REC 2015

Atas

XI Jornadas sobre Sistemas Reconfiguráveis

5 e 6 de fevereiro de 2015

Instituto Superior de Engenharia do Porto

Organizadores:
Manuel Gericota
André Fidalgo
Paulo Ferreira
José Carlos Cardoso

Edição: Comissão Organizadora da REC 2015
Manuel Gericota, André Fidalgo, Paulo Ferreira, José Carlos Cardoso

ISBN: 978-989-98875-2-7

Conteúdo

Aplicações de alto desempenho

Miscelânea

 ISBN: 978-989-98875-2-7

Prefácio

As XI Jornadas sobre Sistemas Reconfiguráveis decorrem no Instituto Superior de Engenharia do Porto, a 5 e 6 de fevereiro de 2015. Esta edição vem na continuação de uma sequência de eventos que teve início no Algarve, onde, em 2005 e sob a responsabilidade da Universidade local, se realizou o primeiro evento, com edições anuais posteriores na Faculdade de Engenharia da Universidade do Porto (2006 e 2011), no Instituto Superior Técnico da Universidade Técnica de Lisboa (2007), no Departamento de Informática da Universidade do Minho (2008), na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (2009), na Universidade de Aveiro (2010), no Instituto Superior de Engenharia de Lisboa (2012), na Universidade de Coimbra (2013) e de novo no Algarve, em Vilamoura (2014) num regresso às origens a comemorar a 10ª edição. Ao longo de todas estas edições, as Jornadas têm conseguido constituir-se como o ponto de encontro anual para a comunidade científica de língua portuguesa com reconhecida atividade de investigação e desenvolvimento na área dos sistemas eletrónicos reconfiguráveis.

O programa das XI Jornadas – REC 2015 – decorre durante um dia e meio, tendo-se procurado, a exemplo das primeiras edições, recriar uma maior interatividade entre os participantes. Mantém-se a apresentação de comunicações de vários tipos sobre trabalhos consolidados ou em curso, facilitando a integração de novos membros no ambiente científico, e criam-se novos espaços para apresentação e debate de ideias e experiências, nomeadamente experiências pedagógicas ligadas ao ensino dos sistemas reconfiguráveis. Assim, procura promover-se o conhecimento e o debate de ideias e de casos e potencia-se a criação de possíveis colaborações futuras.

Este ano, as Jornadas incluem duas apresentações convidadas:

- "FPGAs no espaço - uma aventura nacional", apresentada pelo colega Rodolfo Martins da EVOLEO Technologies, uma empresa sedeada na Maia e que, entre outras atividades, se dedica ao desenvolvimento de hardware e software para aplicações espaciais;
- "Projeto Bambi - Bottom-up Approaches to Machines dedicated to Bayesian Inference", apresentada pelo colega Jorge Lobo do Instituto de Sistemas e Robótica da Universidade de Coimbra, um projeto cujo objetivo é a proposta de uma nova teoria na área da computação probabilística e a sua implementação usando FPGAs.

Agradecemos aos colegas a disponibilidade para partilharem com os participantes da REC 2015 as suas experiências e conhecimentos.

O programa conta ainda com um painel temático subordinado ao tema "Compiling to FPGAs: A decade of improvements, trends, and challenges", dinamizado pelo colega João Cardoso da Faculdade de Engenharia da Universidade do Porto, e com uma mesa redonda, moderada pelo colega Manuel Gericota do Instituto Superior de Engenharia do Porto, onde será abordada a problemática do ensino do co-projeto de HW/SW.

O programa conta ainda com a apresentação de 8 comunicações regulares nas áreas das telecomunicações, teste e fiabilidade, instrumentação e das aplicações de alto desempenho baseadas em sistemas reconfiguráveis. Estas contribuições foram todas aprovadas para apresentação e publicação pelo Comité Científico.

A organização destas Jornadas contou com o apoio de diversas pessoas e entidades, às quais gostaríamos de expressar o nosso agradecimento, nomeadamente e em especial aos autores que contribuíram com os trabalhos incluídos nestas Atas, bem como aos membros do Comité Científico pelo excelente trabalho produzido, concretizado em revisões que, estamos certos, permitiram melhorar a qualidade dos trabalhos submetidos.

Igualmente os nossos agradecimentos aos colegas que se dispuseram a participar no painel temático e na mesa-redonda. Por último, um agradecimento pelo apoio prestado pelos Serviços Económico-Financeiros e pela Presidência do ISEP, respetivamente pelo apoio na gestão financeira e na cedência das salas para o evento.

Esperamos que esta edição das Jornadas constitua, uma vez mais, um espaço para divulgação e discussão dos trabalhos apresentados, bem como de convívio aberto a todos quantos partilham interesses na área dos sistemas eletrónicos reconfiguráveis, contando revê-los nas jornadas do próximo ano.

Manuel Gericota, André Fidalgo e Paulo Ferreira, Instituto Superior de Engenharia do Porto
José Carlos Cardoso, Universidade de Trás-os-Montes e Alto Douro

Comissão Organizadora

Manuel Gericota, André Fidalgo e Paulo Ferreira	Instituto Superior de Engenharia do Porto
José Carlos Cardoso	Universidade de Trás-os-Montes e Alto Douro

Comité Científico

Ana Antunes	Instituto Politécnico de Setúbal
André Fidalgo	Instituto Superior de Engenharia do Porto
Anikó Costa	Universidade Nova de Lisboa – UNINOVA
António Esteves	Universidade do Minho
António Ferrari	Universidade de Aveiro – IEETA
Arnaldo Oliveira	Universidade de Aveiro – IT
Fernando Gonçalves	Instituto Superior Técnico – INESC-ID
Gabriel Falcão	Universidade de Coimbra – IT
Helena Sarmento	Instituto Superior Técnico – INESC-ID
Horácio Neto	Instituto Superior Técnico – INESC-ID
Iouliia Skliarova	Universidade de Aveiro – IEETA
João Bispo	Fac. de Engenharia da Universidade do Porto
João M. P. Cardoso	Fac. de Engenharia da Universidade do Porto – INESC Porto
João Canas Ferreira	Fac. de Engenharia da Universidade 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 Universidade de Lisboa – INESC-ID
José Carlos Alves	Fac. de Engenharia da Universidade do Porto – INESC Porto
José Carlos Cardoso	Universidade de Trás-os-Montes e Alto Douro
José Carlos Metrôlho	Instituto Politécnico de Castelo Branco
José Gabriel Coutinho	Imperial College London
José Silva Matos	Fac. de Engenharia da Universidade do Porto – INESC Porto
Leonel Sousa	Instituto Superior Técnico – INESC-ID
Luís Cruz	Universidade de Coimbra – IT
Luís Gomes	Universidade Nova de Lisboa – UNINOVA
Luís Nero	Universidade de Aveiro – IT
Manuel Gericota	Instituto Superior de Engenharia do Porto

Marco Gomes	Universidade de Coimbra – IT
Mário Calha	Fac. de Ciências da Universidade de Lisboa – LaSIGE
Mário Véstias	Instituto Superior de Engenharia do Lisboa – INESC-ID
Morgado Dias	Universidade da Madeira
Nuno Roma	Instituto Superior Técnico – INESC-ID
Orlando Moreira	Ericsson
Paulo Flores	Instituto Superior Técnico – INESC-ID
Pedro C. Diniz	University of Southern California
Ricardo Chaves	Instituto Superior Técnico – INESC-ID
Ricardo Machado	Universidade do Minho

 ISBN: 978-989-98875-2-7

Comunicações convidadas

 ISBN: 978-989-98875-2-7

FPGAs no espaço - uma aventura nacional

António Sousa
EVOLEO Technologies
Maia – Portugal

Short bio:

António Sousa is EVOLEO's R&D Manager. He holds a Master degree in Electrical and Computer Engineering. He is responsible for the development of embedded systems, both in the VHDL code implementation for Hi-Rel electronics and in SW development over multiple operating systems.

In the last 5 years, he is in charge of the coordination and development of electronics embedded systems for space and industry, being the link between all the SW, HW and Testing activities.

 ISBN: 978-989-98875-2-7

Projeto BAMBI

Jorge Lobo

Instituto de Sistemas e Robótica
Universidade de Coimbra
Coimbra – Portugal

BAMBI - Bottom-up Approaches to Machines dedicated to Bayesian Inference

This project proposes a theory and a hardware implementation of probabilistic computation inspired by biochemical cell signalling. We will study probabilistic computation following three axes: algebra, biology, and hardware. In each case, we will develop a bottom-up hierarchical approach starting from the elementary components, and study how to combine them to build more complex systems. We propose Bayesian gates operating on probability distributions on binary variables as the building blocks of our probabilistic algebra. These Bayesian gates can be seen as a generalisation of logical operators in Boolean algebra. We propose to interpret elementary cell signalling pathways as biological implementation of these probabilistic gates. In turn, the key features of biochemical processes give new insights for innovative probabilistic hardware implementation. We propose to associate conventional electronics and novel stochastic nano-devices to build the required hardware elements. Combining them will lead to new artificial information processing systems, which could, in the future, outperform classical computers in tasks involving a direct interaction with the physical world. For these purposes, the BAMBI project associates research in Bayesian probability theory, molecular biology, nanophysics, computer science and electronics.

Within the BAMBI consortium, ISR-UC will focuses on the emulation hardware implementation and on the computational architecture to be developed, namely in the composition of basic building blocks for probabilistic computation.

*European Commission collaborative FET project BAMBI - Bottom-up Approaches to Machines dedicated to Bayesian Inference - FP7-ICT-2013-C, project number 618024 (www.bambi-fet.eu)

Short bio:

Jorge Nuno de Almeida e Sousa Almada Lobo was born on the 23rd of September 1971, in Cambridge, UK. In 1995, he completed his five year course in Electrical Engineering at Coimbra University, in April 2002 the M.Sc degree and in June 2007, he received the Ph.D degree with the

thesis ‘‘Integration of Vision and Inertial Sensing’’. In 1997 he was a junior teacher in the Computer Science Department of the Coimbra Polytechnic School, and in 1998 joined the Electrical and Computer Engineering Department of the Faculty of Science and Technology at the University of Coimbra, where he currently works as Assistant Professor with tenure. He is responsible for courses on Digital Design, Microprocessors and Computer Architecture.

His current research is carried out at the Institute of Systems and Robotics, University of Coimbra, working in the field of computer vision, sensor fusion, and mobile robotics. Current research interests focus on inertial sensor data integration in computer vision systems, Bayesian models for multimodal perception of 3D structure and motion, and real-time performance using GPUs and reconfigurable hardware. He has been involved on several EU projects, including BACS - Bayesian Approach to Cognitive Systems (FP6-IST-027140) and HANDLE - Developmental pathway towards autonomy and dexterity in robot in-hand manipulation (FP7-2008-231640) and currently in BAMBI - Bottom-up Approaches to Machines dedicated to Bayesian Inference (FET - FP7-ICT-2013-C, 618024).

 ISBN: 978-989-98875-2-7

Painel temático

 ISBN: 978-989-98875-2-7

Compiling to FPGAs: A decade of improvements, trends, and challenges

Moderado por

João Cardoso

Faculdade de Engenharia da Universidade do Porto

Porto – Portugal

Reconfigurable fabrics, especially FPGAs, are being adopted in many areas, from high-performance to embedded computing. They provide solutions for accelerating algorithms and for low cost and efficient performance/energy trade-offs. However, the way we program FPGAs, and specially the required expertize, is still nearly the same as before. Some may argue that the kind of breakthrough provided by RTL/logic synthesis and placement and routing techniques has not yet been seen in the automatic mapping of algorithms and data structures to reconfigurable fabrics!

The complexity of the applications and the hardware capacity of the reconfigurable fabrics are starving for advancements over the traditional use of high-level synthesis tools to map computations from high-level programming languages. The compilation of high-level programming languages (mostly software programming languages) to hardware started in the 80's, with more than twenty years of approaches and limited success as argued by some people. Over the last couple of years, we have seen successful approaches to map computations described in non-traditional FPGA ways, such as the compilation of OpenCL, functional languages, and DSLs (Domain-Specific Languages).

This panel will present an historic perspective about the compilation of high-level programming languages to hardware, with an emphasis on reconfigurable hardware (e.g., provided by FPGAs). The panel will focus on the improvements over the last decade, the trends and the challenges.

We will kick off the panel with 10 minute lightning invited talks that will present some insights about the compilation to FPGAs, followed by an open discussion and interactive panel.

Apresentações:

1. João M.P. Cardoso (UPorto, PT): "Compiling to FPGAs: A decade of improvements, trends, and challenges"
2. José Gabriel Coutinho (Imperial College, UK): "From software description to hardware design - an academic perspective"
3. José Teixeira de Sousa (IST/INESC-ID, PT): "Mapping multimedia applications to reconfigurable fabric - an industrial perspective"
4. João Canas Ferreira (Universidade do Porto /INESC-TEC, PT): "From Application Binaries to Hardware Accelerators: Why, When and How?"

 ISBN: 978-989-98875-2-7

Mesa redonda

 ISBN: 978-989-98875-2-7

O ensino do Co-projeto de HW/SW

Moderado por
Manuel Gericota
Instituto Superior de Engenharia do Porto
Porto – Portugal

O mote para esta mesa redonda surgiu ao ler, num tutorial dedicado precisamente a uma ferramenta para desenvolvimento de sistemas embebidos que misturam o co-projeto de hardware/software com as FPGAs, uma introdução que, vinda de um fabricante, não deixa de ter o seu quê de curioso:

> Embedded systems are complex. Getting the hardware and software portions of an embedded design to work are projects in themselves. Merging the two design components so they function as one system creates additional challenges. Add an FPGA design project to the mix, and the situation has the potential to become very complicated indeed.
>
> EDK Concepts, Tools, and Techniques
> UG683 (v14.6) June 19, 2013, Xilinx

Em face desta realidade assumida pelos próprios fabricantes de FPGAs e das ferramentas de projeto que as acompanham, como podemos ensinar eficazmente e em tempo útil estes conceitos e ferramentas numa sala de aula e/ou num laboratório?

Foi a partir daqui que foi lançado a vários colegas, que em diferentes instituições de ensino superior portuguesas têm a responsabilidade do ensino desta matéria, o desafio para esta mesa redonda.

A mesa redonda conta com a participação de Arnaldo Oliveira da Universidade de Aveiro, Horácio Neto da Universidade de Lisboa, José Carlos Alves da Universidade do Porto e de Luís Gomes da Universidade Nova de Lisboa, sendo moderada por Manuel Gericota, do Instituto Superior de Engenharia do Porto

 ISBN: 978-989-98875-2-7

Telecomunicações

 ISBN: 978-989-98875-2-7

FPGA Applications in 5G Mobile Networks

Diogo Riscado, André Prata, Daniel Dinis, Jorge Santos, Sérgio Julião, João Duarte, Gustavo Anjos, Arnaldo S. R. Oliveira, Nuno B. Carvalho
Instituto de Telecomunicações / Universidade de Aveiro - DETI
Campus Universitário de Santiago, 3810-193 Aveiro, Portugal
{*diogo.riscado, andre.prata, danieldinis, jorgesantos, sjuliao, joao.capela, gustavoanjos, arnaldo.oliveira, nbcarvalho*}*@ua.pt*

Abstract

The traffic and the number of mobile network users is increasing exponentially, leading to successive generations of standards with higher throughput and mobility support, and imposing high requirements in the Radio Access Networks (RANs) to successfully improve the Quality of Service (QoS), manageability, upgradability and cost. In order to achieve these requirements, some new concepts for 5G mobile networks such as Cloud Radio Access Network (C-RAN), Beamforming, Massive Multiple-Input Multiple-Output (MIMO), Software Defined Radio (SDR), between others are being proposed by the academic community and industrial key players in telecommunications. All these concepts require high digital signal processing capabilities, dynamic reconfigurability and high speed logic, which can be adequately addressed by current state-of-art Field Programmable Gate Arrays (FPGAs). In this paper the proposed concepts for next generation mobile networks are discussed, focusing on the importance of the FPGAs capabilities in this domain.

1. Introduction

Mobile networks are subject to an explosive increase in use, in a context of continuous mobility and more stringent levels of Quality of Service (QoS), which imposes demanding requirements to telecommunication networks. During this decade it is predicted an exponential growth of wireless traffic as a consequence of the higher number of connected devices (i.e. smartphones and tablets), the demand for web-intensive services, such as audio and video streaming as well as the spread of Machine-to-Machine (M2M) applications [1]. Therefore, operators are constantly looking for solutions that provide increased capacity and that can improve the QoS and Quality of Experience (QoE). On the other hand, the service enhancements must be done without incurring significant additional Capital Expenditures (CAPEX) and Operational Expenditures (OPEX) costs for the operators [2, 3]. In conjunction, these are some of the main driving forces that will guide the research, experimentation and deployment of next generation, or 5G, mobile networks.

In terms of Radio Access Network (RAN), there are mainly two possible directions for the next generation of wireless access networks: exploiting higher RF-bands (very likely in the millimeter wave range) and the deployment of small-cells (leading to a more dense distribution of antennas and access points). The first one is a suitable solution for point-to-point and/or short-range communications (through microwave links) due to the worst propagation conditions for higher (millimeter wave) frequencies. The second solution can be seen as a reduction of the cell size and increasing its density, improving network coverage and enabling a more aggressive spatial reuse of the spectrum. However, this would not end up with current macro-cells; instead a smart cooperation between small-cells and macro-cells will be needed, in order to provide an enhanced service and achieve a good performance versus cost compromise [4].

Ultimately, 5G and all networks beyond will be extremely dense, flexible and heterogeneous, which introduces many new challenges for network modeling, analysis, design and optimization. The core network will also have to reach unprecedented levels of flexibility and intelligence. Spectrum regulation will need to be rethought and energy and cost efficiencies will become even more critical considerations [5].

Furthermore, with the increasing trend to digitalization motivated by the flexibility and performance allowed by the cognitive and software defined radio approaches, Field Programmable Gate Arrays (FPGAs) will continue to have a rising importance in the design of radio and telecommunication systems.

In this work, the 5G concepts and architectures will be discussed, focusing on the role and applications of FPGAs. The remainder of this work is divided as follows. In Section 2 the main features and enabling technologies proposed for 5G mobile networks will be presented. In Section 3 the FPGA role applied to the 5G technologies will be addressed. Finally, some conclusions are drawn in Section 4.

2. 5G Enabling Technologies

This section is intended to summarize the state-of-the-art of current RAN and its limitations. In addition, the most promising trends and key features of 5G networks are listed and detailed.

2.1. RAN Architecture Evolution

The growing demand for mobile RAN capacity allied with the scarcity of spectral radio resources created the need of increasing the degree of spacial spectral reutilisation, leading to a Heterogeneous Networks (Het-Nets) environment. A Het-Net is defined by the coexistence between today's macro configuration with lower range/capacity or small cells (fig. 1). With this small-cell deployment, the number of radio access points will increase. However, considering the high complexity/cost of the actual 3G (NodeB) and 4G (eNodeB) sites, the network setup costs also grow in a unsustainable way for the operators.

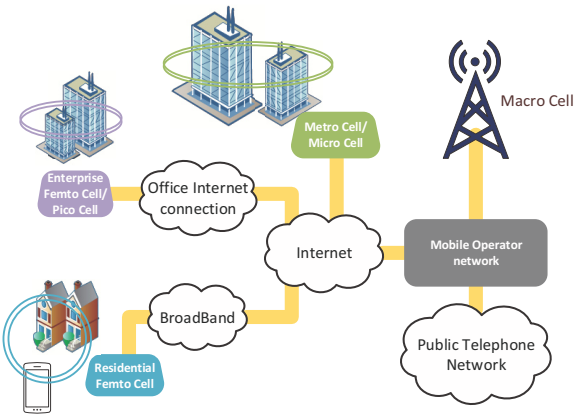


Figure 1. Het-Net typical deployment.

Besides the high initial CAPEX cost to setup a network with this kind of granularity, due to the distributed processing among network sites, the demand in terms of Operation and Maintenance (OAM) will also be critical. Taking these issues in consideration, the new RAN architecture paradigm must be focused in shifting the intelligence/complexity of the network from the edges, to a centralized location. Here, the OAM is performed in an efficient way, as well as, the overall processing resources used for digital signal processing/signal generation can be shared by all the edge nodes dynamically following the traffic patterns along the day. The Cloud Radio Access Network (C-RAN) architecture has been proposed by some important key players in telecommunications such as China Mobile, Huawei, Nokia and others Next Generation Mobile Networks (NGMN) partners as an answer to this trend to a centralized processing.

The C-RAN concept focuses on the separation of the digital Base Band Units (BBUs) from the Remote Radio Heads (RRHs) in order to move them to the cloud for centralized signal processing and management [2, 3]. This new architecture, depicted in fig. 2, is based in the following components: a distributed group of RRHs; a pool of BBUs that are responsible for the L3/L2 and L1 processing tasks; and a high-bandwidth and low-latency optical transport, referred as fronthaul, to link both the above sites. The upper layers of the network are represented as backhaul, where the packed-based Core Network (CN) is implemented.

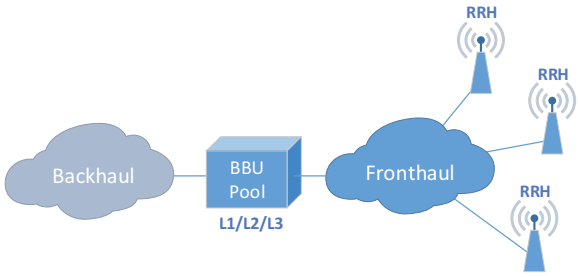


Figure 2. Simplified C-RAN architecture.

2.2. Software Defined Radio

The radio component of the 5G mobile networks must be simple, power efficient and must have a high degree of flexibility, modularity and integration. Therefore the solution for this radio component must be based on enhanced Software Defined Radio (SDR) architectures [6]. The SDR concept it is not new. It was introduced in 1995 by Mitola, who has proposed a new architecture for the radio transceivers, where the waveforms are completely defined in the digital domain, providing great flexibility and efficiency to radio transceivers [6]. The ideal SDR architectures poses some challenges related with the Analog to Digital Converters (ADCs) and Digital to Analog Converters (DACs) operating frequency, dynamic range, power efficiency, among other limitations. At the digital processing side, there are also many challenges due to the required sample rates. However, since most of the operations are highly parallelizable, FPGAs are a convenient implementation platform. Additionally, some new SDR architectures such as All-Digital transmitters and receivers, which are not directly based on high speed ADCs and DACs, can also be a possibility to address the radio component architecture [7, 8]. However, these are topics that still demand a great research effort in order to be more mature to be applied in main stream commercial products and applications.

In the specific case of C-RAN architectures, where the baseband functions no longer reside on the cell site, the RRH modules have much less energy consumption and complexity, which directly should lower their price. In fact, this is a very positive aspect in order to support the feasibility of high-density C-RAN architectures [3]. The challenges associated with this RRH architecture are due to the interface compatibility with the transport layer and also the problems associated with SDR architectures previously stated. It is also important to state that such architectures are not only suitable to the specific case of C-RAN RRH components, but are also convenient to be implemented as an Intermediate Frequency (IF) stage in mmWaves communications either in a point-to-point link scenario or in a short range communication scenario, as it will be presented in the next sections.

2.3. Carrier Aggregation

Regarding the bandwidth, there is a growing consensus that the next mobile generation will require, as minimum, a

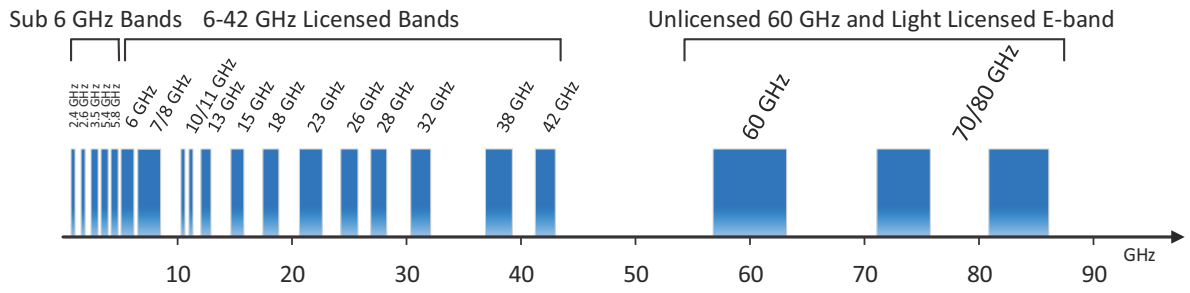


Figure 3. Representation of standardized bands for microwave and mmWaves.

continuous frequency range of several hundred megahertz in the bands below 40 GHz and at least one gigahertz above this frequency [9]. If the searching, selection and allocation of suitable continuous frequency bands does not lead to positive results, Carrier Aggregation (CA) scenarios will be considered, combining a small number of contiguous or non-contiguous portions of the spectrum to achieve the required bandwidth.

Because different frequency ranges have different propagation characteristics, CA will allow the use of lower spectrum ranges that are not adjacent each other but that are narrower and have less propagation loss, while compared with a much larger range in high frequencies. This will still allow the delivery of an ultrahigh data rate (tens of gigabits per second), while making an efficient use of spectrum and using frequency ranges that can be less problematic.

2.4. Millimeter Wave

Until now, terrestrial wireless systems have restricted their operation to the relatively slim range of microwave frequencies, which is often called “beachfront spectrum”, and extends from several hundred MHz to a few GHz. However, since more and more wireless devices need to use this beachfront, it is becoming an ever-precious resource (fig. 3).

Even though beachfront bandwidth allocation can be made significantly more efficient, even by modernizing regulatory standards or by implementing new allocation procedures (such as Dynamic Spectrum Access - DSA - where spectrum is assigned in real time [10]), it seems inevitable that we need to go to higher frequencies. Luckily, there are vast amounts of relatively idle spectrum in the mmWave range of 30-300 GHz, mainly because, until recently, it had been deemed unsuitable for mobile communications due to the rather hostile propagation qualities, including strong path loss, atmospheric and rain absorption, low diffraction around obstacles and penetration through objects, and, further, because of strong phase noise and exorbitant equipment costs [5, 11, 12].

Because of this, mmWave communications have been widely applied mostly on point-to-point communications, such as microwave backhauls. However, it is possible to take profit on the available spectra in mmWave in order to support short-distance (typically less than 100m) high speed communications. As an example, there is an IEEE task force working in a new IEEE 802.11ad standard, which aims to deploy very high throughput Wi-Fi in 60 GHz band [13]. This is a very important capability which will certainly be considered for 5G mobile networks, mainly in small-cell deployment. The use of these higher frequencies in mobile communications seems affordable now, since semiconductors are maturing, their costs and power consumption rapid falling and the other obstacles related to propagation are now considered surmountable [5].

2.5. Massive MIMO

Due to the increasing traffic and number of mobile network users, single antenna traditional communication systems have not been able to follow the progression of wireless cellular and broadband systems. Typical approaches employed to solve this problem are based on multi-antenna systems, or Multiple-Input Multiple-Output (MIMO), in which it is possible to deploy methods such as diversity, spatial multiplexing and beamforming (a capability to produce beams with a specific directivity, through the phase/amplitude interference from all antennas’ signals). With this kind of solution the system’s capacity and its robustness can also be increased. Taking into account that in the mmWaves range the antenna sizes (and thus their costs) will be minimized, multi-antenna systems such as phased arrays and massive MIMO are becoming increasingly a hot topic.

