datasets. In this experiment, the expected values from the analytical model (and, consequently, the obtained speed-ups) had much less fluctuation when comparing with the previous experiments. This is explained by the way each computational task is dependent from the data dimensionality parameter. The tasks are linearly dependent from the data dimensionality. In result, both sequential and hardware/software iteration times grow linearly as the sweep is performed. The only difference between both implementations is in the growth ratio, portrayed by the obtained speed-up.

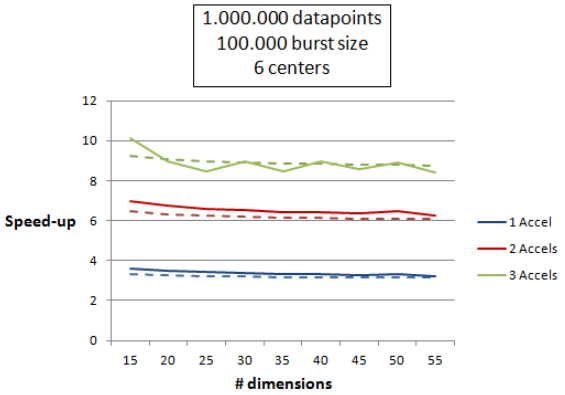


Figure 5. Speed-up results for the dimensions sweep

The 4th and final experiment focused on the number of datapoints in the dataset (see figure 6). Similarly to what happened with the data dimensionality parameter, the iteration times in both sequential and hardware/software implementations are linearly dependent from the number of datasets. As a result, the expected speed-ups throughout the sweep should remain somewhat constant. However, the number of datapoints in the dataset does not have any influence on which of the timing cases is dictating the iteration time, since both $T_{classify_{Accel}}$ and T_{accum} are independent from the number of datapoints. This independence could not be claimed in the previous experiment.

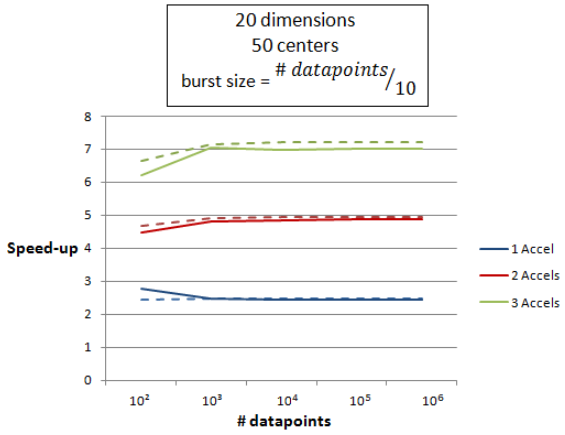


Figure 6. Speed-up results for the datapoint sweep

From all 4 experiments, the observed maximum speed-up was 10.1. However, the analytical model suggests that using more than 3 accelerators can further improve the

maximum speed-up. The model was validated throughout all 4 sweeps and predictions for any given dataset and number of accelerators could be made with a rather low relative error. Depending on the amount of resources available in the target device, the number of accelerators able to be implemented can vary and the consequent speed-up results can change drastically. As an example, the most complex device from the Zynq-7000 family, the Z-7100, could fit up to 144 accelerators. If the algorithm were to be used near to its equilibrium point between both timing cases with such a large number of accelerators, although unlikely in this setup, the expected speed-up could reach up to 165.

6. Conclusions

In this paper, a hardware/software architecture targeting the K-means clustering algorithm was devised, using the recent SoC devices which incorporate FPGA resources and hard-core processors in one single chip. These devices enable the design and implementation of efficient custom hardware solutions, while offering powerful built-in cores, as well as an easy integration between hardware and software domains.

The previous acceleration techniques were studied beforehand, with computation being mostly parallelized between the several computational elements by either splitting the dataset or the centers. The proposed solution chooses a center split, as it provides a much more scalable solution. Furthermore, the solution exploited the parallelism between hardware and software components, by having both the hardware accelerators and the ARM cores performing different computational tasks in parallel.

The solution was tested for its ability to accelerate the clustering algorithm. A timing theoretical model was constructed and several experiments were conducted, which further validated the devised model. Both theoretical and experimental results showed the solution can significantly accelerate the algorithm, even with a few number of accelerators. Predictions were also made for a higher number of accelerators, which could potentially increase the speed-ups even further.

Acknowledgment

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

[1] S. Che, J. Meng, J. W. Sheaffer, and K. Skadron, "A performance study of general purpose applications on graphics processors," in *First Workshop on General Purpose Processing on Graphics Processing Units*, 2007, p. 10.

[2] M. Estlick, M. Leeser, J. Theiler, and J. J. Szymanski, "Algorithmic transformations in the implementation of k-means clustering on reconfigurable hardware," in *Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays*. ACM, 2001, pp. 103–110.

[3] V. Faber, "Clustering and the continuous k-means algorithm." Los Alamos Science, 1994, pp. 138–144.

[4] R. Farivar, D. Rebolledo, E. Chan, and R. H. Campbell, "A parallel implementation of k-means clustering on gpus," in *PDPTA*, vol. 13, no. 2, 2008, pp. 212–312.

[5] M. Gokhale, J. Frigo, K. Mccabe, J. Theiler, C. Wolinski, and D. Lavenier, "Experience with a hybrid processor: K-means clustering," *The Journal of Supercomputing*, vol. 26, no. 2, pp. 131–148, 2003.

[6] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "Fpga implementation of k-means algorithm for bioinformatics application: An accelerated approach to clustering microarray data," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*. IEEE, 2011, pp. 248–255.

[7] M. Kakooei and H. S. Shahhoseini, "A parallel k-means clustering initial center selection and dynamic center correction on gpu," in *Electrical Engineering (ICEE), 2014 22nd Iranian Conference on*. IEEE, 2014, pp. 20–25.

[8] T. Kucukyilmaz, "Parallel k-means algorithm for shared memory multiprocessors," *Journal of Computer and Communications*, no. 2, pp. 15–23, 2014.

[9] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable fpga architecture for k-mean clustering," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1801–1804.

[10] D. Lavenier, "Fpga implementation of the k-means clustering algorithm for hyperspectral images," in *Los Alamos National Laboratory LAUR*. Citeseer, 2000.

[11] Y. Li, K. Zhao, X. Chu, and J. Liu, "Speeding up k-means algorithm by gpus," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, June 2010, pp. 115–122.

[12] W.-C. Liu, J.-L. Huang, and M.-S. Chen, "Kacu: k-means with hardware centroid-updating," in *Emerging Information Technology Conference, 2005.* IEEE, 2005, pp. 3–pp.

[13] C. Ordonez, "Clustering binary data streams with k-means," in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. ACM, 2003, pp. 12–19.

[14] X. Wang and M. Leeser, "K-means clustering for multispectral images using floating-point divide," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*. IEEE, 2007, pp. 151–162.

[15] M. Zechner and M. Granitzer, "Accelerating k-means on the graphics processor via cuda," in *Intensive Applications and Services, 2009. INTENSIVE'09. First International Conference on*. IEEE, 2009, pp. 7–15.

[16] J. Zhang, G. Wu, X. Hu, S. Li, and S. Hao, "A parallel k-means clustering algorithm with mpi," in *Parallel Architectures, Algorithms and Programming (PAAP), 2011 Fourth International Symposium on*, Dec 2011, pp. 60–64.

Índice de Autores

Notas