Massive MIMO (or Large-Scale MIMO) systems can be

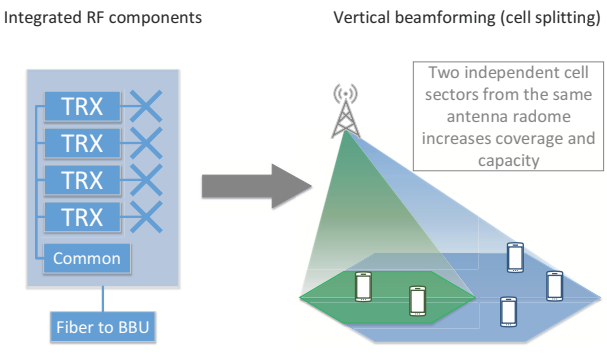


Figure 4. AAS architecture (left) and beamforming technique (right).

stated as a new evolution of the Multi-User MIMO (MU-MIMO) concept, in which the number of base station's antennas is much larger than the number of user terminals. With this approach, it will be possible to focus the radiated energy in any specific direction as well as to minimize the inter and intra-cell interference. This concept can be seen as a group of small (and low power) antenna units, which as a whole, are used to shape beams and to steer nulls in some directions. The ability of shaping beams is essential to maintain a link connection even in the presence of moving nodes while the ability of steering nulls in some direction can be of utmost importance to minimize the interferences with other known nodes. If the antenna pattern can be adjusted in a dynamic fashion, as a response to external conditions, these types of antennas are called AAS or "Smart Antennas".

In a real deployment, a conjugation of the mentioned technologies may take place in the way that the short range of mmWaves is mitigated with a denser small-cell architecture. In turn, AAS take advantage of above 6GHz links due to its high capacity feature.

The next section aims to point out the possible use cases for the FPGA devices in the 5G architectures and features since they will play an important role, either for interface and processing in the scope of the enabling technologies discussed in this section.

3. FPGA Applications

Due to the high density of 5G networks, access points will become more common through Small (micro or femto) cell deployments. State-of-the-art FPGA's lithography with 16 to 28 nm are characterized by low power consumption leading to a lower site footprint and cost-effective solution for operators. In addition, the offer in terms of Intellectual Property (IP) from the main manufacturers eases the integration on FPGAs of industry-standard protocols as well as other high-level components targeted for L1/L2/L3 stack layers. The reconfiguration feature of those devices enables the flexibility on upgrading and reusing such modules, also leading to low-cost deployment of radio/baseband modules. This section addresses how FPGAs may be assumed as an enabling platform in the development of next generation RAN equipment such as the RRHs, BBUs, as well as the backhaul and fronthaul infrastructures.

3.1. Remote Radio Head

The RRH module acts as the transceiver between electrical/optical and RF domains. Its internal structure is depicted in fig. 5. Typical RRH can be split on the baseband interface module (based on standard protocols such as Common Public Radio Interface (CPRI) or Open Radio Interface (ORI)), the Digital Front-End (DFE), where the digital signal processing is performed, the DAC/ADC interfaces and, lastly, the Analog Front-End (AFE), responsible for the operations in the analog domain, i.e., filtering, amplification and I/Q (de)modulation. In addition, a Central Processing Unit (CPU) enables the board-level OAM, as well as other software-based functions. The implementation of digital processing and software modules may take advantage of System-on-a-Chip (SoC) solutions (with ARM processor cores together with FPGA logic on the same chip) for the board-level control and backhaul/fronthaul interface Control & Management (C&M) plane handling due to the peripheral compliance (Ethernet, UART, SPI, GPIO, etc.).

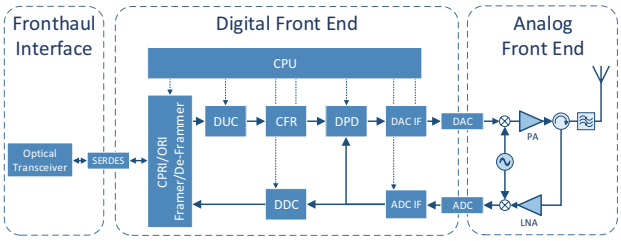


Figure 5. RRH internal architecture.

Besides short/mid term MU-MIMO and LTE-Advanced (LTE-A) with CA (bandwidth up to 100 MHz), 5G will bring new challenges to the development of RRH as the ones mentioned next.

3.1.1. Multi-RAT support

Unlike 2G, 3G and 4G, next generation could not imply a new Radio Access Technology (RAT). Instead, it may be a simple combination of existing RATs targeting multi-mode/multi-band RRHs. This flexibility is enhanced by FPGA-based DFE due to the possibility to develop digital signal processing blocks such as sampling rate and baseband - IF conversions. Unlike Global System for Mobile Communications (GSM) and WiMAX, in terms of sampling rate, Long-Term Evolution (LTE) and Wide-Band Code Division Multiple Access (W-CDMA) share the same timing structure. Here, the equalization is performed through interpolation/decimation FIR and CIC filters, eventually featuring irrational conversion ratios. FPGA built-in multipliers and adders support the filter chain design, with the resource sharing possibility in a multi-channel architecture. Furthermore, the inclusion of Digital Up-Conversion (DUC) and Digital Down-Conversion (DDC) blocks are suitable to meet the SDR heterodyne architecture featuring less RF hardware and better image rejection performance. These blocks are mainly implemented by a filter datapath and Numerically Controlled Oscillators (NCOs) (shareable between different RATs) [14].

3.1.2. AFE-related improvements

The IF sampling method, essential for a mmWaves deployment, is enabled by high-speed DACs and ADCs featuring standard serial interface such as JESD204B (based on Multi-Gigabit Transceiver (MGT) logic), which aims to optimize the board-level layout and reduce power consumption when compared with traditional parallel LVDS/LVCMOS interfaces. The DFE may also include

Crest Factor Reduction (CFR) and Digital Pre-Distortion (DPD) to provide the RRH better analog performance and power efficiency. The first is intended to reduce the signal's Peak-to-Average Power Ratio (PAPR) (which is higher on LTE and W-CDMA waveforms) in order to use the Power Amplifier (PA) more efficiently. The second is used to mitigate the PA non-linear behavior that causes output signal degradation (Error Vector Magnitude (EVM) and Adjacent Channel Leakage Ratio (ACLR)). Despite the need of additional FPGA resources and power consumption, these algorithms are proved to reduce overall RRH OPEX [15].

3.1.3. Evolution towards Smart Antennas

Some 5G paradigms state that the current RRH plus passive antenna may evolve to AAS. Usually, AAS' architectures fall into two groups: Switched Beam System or Beamformed Adaptive System [16]. The main difference lies in the process of beamforming, which, in the former, can be stated as static (one beam is chosen from a range of fixed beam patterns) while the last can be termed as adaptive (it is possible to achieve any antenna pattern in order to fulfill certain requirements of the system). A generic architecture of a *Beamformed Adaptive Receiver System* with M elements comprises the antenna itself, a radio unit, a beamforming unit and a signal processing unit [17]. The radio unit comprises M ADCs and M down-conversion chains. The signal processor unit is responsible to calculate the complex weights through the received signal. The Transmitter's architecture is very similar to the Receiver. The radio unit comprises M DACs and M up-converter chains. A possibility to develop the new RRHs may be simply connecting each element in the antenna array to a separate transceiver element. This evolution step enables:

- Better spectral efficiency;
- Native redundancy and improved thermal performance, which can enhance the system's reliability;
- Compliance with digital beamforming techniques leading to an improvement of signal's Signal-to-Noise Ratio (SNR), link budget and interference avoidance [18].

3.1.4. Challenges

Massive MIMO systems imply hundreds of AAS, which, from an economic and financial service provider's point of view is a major drawback. Although there are, already, practical proof-of- concepts (prototypes) of these type of systems (the Argos [19] and the Ngara [20] testbeds), the evolution path to these type of systems, leverages a goldmine of research challenges [21], in which FPGAs could have a predominant role:

- AAS' Distributed and Real-Time Processing: The need for distributed and real-time coherent signal processing for the amounts of baseband data that the arrays will generate can be relaxed if the FPGAs were used for local processing (an example is presented in [22]);
- AAS' Costs: The high speed parallel processing capability of FPGAs can be exploited to implement the AAS' Radio Units. Moreover, the recent SDR techniques can be used to dismiss the need for up/down-converters chains (IF or RF sampling), or in the future, the continuous developments that are been made in the All-Digital Transmitters/Receivers field, could even bypass the ADCs/DACs components [7, 8].

3.2. Base Band Unit

The centralized baseband units are built on the concept of SDR, allow sharing the processing resources among different protocols in a dynamic manner. Moreover, this structure, allows for software technologies such as Coordinated Multi-Point (CoMP) processing, Multi-Radio Access Technologies (multi-RAT) virtualization, as well as soft and dynamic cell reconfiguration. Thus, the increase in additional carriers, spectral bandwidth and new technologies can be seamlessly supported by assembling multiple baseband units in a centralized office exploiting the improved pooling efficiency that results from centralizing the base band units. C-RAN is an innovative type of radio access architecture and an essential element of the 5G networks. Enabling the deployment of RRH and AAS with a relatively less cost and easy maintenance.

3.2.1. Flexibility

Another important feature of 5G is the long-term coexistence of multi-RAT. Further, 5G networks will have to deal with a enormous number of base stations, deployed dynamically in a heterogeneous fashion and comprised of multi-RAT like, GSM, UMTS, LTE and Wi-Fi which need to be appropriately integrated. However, the massive deployment of multi-RAT access points brings new challenges such as interference, additional backhaul and mobility management, which 5G systems needs to address [23], leading to suboptimal usage of the resources. Moreover, the technologies mentioned earlier on this paper are raising performance levels on the BBUs, i.e. CA and Massive MIMO, in terms of base band signal's bandwidth and processing latency. Thus, FPGAs and SoCs are ideally suited for meeting the future needs of 5G networks. They can enhance the system integration and flexibility to enable telco operators to respond quickly to the demands.

3.3. Fronthaul

A centralized RAN architecture such as C-RAN introduces a new network domain called fronthaul between the BBU pool and the cell sites where the RRHs are located, enabling the transfer of digital radio samples between these two equipments. The split of the traditional BTS in the BBU and RRH components forced the industry to define a common format for radio samples transfer, promoting the interoperability. The CPRI interface, which the specification targets the two lower layers of the OSI model, is one of those formats. In this standard, the exchange of inform-

ation between the BBU and RRH follows a master-slave architecture and is done in a serial way multiplexing in the time domain 3 different types of information flows: user plane In phase/Quadrature (I/Q) data, C&M and synchronization information. The serialization and de-serialization operations required by this interface can be easily implemented using the FPGA embedded MGT. Currently, the specification 6 of CPRI defines line rates up to 10 Gbps and is compliant with 3GPP UTRA-FDD (UMTS/HSPA+), 3GPP E-UTRA (LTE) and 3GPP GSM/EDGE radio interfaces [24, 25].

ORI is an emerging open interface developed by the ETSI specification group. The ORI interface is built on the top of the CPRI interface adding some new functionality in terms of C&M to the original CPRI protocol. One of the main enhancements is its native support of compression. In the case of CPRI, the compression must be done by third party solutions. As will be explained in the next sub-sections, the compression feature plays a crucial role to enable digital radio communications systems.

3.3.1. Challenges

In order to fulfill the strict latency requirements specified by the 3G and 4G cellular standards referred above, digital radio protocols such as CPRI and ORI imposes tight delay limitations, which includes the propagation of the signal through the transmission channel (typically optical fiber or microwave links), limiting the network reach, as well as the additional digital signal processing that must occur on L1 and L2. For instance, in LTE case, the maximum latency budget of 3 ms related to the Hybrid Automatic Repeat reQuest (HARQ) retransmission mechanisms can't be exceeded due the additional source of delay added by this new network domain. Another limitation is the high bandwidth requirement needed to transmit the digitized I/Q radio samples, especially for multi-sector higher order MIMO modes. Actually, the 10 Gbps defined in CPRI specification 6 are not enough to support a 3 sector MIMO 4x4 single 20 MHz LTE carrier, which demands for a line bit rate of 14.7 Gbps. Taking in consideration that the trend for the future 5G networks is continue to increase the MIMO order with massive MIMO scenarios, the development of efficient compression techniques is mandatory to reduce this bandwidth problem. Present state-

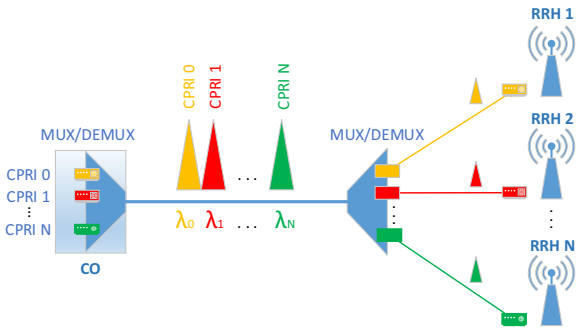


Figure 6. CPRI over WDM techniques.

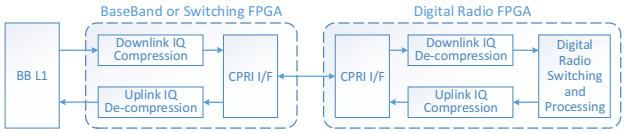


Figure 7. High-level compression blocks in the BBU-RRH link.

of-the-art for compression techniques allows the reach of 3:1 ratios with the target of continue to increase this value. Note that the delay related to the computation of compression algorithms is limited by the specific RAT latency budget referred above, forcing the use of low complexity algorithms running over high speed hardware platforms such as FPGAs [26, 27].

3.3.2. Transport Solutions

Taking into account all the latency and capacity issues referred above, the selection of fiber as the transmission medium is the obvious choice for digital radio transport (D-RoF). The use of dedicated fiber between each BBU-RRH link is a viable solution for operators that already have a large base of dark fiber installed. Another option that is less demanding in terms of fiber resources is the use of passive Wavelength Division Multiplexing (WDM) channels for each CPRI link, such as Coarse Wavelength Division Multiplexing (CWDM) or Dense Wavelength Division Multiplexing (DWDM) technologies. The CWDM technology is a low cost solution that is able to support up to 16 wavelength channels per fiber, each one featuring 2.5 Gbps and reaching transmission distances up to 70 Km. In the case of DWDM, despite a boost of capacity being achieved with up to 80 wavelengths featuring each one 10 Gbps of capacity, the setup cost of this solution is too high [28]. Finally, mapping CPRI in Optical Transport Network (OTN) is also suggested as a possible solution that has the advantage of reach larger network distances due to the Forward Error Correction (FEC) capacity offered by the network. The OTN problem is the delay added by all the processing performed in the network active components. An alternative solution for scenarios where is not possible to install fiber, is the setup of a micro-wave link running at the 2-38 GHz bands, featuring a capacity of 800 Mbps and reaching a distance of 5 to 100 km. For situations where the distance between the BBU and RRH is reduced, E-band mmWaves (71-86 GHz) offer a capacity of up to 2.5 Gbps [3, 29, 30, 31, 32].

The use of high performance FPGAs to improve the fronthaul network performance is not restricted to the implementation of compression algorithms. They can also play a key role in the implementation of high speed interfaces, efficient laser pre-distortion techniques, digital modulation chains that can be used to drive an electro-optical modulator, among others.

3.4. Backhaul

The expected traffic growth in the current decade will put an unprecedented pressure in the backhaul network, setting new substantial requirements of bandwidth and latency. To cope with these new requirements, the existing backhaul links will need a substantial re-engineering. This may pass through evolutionary upgrades, utilization of new techniques, traffic optimizations, or even the exploration of new transmission methods. This section describes the expected evolution path of the backhaul microwave links and how FPGAs can be exploited to realize their architecture physically. Although the main focus are radio links, most of the improvements can be applied to optical fiber links. Further details about optical fiber links were already presented in the fronthaul subsection and can also be applied to the backhaul.

Although radio links present lower bandwidth capacities than the optical fiber ones, their lower CAPEX and the greater flexibility make them a very appealing solution. One of the limitations for the flexibility of the links is the necessity of Line of Sight (LOS) between antennas. However, recent studies proved that links without LOS (NLOS) are feasible in urban environments [33]. The current radio links used in backhaul operate in the microwave band. The problem of using these bands is the fact that the spectrum is almost entirely allocated. This heavy allocation results in very restraining rules by the regulatory authorities, such as very narrow channels. To circumvent these limitations, the best solution is to make a more efficient usage of the spectrum.

To optimize the spectral efficiency, various changes are proposed, like higher order modulations (512, 1024 or higher Quadrature Amplitude Modulation (QAM)) and Co-Channel Dual Polarized (CCDP) links. Although these techniques can drastically increase the throughput, they are not problem-free, mainly the sensitivity of the higher modulation orders to the phase noise, more stringent linearity requirements and the interference between the different polarizations. Thankfully, these problems can be attenuated with techniques such as DPD and Cross-Pole Interference Cancellation (XPIC). While all of these improvements have been known for years, only recently, the newer generations of FPGAs are capable of delivering all of them in the same system, in a cost effective manner [34, 35].

To further increase the link throughput, changes can be made in the bandwidth of the RF channel, namely by optimizing the utilization of the sparse existing spectrum, using wider channels and carrier aggregation. The utilization of wider channels can be complicated. Although it can significantly contribute to a throughput increase, it is dependable on the local regulations and the existing spectrum allocation. Another way to circumvent this problem is to aggregate smaller channels. If the network operator has license for multiple channels, even if they are not contiguous, if they are free, novel backhaul systems can aggregate these multiple channels in the digital domain. Here FPGA, play a fundamental role in the implementation of digital processing at baseband and IF stages, as well as (de)multiplexing data among different carriers, contributing to a significant increase of the system's overall throughput.

Another way to better use the available spectrum is to optimize the traffic in the link. A task easily achieved by the parallel processing power of the FPGAs. In the case of the traffic in the link being Ethernet frames, techniques such as frame header optimization and frame suppression can substantially decrease the overhead inside the link. Payload compression can also be used. But the fact that most traffic data is already compressed and the added latency, make it a far less useful technique [36].

A novel solution to overcome the sparse microwave spectrum is to use different bands. One of the proposals is to use frequencies in the E-band (mmWaves). Although this band can allow for great throughputs, its range is limited by physical factors, like air absorption and rain attenuation. Studies proved that links in these frequencies can reach up to 200m in urban environments, with LOS and non LOS links [37]. While this distances are not optimal for backhaul, they prove that mmWaves can be useful in fronthaul links. Nonetheless, it is predicted that in the future, millimeter Wave links could reach the distance of 5 Km, making them viable for backhaul [32].

The SDR nature of a FPGA based solution allows for an extremely flexible system. This flexibility permits techniques such as Adaptive Coding and Modulation (ACM), variable channel bandwidth selection, asymmetric Uplink (UL)/Downlink (DL) operation and selective channel aggregation. These techniques permit an optimal efficiency of the link for a variety of adverse network conditions.

4. Conclusions

This paper presented an overview of the enabling technologies that many predict to be the basis of the next generation mobile networks. Although the deployment of 5G is targeted only for 2020, the research and prototypes which are already being presented, based on the technologies mentioned above, are promising to reach the expected requirements of the next generation mobile networks. In order to handle the increase of digital signal workload, FPGAs will play an important role in the design of such systems, from the core to the RANs, due to the reasons discusses in this paper. Additionally, the trend for future releases is to converge to massive digital solution such as All-Digital Transceivers and Cloud/Virtualization paradigms, where the reconfigurable hardware will have higher influence, lowering operational costs as well as enhancing QoS.

5. Acknowledgments

This work is funded by National Funds through FCT - Fundação para a Ciência e a Tecnologia under the project PEst-OE/EEI/LA0008/2013.

FCT Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA

References

[1] Cisco, "Cisco Visual Networking Index (VNI): Global Mobile Data Traffic Forecast Update, 2013-2018," February 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

[2] China Mobile, "C-RAN: the road towards green RAN," *White Paper, ver*, vol. 2, 2011.

[3] NGMN Alliance, "Suggestions on Potential Solutions to C-RAN," *White Paper, Version*, vol. 4, 2013.

[4] C. Liu *et al.*, "A Novel Multi-Service Small-Cell Cloud Radio Access Network for Mobile Backhaul and Computing Based on Radio-Over-Fiber Technologies," *Lightwave Technology, Journal of*, vol. 31, no. 17, pp. 2869–2875, Sept 2013.

[5] J. Andrews *et al.*, "What will 5g be?" *Selected Areas in Communications, IEEE Journal on*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[6] J. Mitola, "The Software Radio Architecture," *Communications Magazine, IEEE*, vol. 33, no. 5, pp. 26–38, May 1995.

[7] R. F. Cordeiro *et al.*, "Gigasample Time-Interleaved Delta-Sigma Modulator for FPGA-Based All-Digital Transmitters," in *Digital System Design (DSD), 2014 17th Euromicro Conference on*, Aug 2014, pp. 222–227.

[8] R. Cordeiro, A. Oliveira, and J. Vieira, "All-Digital Transmitter with RoF Remote Radio Head," in *Microwave Symposium (IMS), 2014 IEEE MTT-S International*, June 2014, pp. 1–4.

[9] "The METIS 2020 Project - Laying the foundation of 5G." [Online]. Available: https://www.metis2020.com/

[10] C. Baylis *et al.*, "Solving the spectrum crisis: Intelligent, reconfigurable microwave transmitter amplifiers for cognitive radar," *Microwave Magazine, IEEE*, vol. 15, no. 5, pp. 94–107, July 2014.

[11] T. Wang and B. Huang, "Millimeter-wave techniques for 5g mobile communications systems: Challenges, framework and way forward," in *General Assembly and Scientific Symposium (URSI GASS), 2014 XXXIth URSI*, Aug 2014, pp. 1–4.

[12] Y. Wang *et al.*, "5g mobile: Spectrum broadening to higher-frequency bands to support high data rates," *Vehicular Technology Magazine, IEEE*, vol. 9, no. 3, pp. 39–46, Sept 2014.

[13] IEEE, "*IEEE Standard for Information Technology Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: Enhancements for Very High Throughput in the 60 GHz Band*. IEEE Standard 802.11ad-2012," 2012. [Online]. Available: http://standards.ieee.org/findstds/standard/802.11ad-2012.html

[14] Altera Corporation, "Simplifying Simultaneous Multimode RRH Design," Altera, White Paper, 2009.

[15] Texas Instruments, "Digital Radio Front-End strategies provide game-changing benefits for small cell base stations," Texas Instruments, White Paper, 2013.

[16] F. Gross, *Smart antennas for wireless communications*. McGraw-Hill Professional, 2005.

[17] C. A. Balanis and P. I. Ioannides, *Introduction to Smart Antennas*. Morgan & Claypool Publishers, 2007, vol. 2, no. 1.

[18] Commscope, "Active Antennas: The Next Step in Radio and Antenna Evolution," Commscope, White Paper, 2011.

[19] C. Shepard *et al.*, "Argos: Practical Many-Antenna Base Stations," in *Proceedings of the 18th annual international conference on Mobile computing and networking*. ACM, 2012, pp. 53–64.

[20] H. Suzuki *et al.*, "Highly Spectrally Efficient Ngara Rural Wireless Broadband Access Demonstrator," in *Communications and Information Technologies (ISCIT), 2012 International Symposium on*. IEEE, 2012, pp. 914–919.

[21] E. Larsson *et al.*, "Massive MIMO for Next Generation Wireless Systems," *Communications Magazine, IEEE*, vol. 52, no. 2, pp. 186–195, February 2014.

[22] P. Murphy *et al.*, "FPGAs Help Characterize Massive-MIMO Channels," *Xcell Journal*, no. 89, pp. 18–24, October 2014.

[23] C. Rowell *et al.*, "Towards Green Soft: A 5G Perspective," *IEEE IEEE Wireless Commun. Magazine*, vol. 52, 2014.

[24] Ericsson AB et al., "Common Public Radio Interface (CPRI), Interface Specification v6.0," August 2013. [Online]. Available: http://www.cpri.info/downloads/CPRI_v_6_0_2013-08-30.pdf

[25] JDSU, "Cloud-RAN Deployment with CPRI Fronthaul Technology," JDSU, White Paper, December 2013.

[26] Aviat Networks, "5 Things you Should Know about Fronthaul," Aviat Networks, Tech. Rep.

[27] Light Reading Webminar, *Cloud RAN: The impact of base station virtualization on low latency, high bandwidth front hauling*, Light Reading, Heavy Reading and Xilinx, March 2014.

[28] Fiberstore, "CWDM Technology Transceivers Wiki," 2014. [Online]. Available: http://www.fiberstore.com/CWDM-Technology-Transceivers-Wiki-aid-397.html

[29] Transmode, "Mobile Fronthaul," Transmode, Application Note.

[30] Alcatel-Lucent, "Mobile Fronthaul for Cloud-RAN Deployment," Alcatel-Lucent, Application Note.

[31] Future Network & MobileSummit, "Optical fiber solution for mobile fronthaul to achieve Cloud Radio Access Network," Future Network & MobileSummit, Tech. Rep., 2013.

[32] Tellumat, "Microwave Point to Point Links." [Online]. Available: http://www.tellumat.com/communications/wireless-solutions/micro-p-to-p-links.htm

[33] M. Coldrey *et al.*, "Non-Line-of-Sight Microwave Backhaul in Heterogeneous Networks," in *Vehicular Technology Conference (VTC Fall), 2013 IEEE 78th*, Sept 2013, pp. 1–5.

[34] S. Little, "Is Microwave Backhaul up to the 4G task?" *Microwave Magazine, IEEE*, vol. 10, no. 5, pp. 67–74, August 2009.

[35] R. Branson *et al.*, "Digital Pre-distortion for improving efficiency, linearity and average output power of microwave Point-to-Point power amplifiers used in cellular backhaul telecommunication systems," in *Microwave Symposium Digest (MTT), 2012 IEEE MTT-S International*, June 2012, pp. 1–3.

[36] Aviat Networks, "Improving Microwave Capacity." [Online]. Available: http://www.slideshare.net/AviatNetworks/improving-microwave-capacity

[37] G. MacCartney and T. Rappaport, "73 GHz millimeter wave propagation measurements for outdoor urban mobile and backhaul communications in New York City," in *Communications (ICC), 2014 IEEE International Conference on*, June 2014, pp. 4862–4867.

FPGA-based All-digital FM Transmitter

Rui F. Cordeiro, Arnaldo S. R. Oliveira and José Vieira
Departamento de Electrónica, Telecomunicações e Informática and Instituto de Telecomunicações
Universidade de Aveiro
ruifiel@ua.pt, arnaldo.oliveira@ua.pt, jnvieira@ua.pt

Abstract

New wireless communication scenarios encourage an additional effort to find alternative agile radio systems. The development of more flexible radios capable of adapt themselves to the new network restrictions and user demands requires the use of versatile physical layers. In this paper the concept of all-digital transmitter is explained with a trivial example. A FM transmitter implementation using an all-digital transmitter approach is shown to explain the main characteristics of the all-digital transmitter, its advantages and disadvantages.

The experimental results obtained show the feasibility of this approach as an alternative to conventional radio transmitters, allowing further improvements in terms of system reconfigurability.

1. Introduction

Over the last years in telecommunications, there has been a change of paradigm for modern wireless networks. An increasing number of users has led to the need of more base-stations and access points to fulfill utilization demands. At the same time, there has been a convergence of different wireless standards to a single radio unit, leading to more flexible and cost effective wireless networks.

A possible solution for the next generation wireless communications is the use of Software Defined Radio (SDR). The concept of SDR allows to create a flexible radio for modern wireless communications capable of adapt itself to different scenarios [1]. Using software and digitally reconfigurable hardware, SDR systems are capable of transmitting radio frequency (RF) signals with transmitter and receiver chains implemented by means of software. With the SDR approach, a higher degree of flexibility would be possible for the transmitter and receiver chains.

A new technology that might grant a faster proliferation of SDR systems is the all-digital transmitter (ADT). ADT are a more flexible and potentially more efficient alternative to conventional radio transmitter systems. The principle behind these transmitters is the complete generation of RF signals using only digital logic, with a digital data-path from the baseband up to the RF stage [2].

The are still some limitations for ADT to be used in more demanding wireless communication scenarios. Many times these systems need high speed digital logic to be able to achieve the necessary signal frequencies for most of the standards used. Common standards, such as LTE, WiMaX or WiFi, have carrier frequencies that can go up to the units of GHz. ADT must be able to achieve these clock frequencies to be a competitive alternative for commonly used standards. Keeping a low cost for the transmitter design while satisfying these requisites might be a challenge.

Other limitation yet to be fully addressed, is the noise in these transmitter systems. Most of ADT use some variety of pulse width modulation or delta-sigma modulation which results into a noisy output spectrum. Both modulations create a lot of quantization noise which decreases the system overall power efficiency [3][4] and reduces its flexibility due to the necessary analog filter.

Most of wireless communications use frequency bands in the GHz scale which imply the use of high-end FPGAs and digital transceivers capable of achieve those frequencies. In this paper however, it is shown the basic ADT structure, in order to design a functional transmitter and a practical example of it on a low-cost FPGA. An FM transmitter was chosen due to the lower carrier frequencies needed and the simplicity of the modulation.

The paper, first shows an FPGA-based ADT architecture and its functional blocks are explained in more detail. Then a simple practical project is introduced as a concept example. In this example, frequency modulation was used at the baseband transmitter stage. Details about this particular implementation are in section 3 of this article. Section 4 shows the utilization and experimental results of the transmitter. Lastly, some final remarks and conclusions are made about this work.

2. ADT Basic Concepts

The all-digital transmitter is a more flexible and potentially more efficient alternative to conventional radio transmitter systems. It is a system-on-chip (SoC) solution for Software Defined Radio for their potential flexibility and reconfigurability. While the completely digital generation of RF signals is already a well known concept [5], the full advantages in flexibility, reconfigurability and integration are yet to be fully explored.

A technology that promises to be an interesting platform to the ADT concept is the field-programmable gate array (FPGA). Hardware integration, reconfigurability, flexibility and processing power in modern FPGAs seem to be a convenient solution for a modern fully integrated SDR sys-

tem.

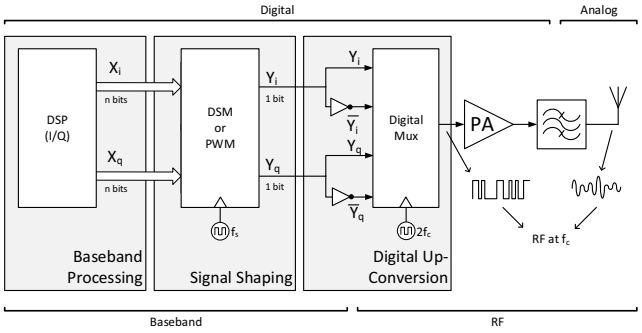


Figure 1. General all-digital transmitter architecture.

The ADT transmitter system presented in figure 1 can be divided into three different blocks: the baseband processing, the signal shaping and the digital up-conversion. The baseband processing block is similar to conventional transmitters with digital baseband processing. This block is responsible for the signal modulations according to a specific protocol or communication standard. These blocks are many times already implemented in FPGA systems due to their processing capability.

The signal shaping block is usually done using delta-sigma modulation (DSM) or pulse width modulation (PWM). The advantage of using these modulations is to have a 2-level representation of the original baseband signal with only two signal amplitude levels. These modulations add a certain amount of noise to the signal, the quantization noise. However, using a high sampling ratio for the signal, it is possible to shape this noise out of the signal's bandwidth. This signal can be recovered afterwards using an analog filter with a passband over signals band.

The digital up-conversion block shifts the baseband signal to the desired carrier frequency. With carrier frequencies being in the hundreds of MHz to tens of GHz, this block has to deal with considerable high clock frequencies. In fact, the output of this block needs a clock frequency at least equal to two times the carrier frequency. Several solutions for this block have been already presented in the past, such as the use of a bandpass delta-sigma modulator [5], high speed multiplexers [6] or multi-gigabit serializers [7].

3. FPGA-based FM All-Digital Transmitter

This section presents the implementation details of the FM all-digital transmitter as an application example of an ADT implemented on a low cost FPGA. An overall system overview is done and in the following subsections each individual block is explained with more detail.

An FM transmitter was designed as a simple explanatory project of FPGA-based all-digital transmitter. Although not a very complex system, the implemented transmitter can give an idea of the working principle for these transmitters.

The entire project is design using Digilent Atlys board with a Spartan-6 FPGA, hence this a very simple project with low resource needs that can be easily ported to other FPGA families. Besides the FPGA, a LM4550 integrated circuit with a AC-97 codec was used for the project Atlys board. The transmitter design was made using Xilinx Design Tools, namely the ISE tool and the System Generator.

The FM ADT presented in figure 2 uses a very simple architecture that can be divided into three main blocks. The baseband signal acquisition, the frequency modulation (FM) stage and the all-digital transmitter.

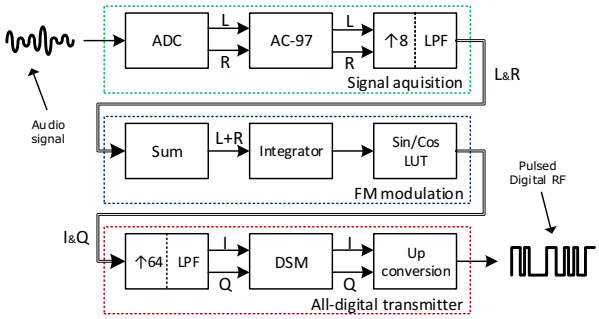


Figure 2. Implemented FPGA-based all-digital FM transmitter.

3.1. Signal acquisition

The baseband signal used was a conventional stereo audio signal sampled using two audio ADCs. The source signal comes from a stereo audio jack connected to the LM4550 integrated circuit in the Atlys board.

The signal is sampled at 48kHz using a LM4550 with a AC-97 audio codec. This signal is then up-sampled for a 384kHz sampling frequency. The up-sampling is necessary to be done before the FM block because the FM signal can achieve up to 100kHz bandwidth.

3.2. Frequency modulation

In this stage, a FM modulation of the up-sampled audio signal is done. The left and right audio signals are summed together for a mono FM signal. This signal is then integrated using an accumulator. The output of the accumulator is used has the phase of a sine and cosine look-up table. This results into a quadrature modulated (IQ) signal (see equation (1) and (2)) with in phase, $s_i(t)$, and in quadrature, $s_q(t)$, components.

$$s_i(t) = cos(2\pi f_\Delta \int_0^t x(\tau)d\tau) \quad (1)$$

$$s_q(t) = sin(2\pi f_\Delta \int_0^t x(\tau)d\tau) \quad (2)$$

Where $x(t)$ is the left and right audio signals added together. The value of the modulation index f_Δ can be adjusted changing the AC-97 codec gain.

 ISBN: 978-989-98875-2-7

3.3. All-digital transmitter

Delta-sigma modulation converts a signal with high resolution into a lower resolution signal at the trade of an higher sampling frequency. The signal is quantized to a 1-bit signal, which is directly converted to a digital 2-level output. This allows to have a signal output directly from a digital buffer instead of using a multi-bit signal and a digital-to-analog converter (DAC).

Using an up-sampled version of the original signal, it is possible to add only a minimum of quantization noise in the signal's bandwidth. For a better signal to quantization noise ratio (SQNR) the DSM loop filter reduces the feedback error for the low frequencies, shaping the noise out of the signal's bandwidth.

With quantization noise further away from the signal's frequency band, wider bandwidth signals can be accepted and the quality factor for the output reconstruction filter can be reduced. The DSM's frequency choice was 24.576MHz which is an integer multiple of the FM block frequency 384kHz. To adjust the sampling frequency for the delta-sigma modulation an interpolation filter with 64 a interpolation factor is needed (all-digital transmitter block in figure 2). A filter with such a narrow band would necessarily have a high order. To save FPGA resources this filter is implemented by cascading three low-pass filters with an interpolation factor of 4 each to achieve the interpolation factor of 64.

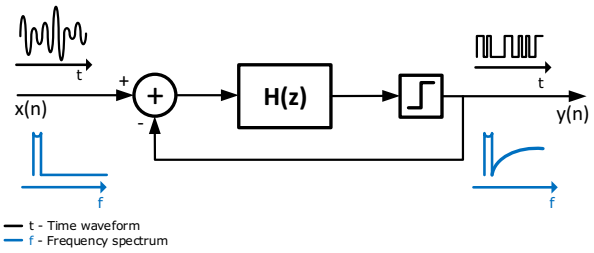


Figure 3. General delta-sigma modulator.

Figure 3 shows a generic delta-sigma modulator. The filter $H(z)$ is the loop filter and can be used to change the noise shaping characteristics of the modulator.

The modulator in this design, presented in figure 4, is a second-order low-pass DSM. It uses a cascade of integrators with distributed feedback topology. The choice of topology was motivated by the lower critical path in the selected one. This is fundamental to allow a higher sampling frequency for the signal which moves away the quantization noise from the signal's band.

The digital up-conversion stage, is a pattern building block followed by a serializer. The pattern building block combines the outputs of both I DSM (v_i) and Q DSM (v_q) and its inverted versions. The word at the pattern building block is a repetition of $[v_i\ \overline{v}_q\ \overline{v}_i\ v_q]$ with a sampling frequency two times superior to the desired carrier frequency.

Because the delta-sigma modulators frequency is 24.576MHz, the easier frequency to achieve in the FM

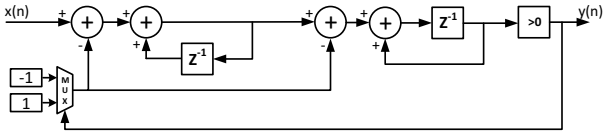


Figure 4. Second-order delta-sigma modulator used in the FM transmitter.

band is 98.304MHz. This means that the output signal must have a 196.608MHz clock. For this purpose a OSERDES 8-to-1 serializer was used with the pattern $[v_i\ \overline{v}_q\ \overline{v}_i\ v_q\ v_i\ \overline{v}_q\ \overline{v}_i\ v_q]$

3.4. FPGA Resources

For real communication scenarios it is of great importance to have transmitter systems that don't consume many hardware resources. If an FPGA-based ADT can be implemented with low usage of logic resources, one can take advantage of the already present baseband processing unit and use the same SoC to add the all-digital transmitter. This not only adds reconfigurability options to the radio transmitter, but also allows to decrease the cost of the system.

Although this is merely a demonstrating example of a simple all-digital architecture, one can analyze the logic resources of the transmitter to understand the low resource utilization even for a small FPGA. Table 1 shows the resources used for the implemented FM modulator.

Table 1. All-digital FM transmitter total occupied resources on Spartan-6 FPGA

Logic resources	Used	Total	Percentage used
Flip Flops	3404	54576	6%
LUTs	1912	27288	7%
DSP48A1	52	58	89%
OSERDES2	2	376	1%

It is possible to see that for the majority of the resources the ADT occupies a small portion of the FPGA logic. The most occupied resource is the DSP units with 89% of occupation. This however only happens because this FPGA family has a very low number of DSPs when compared with other more recent families. Moreover, the high order interpolation filter used cause a great relative occupation of this type of resource.

4. Experimental Results

For testing, some different types of signal were used. Figures 5 to 7 show the resulting signals from different frequency sine waves and audio music being transmitted over the FM ADT.

Figure 5 shows a large span area to show the harmonic repetitions generated by the square wave carrier. The repetitions happen because the up conversion wave is a square

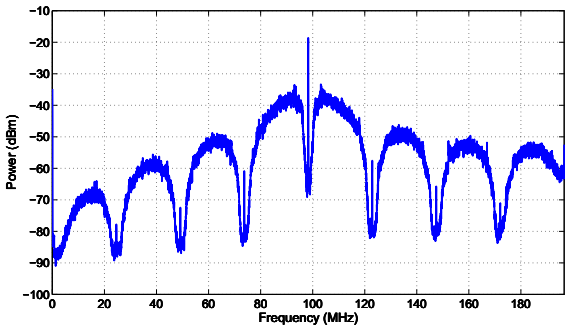


Figure 5. Transmitted signal with 200MHz frequency span. The carrier frequency is centered at 98.3MHz

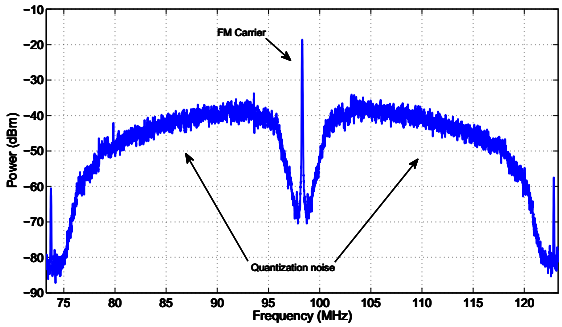


Figure 6. Transmitted signal with 50MHz frequency span. The carrier frequency is centered at 98.3MHz

wave, and the baseband signal will appear at the fundamental carrier frequency and at its harmonics.

In figure 6 a smaller span is used to show the noise shaping characteristic of the delta-sigma modulation. It is possible to see that this system generates a considerable amount of quantization noise outside the signal's bandwidth which is not desirable. Nonetheless, this noise is shaped out of the signal's band by the DSM, which makes the possible to recover the transmitted signal.

Figure 7 shows a more detailed view of the signal's

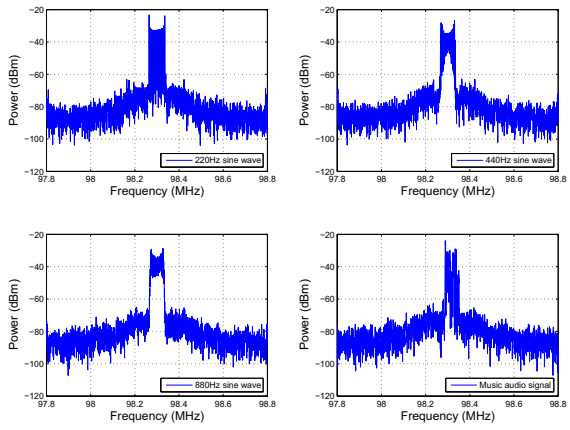


Figure 7. Signal's bandwidth detail at the carrier frequency with different input signals: a 220Hz, 440Hz and 880Hz sine waves and music audio signal.

bandwidth. Four different signals are shown including a real audio signal. The frequency modulation index was set so that the modulated signal has about 80kHz bandwidth.

The FM signal can effectively be transmitted over the FM band without any additional use of amplifier for a reasonable radius in the order of tens of meters. Further amplification and filtering would be necessary to have a fully functional FM transmitter system, with a completely digital RF generation.

5. Conclusions

In this paper it is shown a very simple yet practical FM transmitter using only digital logic. The FPGA embedded FM transmitter was implemented in a low cost FPGA development board and it was shown to work as expected. The concept of all-digital transmitter is shown with a very practical and simple example. Such systems benefit from the digital signal processing capabilities and digital logic reconfiguration to add an increased degree of flexibility to the conventional RF transmission, getting closer to the ideal SDR system.

Acknowledgment

This work was partially supported by the Portuguese Foundation for Science and Technology (FCT) under PhD scholarship ref. SFRH/BD/91533/2012 and by the Instituto de Telecomunicações under project CReATION (ref. EXCL/EEI-TEL/0067/2012.).

References

[1] J. Mitola and G Q. Maguire. Cognitive radio: making software radios more personal. *Personal Communications, IEEE*, 6(4):13–18, Aug 1999.

[2] M.M. Ebrahimi, M. Helaoui, and F.M. Ghannouchi. Delta-sigma-based transmitters: Advantages and disadvantages. *Microwave Magazine, IEEE*, 14(1):68–78, 2013.

[3] M.M. Ebrahimi, M. Helaoui, and F.M. Ghannouchi. Improving coding efficiency by compromising linearity in delta-sigma based transmitters. In *Radio and Wireless Symposium (RWS), 2012 IEEE*, pages 411–414, Jan 2012.

[4] N.V. Silva, A.S.R. Oliveira, and N.B. Carvalho. Design and optimization of flexible and coding efficient all-digital rf transmitters. *Microwave Theory and Techniques, IEEE Transactions on*, 61(1):625–632, 2013.

[5] J. Keyzer, J. Hinrichs, A. Metzger, M. Iwamoto, I. Galton, and P. Asbeck. Digital generation of rf signals for wireless communications with band-pass delta-sigma modulation. In *Microwave Symposium Digest, 2001 IEEE MTT-S International*, volume 3, pages 2127–2130, 2001.

[6] M. Helaoui, S. Hatami, R. Negra, and F.M. Ghannouchi. A novel architecture of delta-sigma modulator enabling all-digital multiband multistandard RF transmitters design. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 55(11):1129–1133, 2008.

[7] N.V. Silva, A.S.R. Oliveira, U. Gustavsson, and N.B. Carvalho. A novel all-digital multichannel multimode RF transmitter using delta-sigma modulation. *Microwave and Wireless Components Letters, IEEE*, 22(3):156–158, 2012.

Aplicações de alto desempenho

 ISBN: 978-989-98875-2-7

Host to Accelerator Interfacing Framework for High-Throughput Co-Processing Systems

Nuno Neves, Pedro Tomás, Nuno Roma

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Rua Alves Redol, 9, 1000-029 Lisboa - Portugal

{*nuno.neves, pedro.tomas, nuno.roma*}*@inesc-id.pt*

Abstract

To mitigate the design effort related with the interface between custom accelerators implemented in FPGA devices and the host computer through high-speed PCI Express interconnections, a generic and highly flexible framework is presented. The framework comprehends both a hardware layer in the FPGA device, composed of a generic accelerator architecture and a communication interface bridge, and a software layer at the host, comprising a device driver and an API to facilitate application development. Complementary to the presentation of the involved hardware resources and resulting communication throughput, it is also discussed the advantages of deploying such a complete toolchain for FPGA devices through two real case studies, presenting application scenarios that take advantage of the proposed framework as a base system.

1. Introduction

The increasing demand for computational processing power observed along the past decade has driven the development of heterogeneous systems composed of one or more processing devices. Such systems typically include a host General Purpose Processor (GPP) and one or more accelerating devices, such as a Graphics Processing Unit (GPU) or a Field-Programmable Gate Array (FPGA), each integrating multiple Processing Elements (PEs). The adoption of such systems allows for significant application accelerations, which in most cases not only improves the processing performance but also increases its energy efficiency. However, in what concerns the investigation of dedicated accelerator architectures in FPGA devices, researchers usually struggle when creating communicating data schemes between the dedicated hardware and a host computer.

Although there are several well-established toolchains that abstract such issues for GPU devices (such as CUDA and OpenCL), not enough attention as been given to FPGA-based accelerators, specially in what concerns the communication between the host computer and the FPGA device. This probably results from the fact that FPGA devices are still mostly seen as prototyping devices and only recently began being used as CPU-coupled devices, targeting the deployment of dedicated co-accelerators.

Even though some recent devices are already deployed with tightly-coupled CPUs, embedded in full System-On-Chips (SoCs) that already integrate all the necessary communication infrastructures between the CPU and the reconfigurable fabric (e.g. Xilinx Zynq devices), the communication is still an issue in stand-alone FPGA devices. This is mainly due to the fact that, in such cases, interfacing an FPGA design with a host CPU requires the latter to adhere to complex interface protocols, specific to the available interconnection link (e.g. PCI Express (PCIe)). Moreover, if satisfactory performance is to be achieved, a reasonable amount of effort must go into the definition and implementation of the hardware structures that support and implement those interfaces, such as communication bridges or Direct Memory Access (DMA) controllers.

Although modern tools/compilers already support the translation of C- or OpenCL-based kernels to RTL (e.g. Vivado from Xilinx and AOCL from Altera), as well as their integration in fully functional FPGA-based systems, the communication between the host and the FPGA must still be deployed by the developer. Furthermore, custom or adapted device drivers must also be created to handle all the low-level interactions between the host and the device, and provide a device-programming interface for the host. Moreover, despite the current availability of extensive documentation concerning the usage of the various Application Programming Interfaces (APIs) provided by the Linux kernel to interface with these types of devices, developing a device driver from scratch is rather time consuming and error prone. To avoid such drawbacks, a number of communication libraries and device drivers have already been proposed that are capable of interfacing a host CPU and an FPGA through a PCIe interconnection [1, 2, 3]. Some of them also present a part of the communication chain. However, most of the remaining communication and interfacing infrastructures are implementation-specific and must still be defined.

To tackle the described issues, this manuscript presents a generic framework for interfacing a host CPU with custom accelerators implemented in FPGA devices, through a high-speed PCIe interconnection. The framework deploys

a fully and integrated application stack, ranging from high-level software to low-level hardware layers, comprising: *i)* a generic and scalable accelerator (comprising one or more PEs), composed of a communication and management controller and the basic building blocks for creating custom accelerators (supporting custom intercommunication networks and shared memory topologies); *ii)* an FPGA base system, containing a Host Interface Bridge (HIB) interfacing the custom accelerator with the PCIe interconnection; *iii)* a device driver largely based on the driver provided in the MPRACE framework [1], that can be easily adapted to the proposed framework; and *iv)* a low-level API that provides micro-routines for architecture-independent control over the accelerator's PEs and for the data transfers between the host and the accelerator's global memory.

The manuscript describes the proposed framework starting from the lower hardware layers up to the higher-level software layers. Hence, the remainder of the manuscript is organized as follows: Section 2 provides a high-level description of the proposed framework, introducing each of its components and the stack organization; Section 3 presents a generic and scalable accelerator architecture and details the accelerator controller; Section 4 describes the device driver and the required hardware modules to interface the PCIe link with the accelerator; Section 5 presents a discussion on the advantages of the proposed framework, by considering two case studies where the herein presented framework is used as a base system; Section 6 concludes the manuscript, addressing the main contributions and achievements.

2. Heterogeneous Computing Framework

The devised framework targets heterogeneous systems comprising a CPU that is connected to an FPGA device, through a high-speed interconnection. In the scope of this manuscript the system is assumed to integrate a Xilinx 7-Series FPGA device, connected to a host x86 CPU through a PCI Express interconnection. This not only results from the chosen base device driver (that targets older PCIe-connected Xilinx devices), but also from the ease of IP core integration in the system provided by the latest Xilinx toolchains. However, depending on the availability and support for PCIe communication, and integration with custom hardware, the proposed framework can be adapted to be used with different FPGA devices.

As previously mentioned the framework stack is composed of both software and hardware layers, integrated in the CPU and FPGA device, respectively. A base system is deployed on the reconfigurable fabric of the FPGA device, composed of two main hardware modules (see Fig. 1): *i)* a Host Interface Bridge (HIB), that handles all communication with the PCIe, and *ii)* the accelerator itself. The accelerator comprises a specially devised controller for host-communication and execution management and monitoring, an area reserved for the user's custom processing architecture and a global memory.

Throughout this manuscript the accelerator's architecture is maintained as generic and scalable as possible, i.e.

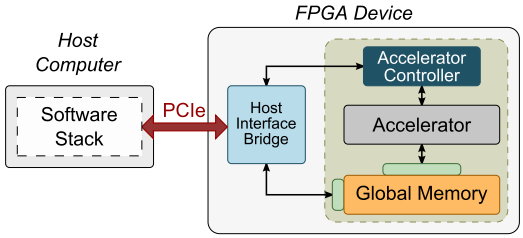


Figure 1. Heterogeneous computing framework overview.

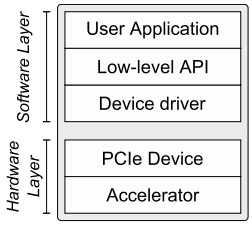


Figure 2. Framework integrated stack, depicting both the software and the hardware layers.

only the main structures and interfaces are actually described, since they must be compatible with the remainder of the system. Everything from PE architectures, to internal bus topologies and memory hierarchy is assumed as application-specific and are left for the user to implement.

Accordingly, the accelerator's processing architecture is designed so that the user can deploy any sort of dedicated processing structures with any sort of organization, ranging from high-footprint dedicated single-core processors [4], to dense and heterogeneous many-core processing structures [5] or even multiple individual accelerators (e.g. systolic arrays [6]). Hence, each PE is deployed through a generic wrapper that contains the custom hardware modules and the necessary structures and interfaces for its integration in the system.

The communication with the deployed PEs is achieved with a message-passing-type backbone communication infrastructure that connects the accelerator controller to each of the PEs. Moreover, both the included global memory and the internal communication structures, which constitute the memory hierarchy (e.g. cache structures or First-In First-Out (FIFO) buffers) and PE inter-communication structures (e.g. shared bus interconnection or Network-on-Chip (NoC)), can be tailored according to user and application requirements.

The software layer is composed of two modules deployed in the CPU (see Fig. 2): *i)* a device driver, that maps a PCIe device to the CPU address space and allows direct communication between the accelerator and the operating system; and *ii)* an API that provides higher level routines that can be fully integrated in a user application in order to load code and data to the processing cores local memory, manage the PEs execution and perform coarse-grained data transfers between the CPU and the accelerator's global memory.

 ISBN: 978-989-98875-2-7

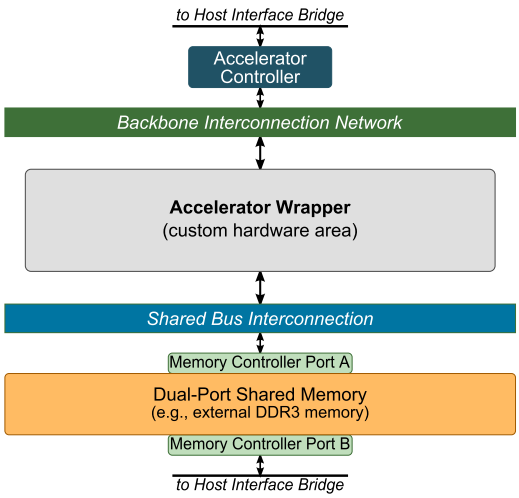


Figure 3. Top-level accelerator architecture.

3. Scalable Heterogeneous Accelerator

In order to aid the integration of dedicated processing hardware in the framework, a heterogeneous accelerator (comprising the lower level layer of the framework) was devised and made as generic and scalable as possible. In fact, it was designed in order to provide only the necessary infrastructures to include the user's custom hardware modules. For such purpose, the accelerator is composed of four main structures (see Fig. 3): *i)* an *accelerator controller*, responsible for interfacing the accelerator and the HIB (detailed in Section 4.1), and controlling and monitoring the accelerator's execution; *ii)* an *accelerator wrapper*, whose basic building block is a generic *processing core*, containing the required modules and interfaces to integrate custom PEs in the accelerator; *iii)* a *backbone communication structure* to connect the *accelerator controller* to the PEs; and *iv)* a *global shared memory* accessible by the host CPU (through the HIB) and by the accelerator's PEs (through a simple shared bus, replaceable by a custom structure suited to the user application). The following subsections describe each of the accelerator's components in detail.

3.1. Processing Core

As described above, the heterogeneous accelerator architecture was designed by following a fully modular approach. In particular, each processing core was designed in order to be as independent as possible of the underlying PE architecture. Each processing core (see Fig. 4) is composed of: *i)* an area reserved for the PE itself; *ii)* a local private memory, with configurable size; *iii)* an internal communication interface; and *iv)* a core controller, for receiving commands from the accelerator controller, for loading code and data to the local memory and signaling the accelerator controller when the PE finishes execution.

This approach allows a wide range of PE architectures to be used within the accelerator. In fact, the only imposed restriction to the architecture of the PEs is concerned with the provision of straightforward communication interfaces

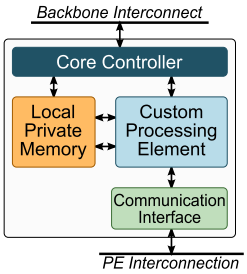


Figure 4. Processing core wrapper architecture. The PEs architecture can be custom-tailored according to user and application requirements.

to the attached core controller, to the local memory and to the bus interface.

Given that a large number of individual processing cores can be deployed, an optional higher-level structure is also provided that allows the user to group the cores in processing clusters. This not only provides a better organization of the processing structure, in terms of monitoring and control, but also allows the user to define clusters of cores with different architectures and specific internal communication structures, envisaging truly heterogeneous processing structures. Moreover, it allows the user to include a cluster local memory or cache, further increasing the system flexibility by allowing a number of different memory hierarchies to be deployed.

3.2. Backbone Communication Infrastructure

The communication between the accelerator controller and the PEs was achieved by a specially devised bus interconnection, partly based on the AXI-Stream Protocol [7]. The implemented interconnection provides single-cycle communication between a master and up to 16 peripherals. Consequently, this interconnection features two independent unidirectional channels: a one-to-many channel and a many-to-one channel.

The first channel routes data signals from the master to the peripherals in one of two ways: *i)* by establishing a direct connection between the master and one of the peripherals, by using a decoder driven by the identification of one of the peripherals; or *ii)* by making use of a broadcast mask to simultaneously connect to a set of the peripherals.

The second channel routes data signals from each of the peripherals to the master. The channel is managed by a round-robin arbiter, driven by request and acknowledge signals, from and to the peripherals, respectively.

Finally, since any number of processing cores can be deployed, a bus bridge (capable of registering and forwarding data signals) was devised to allow the creation of multiple instances of the interconnection bus and stack them together in multi-level way. With this infrastructure, it is possible to create an interconnection bus, capable of supporting an arbitrary number of levels and allowing the connection of all the processing cores to the accelerator controller, provided that each level adds a clock cycle of latency to the communication, necessary to register and forward data signals between consecutive levels. The additional clock cycle

per consecutive bus level aids in keeping the critical path of the interconnection from increasing significantly, resulting in a minimal impact in the maximum operating frequency of the bus interconnection.

3.3. Accelerator Controller

A controller was specially devised to manage the accelerator. The controller's architecture (depicted in Fig. 5) comprises a set of *control registers* and *status registers*, a state machine that handles requests from the host and issues commands to the accelerator and an interface to the backbone intercommunication.

Besides interfacing the accelerator with the host CPU through the HIB, the controller is also responsible for managing and monitoring the processing cores execution and for loading the cores' local memories. The controller is directly mapped to the host's user address space (as described in Section 4). This allows the host to send commands to the controller by configuring a set of *control registers* mapped on its address space. Hence, a host memory address is reserved for each of the registers and to access them from the host it is only required to issue a read/write to the corresponding address. Also, a set of *status registers* (with a configurable number and size) according to the user's processing structure organization, is maintained by the controller indicating the execution state of each processing core ('1' or '0', indicating running and stopped states). This way, the host can easily monitor the cores' execution by reading the values in each register.

Three commands are available for the host to control the accelerator, and hence the processing cores: *i)* `RST` issues a system reset, reverting all cores to a reset state and clearing the values of the status registers; *ii)* `RUN` starts the execution of specific cores, according to the command configuration; and *iii)* `LOAD` sends a word to a number of core local memories, according to the command configuration. To allow the issue of these commands to the accelerator by the controller, the host must conveniently configure the following four control registers:

- **`command`** - indicates the command to be executed;
- **`brdcast`** - command broadcast mask, which indicates to which cores the command is sent to. The mask must be defined by the user according to the processing structure organization;
- **`procid`** - identification of the target processing core to which the command is sent, in case the broadcast mask was not set, allowing commands to be sent to individual cores;
- **`address`** - only used for the `LOAD` command, indicating the address in the cores' local memory to which the word is to be copied.

As it will be described in Section 4, the accelerator controller will be memory mapped to the host address space. This way, each of the described registers can be accessed in the accelerator's assigned address range, starting at a base

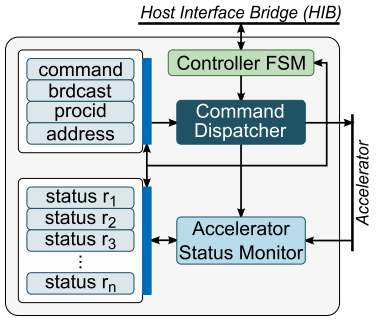


Figure 5. Accelerator controller architecture. A FSM is used to handle all requests from the host CPU, activate the command dispatcher that issues commands to accelerator according to the control registers and report the accelerator status to the host.

address (hereafter referred to as `BASEADDR`) and assuming 32-bit memory locations. This way, the host reaches each of the control and status registers starting at the address `BASEADDR+4`. The `BASEADDR` position is used by the host to issue the command itself, i.e. when the controller receives a write request to this address, it issues a command to the accelerator according to the configuration of the control registers. If a `LOAD` command is to be issued, the controller sends the data word (sent by the host to the `BASEADDR` location) to the local memories of the targeted cores. Furthermore, when the `LOAD` command completes, the address in the `address` control register is automatically incremented, which means that when the host is loading the core's memories, it is only required to configure the command once and send all the subsequent data.

When a `RUN` command is issued, the controller updates the status registers according to the processing cores being started. Accordingly, a monitoring module was devised that receives messages from the processing cores, upon their execution completion, and updates the corresponding status registers.

3.4. Shared Memory Access

To allow each processing core to access the global memory, a different interconnection module was derived from the backbone bus, in order to provide a shared bidirectional channel. Hence, while maintaining the same base structure, it is possible to obtain an arbitrated shared bus interconnection. This is achieved by adding a memory address signal and by including memory and core interfaces that deploy a simple shared bus protocol.

Hence, the devised interconnection provides a simple connection to the main memory by the processing cores. This eases the development of custom accelerators at an early stage, in the sense that the user only needs to include custom hardware modules (in the accelerator wrapper) in order to deploy a fully functional processing structure, supported by a global memory. In fact, the bus can be easily replaced by more complex structures, such as cache-based memory hierarchies, distributed memory structures, or even FIFO-based stream interconnections.

The devised global memory provides a dual-port configuration and is managed by a dedicated memory controller (either a DDR3 controller for off-chip memory or a standard BRAM on-chip implementation). The dual-port configuration makes the memory simultaneously accessible by the accelerator's PEs and by the host (through the HIB). Since the memory hierarchy and the interconnection topologies are implemented by the user, it is the user's responsibility to maintain the memory consistency, i.e. when performing transfers to/from the global memory, the host must assure that no memory regions being accessed by the PEs become corrupted.

4. Host-Accelerator Communication

Having defined a generic and scalable accelerator architecture, the framework must be capable of enabling efficient communication means between the accelerator and the host computer. For such purpose, since the FPGA device is connected to the host CPU via a PCIe interconnection, appropriate interfaces for PCIe must be provided on both sides. For such purpose, a Host Interface Bridge (HIB), that interfaces the PCIe and the accelerator, was devised and implemented on the FPGA device, and an appropriate device driver, to recognize the accelerator as a PCIe device and map it to the host's address space, was deployed on the CPU. Finally, an API was created, allowing the user to easily program and communicate with the accelerator.

4.1. Host-Accelerator Communication Bridge

The HIB comprises all the required hardware modules to handle the communication between the PCIe and the accelerator, being its main module a bridge between the PCIe port of the FPGA device and an internal bus instantiated in the reconfigurable fabric. The AXI Bridge for PCI Express IP core [8] module, available in the latest Xilinx FPGA devices, performs that exact function, while simultaneously providing an internal AXI interface. Moreover, it is compatible with the adopted device driver detailed below.

The communication between the AXI bridge and the remaining modules of the HIB is achieved by an AXI interconnect. This way, the accelerator controller can be directly accessible from the host, by attaching it to the AXI interconnect as a slave and by mapping its assigned address range to the PCIe Base Address Registers (BARs) with the device driver (see Section 4.2).

As briefly mentioned before, the host must be capable of sending commands to the accelerator's controller and performing transfers to/from the global memory. Being the first directly performed through the PCIe BARs, the latter function can be implemented with the aid of a DMA controller, which directly connects the AXI Bridge core to the global memory. This way, coarse-grained data transfers to/from the accelerator's global memory can be efficiently performed from the host CPU.

When making use of a DMA controller placed on the reconfigurable fabric, the host CPU only needs to configure a handful of registers and set up a small number of descriptors, leaving the controller with the responsibility of handling the rest of the heavy-lifting. However, it must be taken into account that when a buffer is allocated in the CPU's kernel space (and thus within the kernel logical address space) the resulting pointer is guaranteed to reference a continuous block of memory. Therefore, setting up a DMA operation would be as simple as providing the DMA engine with a start address and the number of bytes to transfer. However, it becomes more complex when dealing with user space buffers, as their mapping to physical addresses is hardly ever contiguous, instead being scattered across many physical pages. To transfer a user buffer of arbitrary length, it is then necessary to access these pages sequentially and set up a DMA operation for each. Naturally, this would greatly compromise the benefit of using a DMA mechanism, by requiring the CPU intervention each time a new page is to be transmitted. This problem is made even worse by the fact that these pages are usually small (4 kB in x86 systems). To overcome this limitation, a scatter-gather DMA controller must be used. By setting up a chain of descriptors describing the starting position and size of every memory block to be transferred, the CPU is only required to intervene once by transaction, thus maximizing the throughput and minimizing the CPU usage. The AXI DMA engine [9] available on the Xilinx IP Catalog already includes a scatter-gather controller, which results in a useful self-contained solution.

Apart from the DMA engine, an additional AXI interconnect, a Block Memory RAM (BRAM) and an AXI-to-AXI bridge connector are included. The BRAM's sole purpose is to provide storage space for the descriptors used by the scatter-gather controller of the DMA engine. The AXI-to-AXI bridge, connects both instances of the AXI interconnection, to allow the master port of the AXI Bridge for PCI Express to access the BRAM, to write descriptors and to read back their status, while simultaneously allowing the scatter-gather master port of the DMA engine to access the BRAM's descriptors. The DMA engine is also connected to the first AXI interconnect and its master interface is attached to the slave port of the AXI Bridge, allowing the DMA to issue write and read requests to the host system.

Fig. 6 details the HIB architecture and the main connections between its components.

4.2. Device Driver

In modern operating systems, multiple address spaces co-exist, namely user, kernel and physical address spaces. The first two make use of virtual addresses in order to present an apparently continuous block of memory to the applications running on these layers. However, they must be mapped into physical addresses, which correspond to actual physical positions on a memory device. Moreover, a distinction exists in some architectures between the physical addresses and the bus addresses. On the x86 architecture, however, they are the same. Hence, when a PCIe device is detected by the host operating system and upon its configuration registers and BARs are mapped into the physical address space, only applications running on the

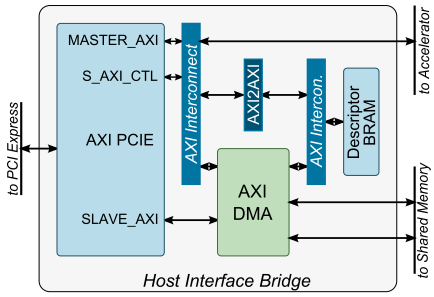


Figure 6. Host Interface Bridge, representing all the connections and modules necessary to interface the PCIe and the accelerator. Notice that the DMA subsystem directly connects to the shared memory, enabling high-throughput data transfers.

kernel space may access these resources. Thus, in order to present the device to user space applications, a device driver must be provided that abstracts it as a simple device file, on which *open()*, *read()*, *write()* and *ioctl()* calls may be performed. Then, the device driver can handle all the low-level interactions with the accelerator, starting on the PCIe bus protocol itself, up to reading and writing its configuration registers or setting up interrupt handlers.

The MPRACE framework [1] provides a generic PCIe driver for interfacing with FPGA-based accelerators. In addition to the device driver capabilities for configuring the PCIe device, creating the corresponding device node and allowing for register accesses through the *ioctl()* interface, it also includes a C and a C++ implementation of a user-space interface, providing an easy access to more complex operations, such as allocating kernel memory buffers, mapping user-space buffers into device space, as well as mapping and unmapping the BAR area into user-space.

To ensure that the device is properly recognized and configured during the device enumeration phase performed by the BIOS during the host boot phase, it is necessary to properly set the ID values and Class codes. These are made available on the PCIe configuration space of the device and indicate the vendor and type of device present on the PCIe slot. This way, the host can recognize the device as a Xilinx Corporation Device acting as a multimedia controller.

The last step that is also required to ensure that the BIOS and the host operating system successfully recognize and configure the device is the configuration of the PCIe BARs. The purpose of these registers is twofold. Initially, they serve as a mechanism for the device to request blocks of address space in the system memory map. Once this step is completed, the BARs are used to perform address translation between the host and the device.

At this point, a PCIe base system is available in the FPGA device and the operating system is now capable of mapping the BARs of the PCIe device into the bus address space (the same as the physical address space in x86 architectures) performing address translation between the physical address space and the bus address space. In other words, mapping this memory range to user space makes it possible to access the device's AXI address space from within the host machine.

4.3. Low-Level API

The final part of the framework, corresponding to the top level of the integrated stack, comprises an API that interfaces the device driver with a user application. By making use of the tools provided in the MPRACE driver, this API provides routines to initialize and close the PCIe device file and the DMA controller interfaces, to issue commands to the accelerator controller, and to perform data transfers to/from the global memory. Each of these routines that involve issuing a command, start by configuring the appropriate control registers, triggering the command and awaiting its completion, by following the already described procedure.

To allow the host to monitor the accelerator processing cores' execution, a data structure is maintained with the last fetched execution state of the PEs. A routine was also devised that reads the status registers, updates the data structure with the current accelerator status and returns a list with the observed status changes.

Envisaging the deployment of multi-threaded applications, the operations on each of the provided routines was optionally made atomic with the aid of the PTHREADS library mutexes. This way, the user can create a monitoring thread that automatically samples the status registers or develop concurrent applications that issue multiple kernels to the accelerator, without incurring in errors resulting from multiple commands being simultaneously issued.

5. Framework Evaluation

The evaluation of the proposed framework is presented through a hardware resource utilization study, together with a characterization of the achievable communication throughput between the host and the accelerator's global memory. Moreover, two case studies are presented, corresponding to two distinct accelerator systems that already made use of this framework in their base system. The first case study presents the HotStream framework [10], which targets the streaming of complex data patterns to computing kernels; while the second case study presents a morphable hundred-core architecture that allows the automatic reconfiguration of the accelerator's PEs, according to runtime system and application performance and energy requirements, targeting energy-aware computation [5]. Even though the chosen systems are based on early versions of the framework, both resulted in major contributions in their respective research areas. This validates the usefulness of having prototyping frameworks that abstract and mitigate the complexity of the interface and data transfer tasks in accelerators implemented in CPU-coupled FPGA devices.

5.1. Hardware Resources and Communication Throughput Evaluation

To evaluate the proposed framework, it was prototyped in a Xilinx Virtex-7 FPGA (XC7VX485T), connected through an 8x PCIe Gen 2 link to a host computer with an Intel Core i7 3770K, running at 3.5 GHz. The Syn-

 ISBN: 978-989-98875-2-7

Table 1. Evaluation of the framework's base system in terms of hardware resources.

	Registers	LUTs	RAMB36E1
AXI4 BUS	21041 (3%)	17325 (6%)	4 (<1%)
AXI4-Lite Bus	248 (<1%)	483 (<1%)	1 (<1%)
AXI Bridge	12595 (2%)	17954 (6%)	0 (0%)
DMA	2891 (<1%)	2930 (<1%)	1 (<1%)
Descriptor BRAM	864 (<1%)	1242 (<1%)	3 (<1%)
Total	36789 (6%)	39934 (13%)	9 (1%)

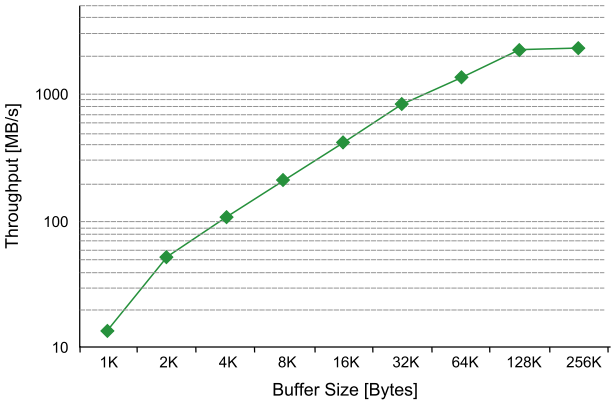


Figure 7. Communication throughput between the host and the accelerator's global memory, measured with several buffer sizes.

thesis and Place&Route procedures were performed using Xilinx ISE 14.5. On the Intel Core i7, cycle accurate measurements were obtained by using the PAPI library.

The resource utilization of the HIB and the accelerator's controller is presented in Table 1. Despite the complexity of some of the instantiated modules, it can be observed that the hardware overhead imposed by the base system is still less than 13%. This way, most of the FPGA resources are left available for the user to implement large scale acceleration architectures.

The communication throughput was evaluated by setting up transfers between the host CPU and the accelerator's global memory. The results, presented in Fig. 7 where obtained by performing several transfers with different buffer sizes and by measuring the time between the writing of the descriptor to the AXI DMA registers and the moment at the end of the receiving operation. In order to minimize as much (as possible the impact) of non-deterministic behaviour on the host machine, the transfers where performed several times for each buffer size and the mean value was extracted.

As it can be observed from Fig. 7, the achievable throughput rises with the increase of the buffer size, allowing an easier mitigation of the startup and other overheads present on the system. The maximum attained throughput is achieved with 256 KB buffers, with the transfer rate saturated at about 2.3 GB/s.

The obtained values can be explained by the fact that the PCIe bridge and DMA engine must be constrained at a maximum operating frequency of 100 MHz, in order to ensure an accurate frequency-lock. Hence, the highest bandwidth the AXI interface can provide is 1.6 GB/s (16B x 100 MHz) in each direction. This creates a throughput ceiling that limits the achievable performance to 3.2 GB/s. The fact that the attained throughput was 72% of the theoretical limit, may be justified by arbitration latencies, idle periods between successive bursts and setup overheads.

5.2. Case Study A: HotStream Framework

The HotStream framework [10], which drove the first stages of the development of the herein presented framework, proposed a novel architecture for the development of efficient and high-performance stream-based accelerators. The proposed accelerator framework is capable of simultaneously processing an arbitrary number of stream-based kernels, providing pattern-based data accesses with two granularity levels: a coarse-grained data access from the Host to the accelerator processing engine, to maximize the transmission efficiency; and a fine-grained data access between PEs within the accelerator, to maximize data reuse. The latter makes use of an innovative Data Fetch Controller, which extracts the data streams from an address-based shared memory, by using access patterns of arbitrary complexity. Moreover, instead of accomplishing the pattern description by traditional descriptor-based methods, the HotStream framework relies on a micro-coded approach, supported on a compact Instruction Set Architecture. This allows an efficient description of data streams with complex memory access patterns, without compromising the address issuing rate.

When compared to state-of-the-art pattern generation mechanisms, the HotStream framework achieved considerable gains in terms of the hardware resources, as well as in what concerns the storage requirements of the pattern description code. A block-based matrix multiplication accelerator, operating over large matrix sizes, was adopted as the evaluation benchmark. Experimental results showed that conventional solutions that do not exploit data-reuse can be easily constrained by the PCIe communication link. On the other hand, by using the proposed framework, it was possible to increase the core available bandwidth by 4.2x, in turn leading to a 2.1 performance speedup in a 4x data parallelism approach. Furthermore, by easing the implementation of more complex solutions, further speed-ups can still be achieved. In particular, it allows the implementation of the matrix reduction step directly on the accelerator and to alleviate the computational requirements of the host CPU. The final solution allowed for a reduction in host memory requirements by 42x, consequently allowing the processing of much larger matrices.

5.3. Case Study B: Morphable Architecture for Energy-Aware Computation

The system proposed in [5] comprised a new morphable heterogeneous computing platform, integrating hundreds of processing cores and offering runtime adaptation capabilities to meet the performance and energy constraints imposed by the application under execution. The adaptive na-

ture of this platform was achieved by monitoring, in real-time, the performance of the computational cores while they execute the application kernels, and by determining the processing architectures and topologies that maximise the performance and/or energy efficiency. To perform this decision, a Hypervisor software module was devised, responsible for scheduling the computing kernels to the available processing cores, and for triggering the reconfiguration of the currently instantiated cores, by issuing appropriate commands to an on-chip reconfiguration engine.

The reconfiguration engine module performs the actual adaptation, by exploiting the existing partial dynamical reconfiguration mechanisms of modern FPGA devices. To perform the adaptation of this hundred-core heterogeneous platform, different algorithms (policies) are considered, corresponding to typical optimisation goals, namely: minimization of the execution time; maximization of the processing performance for a given power-ceiling; and minimisation of the power consumption, while guaranteeing a minimum performance level.

The system and the corresponding policies where evaluated with a set of computation kernels from the algebraic domain. The obtained experimental results demonstrated that the optimization policies provide a significant reduction of both the execution-time and energy consumption when compared with static homogeneous or heterogeneous implementations with a fixed number of cores. The proposed reconfigurable system achieves performance gains between 2x and 9.5x, whereas the energy consumption was reduced between 2x and 10x. Accordingly, the morphable heterogeneous structure has shown to be a highly viable and efficient approach to provide an adaptable architecture, being able to morph the computing cores according to the instantaneous system restrictions, resulting not only in improved performances but, more importantly, in an energy-aware computing platform.

6. Conclusion

A generic framework for interfacing a host CPU with custom accelerators implemented in FPGA devices, through a high-speed PCI Express (PCIe) interconnection was proposed in this manuscript. The deployment of this framework aims at mitigating the most time-consuming design tasks that emerge when developing these kinds of systems, which involve the communication between the host CPU and the accelerating device and the lack of proper tools for application development in such devices. The proposed framework deploys a fully integrated application stack, comprehending: *i)* an FPGA base system, comprising an Host Interface Bridge (HIB) that implements all the required hardware modules to connect a custom accelerator to the PCIe interconnection; *ii)* a generic and scalable many-core accelerator, composed of a communication and management controller and the basic building blocks for creating custom accelerators (supporting the inclusion of custom intercommunication networks and shared memory topologies); *iii)* a device driver, capable of interfacing the CPU and the FPGA through a PCIe interconnection; and *iv)* a low-level API that provides convenient routines for controlling the accelerator's PEs and the data transfers between the host and the accelerator's memory.

An evaluation of the base system showed that the required hardware to interface the PCI Express interconnection with a custom accelerator impose a minor overhead, leaving most of the reconfigurable fabric of the FPGA device free for the user to deploy custom hardware modules. Moreover, the communication between the host CPU and the accelerator's global memory achieved throughputs as high as 2.3 GB/s. In order to demonstrate the potential of the framework, two systems that already made use of the framework as base system were also discussed.

7. Acknowledgements

This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project Threads (ref. PTDC/EEA-ELC/117329/2010) and project PEst-OE/EEI/LA0021/2013.

References

[1] Guillermo Marcus, Wenxue Gao, Andreas Kugel, and Reinhard Manner. The MPRACE framework: an open source stack for communication with custom FPGA-based accelerators. In *Programmable Logic (SPL), 2011 VII Southern Conference on*, pages 155–160. IEEE, 2011.

[2] Matthew Jacobsen and Ryan Kastner. Riffa 2.0: A reusable integration framework for fpga accelerators. In *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pages 1–8. IEEE, 2013.

[3] Jian Gong, Tao Wang, Jiahua Chen, Haoyang Wu, Fan Ye, Songwu Lu, and Jason Cong. An efficient and flexible host-fpga pcie communication library. In *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, pages 1–6. IEEE, 2014.

[4] Tiago Dias, Nuno Roma, and Leonel Sousa. Unified transform architecture for avc, avs, vc-1 and hevc high-performance codecs. *EURASIP Journal on Advances in Signal Processing*, 2014(1):1–15, 2014.

[5] N. Neves, H. Mendes, R. Chaves, P. Tomás, and N. Roma. Morphable hundred-core heterogeneous architecture for energy-aware computation. *To appear in IET Computers & Digital Techniques*, 2014.

[6] Nuno Sebastião, Nuno Roma, and Paulo Flores. Integrated hardware architecture for efficient computation of the-best bio-sequence local alignments in embedded platforms. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(7):1262–1275, 2012.

[7] ARM, Ltd. *AMBA® 4 AXI4-Stream Protocol, v1.0*, March 2010. http://infocenter.arm.com.

[8] Xilinx. LogiCORE IP AXI Bridge for PCI Express. Technical Report PG055, Xilinx, 2012.

[9] LogiCORE IP AXI DMA v6.03a. Technical Report PG021, Xilinx, 2012.

[10] Sérgio Paiágua, Frederico Pratas, Pedro Tomás, Nuno Roma, and Ricardo Chaves. Hotstream: Efficient data streaming of complex patterns to multiple accelerating kernels. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*, pages 17–24. IEEE, 2013.

Transparent Binary Acceleration via Automatically Generated Reconfigurable Processing Units

Nuno Paulino, João Canas Ferreira, João M. P. Cardoso
INESC TEC and Faculdade de Engenharia,
Universidade do Porto, Portugal
{*nuno.paulino, jcf, jmpc*}*@fe.up.pt*

Abstract

In order to accelerate the execution of embedded applications, the most demanding kernels may be re-targeted to a hardware accelerator.

In this paper, an approach is presented which automatically detects repeating instruction traces of an application. The traces are used to define a Reconfigurable Processing Unit (RPU) that is coupled to a host processor executing the application. A transparent migration mechanism shifts execution of the detected kernels from the main processor to the RPU. Past system and RPU implementations are presented and compared to a recent RPU design, which improves speedups by exploiting inter-iteration instruction parallelism and loop-pipelining via modulo scheduling.

For a set of 12 benchmarks, the latest RPU design achieves a geometric mean speedup of 5.44×*, improving performance and lowering resource requirements over previous implementations.*

1. Introduction

Many embedded applications require efficient implementations to meet their performance or power consumption requirements. A possible approach to attain these performance targets is to shift execution of computationally demanding algorithms to dedicated hardware.

Analysing an application, determining which portions are good candidates for hardware acceleration and migrating them is known as hardware/software partitioning, and is a well known strategy in the embedded domain [1, 2].

These approaches rely on a target heterogeneous architecture. The application will have to be re-targeted so that it can execute on two or more inter-operating hardware components. Targeting a fully custom hardware accelerator requires an even greater design effort than the use of pre-defined hardware structures.

Binary acceleration approaches attempt to automate this partitioning process via automated runtime or compile-time analysis of the application. The result of these processes is typically a configuration/specification of an accelerator. Accelerator designs vary per approach, targeting different types of traces, use different types of resources and support different capabilities [3, 4, 5].

We presented several implementations of an approach based on augmenting a General Purpose Processor (GPP) with an automatically generating a Reconfigurable Processing Unit (RPU), which accelerates repetitive binary traces [6, 7, 8, 9]. The RPU acts as an accelerator to which calculations can be offloaded in a manner transparent to both the GPP and programmer. The RPU is generated automatically by translation/scheduling tools which define the RPU's resources, interconnections and configurations.

Given that, this paper presents: (1) a short summary of the general approach, together with previous system and RPU architectures; (2) validation of the architectures with up to 37 benchmarks; (3) a comparative performance analysis of the several implementations; (4) issues to be addressed in future work in order to increase performance.

This paper is organized as follows. Section 2 reviews related work , while Section 3 explains the acceleration approach. System-level and RPU architectures are explained in Sections 4 and 5, respectively. Experimental results are discussed in Section 6 and Section 7 concludes the paper.

2. Related Work

There are a number of works regarding transparent acceleration based on runtime profiling and accelerator synthesis/configuration. One example is the WARP approach, with its latest design presented in [3]. The system has a fine-grain reconfigurable architecture loosely coupled to a MicroBlaze processor. An additional processor is responsible for detecting frequent inner loops at runtime. Candidate loops are passed to on-chip CAD tools, which target a fine-grained reconfigurable accelerator. Execution of the loops is migrated to the accelerator, which can perform up to one memory access and has dedicated MAC unit and address generators for regular patterns. The system was implemented in Virtex-II Pro and Spartan-3 FPGAs, and speedups up to 5.8× were achieved with energy consumption reductions of 49 % compared to a single MicroBlaze. The capability to accelerate threads is addressed in [10]. Speedups of up to 109× are reported when compared to 4 ARM11 cores, while requiring 36× as many resources.

The Dynamic Instruction Merging (DIM) approach [5] also translates binary traces at runtime, targeting a Coarse Grain Reconfigurable Array (CGRA). The array is tightly-coupled and organized in rows. Each containing several

ALUs, a memory unit and a multiplier. A row receives data from the preceding row, directs its outputs to the next row and is also capable of forwarding data. Inputs are fetched from the processor register file. Basic blocks are detected at runtime by monitoring the instruction stream of a MIPS processor. An average speedup of 2.7× with 2.35× less energy over a single MIPS processor is achieved for the MiBench benchmark set.

An implementation capable of accelerating both inner and outer loops is presented in [11], which proposes a large CGRA design supported by a LLVM based compilation flow. The CGRA is composed of a set of clusters containing computing elements, memories and an interconnection network. Clusters behave as pipelined datapaths, are connected to their nearest neighbours and share access to external memory. Data is mapped to cluster memory banks at compile-time and pre-fetched according to runtime control. Several clusters can be configured to execute loop-pipelined iterations of the same loop. An implementation on a Virtex-7 FPGA is compared to a dual-core ARM. For three applications, speedups by accelerating inner loops alone range from 1.4× to 3.4×. Speedups can be doubled by parallelizing two inner loop instances onto available resources.

Modulo-scheduling is very effective in pipelining loops and thus some CGRAs exploit this execution model. To achieve efficient and fast scheduling, a challenge for heterogeneous architectures, several works propose scheduling optimizations or schedule-friendly architectures.

In [12], a configurable loop accelerator template for modulo-scheduled loops is presented. The accelerator contains a single row of Functional Units (FUs) attached to output FIFOs and feedback interconnection logic. The inter-FU connectivity is specialized at design time to the minimum required to implement the equivalent Control and Dataflow Graph (CDFG) edges. The authors present a study regarding the amount of flexibility required of a loop accelerator to support a given set of CDFGs. Starting from a baseline CDFG, a compile time toolflow creates a non-programmable loop accelerator instance which implements that single CDFG. Flexibility is then introduced in the form of additional FU capability and multiplexers at the input stage. Additional CDFGs are then scheduled on the accelerator by a constraint-driven modulo scheduler.

The approach presented in [13] addresses the speculation introduced when modulo scheduling cyclic CDFGs. Nodes which form a closed circuit are modelled as a clustered node, until all backwards edges are eliminated, thereby avoiding speculative scheduling. Each clustered node is then scheduled in order. A cost function penalizing latency is used to ensure that the cluster of nodes does not violate the Iteration Interval (II). A total of 390 CDFGs are used to evaluate the scheduler targeting a CGRA with 16 FUs, reducing scheduling times up to 170× relative to a state-of-the-art, edge-centric modulo scheduler [14].

An approach based on the Samsung Reconfigurable Processor (SRP) is presented in [15]. The SRP is composed of a VLIW with an attached CGRA. Inner loops are modulo-scheduled to the CGRA portion. The CGRA has 16 FUs, configured manually at design time. The two halves of the SRP share a central register file, and up to four concurrent load/store operations can be performed by the CGRA. The approach presents an automated compile-time flow to detect instructions patterns in the application and replace them with the respective intrinsics supported by a given SRP instance. An LLVM-based flow generates intrinsics, based on high-level architecture and intrinsics definitions, and automatically replaces matching instruction patterns in the application. Generated intrinsics cover 75 % of the possible executable intrinsics of 11 SRPs. The identification step replaces 90 % of the instruction patterns found by manual identification with the intrinsics.

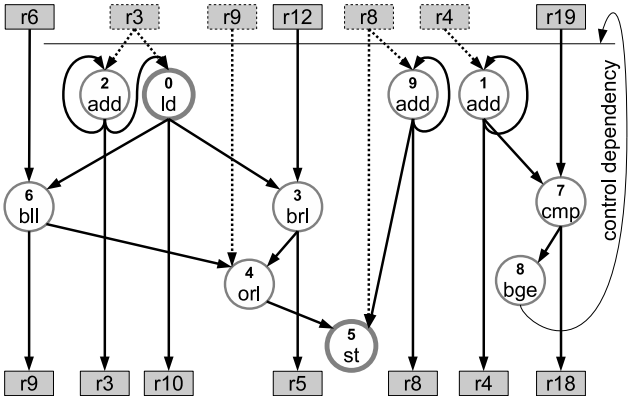


Figure 1: CDFG for a loop of the *blit* benchmark

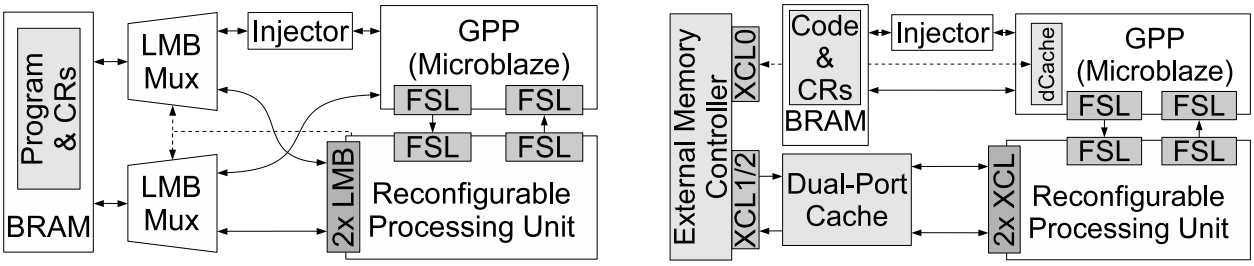
Like these approaches, the implementations presented in this paper are based on transparent binary acceleration. Dedicated accelerators are generated offline from frequent traces detected during a post-compile simulation step. At runtime, execution of detected traces is migrated transparently to the loosely coupled coarse-grained accelerator, which exploits Instruction Level Parallelism (ILP) and loop pipelining.

3. General Approach

The approach presented in this work and previous similar implementations are based on: automatically detecting a type of repeating binary trace called Megablock [16]; selecting a set of detected traces; translating the traces into CDFGs which extract the latent instruction parallelism of a single trace iteration; generating an RPU instance by exploiting the intra-iteration ILP of the CDFGs and their IIs to perform loop-pipelining.

Megablocks A Megablock trace represents a repetitive sequence of instructions. The trace includes all instructions executed by the GPP. That is, both the data and control-flow are captured. The control-flow is determined by the conditional *branch* operations that cause the execution to break from the repetitive pattern. Unlike basic blocks, the trace can contain any number of *branch* instructions. That is, the Megablock is a single-entry multiple-exit pattern which typically represents a single execution path through a loop.

Fig. 1 shows the CDFG of a Megablock detected in the *blit* benchmark. Nodes are equivalent to the trace instructions of a single iteration of the trace and edges represent



(a) System with support for RPU local memory access

(b) System with support for RPU external memory access

Figure 2: Overview of system architectures with a GPP augmented by a RPU and transparent migration mechanism

dependencies between nodes. Most nodes shown are data operations, along with a *load*, a *store* and the *bge* node which represents the *branch* instruction.

Boxes on the top correspond to MicroBlaze registers which contain the inputs to a single iteration, i.e., *live-ins*, and boxes on the bottom are registers which hold results from one iteration, i.e., *live-outs*. Registers with a solid line are never modified throughout all iterations. Registers that are updated on every iteration appear both on the top and bottom and, on the top, are represented with dotted lines.

The connections established by dotted edges are valid for the first iteration. For subsequent iterations, connections are established according to the nodes which produce new register values. If the number of iterations to execute is known, then only these direct dependencies would be needed to compute the recursive II. However, in the implementations presented here, the number of loop iterations may be determined at runtime by live values. Therefore an iteration can only be initiated after it is known that the previous one will complete, i.e., only after all *branch* nodes are evaluated. Hence, for this example, the *bge* node sets an II of 3.

Migrating to Hardware A fully runtime system capable of autonomously detecting and migrating such traces to self-adaptive hardware would provide a competitive alternative to manual hardware/software partitioning.

Previous work, and the approach presented in this paper, are implementations towards that end, which are based on a mixed offline/online methodology: (1) the target application is executed in a cycle accurate Instruction Set Simulator (ISS) to detect Megablocks; (2) selected Megablocks are given to translation tools to generate a tailored RPU instance; (3) during runtime, a transparent migration mechanism offloads execution to the RPU. The systems presented in the next section follow this methodology. A MicroBlaze soft-core is used as the GPP and the systems were developed and implemented for FPGA targets.

4. System Architectures

Two of the implemented system architectures are shown in Fig. 2: an organization where the RPU and GPP share access to a local data Block RAM (BRAM) (Fig. 2a) [7] and a second where the RPU accesses the GPP's external data memory (Fig. 2b) [8].

An implementation of the system shown in Fig. 2a is discussed in [7]. The system contains: a MicroBlaze GPP; local BRAMs containing code and data, the tailored RPU; two multiplexer modules, allowing for the GPP and RPU to share the data memory; and the *injector* module, a transparent migration mechanism which shifted execution to the RPU. The RPU architecture is based on a 2D array of FUs organized into rows which exploited intra-iteration ILP and did not perform loop pipelining. A previous implementation using a similar RPU architecture did not provide support for memory access [6]. As a result, the traces that were supported for acceleration were shorter an less representative of real data-oriented applications. Although the storage provided by BRAMs may be limited when compared to off-chip memory, higher end devices contain up to 68 kB of RAM [17], which may be adequate for some applications.

The system layout of Fig. 2b is discussed in [8]. In this case, the local BRAMs only hold code. The bulk of the program data is held in external memory which the RPU accesses using a custom dual-port data cache. The RPU architecture presented in [8] is based on directly translating a set of CDFGs into a configurable pipelined datapath, to exploit both intra-iteration ILP and loop pipelining.

Apart from the locations of data and code memory, the RPU is generated and made use of in the same manner in all system implementations. At runtime, the *injector* monitors execution. When the address corresponding to the start of a translated trace is detected, the *injector* places sends an absolute jump instruction to the GPP. At the target address of the jump is a tool-generated Communication Routine (CR). By executing it, the GPP sends operands to the RPU, waits for completion of RPU execution, retrieves results to its register file and returns to the address where software execution was interrupted.

5. RPU Architectures

All RPU designs discussed here are based on parameter-oriented templates. A specialized RPU instance is created at synthesis time according to tool-generated parameters, which result from automated translation of a set of CDFGs. Depending on the selected configuration, the RPU accelerates one out of the set of translated loops. These parameters dictate which FUs the RPU contains, the interconnections between them and control logic/instruction words to command execution (e.g., configuring interconnections, enabling FUs or writing results to registers).

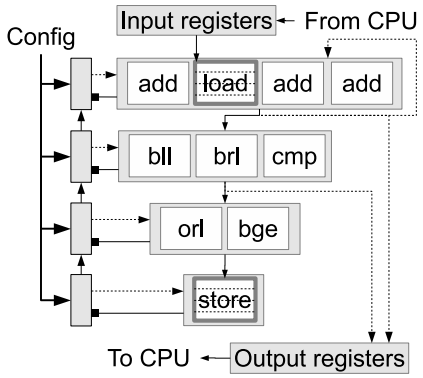


Figure 3: 2D Pipelined RPU architecture

All RPU designs support all integer arithmetic operations (including division by a constant) and logical operations. All FUs are have a latency of one clock cycle, except the constant division and multiplication FUs (3 clock cycles). The latency of the *load/store* FUs varies per implementation.

The translated CDFGs include the control-flow of the loops, i.e. the *branch* instructions. An additional class of *exit* operations implements these instructions. Implementing the control flow allows for more flexible loop acceleration by supporting dynamically determined iteration counts.

Early systems did not support memory access by the RPU. This prevented acceleration of loops with load/store operations, which are the most common in data processing. The architectures presented here are capable of up to two concurrent memory accesses. An access is a single, byte enabled, 32-bit read/write transaction.

5.1. Pipelined RPU

Structure The execution model for the RPU of [8] relies on a pipelined datapath layout, enhancing the design presented in [7]. The example in Fig. 3 shows a set of FUs organized into stages, per-stage control modules and connections between stages. The two memory ports are omitted. The inter-stage connectivity was customized based on the translated CDFGs. Support for backwards connections allowed for exploitation of the IIs of the loops. The array in Fig. 1 is an implementation of the example shown in Fig. 1.

The array was customized by translating CDFG nodes into FUs. The resulting structure is a near direct implementation of the set of CDFGs, with the required control to configure the array to execute one out of the set of CDFGs. As each CDFG was translated, existing FUs would be reused between configurations. List scheduling was employed to exploit node mobility and increase FU reuse. Specialized multiplexers feed the FU inputs, implementing the minimum required connectivity. A multiplexer can fetch output values from any FUs or can provide hard-coded constant values, which originate from nodes in the CDFG that have constant operators. If the multiplexer needs only one input, it is optimized away. The array is configured prior to execution by setting the multiplexer outputs through a single 32-bit configuration word sent by the *injector*. That selection is held constant throughout the execution of the loop, i.e., the array behaves like a static pipelined datapath.

There can by any number of *load/store* FUs on the array. Actual memory accesses are performed by the two memory ports. Each port handles half of the *load/store* FUs on the array, according to a *round robin* type arbitration scheme.

Execution To control execution, there are no instruction words or static control schedules. Stages are enabled dynamically based on inter-stage dependencies. This simplifies support for variable memory access latency and requires less complex translation tools. Each stage's control module issues an *enable* signal to the stage, and receives back a *valid* signal when that stage has finished executing. The valid signals of all stages are fed to all control modules. An *enable* is issued to a stage only after all the required data has been produced by one or more other stages. To start the execution, a signal jump-starts the first stage, and execution flows from that point until an exit condition is triggered.

5.2. Modulo Scheduled 1D RPU

The 2D pipelined architecture suffers from idle time, meaning that its resources are underused. Although the architecture conceptually allows for loop pipelining, the contention for memory access is detrimental to performance. Also, instances which implement multiple large CDFGs might be too costly in terms of resources and too inefficient in terms of operating clock frequency. The latest RPU architecture presented in detail here, and briefly described in [9], shown in the top of Fig. 4, approaches the problem by considering a 1D RPU and modulo scheduling. A single row of FUs includes all the resources required to execute the single cycle instructions corresponding to the steps of a modulo schedule. The modulo schedule for the CDFG if Fig. 1 is shown in the bottom half of Fig. 4.

Modulo Scheduling To construct an RPU instance, the modulo scheduler takes a set of CDFGs, computes their minimum IIs, and schedules them one at a time. Instead of iteratively increasing the II so that the loop is schedulable given the resource limitations of the architecture, FUs are added during scheduling to meet the chosen II.

The rows in the scheduling table correspond to time steps, and the columns to FUs. The node numbers correspond to the numbers shown in Fig. 1. Nodes with a solid line correspond to one iteration. Nodes with a dotted line are operations of the next iteration, shifted to a later time according to the II. The CDFG has an II of 3 clock cycles, determined by the control dependency with the *bge* node.

The table shows the prologue, steady state (i.e. modulo reservation table) and epilogue of the schedule. Considering the II and that only six time steps are required to schedule the first iteration, only two iterations need to be overlapped to achieve a steady state. An iteration is completed every three clock cycles by executing nodes as scheduled in the three time steps of the steady state.

Nodes are scheduled top-to-bottom by computing each node's earliest and latest possible scheduling time, t_e and t_l. At the start of the scheduling process, only the two *ld/st*

 ISBN: 978-989-98875-2-7

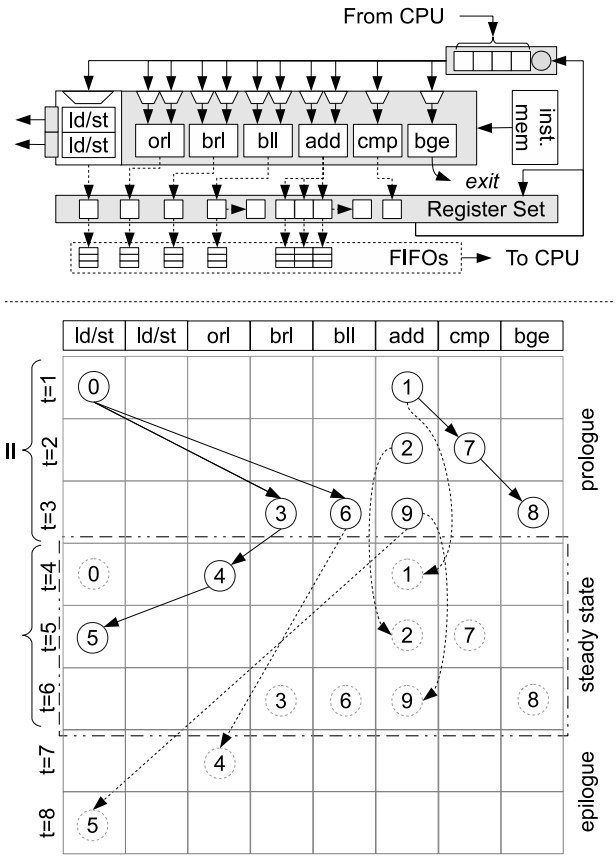


Figure 4: 1D RPU architecture capable of executing modulo-scheduled loops; schedule for a loop of the *blit* benchmark

units are available. Required FUs are added as a CDFG is scheduled. They are then reused while scheduling additional loops. The mentioned FU latencies are considered while scheduling the nodes. Since the data memory used while testing this RPU implementation was local BRAM, the *load/store* FUs have a fixed latency of 2 clock cycles.

Computing the t_e for a node involves checking every source node for its scheduled time. For most nodes, all predecessors have already been scheduled so the calculation is trivial. Source nodes which are either (1) yet unscheduled or (2) downstream relative to the node being scheduled, are ignored in the computation of t_e values. For each node, the t_l value is first set for $+inf$. For every sink node the calculation of t_l: (1) continues recursively through that node or (2) if that node is upstream to the node being scheduled (i.e. there is a closed circuit) and has already been scheduled then the tentative t_l can be no greater than $t_u + II$, where t_u is the scheduled time of the sink node. This ensures that the closed circuit stays in the bounds of the II.

After all schedules for a set of CDFGs are built, the structure of the RPU is configured.

RPU Structure The structure shown in Fig. 4 is a synthetic example of an instantiation. The basic components are: two *load/store units*, a row of FUs, a register set to hold FUs outputs, an instruction memory which controls execution, and input/output register sets to communicate data to/from the GPP (in the top and bottom of Fig. 4). The two memory access units are present for every RPU instantiation. Like earlier designs, FU inputs are fed by specialized multiplexers. An FU input may receive: values from the FU output register set, values from input register set, which remains constant through execution, or constant values.

The register pool is composed of 32-bit registers fed by the FUs. Each FU feeds only a pre-determinate set of registers. The size of this set varies per implementation and depends on the time-to-live of the produced values, which is determined by when the consumer-producer node pairs are scheduled relative to each other. In the example of Fig. 4, the *add* FU required three output registers. Node *2* is executed after *1*, but before the output of the latter is consumed, therefore at least two registers are required. The same happens with node *9*. If the output of nodes *1* or *2* had been consumed prior to executing node *9*, one of the two registers could have been used.

Since iterations are overlapped, some nodes will be executed multiple times before the respective consumer nodes execute. This happens when a consumer node is scheduled at a time t_c greater than $t_p + II$, where t_p is the scheduled time of the producer node. To solve this, values can also be moved between registers in the pool. The first output of node *9* is consumed by node *5* at time $t = 8$, after a second output by node *9* is produced. The third output register of the *add* FU saves its value to an additional register. The number of registers required in these cases is equal to $(t_c - t_p/II)$.

The output FIFOs hold values which the GPP will read back into its register file. A complete set of output values is ready when an iteration is complete. The FIFOs are required since several values for a single output may be produced before that, due to iteration overlap.

Execution Execution is controlled cycle-by-cycle by the instructions generated by the scheduler. The width of an instruction varies per instance. One instruction corresponds to one time step and controls: which FUs are enabled, the multiplexer selection, the Program Counter (PC), when to commit values form the FIFOs to the output registers, and when execution ends. Per loop, a sequence of instructions implements the prologue, steady state and epilogue of the respective schedule.

When the RPU is signalled to start, the PC is set to the address containing the first instruction of that loop. During the prologue the iterations begin to overlap, incrementing the PC by one for every instruction. Execution then reaches the steady state. The second to last instruction of the steady state updates the PC to $PC = PC - II$–1. The steady state is executed until an exit is triggered. This causes the instruction-coded update to the PC to be ignored and the execution continues through the epilogue. During the epilogue, no new iterations are initiated. The RPU completes iterations initiated before the one which triggered the exit condition.

6. Experimental Evaluation

The systems and RPU architectures shown in the previous sections were implemented on a Spartan-6 LX45 FPGA

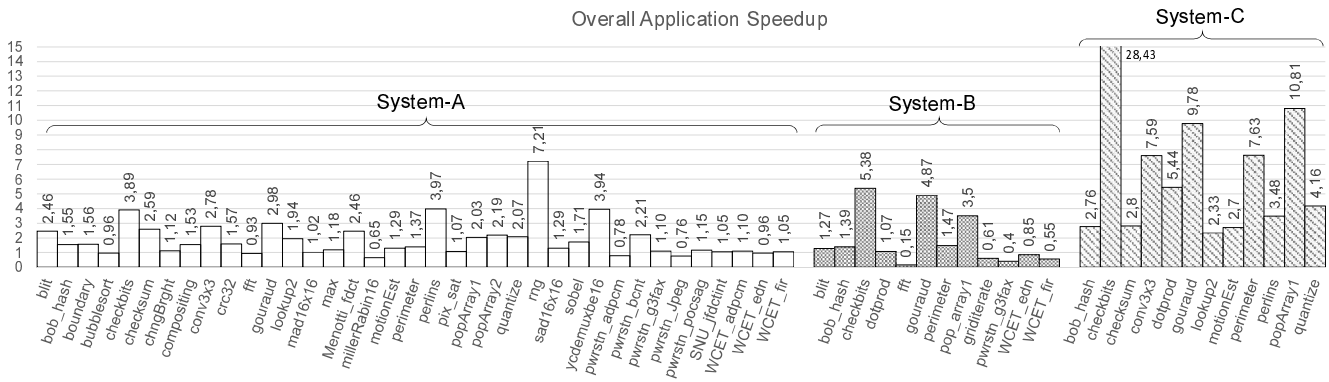


Figure 5: Overall Application speedups for the three system implementations

board. Two systems follow the layout shown in Fig. 2a (system *A* [7] and system *C* [9]) and another employs the architecture shown in Fig. 2b (system *B* [8]). The RPU architecture used for each is, respectively, a non-pipelined 2D oriented RPU, the most recent architecture shown in Fig. 4, and the pipelined 2D RPU shown in Fig. 3. The systems also contain a UART and a custom timer module to measure total application runtime, runtime of accelerated traces and communication overhead.

A total of 37 benchmarks were used to validate system *A*. The remaining systems have so far been tested with subsets of these benchmarks. Given that system *B* has access to a large external memory, an additional *gridIterate* benchmark was used (taken from a stereo navigation application [18]).

In this section the performance of the modulo-scheduled RPU architecture is evaluated and compared to previous implementations in terms of performance and resources. When comparisons are made between systems, only the subset of common benchmarks is considered.

6.1. Performance

Overall application speedups are shown in Fig. 5 for all systems. The geometric mean speedups are, respectively, 1.60×, 1.13× and 5.44×. The average number of executed instructions per clock cycle (IPC) is 2.42, 2.24 and 9.78. Table 1 also contains geometric mean speedups, average number of executed IPC, average number of FUs on the RPU and the average number of instructions in the Megablocks.

For systems *A* and *C*, the RPU accesses local data memory with a latency of 1 clock cycle. Differences in performance can be analysed by considering only the RPU design. The acceleration achieved by system *A* is derived only from inter-iteration ILP. Considering only the 11 common benchmarks between systems *A* and *C*, the geometric mean speedups are 2.25× and 5.44×, and the respective IPC_{HW} values are 2.35 and 10.49.

Since the RPU in *B* implements a direct translation of the loop CDFGs, the potential IPC_{HW} for these cases can be derived from the number of instructions in one loop iteration and the II. The mean potential IPC_{HW} for this set was 15.53. However, the RPU executes only 2.51 instructions per clock cycle, a small increase relative to the IPC_{HW} of 2.10 achieved with the non-pipelined version in system *A*. Additionally, the mean geometric speedup decreases to 1.21× for *B*, relative to the achieved 1.62× for *A*. Since the execution of the software-only portions of the application suffers from external access latency as well, the accelerated traces represent less of the total execution time. In system *A*, the average times spent in software-only execution and the total execution times are 6.82 μs and 1603 μs, respectively. For system *B* the values are 1239 μs and 2602 μs.

Although system *B* performs loop pipelining, it suffers not only from the memory access latency but also from contention from the numerous *load/store* to the two available memory ports. In both system *A* and *B*, there could be any number of *load/store* FUs which were handled by the arbitration logic. The arbitration scheme would work efficiently only when there were two or one load or store FU in the entire array allowing for uninterrupted execution. However, the array has an average of 5.26 and 2.67 *load* and *store* FUs (considering all benchmarks from both systems). Since the array would halt waiting for completion of accesses, this contention introduced idle time. In system *B*, multiple stages were executed simultaneously, increasing contention.

For example, the *gridIterate* benchmark executes for the longest time, processes the most data and contains the largest accelerated trace. Out of the 120 instructions of the trace, 41 are memory accesses. Given the recursive driven II of 5, the potential IPC was 24. However, the RPU's execution model and the external access latency lower the IPC_{HW} to 0.49.

The modulo scheduled approach manages to exploit the recursive driven II for nearly all cases. Only in the case of *conv3x3* was the II determined by resource restrictions. That is, the number of memory operations in the CDFGs, since there are only two memory ports. However, unlike the passive translation approach of the RPU design of system *B*, the operations are efficiently scheduled so that accesses are pipelined, making constant use of the memory ports. Also, the RPU execution does not halt waiting accesses to complete, since translation assumes a known memory latency and generates the schedules accordingly. Considering only the CDFG for *gridIterate*, the minimum II for an implementation in the new RPU architecture would be dictated by the 33 memory accesses per iteration. This yields an II of 17, and an IPC_{HW} of 7.05. However, this does not account for external memory access latency, which could be mitigated

by tuning the scheduler to a mean access latency.

On average, the attained speedups for system *C* correspond to 84 % of the their speedup upper bound, whose geometric mean is 6.83×. The upper bound is the speedup that would be achieved executing the accelerated traces at their minimum possible IIs.

6.2. Resource Usage

Table 1 contains average resource requirements and synthesis frequency for the three systems and for three comparisons considering common benchmark subsets. The first three columns contain the average synthesis frequency of the RPU designs and the average required number of Lookup Tables (LUTs) and Flip Flops (FFs).

The mean number of LUTs and FFs required by the RPU design in system *A* are 3837 ($max = 18626$, $min = 695$, $\sigma = 4168$) and 2634 ($max = 8374$, $min = 959$, $\sigma = 1815$), respectively. This corresponds to an average of 49 % and 54 % of the total LUT and FF of the system. The RPU requires 3.42× the number of LUTs and 3.95× the number of FFs that of a MicroBlaze processor. The average synthesis frequency was 97.2 MHz ($\sigma = 34 MHz$). For system *B*, the LUT and FF requirements are 3403 ($max = 7771$, $min = 1333$, $\sigma = 4168$) and 5281 ($max = 16332$, $min = 2113$, $\sigma = 3909$), which correspond to 37 % and 51 % of the system. When compared to a MicroBlaze 3.04× and 7.92× the LUTs and FFs are required. The average synthesis frequency was 127.8 MHz ($\sigma = 24 MHz$).

The pipelining cost of system *B* increases its average required number of FFs from 2031 to 4494, relative to system *A*. The mean synthesis frequency increases from 111 MHz to 129 MHz. This is most due to optimizations regarding the logic handling memory access and arbitration for the *load* and *store* FUs, where the critical path of the design was most found for these implementations.

For system *C*, the LUT and FF used correspond to 45 % and 53 % of the total number of the respective resources. The RPU requires 2.37× the LUTs and 3.83× the FFs of a MicroBlaze. However, the relative mean number of required slices is only 1.23×. The 2D oriented RPU architectures in *A* and *B* are synthesized for *Speed*, while the synthesis of the modulo-scheduled architecture in *C* targets *Area* at *high* effort. Despite this, the new architecture achieves comparable operating frequencies for the same resource requirements. Earlier designs required a synthesis target of *Speed* to reach operating frequencies which did not force a decrease in the system operating frequency. This however still happened for the larger RPUs in both systems *A* and *B*.

6.3. Open Issues

Further performance improvements can be attained by enhancing the methodology and architectures presented.

Achieved acceleration is dependent on the efficiency of the Megablock detection step. Currently, a Megablock represents a single-path trace through a loop. If a trace originates from a loop with an *if-else* clause, this means that there might be two possible traces with the same starting address. Current limitations impose that only one path be implemented since the ambiguity cannot be resolved. Even if no ambiguity existed, both paths could be equally frequent and, for instance, execute alternatively In this case, the RPU would perform few iterations per call and a greater overhead would be introduced. Solving this issue entails both adding support for conditional execution on the RPU and the ability to extract multi-path traces during profiling, to enable parallel execution of loop paths in hardware.

Additional information could also be extracted during profiling. One way is to enhance the runtime-only analysis with compiler-generated meta-information, so as to assist Megablock identification. A further challenge is the identification of specific memory access patterns, in particular data streaming, to enable the deployment of very efficient, customized memory interfaces.

Given the impact of external memory access on acceleration, a pre-fetch mechanism could be implemented to mitigate the access latency, given that access pattern information was known. Furthermore, it is trivial to enhance the new RPU architecture with support for higher latency floating-point operations, which will are becoming less costly in terms of resource given recent FPGAs with dedicated floating-point units [19]. To make better use of resources when accelerating multiple Megablocks, a method is still required to determine the optimal amount of resources for an efficient performance/resource trade-off.

Finally, the design of an autonomous adaptable system involves addressing the challenges of: runtime binary analysis and translation, and on-chip RPU generation. A simple analysis of compiler generated binary can be done as in [20] for data-dependences and alias analysis (e.g., used for data partitioning), but the possibility to perform this kind of analysis at runtime is still an open problem. To generate the RPU on-chip, a combination of extendible, regular architectures and partial reconfiguration (just-in-time hardware compilation) could be explored.

7. Conclusion

This paper presented an overview of several system implementations where a GPP is augmented with an automatically generated Reconfigurable Processing Unit. Two different RPU architectures are presented and compared. Instantiation of an RPU is done via a specialization of a generic template. Instruction patterns are extracted from runtime traces generated by a simulation step, used to specialize the RPU and then accelerated at runtime in a manner transparent to the GPP.

The most recent RPU design exploits inter-iteration parallelism and performs loop pipelining through modulo scheduling. The increased performance and reduced resource requirements are compared to previous RPU implementations.

Ongoing work focuses on the evaluation of the approach with more complex benchmarks. Future work plans include extensions to the Megablock, the use of memory disambiguation techniques, and the migration of the entire mapping scheme to runtime.

Table 1: Average RPU characteristics, resource requirements, and Speedups over a single MicroBlaze for the three systems and for comparisons of common benchmark subsets

	System	Synt. Freq (Mhz)	LUTs	FFs	#Insts.	#FUs	IPC_{HW}	Speedup
	A[1]	97.2	3837.3	2633.8	41.6	48.3	2.42	1.60
	B[2]	127.8	3403.9	5281.8	29.3	33.25	2.24	1.13
	C[3]	131.2	2661.4	2551.8	48.7	13.7	9.78	5.44
Comparison	System	Synt. Freq (Mhz)	LUTs	FFs	#Insts.	#FUs	IPC_{HW}	Speedup
A vs B	A	110.9	3341.2	1751.6	22.5	29.4	2.10	1.62
	B	128.9	3174.3	4214.3	21.9	26.8	2.51	1.21
B vs C	B	128.5	1915.7	3359.5	25.5	24.3	3.76	2.42
	C	132.3	1636.0	1593.2	25.5	13.7	10.81	8.37
A vs C	A	120.2	2791.4	1990.3	51.3	51.6	2.35	2.25
	C	130.4	2794.6	2277.0	52.0	14.7	10.49	5.44

[1]local memory, 2D non-pipelined RPU; [2]external memory, 2D pipelined RPU; [3]local memory, 1D modulo scheduled RPU;

References

[1] Katherine Compton and Scott Hauck. Reconfigurable computing: A survey of systems and software. *ACM Comput. Surv.*, 34(2):171–210, June 2002.

[2] João M. P. Cardoso, Pedro C. Diniz, and Markus Weinhardt. Compiling for reconfigurable computing: A survey. *ACM Comput. Surv.*, 42(4):13:1–13:65, June 2010.

[3] Roman Lysecky and Frank Vahid. Design and implementation of a microblaze-based warp processor. *ACM Trans. Embed. Comput. Syst.*, 8(3):22:1–22:22, April 2009.

[4] Nathan Clark, Amir Hormati, and Scott Mahlke. VEAL: Virtualized execution accelerator for loops. In *35th Intl. Symp. on Computer Architecture*, pages 389–400, june 2008.

[5] Antonio Carlos S. Beck, Mateus Beck Rutzig, and Luigi Carro. A transparent and adaptive reconfigurable system. *Microprocessors and Microsystems*, 38(5):509–524, 2014.

[6] João Bispo, Nuno Paulino, Joao M.P. Cardoso, and Joao C. Ferreira. Transparent runtime migration of loop-based traces of processor instructions to reconfigurable processing units. *Intl. Journal of Reconf. Computing*, 2013. Article ID 340316.

[7] Nuno Paulino, João C. Ferreira, and João M.P. Cardoso. Architecture for transparent binary acceleration of loops with memory accesses. In Philip Brisk, João G. de F. Coutinho, and Pedro C. Diniz, editors, *Reconfigurable Computing: Architectures, Tools and Applications*, volume 7806 of *LNCS*, pages 122–133. Springer Berlin Heidelberg, 2013.

[8] Nuno Paulino, João C. Ferreira, and João M. P. Cardoso. Trace-based reconfigurable acceleration with data cache and external memory support. In *2014 IEEE Intl. Symp. on Parallel and Distributed Processing with Applications*, pages 158–165, Aug 2014.

[9] Nuno Paulino, João C. Ferreira, João Bispo, and João M. P. Cardoso. Transparent acceleration of program execution using reconfigurable hardware. In *Design, Automation, and Test in Europe (DATE), Hot Topic - Transparent use of accelerators in heterogeneous computing systems*, Grenoble, France, Mar 2015. (to appear).

[10] Greg Stitt and Frank Vahid. Thread warping: Dynamic and transparent synthesis of thread accelerators. *ACM Trans. Des. Autom. Electron. Syst.*, 16(3):32:1–32:21, June 2011.

[11] Jason Cong, Hui Huang, Chiyuan Ma, Bingjun Xiao, and Peipei Zhou. A fully pipelined and dynamically composable architecture of CGRA. In *Proc. of the IEEE Intl. Symp. on Field-Programmable Custom Computing Machines*, pages 9–16, Washington, DC, USA, 2014. IEEE Computer Society.

[12] Kevin Fan, Hyun hul Park, Manjunath Kudlur, and S ott Mahlke. Modulo scheduling for highly customized datapaths to increase hardware reusability. In *Proceedings of the 6th Annual IEEE/ACM Intl. Symp. on Code Generation and Optimization*, pages 124–133, 2008.

[13] Taewook Oh, Bernhard Egger, Hyunchul Park, and Scott Mahlke. Recurrence cycle aware modulo scheduling for coarse-grained reconfigurable architectures. In *Proc. of the 2009 ACM SIGPLAN/SIGBED Conf. on Languages, Compilers, and Tools for Embedded Systems*, pages 21–30, 2009.

[14] Hyunchul Park, Kevin Fan, Scott A. Mahlke, Taewook Oh, Heeseok Kim, and Hong-seok Kim. Edge-centric modulo scheduling for coarse-grained reconfigurable architectures. In *Proc. of the 17th Intl. Conf. on Parallel Architectures and Compilation Techniques*, pages 166–176, 2008.

[15] Narasinga Rao Miniskar, Soma Kohli, Haewoo Park, and Donghoon Yoo. Retargetable automatic generation of compound instructions for CGRA based reconfigurable processor applications. In *Proc. of the 2014 Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, pages 4:1–4:9, New York, NY, USA, 2014. ACM.

[16] João Bispo and João M. P. Cardoso. On identifying segments of traces for dynamic compilation. In *Intl. Conf. Field-Programmable Logic Appl. (FPL'10)*, pages 263–266, 2010.

[17] Xilinx. 7 Series FPGAs overview. www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf, 2014. [Accessed 28-nov-2014].

[18] José Gabriel Coutinho Zlatko Petrov (eds.) João M. P. Cardoso, Pedro Diniz, editor. *Compilation and Synthesis for Embedded Reconfigurable Systems: An Aspect-Oriented Approach*. Springer, 1 edition, 2013.

[19] Altera. Arria 10 core fabric and general purpose I/Os handbook. www.altera.com/literature/hb/arria-10/a10_handbook.pdf, 2014. [Accessed 28-nov-2014].

[20] Gaurav Mittal, David C. Zaretsky, Xiaoyong Tang, and P. Banerjee. Automatic translation of software binaries onto fpgas. In *Proc. of the 41st Annual Design Automation Conference*, pages 389–394. ACM, 2004.

Three Channel G-Link Transmitter Emulation in FPGA for the ATLAS Tile Muon Digitizer Board

José Domingos Alves[1], Agostinho Gomes[1], Guiomar Evans[2, 3], José Soares Augusto[2, 3]
jalves@lip.pt *agomes@lip.pt* *gevans@fc.ul.pt* *jaaugusto@fc.ul.pt*

[1]Laboratório de Instrumentação e Física Experimental de Partículas, Av. Elias Garcia 14-1, Lisbon 1000-149, Portugal
[2]Faculdade de Ciências da Universidade de Lisboa, Campo Grande, Lisbon 1749-016, Portugal
[3]Centro de Física da Matéria Condensada, Campo Grande, Lisbon 1749-016, Portugal
[4]Instituto de Engenharia de Sistemas e Computadores - ID, Rua Alves Redol 9, Lisbon 1000-029, Portugal

Abstract

In this paper a three channel implementation of a G-Link transmitter in a Spartan-6 FPGA[1] from Xilinx, hosted on the "Tile Muon Digitizer Board", recently assembled to be used on the ATLAS[2] trigger system, is presented. This implementation was performed using the high speed serial transceivers (GTP transceivers) embedded in the FPGA and VHDL[3] custom modules. Communication tests with the suitable components were performed to evaluate the design.

1. Introduction

The Tile Calorimeter (TileCal) is a sub detector of the ATLAS detector of CERN (European Organization for Nuclear Research). It provides accurate measurement of the energy of jets of particles from the proton-proton collisions that occur during the LHC (Large Hadron Collider) experiments [1], the measurement of Missing Transverse Energy and the identification of low-pt muons. It has a trigger system that is used to select events produced on LHC collisions in each 25 ns or 50 ns. This trigger system combines specialized hardware (first level trigger) and specific software (second and third level trigger). The D-cells of the TileCal will be added to the trigger system to reduce the rate of creation of fake triggers generated by particles of low momentum. For this purpose a board which was coined as *Tile Muon Digitizer Board* (TMDB) was designed and assembled to interface the D-cells signals with the ATLAS Sector Logic (SL) of muons. The communication of TMDB with the muons' SL is performed using a three channel G-Link, as physical layer. The TMDB receives and digitizes the signals from the D-cells in the outer layer of the TileCal Extended Barrel. These cells are named as D5 and D6. The TMDB prepares trigger primitives for the muon Sector Logic. Each TMDB connect to 3 Sector Logic boards, and each 25 ns three words of 16 bits are sent through G-Link channels. Therefore, this communication link needed to be implemented and tested, and it is the subject of this paper. There are 128 modules of TileCal to be interfaced with 48 Sector Logic boards, which implies that 16 TMDB boards must be used. The TMDB hosts two different FPGAs to handle different functions, a Xilinx Spartan-6 FPGA (*core FPGA*) and an Altera Cyclone III FPGA (*VME[4] FPGA*). The *Core FPGA* is responsible for processing the digitized inputs from the *Analog Stage* (Analog-to-Digital Converters), which receives the 32 analog signals from the 16 TileCal cells. The *VME FPGA* used implements VME support, needed for the VME crate where the board will be hosted. The output data is available through optical links (using G-Link).

This paper is structured in 4 sections. Section 2 is dedicated to the G-Link chip-set specification. The description of the implementation of the three channels G-Link is presented in section 3. Finally, in section 4 there are the final considerations and conclusions about this work.

2. G-Link Chip-Set Specification

The G-Link chip-set HDMP-1032/1034 can be used to build a serial high speed data link for point-to-point communication, providing data rates that extend from 0.208 to 1.120 Gbps [3]. Examples of its applications are: cellular base stations, ATM switches, video/image acquisition [3, 4]. For example, the ATLAS Tile Calorimeter front-end and the back-end electronics use the Agilent HDMP-1032/1034 Transmitter/Receiver chip-set as the physical layer. This chip-set uses the *Conditional Inversion Master Transition* (CIMT) coding scheme to encode/decode the data. The CIMT coding scheme ensures the DC balance of the serial line [3, 4]. This encoding scheme uses three types of words: *data words*, *control words* and *idle words*. Each word to be transmitted consists of a *Word-Field (W-*

[1] FPGA - **F**ield **P**rogrammable **G**ate **A**rrays
[2] ATLAS - **A** **T**oroidal **L**HC Apparatu**S**
[3] VHDL - **V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits) **H**ardware **D**escription **L**anguage

[4] VME - **V**ersa **M**odule **E**uropa

Field) followed by a *Coding Field (C-Field)*. The *C-Field* is used to flag the type of word (*data*, *control* or *idle* words) to be transmitted and consists of 4 bits, which are added to the data input of 16 bits, before transmission. The *C-Field* has a master transition in the middle (Fig. 1), which allows the receiver to check for this transition in received data in order to perform word alignment and error detection. This master transition also serves as the reference for the receiver clock recovery circuit. When neither data words or control words are being sent, idle words are transmitted to maintain the DC balance in the serial link and allow the receiver to sustain frequency and phase lock [3, 4]. The DC balance of the serial line is ensured by monitoring the disparity (disparity is defined as the total number of high bits minus the total number of low bits) of successive encoded data words. The words are conditionally inverted in accordance with the disparity of the previously sent bits.

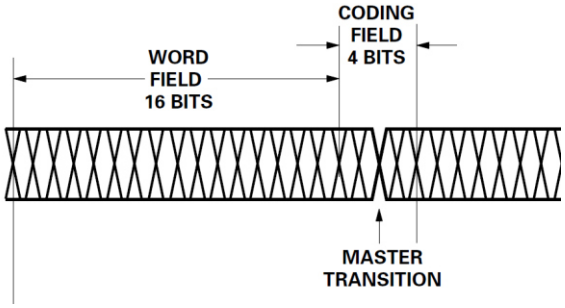


Fig. 1: CIMT encoding scheme [3].

If the sign of the current word is the same as the sign of the previously transmitted bits, then the word is inverted. If the signs are opposite, the word is not inverted. No inversion is performed if the word is an *idle word*. If a word is inverted before its transmission, it will be inverted again at the receiver to recover the original word [3, 4].

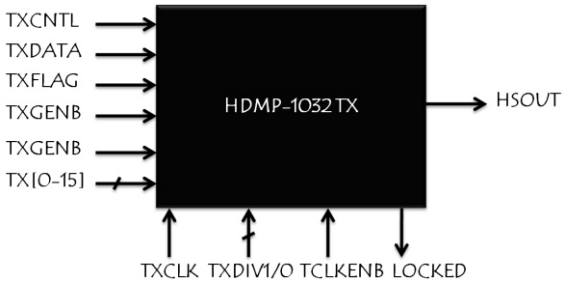


Fig. 2. HDMP-1032 Transmitter [3].

In Fig. 2, a representation of the HDMP-1032 G-Link Transmitter chip with its main inputs and the serial output (HSOUT) is presented. The data is encoded regarding the values of the input signals: TXFLAG, TXDATA and TXCNTL. If TXCNTL is high means that a *Control word* should be sent, no matter the value on the TXDATA input. In control mode, only the bits TX[0-13] are sent, as the chip uses only the first 13 bits of TX to construct a *Control Word*, the last two bits are ignored and the *W-Field[7-8]* are replaced by "01", as shown in table 1.

Type	FLAG	W-Field			C-Field
Data Word	0	TX[0-15]			1101
	1				1011
Control Word	-	TX[0-6]	01	TX[7-13]	0011
Idle Word	1a	1111111	11	0000000	0011
	1b		00		

Table 1. Coding scheme [3].

If TXCNTL is low and TXDATA is high, it sends TX[0-15] and a *C-Field* encoded as a *data word*. The flag bit (TXFLAG) and can optionally be used as an extra bit for applications where the data to be transmitted is 17 bits wide. When this option is selected the TXFLAG input is sent as an extra bit encoded in the *C-Field*, when it is not the FLAG internally alternates between '0' and '1'. If neither TXCNTL nor TXDATA are high, the transmitter chip assumes that the link is not being used. In this case, it submits *idle words* to maintain the DC balance on the serial link allowing the receiver to maintain frequency and phase lock. Two *idle words* (table 1) can be sent, accordingly with the disparity of the last transmitted word. If the disparity sign of the previously transmitted bits is low, the *idle word 1a* is sent, while if disparity sign is high *idle word 1b* is sent.

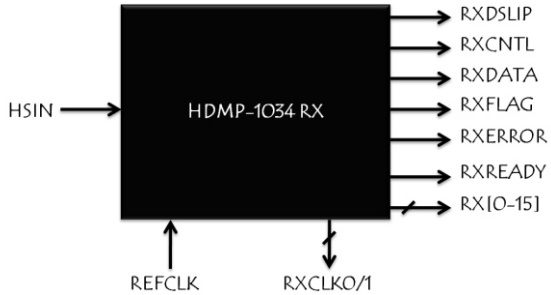


Fig. 3. HDMP-1034 Receiver [3].

In Fig. 3, a representation of the HDMP-1034 G-Link receiver chip, is also presented, with its serial input and outputs pins. It converts the serial data signal sent from the HDMP-1032 transmitter, into 16 bit wide parallel data, or into 17 bit wide parallel data if the TXFLAG was sent as an extra bit. It performs the following functions: frequency lock, phase lock, encoded word synchronization, de-multiplexing, word decoding and encoding error detection. It also performs clock recovery from incoming data stream and all internal operations are synchronized with the recovered clock. It decodes the 4 bits *C-Field* and determines whether the 16 or

17 bit *Word-Field* consists in: normal or inverted; *data*, *control*, or *idle words*; or errors. The flag bit is also decoded from the data word. If RXDATA is high it indicates that a *data word* is detected by the receiver. If RXCNTL is high it indicates that a *control word* is detected by the receiver. An idle word is detected by the receiver if RXDATA, RXCNTL and RXERROR are low [3, 4].

3. The Proposed Design

As the present muon's SL boards was not upgraded already, it hosts the original HDMP-1034 receiver chip to de-serialize data from TMDB, it is required that the TMDB hosts transmitter chips. As the G-Link chip-set is obsolete and is not being produced, the possible solution is the emulation of its functionalities on the Xilinx Spartan-6 FPGA hosted on the board. This FPGA has high speed serial transceivers embedded on it, which makes possible the emulation. In Fig. 4, the block diagram of the implemented design is presented. A VHDL custom *CIMT encoder* module was developed. This encoder is able to perform the required encoding scheme described in section 3. It monitors the disparity of data, ensuring that the number of 1 (ones) and 0 (zeros) are balanced, in order to ensure the DC balance on the serial line. Also, a VHDL custom module, which we called *CSM (Control State Machine),* was developed. This module was implemented to ensure that a set of *idle words* are sent at link start-up, required for the receiver chip to perform the synchronization with the incoming data.

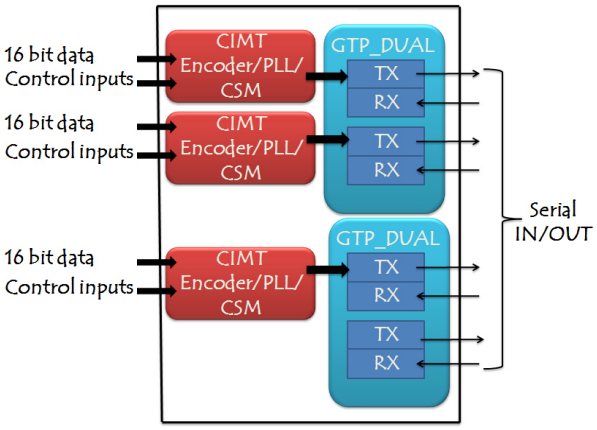


Fig. 4. Block diagram of the implemented design.

This set of *idle words* are sent after the *CSM* module receives a flag *(Flag_GTP_ready)* from the GTP transceiver flagging that it is ready to receive data to be serialized (Fig. 5). When it sends all *idle words*, it outputs a flag (*Flag_TX_ready*) that flags that user data can be put on the 16 bit input *TX (15:0)*, to be transmitted. This process is illustrated in Fig. 5, where a one channel transmitter is presented, with more details. The CSM block takes three user command signals: *idle*, *data* and *control*; and generates the input control signals (*TX_CNTL* and *TX_DATA*) for the *CMIT encoder* module, accordingly with table 2.

SCM user commands			CIMT encoder signals		CIMT encoder output
Idle	data	control	TX_CNTL	TX_DATA	
1	0	0	0	0	Idle word
0	1	0	0	1	Data word
0	0	1	1	0	Control word

Table 2. User logic commands and encoder control signals.

The user logic defines what is sent by the channel (*idle word*, *data word* or *control words*) by managing these command signals, as presented in table 2. Other states of command signals not presented in the table are not considered. The *PLL* block takes the input clock of 40 MHz from the LHC TTC (Trigger Timing Control) system, and produces 3 clock signals: 160 MHz and 80 MHz both for the GTP; and also a 40 MHz, required for the other blocks. User logic data (16 bit parallel) is directly put on the *TX (15:0)* input of the encoder (at a frequency of 40 MHz, in this case), where it is coded and the *C-Field* added. Therefore, the encoder output is a 20-bit word in each 25 ns time interval, which is sent to the GTP transceiver, where it is serialized and transmitted through optical links at a rate of 800 Mbps.

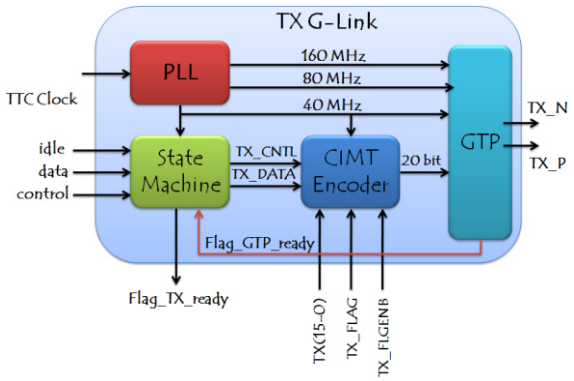


Fig. 5. One channel G-link transmitter.

The GTP transceiver [5] is a power-efficient transceiver embedded in Spartan-6 FPGAs. It is highly configurable and tightly integrated with the programmable logic resources of the FPGA. It supports multiple industry standards with the following line rates: 614 Mbps to 810 Mbps, 1.22 Gbps to 1.62 Gbps and 2.45 Gbps to 3.125 Gbps. It support fixed latency and deterministic modes [5], which is required for trigger applications like this.

They are organized in pairs (GTP_DUAL) on the FPGA. A GTP_DUAL is composed by two GTP instantiations (two TX/RX pairs). In our design we use two GTP_DUAL, as we needed 3 TX (two from 1 GTP_DUAL and one from another GTP_DUAL). Each GTP transceiver includes an independent transmitter, which consists of a PCS (Physical Coding Sub-layer) and a PMA (Physical Medium Attachment sub-layer) block. Parallel data flows from the user logic into the FPGA TX interface, through the PCS and PMA, and then goes out the TX driver as high-speed serial data [5], as shown in Fig. 6.

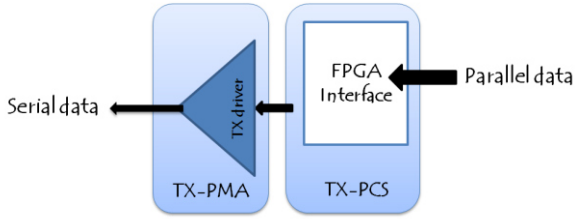


Fig. 6. Simplified block diagram of the GTP TX [5].

The FPGA interface is able to accept 8, 10, 16, 20, 32, and 40 bits wide parallel data [5]. In this application the FPGA interface was configured to accept 20 bits parallel data and configured to work at 800 Mbps bit rate.

3.1. Tests and Results

In figure 7, a TMDB prototype in the VME crate is presented. Communications tests at 800 Mbps (through optical fibers) between the TMDB (using the implemented design) and a board hosting an original receiver chip (called PT5, which is an ATLAS Read-Out Driver - ROD emulator) was performed, as shown in Fig. 8.

Fig. 7. TMDB board in the VME crate.

The first tests show that correct communications was achieved: the data sent by the TMDB was correctly received and read on the PT5, as we present in Fig. 9.

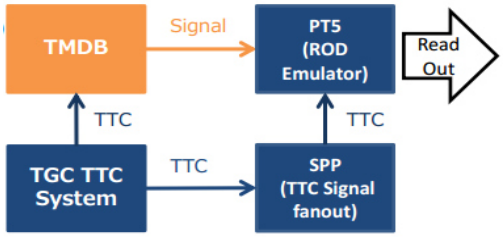


Fig. 8. Communication tests scheme between TMDB and PT5 [6].

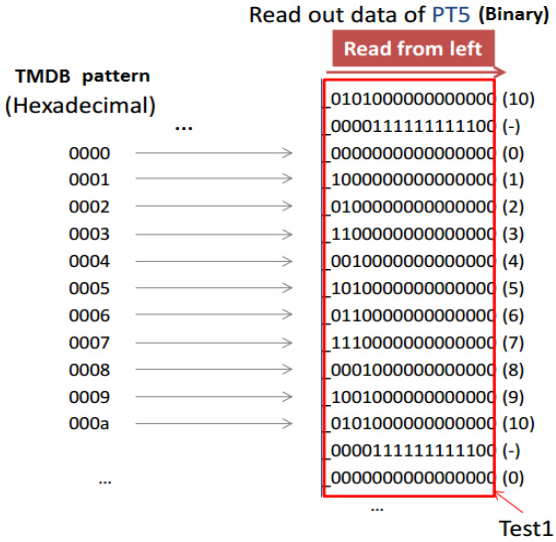


Fig. 9. First result of G-Link communication test TMDB-PT5 [6].

In this test, in each clock cycle of 25 ns, the 16 bit inputs to our design are incremented as *"0, 1, 2,..., a"* and the patterns dumped by PT5, appeared as expected [6]. At this stage, we were able to perform communication (at 800 Mbps) between an emulated version of the G-Link transmitter and the original receiver chip. After this step, we advance to test the design with the final target, the Sector Logic boards, which also host the original G-Link chip to receive the data from TMDB (Fig. 10).

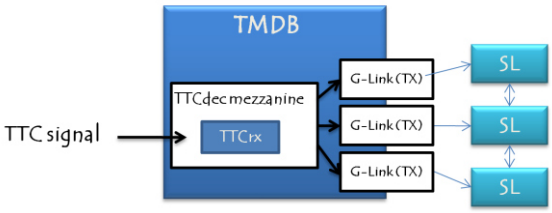


Fig. 10. Communication tests scheme TMDB-SL [7].

The SL boards output 8 bits patterns, which are coded (as in Fig. 11) from the 16 bits received from the TMDB. Therefore, a test pattern, for which we know in advance what will be the result on SL, was used. For instance, we configure our design to transmit the pattern "0000010000100110", for which accordingly with the coding made on the SL,

the readout from this SL should be “00100111”, and this is exactly what we got from SL [7]. This result is presented in the Fig 11.

Fig. 11. First results of the communication tests TMDB-SL [7].

This first result shows a correct communication between the TMDB-SL. Again we have a correct communication between an emulated version of the G-Link transmitter with the original receiver chip. It was the first time that a communication channel between SL boards with another board hosting an emulated version of G-Link, was implemented. Despite, these first correct results, which show a correct communication between TMDB-PT5/SL, using our design, further tests indicates that in some random period of transmissions, data is not correctly received. The investigation of this issue with the aim of checking where the source of error (G-Link transmitter, the PT5/SL hosting the original chip, channel or a bad clocking scheme, etc...) was made by another group of work that also collaborates on the TMDB developments and the results is presented in [9]. We have suggested before, that the error rate should be determined to help addressing this issue and that in our design, we use a PLL to generate all clock signals from the 40 MHz TTC clock, so we also suggested that different clocking can be considered in order to evaluate which clocking schemes lead to a lower bit error rate. However, as reported in [9], three issues was identified and corrected: (1) the clock constrains of the Xilinx Spartan-6 FPGA was not described properly in the design; (2) in SL the received signal can be sampled on either positive or negative of REFCLK of HDMP-1034 RX. Therefore, in the setup of figure 8, a positive edge was used and an error rate of 25 % was observed, while using the negative edge the error rate was reduced to the level of 1 x 10^{-7}, and (3) the clock frequency constrain was set to 40 MHz, but it should be 40.08 MHz, which is the value of the LHC TTC clock. After addressing these issues, it was reported that the G-Link stability was greatly improved [9]. We also checked the latency of our design, which is presented on table 3. We should note an interesting feature of our custom *CIMT encoder*: it performs all the operations (encoding, disparity monitoring and DC balance ensuring, as described in section 3) using only 2 clock cycles. The lower and fixed latency mode used in GTP allows a fixed and deterministic latency of about 3.2 clock cycles. The total latency of our design is about 5.2 clock cycles.

Block	#Clock cycles of 25 ns	Period (ns)
CIMT encoder	2	50
GTP TX	~3.2	80
Total	5.2	130

Table 3. Latency results of our G-link transmitter.

Comparing with the latency of the original transmitter chip (which is about 1.4 clock cycles) we note that our design has, of course, higher latency, mainly introduced by the GTP operations. But the latency value of our design complies with the TMDB specifications. In table 4, the hardware resources used on the Spartan-6 FPGA to implement our design are also presented.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	
Slices Registers	376	184304	1%
Slices LUTs	808	92152	1%
Bonded IOBs	5	396	1%
Specific Feature Utilization			
BUFIO2/BUFIO2_2CLKs	1	32	3%
BUFIO2FB/BUFIO2FB_2CLKs	1	32	3%
BUFG/BUFGMUXs	6	16	37%
GTPA1_DUAL	2	4	50%
Slice Logic Distribution			
Occupied Slices	365	23038	1%
LUT Flip-Flop pairs	899	2697	33%
MUXCYs	304	46076	1%

Table 4. Hardware logic resources used to implement the 3 channel G-Link transmitter.

5. Conclusions

In this paper we presented an emulation of a three channel G-Link transmitter implemented on a Xilinx Spartan-6 FPGA, and supported on its embedded GTP transceivers. The FPGA is hosted in the *Tile Muon Digitizer Board (TMDB*, which will be used for trigger purposes in the LHC ATLAS detector. Communication tests between our emulated G-Link transmitter and boards hosting the original receiver chip were performed to validate the design. The first tests have shown that a correct communication was achieved, but in further tests indicate some random period of times where data is corrupted. Works to address this issue was performed by another team work that also collaborates in the TMDB developments and presented in [9]. An interesting feature of this design is the relatively low total latency, of about 5.2 clock cycles, which compares very favourably with other implementations of the emulation of the G-Link chip-set, for instance the works presented in [4, 8].

References

[1] The Large Hadron Collider, the LHC Study Group, CERN/AC/95-05, 1995. https://cds.cern.ch/record/291782/files/cm-p00047618.pdf

[2] Augusto Cerqueira on behalf of ATLAS/Brazil group, *Tile-Muon Digitizer Board*, TMDB Technical review, 25 July 2014, CERN. https://indico.cern.ch/event/326618/contribution/0/material/slides/0.pdf

[3] Agilent HDMP1032/1034Transmitter/Receiver Chipset Datasheet. http://www.physics.ohiostate.edu/~cms/cfeb/datasheets/hdmp1032.pdf

[4] José Alves, José Silva, Guiomar Evans, José Soares Augusto, *"Emulation in FPGA of G-Link Chip-set of Tile Calorimeter Electronic System*", Conference on Electronics, Telecommunications and Computers – CETC 2013, December 5 and 6, 2013.

[5] Xilinx, Spartan-6 FPGA GTP Transceivers (Advance Product Specification) UG386 (v2.2) April 30, 2010. http://www.xilinx.com/support/documentation/user_guides/ug386.pdf

[6] Makoto Hasegawa, Takuto Kunigo, *Tile Muon connection test report*, June 19, 2014, CERN.

[7] Takuto Kunigo, Makoto Hasegawa, José Domingos Alves, *G-Link connection test between TGC Sector Logic and TMDB in blg. 175 report*, Tile-muon Trigger meeting, June 30, 2014, CERN. https://indico.cern.ch/event/327432/contribution/0/1/material/slides/0.pdf

[8] A. Aloisi, F. Cevenini, R. Giordano, V. Izzo, "An FPGA-based Emulation of the G-Link Chip-Set for the ATLAS Level-1 Barrel Muon Trigger", Topical Workshop on Electronics for Particle Physics, Paris, France, September 2009. http://indico.cern.ch/event/49682/session/30/contribution/80/material/paper/0.pdf

[9] M. Ishino, "*A report of G-link test between TMDB and Sector-Logic",* December 2014. https://indico.cern.ch/event/360085/contribution/0/material/0/1.pdf

Miscelânea

 ISBN: 978-989-98875-2-7

Redundância Modular Tripla baseada em *Bitstreams* Parciais para Multiplas Partições da FPGA

Victor M. Gonçalves Martins*†, João Gabriel Reis*, Horácio C. C. Neto†, Eduardo Augusto Bezerra*
*Departamento de Engenharia Elétrica e Electrônica
Universidade Federal de Santa Catarina
88040-900 Florianópolis, Santa Catarina - Brasil
{victor.martins, joao.reis, eduardo.bezerra}@eel.ufsc.br
†Electronic System Design and Automation - INESC-ID
Rua Alves Redol, 9; 1000-029 Lisboa, Portugal
{marte, hcn}@inesc-id.pt

***Resumo*—Em alguns sistemas com *Field Programmable Gate Array*s (FPGAs) a confiança é fulcral. O sistema não pode falhar e precisa de estar disponível pelo maior período de tempo possível. Para reduzir a ocorrência de falhas, mesmo que uma falta ocorra no sistema, uma política de *Triple Modular Redundancy* (TMR) é uma boa opção. Mas se ocorrer uma falta permanente no hardware que implementa o TMR, a vantagem da redundância pode ser destruída. Neste trabalho, é apresentada uma solução de baixo custo que dá a um sistema TMR a capacidade de Detecção, Isolamento e Recuperação de Faltas, do inglês *Fault Detection, Isolation and Recovery* (FDIR). A metodologia proposta utiliza o fluxo desenvolvido *Assisted Design Flow* (ADF) que implementa *Partial Bitstream for Multiple Partitions* (PB4MP), o qual habilita a realocação de módulos em diferentes *Reconfigurable Partition*s (RPs) com o mesmo *bitstream* ou através de uma operação de troca de memória de configuração interna. Os resultados mostram que com um pequeno aumento de memória de programa do sistema projectado, é possível reduzir a probabilidade de a ocorrência de duas faltas permanentes anularem a capacidade de TMR, de 86% numa implementação standard para 40%.**

Keywords*—Bitstream para Múltiplas Partições; Recuperação de FPGA

I. Introdução

O uso de *Field Programmable Gate Array*s (FPGAs) em sistemas digitais é cada vez maior, e em alguns deles é muito crítico se alguma falta forçar o sistema a falhar. Mesmo que passem com sucesso por todos os testes submetidos no final do processo de fabrico, não há garantia que fiquem para sempre imunes a faltas permanentes. Na realidade, estes dispositivos também sofrem de envelhecimento, o que resulta numa evolução natural de degradação física [1]. O ritmo de envelhecimento depende de vários factores como a maturidade da tecnologia usada, o controlo da qualidade de fabrico, temperatura e choques térmicos (relacionados com o ambiente onde o próprio opera), tensão de funcionamento, dimensão, e até mesmo a complexidade do circuito [1]. Como a densidade de recursos numa FPGA continua a aumentar a cada nova geração de tecnologia, preocupações com a confiabilidade, como *Negative Bias Temperature Instability* (NBTI), tornaram-se um factor relevante [2]. Este tipo de faltas não destroem categoricamente os recursos da FPGA, mas provocam o aumento dos seus tempos de propagação [3].

O esforço do trabalho proposto é reforçar a capacidade de recuperação do sistema implementado, de modo a incrementar a disponibilidade e confiabilidade da FPGA em relação ao processo de degradação por envelhecimento [4].

Neste artigo é proposta uma nova estratégia baseada em *Triple Modular Redundancy* (TMR) [1], onde a implementação do TMR segue autonomamente a filosofia *Fault Detection, Isolation and Recovery* (FDIR) [5]. Para que isto seja possível foi desenvolvido um *Assisted Design Flow* (ADF) para implementar uma solução baseada em *Partial Bitstream for Multiple Partitions* (PB4MP). Esta abordagem tem como principal foco sistemas embarcados em FPGAs que possuem núcleos de processadores ou equivalentes através de *softcores* [6]. Embora tenha sido desenvolvida para o dispositivo Virtex-6 XC6VLX240T da Xilinx, a metodologia proposta pode ser igualmente utilizada nas famílias mais recentes da Xilinx que suportam *Partial Reconfiguration* (PR). Os resultados mostram que com um reduzido incremento de memória utilizada pelo processador, é possível detectar uma falta permanente e recuperar o TMR implementado na FPGA. O volume de memória exigida pode ser desprezável quando o próprio sistema inclui um *Operation System* (OS) como o *Embedded Parallel Operating System* (EPOS) [7].

Este documento é organizado do seguinte modo. Secção II que descreve o estado da arte. Secção III que descreve detalhes técnicos relativos à família de FPGAs Virtex-6, necessários para este trabalho. Secção IV apresenta a arquitectura proposta, o fluxo de operações associado, e a correspondente caracterização dos requisitos de memória e processamento. Secção V sumariza um caso de estudo e apresenta os respectivos resultados. Secção VI conclui este documento e descreve trabalhos futuros.

II. Trabalhos Relacionados

Esta secção apresenta os trabalhos relacionados como modos de implementação de sistemas numa FPGA, que sigam a política FDIR [5], e as técnicas e tecnologias disponíveis que permitem a implementação.

A. Detecção, Isolamento e Recuperação de Faltas (FDIR)

O uso da arquitectura TMR [1] não é recente. A equipa de Montminy [8] apresenta um sistema com TMR, onde usa a capacidade PR da FPGA. Neste trabalho, a detecção das faltas

é feita de forma automática. Quando uma falta origina uma falha, o módulo irá produzir uma resposta diferente da dos restantes. A unidade que analisa os três resultados e vota no correcto irá detectar a existência da falha e identificar imediatamente o respectivo módulo. Detectado o módulo com falta(s), a correspondente partição é excluída, o módulo é alocado numa partição suplente e o sistema retorna ao normal funcionamento. Esta abordagem obriga a sacrificar uma considerável quantidade de recursos para que seja possível a realocação dos módulos. Para além disso, o processo de realocação obriga à existência de uma biblioteca de *bitstreams* parciais, que para além da necessária manutenção, exige memória externa.

Dumitriu [9], embora sem TMR, desenvolveu um *framework* intitulado *Dynamic Partially Reconfigurable* (DPR). Esta plataforma é constituída por várias *Reconfigurable Partition*s (RPs) designadas por *slots*. Nesses *slots* são alocados módulos de hardware, que são identificados pelo autor como *Collaborative Macro-Functional Unit*s (CMFUs). Cada CMFU possui um *Built-In Self-Test* (BIST) implementado o que permite o seu teste periódico. Quando uma falta é detectada pelo teste, o CMFU é realocado noutro *slot* que se encontre livre, e o *slot* actual é marcado como defeituoso.

B. Manipulação da Configuração da FPGA

O advento da RP trouxe uma enorme flexibilidade na forma de como podemos utilizar os recursos de uma FPGA. No trabalho [10], a implementação do sistema foi condicionada de modo a ser aproveitada a capacidade de observar e controlar os Flip-Flops da FPGA através do *Internal Configuration Access Port* (ICAP). Desta forma, foi possível a implementação de um BIST através de um *Virtual Scan Chain*, evitando o gasto extra de recursos do dispositivo. O controlo do BIST é realizado pelo OS que corre no microprocessador do sistema e que possui suporte para usar o ICAP.

O uso de ferramentas de PR como o PlanAhead da Xilinx [11], permite definir RPs e gerar individualmente um *bitstream* parcial para cada módulo que se pretenda alocar numa determinada RP. Esta individualidade acontece fundamentalmente por duas razões:

- por norma, o roteamento entre recursos externos à RP passam pelo interior da mesma. O que obriga a que o parcial *bitstream* possua para além da implementação do módulo desejado o roteamento do sistema externo;
- não há controlo sobre o roteamento que faz o interface entre o módulo implementado na RP e o restante sistema implementado na FPGA. Podendo por isso, dois módulos com o mesmo interface definido, ficarem com roteamento relativo bem distinto.

Relativamente à primeira razão, a própria Xilinx disponibiliza o *Isolation Design Flow* (IDF) [12]. Respeitando este fluxo temos a garantia que o módulo implementado numa RP fica completamente isolado do restante sistema implementado na FPGA.

Em relação à compatibilidade do interface de várias RPs, Yoshihiro [13] apresenta um fluxo que procura solucionar esse problema. Executa uma primeira vez o normal fluxo PR até ao final do "Place and Route" e com o comando "PR2UCF" extrai as posições dos *Proxy Logics* de todas as RPs. Opta por um conjunto de *Proxy Logics* de uma das partições e constrói um ficheiro de restrições forçando as localizações dos *Proxy Logics* das restantes partições a serem iguais de uma foram relativa. Seguindo a mesma filosofia, com a ferramenta "Directed Routing Constraints" do "Xilinx FPGA Editor", extrai o roteamento dos sinais que fazem parte do interface de uma partição de referência. Depois com essa informação força o roteamento dos sinais que constituem o interface das restantes PRs. Isso é feito através da adição de regras no mesmo ficheiro de restrições. Após esta actualização do ficheiro de restrições, é executado novamente todo o fluxo e gera o *bitstream* com a implementação desejada.

Embora muito interessante, este processo é manual e não cobre as seguintes necessidades:

- Situações em que um sinal de saída de uma partição possua um *fanout* maior que 1. Neste cenário, o sinal tem vários destinos;
- Situações em que a saída do sinal de entrada de uma partição tenha um *fanout* maior que 1. Nesta situação, a entrada do módulo é apenas um dos vários destinos do sinal. Muitas vezes, o mesmo sinal é entrada de várias partições;
- Forçar que árvore de sinal de relógio seja relativamente a mesma. Como cada região de relógio de uma FPGA possui várias árvores de sinais de relógio, durante o "Place and Route" a ferramenta poderá optar pela atribuição de árvores relativamente diferentes.

C. Contribuições

A principal contribuição deste trabalho é propor uma forma de implementar um sistema com TMR numa FPGA com capacidade de recuperação de faltas permanentes. O aumento da confiabilidade é obtido através do uso do fluxo ADF-PB4MP, o qual, com algumas restrições extra permite obter a capacidade de trocar módulos entre RPs. Com este novo mecanismo é possível dotar a implementação do TMR da capacidade de recuperação de faltas permanentes, sem necessidade de utilização de recursos extra da FPGA em partições suplentes como nas propostas [8] [9]. A principal ameaça considerada são as faltas permanentes, que podem ser originadas por uma danificação do dispositivo, ou algo que embora não destrua o recurso possa aumentar o seu tempo de resposta. Esta proposta deve ser complementar ao uso de soluções como o *Soft Error Mitigation* (SEM) [14], de modo a que um *Single Event Upset* (SEU) não possa ser interpretado como uma falta permanente. Embora por uma questão de celeridade na demonstração se tenha usado a plataforma do trabalho [6], dado o risco que é o votador sofrer alguma falta, é conveniente usar um processador como LEON3 [15] que tem tolerância a faltas, incluindo SEU, e possui inclusive uma versão para aplicações espaciais.

III. Arquitectura da família de FPGAs Virtex-6

O mecanismo PB4MP proposto é complexo e, por isso, exige conhecimento profundo da FPGA usada. Portanto, esta secção revê alguns detalhes importantes relativos à família de FPGAs Virtex-6 da Xilinx [16].

A. Recursos e Arquitectura da Memória de Configuração

Os recursos disponíveis numa FPGA são compostos por *Configurable Logic Block*s (CLBs), *Digital Signal Processing*s (DSPs), BRAMs, IOBs e módulos de gestão de sinais de relógio (*Phase Locked Loop*s (PLLs)). Estes recursos estão organizados numa matriz, linhas (*rows*) x colunas (*columns*), e onde cada um tem um bloco de roteamento associado (*switch box*). Cada linha é composta por uma secção de cada coluna de recursos. Relativamente à fracção da coluna com CLBs, esta inclui 40 CLBs na Virtex-6 (50 nas 7-Series). Para configurar cada secção de 40 CLBs é necessário um grupo de *frames* de memória consecutivos. Onde cada *frame* é composto por 81 palavras de 32 bits. A primeira parte do grupo de *frames* guarda a informação relativa ao roteamento que configurar os blocos de roteamento. Os restantes incluem a configuração das *Look Up Table*s (LUTs) presentes nos 40 CLBs, as propriedades dos *Slices* e respectivos multiplexers, e também os parâmetros INIT, que definem o valor inicial de cada Flip-Flop presente no grupo de 40 CLBs.

B. Processo de Geração da Configuração da FPGA

É necessário um profundo conhecimento dos processos de PR e Isolamento para implementar PB4MP. Existem importantes capacidades no fluxo PR que são essenciais para esta proposta. Essas capacidades estão seguidamente listadas:

- As ferramentas de síntese da Xilinx para PR adicionam uma primitiva *LUT1* em todos os sinais de entrada e saída de uma RP. São designados por *Proxy Logic* e a sua posição na FPGA pode ser controlada por intermédio de restrições [11];
- Adicionando a propriedade `Isolated="true"` no ficheiro xpartition.pxml, as ferramentas forçam o módulo implementado nessa RP a ficar completamente isolado do resto do sistema [12];
- Usando a restrição `SCC_BUFFER` nos sinais que conectam aos *Proxy Logics* no lado do resto do sistema, é possível garantir que a origem dos sinais que conectam os módulos implementados nas RPs ao resto do sistema têm um *fanout*=1 [12];
- O "Directed Routing Constraints" permite-nos forçar o roteamento de um sinal na FPGA [17].

IV. TMR proposto baseado em PB4MP

Neste documento, é proposto um sistema que recorre a TMR com capacidade de recuperação de faltas permanentes. Como é implementado através da ferramenta PB4MP desenvolvida, o aumento da confiabilidade é obtido sem necessitar de mais recursos da FPGA. Contudo, o PB4MP usa capacidades de PR [11] e IDF [12] para alcançar a desejada implementação.

A. Arquitectura TMR

Basicamente, o mecanismo TMR implica implementar três vezes o mesmo módulo/função. Todos os módulos funcionam em paralelo e fazem exactamente o mesmo. Os resultados dos três módulos são analisados por um outro módulo que vota num resultado correcto. Deste modo, mesmo que um módulo produza um resultado errado, os outros dois garantem um resultado correcto. O ponto fraco desta estratégia é o módulo que vota, que se falhar, todo o sistema TMR falha. A principal vantagem do TMR é a capacidade de detectar e corrigir erros, e a principal desvantagem é a necessidade de usar três vezes mais hardware [1].

B. A Adaptação do PB4MP para o TMR Proposto

Soluções como [8] [9] que usam o fluxo normal de PR implicam uma inevitável limitação: é necessário gerar um *bitstream* parcial para cada módulo em cada RP. Isto significa que, se existirem N módulos, M RPs e desejarmos poder alocar qualquer módulo em qualquer RP, será necessário criar uma biblioteca com $N * M$ *bitstreams* parciais.

Por causa desta limitação foi desenvolvido o PB4MP. Usando uma compilação de informações do IDF [12], do trabalho prévio realizado por outro autor [13] e da documentação da Xilinx [17], desenvolveu-se uma ferramenta usada no ADF.

O ADF possui os seguintes requisitos:

- Todas as RPs necessitam de incluir precisamente a mesma quantidade de recursos da FPGA exactamente com a mesma distribuição física;
- Os limites verticais da RP (superior e inferior) têm de coincidir com as regiões de relógio da FPGA;
- As RPs só podem incluir pinos de entrada/saída da FPGA se eles nunca forem usados pelo sistema;

Como resultado, o ADF permite implementar um sistema numa FPGA, onde todas as RPs possuem o mesmo *interface* físico, o qual permite usar apenas um *bitstream* parcial para cada módulo que pode ser alocado em qualquer RP.

Para tirar vantagem do uso do PB4MP no TMR, uma lista adicional de restrições é acrescentada no processo ADF. Esta lista é elaborada usando a restrição `CONFIG PROHIBIT` que nos faculta a opção de seleccionar recursos da FPGA que pretendemos que não sejam usados. Nesta primeira versão, a política de selecção foi excluir uma coluna completa de recursos a cada três colunas. Para alcançar a confiabilidade desejada, o processo de exclusão é feito sem nunca excluir uma coluna com a mesma posição relativa, que outra já excluída noutra RP, tal como é ilustrado na Figura 1.

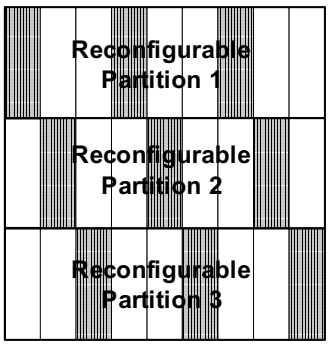


Figura 1. Distribuição de Recursos nas RPs do TMR

Com esta distribuição, mesmo que um módulo sofra uma falta permanente, é possível trocar a RP com outro módulo que

não usa o mesmo recurso onde ocorreu a falta permanente. Esta restrição `CONFIG PROHIBIT` exclui recursos (CLB, DSP, BRAMs, etc) mas não impossibilita o uso dos blocos de roteamento associados. No entanto, não sendo os recursos utilizados, não existe roteamento para esses recursos, o que significa que os blocos de roteamento não são usados, ou são usados de uma maneira diferente.

À primeira vista, esta distribuição pode significar um aumento extra de 33% de recursos da RP, no entanto, num sistema implementado numa FPGA, devido ao congestionamento do roteamento, muitas vezes a taxa de utilização de recursos é inferior a 70%. Quando o roteamento é limitado ao tamanho de uma RP, este congestionamento é ainda maior. Portanto, é usado este facto para que na realidade seja possível realizar esta distribuição, sem ter um custo em hardware da FPGA.

C. Assisted Design Flow *(ADF) para o TMR Proposto*

A Figura 2 mostra o fluxo completo da implementação de um sistema em FPGA baseado em PB4MP. Tem como base o fluxo das normais ferramentas da Xilinx (versão 14.7 do ISE e PlanAhead) e requer apenas uma pequena intervenção do projectista na fase inicial da elaboração do hardware, tal como no processo IDF [12]. As restantes etapas são automáticas.

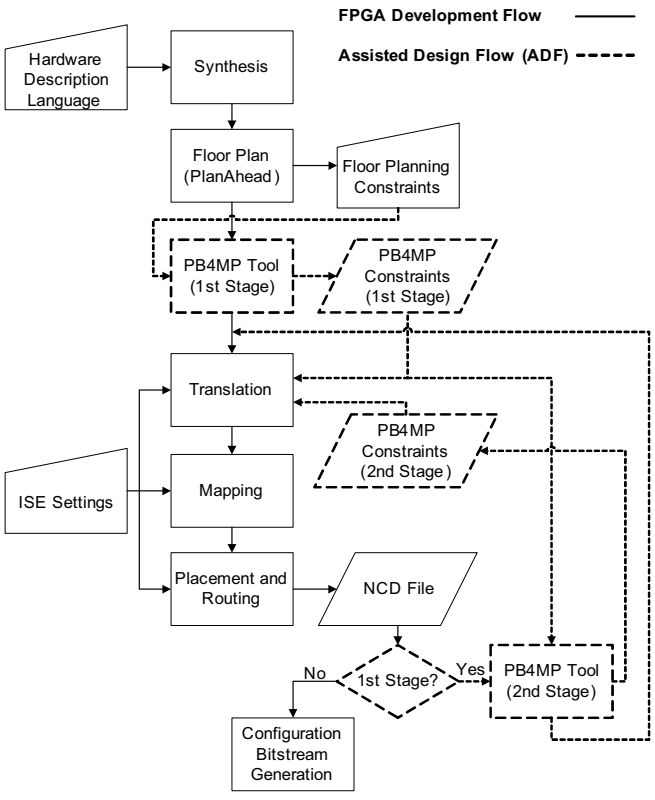


Figura 2. Fluxo de Desenvolvimento em FPGA + *Assisted Design Flow* (ADF)

A base do fluxo é a sequência *Synthesis* $\rightarrow$ *Floor Plan* $\rightarrow$ *Translation* $\rightarrow$ *Mapping* $\rightarrow$ *Placement and Routing* $\rightarrow$ *Configuration Bitstream Generation*. É importante que o *Hardware Description Language* (HDL) esteja estruturado de modo a que cada módulo seja implementado na RP desejada. Outro passo que necessita da atenção do projectista é o *Floor Plan*, onde este decide a localização dos recursos das RPs. É imperativo seguir as regras identificadas na Secção IV-B.

Após este ponto, a ferramenta PB4MP gera o primeiro segmento de restrições. Estas restrições serão responsáveis pela inclusão dos `SCC_BUFFER`. Nesta fase, a ferramenta define uma janela de possíveis localizações para os `SCC_BUFFER`. Esta é uma forma de dar alguma liberdade aos próximos passos para que a localização dos recursos da FPGA e o seu roteamento seja mais eficiente.

No final da primeira operação de *Placement and Routing*, um ficheiro NCD é gerado. A ferramenta PB4MP é então chamada de novo, e usando a ferramenta *Directed Routing Constraints* do `fpga_edline.exe`, é extraído o roteamento dos sinais que interconectam os módulos nas RPs ao resto do sistema. Com esta informação, é criado um segundo grupo de restrições e novamente é executada a sequência *Translation* $\rightarrow$ *Mapping* $\rightarrow$ *Placement and Routing*. No final dos dois estágios o *bitstream* é gerado e pode ser carregado na FPGA.

D. Memória Utilizada e Tempo de Execução

A memória utilizada e o tempo de execução do software usado na implementação desta capacidade de recuperação depende de vários detalhes da estratégia adoptada. A base da estratégia é sempre a mesma, a troca entre RPs, mas pode ser feita de duas maneiras diferentes:

- A configuração dos módulos fica no exterior da FPGA numa memória com os *bitstreams* parciais;
- A troca é realizada movendo a correspondente memória de configuração da FPGA.

A opção preferencial é a segunda porque não precisa de nenhum hardware extra. Mesmo assim, esta via pode exigir mais ou menos memória, ou tempo de execução. Se o OS tiver disponível uma considerável quantia de memória livre, é possível ler e escrever toda a memória de configuração correspondente. Mas se a memória livre for muito limitada, a opção é ler e escrever *frame* por *frame*. Neste trabalho a escolha foi esta última.

De modo a avaliar o consumo de memória e o tempo de execução, um grupo de equações foram definidas tendo em conta a Tabela I e a Tabela II. A Tabela I descreve todas as funções em software e a respectiva memória usada (compilado usando gcc para a arquitectura MIPS32), e os correspondentes tempos de execução para a família Virtex-6. A segunda coluna mostra a quantidade de memória requerida para a parte do código (M_{CODE}). A última coluna detalha todos os parâmetros que influenciam o tempo de execução indicado o número de ciclos de relógio necessários. A Tabela II descreve todos os parâmetros do hardware que influenciam o tempo de execução e a memória usada para dados (M_{DATA}).

A memória requerida é dividida em duas partes: código (M_{CODE} discriminado na Tabela I) e temporária para dados (M_{DATA}). A secção de dados é a memória necessária para guardar o endereço inicial das RPs, quatro bytes para cada uma, e o armazenamento temporário do conteúdo da memória de configuração da FPGA, durante a troca de RPs. No modo mais económico, onde a operação leitura/escrita é

Tabela I. ROTINAS EM SOFTWARE COM VALORES INDIVIDUAIS DE MEMÓRIA E TEMPOS DE EXECUÇÃO

Descrição da Rotina em Software	Memória (Bytes)	Execução (Ciclos)
RP_swap(rp1, rp2): Função que usa todas as restantes rotinas para implementar o algoritmo que troca duas RPs.	418	– – –
XHwICAP Base Driver: Base do driver usado para comunicar com o ICAP (inclui as funções Init(), SelfTest() e GetConfigReg()).	7012	– – –
XHwICAP DeviceReadFrame(): Rotina usada para ler um *frame* da memória de configuração da FPGA.	554	NC_{RF} (3720)
XHwICAP DeviceWriteFrame(): Rotina usada para escrever um *frame* na memória de configuração da FPGA.	784	NC_{WF} (4010)
Total M_{CODE}	**8768**	– – –

Tabela II. CARACTERÍSTICAS DAS PARTIÇÕES

Parâmetros da RP	Descrição dos Parâmetros dos Módulos em Hardware
$Size_{Frame}$	Número de bytes num *frame*.
N_{RPs}	Número de RPs no sistema.
$N_{RPFrames}$	Número de *frames* que possui a configuração de uma RP.
$Frequency$	Frequência do relógio do sistema (usado pelo CPU).

feita ao *frame*, é necessário o dobro do tamanho de um *frame* $Size_{Frame}$ (equação 1a). No caso mais exigente, quando a troca é feita num bloco só, é necessário o dobro do tamanho da configuração de memória de uma RP. Isto significa duas vezes o tamanho do *frame* $Size_{Frame}$ por cada um dos $N_{RPFrames}$ *frames* (equação 1b).

$$M_{DATA} = N_{RPs} \times 4 + Size_{Frame} \times 2 \quad (1a)$$

$$M_{DATA} = N_{RPs} \times 4 + Size_{Frame} \times N_{RPFrames} \times 2 \quad (1b)$$

A equação 2 calcula a quantidade de memória necessária para implementar a capacidade de recuperação desejada. Este total de memória final depende da família de FPGA, porque o tamanho do *frame* $Size_{Frame}$ varia, e também da quantidade de *frames* que inclui cada bloco usado em cada operação leitura/escrita.

$$MEM_{Total} = M_{CODE} + M_{DATA} \quad (2)$$

O número total de ciclos de relógio exigidos para executar uma troca completa entre duas RPs (NC_{Total}) é determinado por dois factores (3): família de FPGA, por causa do tamanho do *frame* $Size_{Frame}$, que influencia o número de ciclos consumidos pelas rotinas *XHwICAP DeviceReadFrame()* (NC_{RF}) e *XHwICAP DeviceWriteFrame()* (NC_{WF}); e o tamanho de cada RP $N_{RPFrames}$, que decide quantos *frames* precisam de ser lidos/escritos.

$$NC_{Total} = (NC_{RF} + NC_{WF}) \times N_{RPFrames} \times 2 \quad (3)$$

Após obter o número total de ciclos de relógio, com a equação 4 obtém-se o tempo requerido para trocar duas RPs em segundos (T_{Swap}).

$$T_{Swap} = \frac{NC_{Total}}{Frequency} \quad (4)$$

V. RESULTADOS EXPERIMENTAIS E SUA DISCUSSÃO

De modo a mostrar esta nova proposta, construiu-se um sistema na FPGA usando a mesma plataforma do trabalho [6], tal como ilustra a Figura 3. Esta é a nossa plataforma que suporta reconfiguração dinâmica e é implementada na FPGA XC6VLX240T da Xilinx.

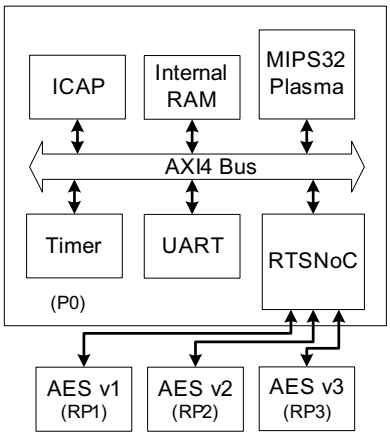


Figura 3. Diagrama do Sistema

É baseada numa implementação de um MIPS32, o *soft-core* Plasma, que é livre e encontra-se disponível no Opencores [18]. A estrutura interna de comunicação do CPU é baseada na família de protocolos AXI4, que se tornaram o standard na indústria para a interconexão baseada em *bus*. A reconfiguração do hardware é realizada pelo interface ICAP interligado ao AXI4, que é gerido pelo OS.

O OS que corre no *Central Processor Unit* (CPU) Plasma é o EPOS [7], que providencia o suporte necessário para implementar as necessidades do sistema e a comunicação com os restantes módulos.

RP1, RP2 e RP3 representam as RPs. Cada uma contém um componente *Advanced Encryption Standard* (AES), e as três versões do AES são implementadas seguindo a distribuição de recursos como a apresentada na Figura 1.

Foi escolhido um esquema de interconexão para RPs baseado na *Real-Time Star Network-on-Chip* (RTSNoC) [19]. Uma interconexão baseada num *bus* foi descartada pois não é a opção mais adequada para projectos mais heterogêneos em que os componentes de hardware têm papéis activos [20]. A RTSNoC consiste em *routers* com uma topologia em estrela que pode ser organizada de modo a formar uma malha 2-D. Cada *router* tem oito canais bidireccionais que podem ser conectados a componentes em hardware ou a canais de outros *routers*. Cada AES presente numa RP é conectado a uma porta de um *router* RTSNoC e um porto do *router* é conectado a uma ponte AXI4.

A síntese de cada módulo AES reportou a necessidade de 4172 LUT *Flip-Flop pairs*. Como na família Virtex-6, cada secção de coluna de CLBs possui 40 CLB, e cada um

possui 8 LUT *Flip-Flop pairs*, então cada secção possui um total de 240. Por isso são necessárias pelo menos 18 destas pequenas colunas. No entanto, devido ao congestionamento do roteamento durante a operação de *Placement and Routing*, foi necessário aumentar a disponibilidade de recursos para 27 colunas. Este incremento de recursos forçado permitiu-nos aplicar a desejada política de distribuição de recursos sem qualquer incremento de recursos.

O seguinte pseudo-código explica como funciona o algoritmo da nossa aplicação de demonstração.

```
void main_application(){
  ...
  while (1){
    ...
    send_data(&AES1, input_data);
    send_data(&AES2, input_data);
    send_data(&AES3, input_data);
    ...
    get_data(&AES1, output_data[1]);
    get_data(&AES2, output_data[2]);
    get_data(&AES3, output_data[3]);
    AES_error = vote_AES(output_data, result_out);

    if (AES_error) {
      switch () {
      case 1:
        if (last_AES_error == 1) RP_swap(&RP1, &RP3);
        else RP_swap(&RP1, &RP2);
        break;
      case 2:
        if (last_AES_error == 2) RP_swap(&RP2, &RP1);
        else RP_swap(&RP2, &RP3);
        break;
      case 3:
        if (last_AES_error == 3) RP_swap(&RP3, &RP2);
        else RP_swap(&RP3, &RP1);
        break;
      default:
        break;
      }
      last_AES_error = AES_error;
    }

    ...
  }
}
```

No algoritmo implementado no OS, os dados de entrada são enviados para os três módulos AES. Após o tempo de processamento necessário, todos os resultados são lidos dos mesmos módulos. Os três resultados são analisados por uma função, a qual se detectar alguma diferença em algum dos módulos retorna a indicação do módulo que falhar. Nesse cenário haverá uma troca de módulos entre RPs. Existindo um mecanismo complementar como o SEM, que salvaguarda o sistema dos SEU, com esta troca de módulos uma falta permanente será contornada, mesmo que ela só tenha provocado o aumento no tempo de resposta.

Todos os AES estão alocados numa RP e cada uma tem as características presentes na Tabela III.

Tabela III. CARACTERÍSTICAS DE CADA RP

RP Parâmetros	Valores dos Parâmetros
$Size_{Frame}$	324
N_{RPs}	3
$N_{RPFrames}$	1744 (1232 for CLBs + 512 for BRAMs)
$Frequency$	50MHz

Usando a equação 1a obtém-se $M_{DATA} = 660$ bytes, e com a equação 2 conclui-se que o TMR proposto precisa apenas de 9498 bytes de memória extra do sistema principal. Relativamente ao tempo de execução, para a troca entre duas RPs, pela equação 3, são necessários 13990600 ciclos de relógio. Recorrendo à equação 4, isto corresponde a 279,8 ms para trocar a alocação entre duas RPs. Embora seja um valor considerável, é algo pouco provável de acontecer.

Assumindo que os módulos AES utilizam 66% dos recursos de cada RP, é possível definir $Fail2 = \{$A área de recursos da FPGA incluídos nas três RPs sofreu duas faltas permanentes que fez o TMR falhar irremediavelmente$\}$, e calcular manualmente a probabilidade $\mathbb{P}(Fail2)$ para cada implementação de TMR presente na Tabela IV.

Tabela IV. COMPARAÇÃO DE TMRS

TMR Versão	$\mathbb{P}(Fail2)$	Hardware Extra
Standard TMR [1]	$\frac{13}{15} = 86,7\%$	0%
Montminy [8]	$\frac{9}{13} = 69,2\%$	PR *Overhead* + 25%
Com PB4MP	$\frac{2}{5} = 40\%$	PR *Overhead*

A primeira linha da Tabela IV é uma versão normal, sem qualquer mecanismo de recuperação. Na segunda, é uma versão do TMR proposto no trabalho [8] com apenas uma RP extra para o processo de realocação. Na última, é o trabalho proposto, e é de longe a opção que permite alcançar o melhor grau de disponibilidade, necessitando de menos de 10 Kbytes e o normal incremento de hardware associado à implementação de PR.

VI. CONCLUSÕES E TRABALHO FUTURO

O principal objectivo deste trabalho era propor um sistema com TMR utilizando a ferramenta PB4MP desenvolvida, seguindo como base o fluxo das ferramentas do ISE da Xilinx. A abordagem proposta aumenta consideravelmente a disponibilidade de um sistema com TMR com um incremento de memória muito limitado, e um aumento residual de hardware.

O PB4MP pode ser usado noutros sistemas onde seja importante expandir o seu tempo de vida. Sem um TMR será necessário prover os módulos com um mecanismo de teste como [10], para conseguir implementar uma política de FDIR.

Como trabalho futuro, tenciona-se portar o ADF-PB4MP para o fluxo do Vivado da Xilinx, e existe o desejo de conseguir aumentar o tempo de vida da FPGA através da redução do efeito de stress causado pelo NBTI.

AGRADECIMENTOS

Este trabalho foi financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

REFERÊNCIAS

[1] I. Koren and C. M. Krishina. *Fault-Tolerant System*, chapter 2-Hardware Fault Tolerance. Morgan Kaufmann, 2007.

[2] R. K. Mishra, A. Pandey, and S. Alam. Analysis and impacts of negative bias temperature instability (nbti). In *Proc. of the IEEE Students Conference on Electrical, Electronics and Computer Science (SCEECS)*, pages 1–4, Bhopal, India, March 2012.

[3] J. B. Velamala, V. Ravi, and Yu Cao. Failure diagnosis of asymmetric aging under nbti. In *Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 428–433, San Jose, USA, November 2011.

[4] A. Avizienis, Jean-Claude Laprie, B. Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. In *Proc. of the IEEE Transactions on Dependable and Secure Computing*, pages 11–33, January 2004.

[5] A. Guiotto, A. Martelli, and C. Paccagninis. Smart-fdir: use of artificial intelligence in the implementation of a satellite fdir. ftp://ftp.estec.esa.nl/pub/wm/anonymous/wme/Web/SmartFDIR2003.pdf, June 2003.

[6] Tiago R. Mück and Antônio A. Fröhlich. Seamless integration of hw/sw components in a hls-based soc design environment. In *Proc. of the International Symposium on Rapid System Prototyping (RSP)*, pages 109–115, Montreal, Canada, October 2013.

[7] The EPOS Project. Embedded parallel operating system. http://epos.lisha.ufsc.br, 2014.

[8] D. P. Montminy, R. O. Baldwin, P. D. Williams, and B. E. Mullins. Using relocatable bitstreams for fault tolerance. In *Proc. of the Adaptive Hardware and Systems (AHS 2007). NASA/ESA Conference*, pages 701–708, Edinburgh, Scotland, August 2007.

[9] V. Dumitriu and Lev Kirischian. Soc self-integration mechanism for dynamic reconfigurable systems based on collaborative macro-function units. In *Proc. of the International Conference on Reconfigurable Computing and FPGAs (ReConFig '13)*, pages 1–7, Cancun, Mexico, December 2013.

[10] Victor M. G. Martins, F. Ferlini, Djones V. Lettnin, and Eduardo A. Bezerra. Low cost fault detector guided by permanent faults at the end of fpgas life cycle. In *Proc. of the IEEE Latin America Test Workshop (LATW)*, pages 1–6, Fortaleza, Brasil, March 2014.

[11] Xilinx Inc. Partial reconfiguration tutorial: Planahead design tool v14.5, April 2013. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/PlanAhead_Tutorial_Partial_Reconfiguration.pdf.

[12] Xilinx Inc. Isolation design flow. http://www.xilinx.com/applications/isolation-design-flow.html, 2014.

[13] Y. Ichinomiya, S. Usagawa, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi. Designing flexible reconfigurable regions to relocate partial bitstreams. In *Proc. of the IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, page 241, Toronto, Canada, May 2012.

[14] Xilinx Inc. Logicore ip soft error mitigation controller user guide v3.1, October 2011. http://www.xilinx.com/support/documentation/ip_documentation/sem/v3_1/ug764_sem.pdf.

[15] Aeroflex Gaisler. Leon3 processor, December 2014. http://www.gaisler.com/index.php/products/processors/leon3.

[16] Xilinx Inc. Virtex-6 fpga configuration user guide v3.8 (ug360). http://www.xilinx.com/support/documentation/user_guides/ug360.pdf, August 2014.

[17] Xilinx Inc. Constraints guide v14.5 (ug625). http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/cgd.pdf, April 2013.

[18] OpenCores. Opencores. http://opencores.org, November 2014.

[19] Marcelo D. Berejuck. Dynamic Reconfiguration Support for FPGA-based Real-time Systems. Technical report, Federal University of Santa Catarina, Florianópolis, Brazil, 2011. PhD qualifying report.

[20] G. De Micheli, C. Seiculescu, S. Murali, L. Benini, F. Angiolini, and A. Pullini. Networks on chips: from research to products. In *Proc. of the 47th Design Automation Conference (DAC)*, pages 300–305, Anaheim, USA, June 2010.

 ISBN: 978-989-98875-2-7

A Universal Technique of Printing to VGA using FPGAS

Tiago M.A. Santos, Pedro L. Monteiro, V. Brito
Anikó Costa
tma.santos@campus.fct.unl.pt,
pm.monteiro@campus.fct.unl.pt, v.brito@campus.fct.unl.pt,
akc@fct.unl.pt

Abstract

This paper proposes to analyse one of the many concepts approached by FPGA. Hence, the purpose of this paper is to give an overview on what a FPGA is, their capabilities and interfaces such as VGA, and, finally, to present a solution using VHDL for resizing and printing on VGA through FPGA. Thus, the solution is based on a mathematical expression that can resize, to practically any scale, a character saved in the ROM block. VHDL was the main tool used to implement this algorithm since it presents the appropriate tools to fulfil the aforementioned objective.

1. Introduction

VHDL (Very High Speed Integrated Circuit Hardware Description Language) has been used for almost twenty years now, mostly as a broad language for modelling systems. These systems are implemented in a variety of circuit boards, many of which, non-reprogrammable, hence of a one-time use. There is no need to expand on how expensive this type of implementation is. Therefore, a need for more versatile, sometimes smaller, easier to transport, resulting in more educational, boards, urged. For example, FPGA (Field-Programmable Gate Array) based development kits from Digilent, the Basys2 board with Spartan-3E FPGA.

These development kits are adequate to use within digital system teaching. The board includes several interfaces, such as PS/2 or VGA (video graphics array) [1].

This type of boards are versatile enough to be used in digital system development, namely within digital system design teaching based on system modelling and its implementation using VHDL.

Taking into account the fact that this type of boards have a VGA port interface, it becomes adequate to use, as a way for students to exercise their knowledge of VHDL, by implementing a VGA module, which is used to print a character on the VGA screen. This is how FCT-UNL students deal with VHDL. To do so, they need some knowledge on VGA, the most known and widely used standard for graphics.

This paper proposes a simple and didactic algorithm to output a number of different characters, defined by the programmer, using 8x16 matrixes, through VGA using both VHDL and FPGA, along with some knowledge on VGA and ROM.

In section 2 some basic background knowledge on FPGA, VHDL and VGA is presented. In section 3 is explained how a character is represented in the ROM, followed by the explanation of how to print onto the VGA screen in section 4. In section 5, the implementation results are presented. In section 6, the current proposal is compared with previous works and, finally, in section 7 conclusions are drawn.

2. Technological Background

FPGA and Spartan3E

Field-Programmable Gate Array (FPGA) is an integrated semiconductor circuit designed to process digital data. The purpose of this equipment is to be programmed by either the consumer or the designer.

There are three main components which constitute a FPGA: Input and Output blocks (IOB), configurable logic blocks and Memory dedicated blocks. All these components are connected by matrixes.

IOB are the responsible for the interface and the outputs of the logic Blocks, which can be referred to as buffers that contain inputs or outputs of the FPGA.

Configurable logic blocks are composed by flip-flops and the use of combinatory logic, allowing the consumer or designer to make functional logical operations.

The memory blocks provide a way to allocate programs into the FPGA.

Lastly, the interconnected matrixes, the programmable ones, typically have the same width which enables the connection of the previous three mentioned [2].

In this paper, a specific FPGA will be focused (Spartan3 made by Xilinx Inc.). This particular board has, in addition to those mentioned before, three more available characteristics: PROM, Multiplier block and a Digital Clock Manager.

The PROM is a flash configurable memory block capable of storing data in a 2Mbit form. The Multiplier block receives two 18bit binary numbers and calculates the product. The Digital Clock Manager offers the possibility of self-calibration. With this block it is possible to distribute, delay, multiply, divide, and phase-shift clock signals by only using digital methods. The board also includes two SRAM 256K by 16k asynchronous blocks.

The Spartan3 is a compact board, offering a most decent network between all its components and multiple interfaces, thus being easy to use and learn [3].

VHDL

Hardware description languages (HDL's) are designed for describing the behaviour of physical devices and processes, a task commonly called modelling. Models written in a HDL are used as input to a suitable simulator to analyse the behaviour of such devices. HDL's have been used since the 1960's to model and simulate applications as diverse as digital and analogic electronic systems, fluid concentrations in chemical processes, and parachute jumps [4].

Modern HDL's support the description of both behaviour and structure characterization of the system. The structural mechanisms of an HDL allow a user to compose the model of a complete system from reusable model components stored in a library. Stored components are assembled into a design hierarchy that often closely resembles the decomposition of the system into subsystems and sub subsystems. The behavioural mechanisms of an HDL allow a user to express the operation of a subsystem at various levels of abstraction: very detailed, highly abstract, or anything in between. [4]

HDL's can be divided into digital, analogic, and mixed-signal HDL's, depending on the available language constructs. Digital HDL's, such as VHDL or Verilog, are based on event-driven techniques and a discrete model of time. They support the modelling of digital hardware at abstraction levels, from system level down to gate level [5].

VHDL has been recognized as a suitable language as it presents many useful features:

a) Ability to describe both the structure and behavior of a system in a unique syntactical framework;
b) Widespread use in digital design and inherent hierarchical abstraction description capabilities;
c) Recognition as a viable framework for developing high level models of digital systems (block diagrams, etc.), even before the decision between hardware or software decomposition of the functions takes place, and for supporting hybrid (i.e., mixed abstraction levels) simulation models;
d) Capability to support test activities [6].

VGA

Video Graphics Array (VGA) refers specifically to the display hardware standard first introduced with the IBM PS/2 line of computers in 1987, but given its widespread adoption throughout all the brands it has also come to mean either an analogic computer display standard, the 15-pin D-sub-miniature VGA connector or the 640x480 resolution itself.

VGA analogic interface is used for high definition video, including resolutions of 1080p and higher. While the transmission bandwidth of VGA is high enough to support even higher resolution playback, there can be picture quality degradation depending on cable quality and length. How discernible this degradation is depends on the individual's eyesight and the display [7].

VGA is referred to as an "Array" instead of an "adapter" because it was implemented from the start as a single chip—an application-specific integrated circuit which replaced both the Motorola 6845 video address generator as well as dozens of discrete logic chips that covered the full-length ISA boards of the MDA (monochrome display adapter, CGA (color graphics adapter), and EGA (extended graphics adapter. Its single-chip implementation allowed the VGA to be placed directly on a PC′s motherboard with a minimum of difficulty, since it only required video memory, timing crystals and an external RAMDAC (a buffered digital-to-analog converter). As a result, the first IBM PS/2 models were equipped with VGA on the motherboard, in contrast to all of the "family one" IBM PC desktop models—the PC, PC/XT, and PC AT — which required a display adapter installed in a slot in order to connect a monitor [8].

The intended standard value for the horizontal frequency of VGA is exactly the double the value used in the NTSC-M video system, as this made it much easier to offer optional TV-out solutions or external VGA-to-TV converter boxes at the time of VGA's development. The formula for the VGA

horizontal frequency is (60 ÷ 1001) × 525 kHz = 4500 ÷ 143 kHz ≈ 31.4685 kHz. All other frequencies used by the VGA card are derived from this value by integer multiplication or division. Since the exactness of quartz oscillators is limited, real cards will have slightly higher or lower frequency [8].

3. Defining Characters in the ROM

For the demonstration, it is used ROM arrays, which can store a limited set of characters, depending on how they are initialized. Through the next topics, some simple VHDL lines of code will be presented. In order to use them, some IEEE standard libraries must be used.

ROM (Read Only Memory), in VHDL, is defined as a STD_LOGIC_VECTOR constant, which can store a limited set of bits, depending on the amount of memory available.

It is used the character "#", represented in an 8x9 matrix, as shown in Fig. 1, which will later be enlarged vertically to an 8x16 matrix (128 bits), through the addition of blank lines to its top and bottom.

1	0	1	1	0	1	1	0	0	"6C"
2	0	1	1	0	1	1	0	0	"6C"
3	1	1	1	1	1	1	1	0	"FE"
4	0	1	1	0	1	1	0	0	"6C"
5	0	1	1	0	1	1	0	0	"6C"
6	0	1	1	0	1	1	0	0	"6C"
7	1	1	1	1	1	1	1	0	"FE"
8	0	1	1	0	1	1	0	0	"6C"
9	0	1	1	0	1	1	0	0	"6C"
	1	2	3	4	5	6	7	8	

Fig. 1. Matrix representation of the # character.

As it can be observed, each matrix's position is represented either by the bit 0 or 1, the latter being responsible for defining the character colour. After drawing the character, it must be codified into a STD_LOGIC_VECTOR constant, in which every matrix row is represented by a byte. To do so, one must translate every row into its hexadecimal representation, as shown in the highlighted first row of Fig. 1 (being the most significant bit the one on the left).

After that, every hexadecimal byte must be placed side by side, for example, the 8x9 '#' character is represented by "6C6CFE6C6C6CFE6C6C". As aforementioned, it is used 8x16 matrixes instead of 8x9. Hence, 3 blank lines on the top and 4 on the bottom of the character must be added. Thus, the character will be stored as X"0000006C6CFE6C6C6CFE6C6C00000000". The next code lines represent how the character must be stored in VHDL language.

```
   constant character:
STD_LOGIC_VECTOR( 0 to 8*16-1 ) :=
      x"0000006C6CFE6C6C6CFE6C6C000
00000";
```

4. From ROM to the Display

Now that a STD_LOGIC_VECTOR constant has been created, one must have a simple way to display it using the VGA interface.

Fig. 2 represents the block diagram of the display's architecture. The VGA controller is the module where the controller for the synchronization is implemented. Its outputs are v_sync and h_sync (the signals for the vertical and horizontal synchronization) and the current coordinates of the pixel to be printed. The Arithmetic Controller module processes this coordinates and chooses the colour in which to write them, through the use of the previously generated constant.

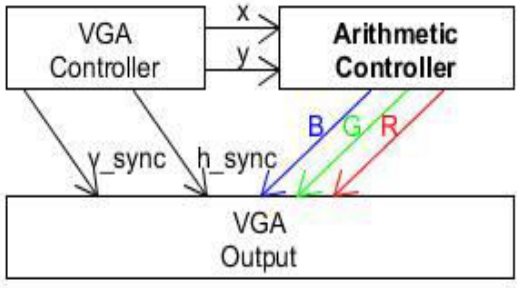


Fig. 2. Block diagram of VGA Interactions.

To simplify this process, was used the following mathematical equation:

$$F(x,y) = int\left(\frac{y-y_i}{k_y}\right) * 8 + int\left(\frac{x-x_i}{k_x}\right) \quad (1)$$

With the following initial conditions:

a) $y_i, x_i \in N_0$;
b) $y \in [y_i, y_i + 16k_y[$;
c) $x \in [x_i, x_i + 8k_x[$;
d) $k_y, k_x = \{ 2^k : k \in N_0 \}$.

Being x and y the current coordinates of the display; x_i and y_i the initial pixel coordinates (x_i, y_i) where one wishes to start displaying its character; k_x and k_y the factors by which we intend to expand the character; the function $int(a/b)$ represents the integer division of a by b.

If implemented correctly, following the initial conditions presented before, F will always give values inside the range of $[0,127]$. Hence, one can now know if the display should present a colour or not, by getting the value of the constant for the given $F(x,y)$ index ($\texttt{character}(F(x,y))$). If more than one character is stored in the constant, a simple trick can be added to the function F.

$$F(x,y) = int\left(\frac{y-y_i}{k_y}\right) * 8 + int\left(\frac{x-x_i}{k_x}\right) + 128N_i \quad (2)$$

a) $N_i \in N_0$;

Being N_i the index of the character to write.

5. Implementation Results

The proposed algorithm was implemented with the language VHDL, using Spantan-3E development boards, under the scope of the Digital Systems Design course, for master's degree students, held at FCT-UNL.

The following Fig. 3 shows the display of a set of characters, with different sizes and colours, using the algorithm proposed in this paper.

Fig. 3 Example output of printing to the display.

6. Related Work

Guohui Wang et. al [9] proposed a similar method, one which stored the character inside a matrix with different dimensions. Since he was working with characters from the Chinese alphabet, 8x16 matrixes weren't an option.

The proposed equation doesn't accommodate any other alphabet than the Latin. Although, it can be easily modified to store characters in different sized matrixes, in order to use other alphabets.

In their work, a string of characters is presented, each with a one-pixel width, written using a simple pointer that ran through the stored matrix. Thus, their work doesn't focus as thoroughly on the displaying part.

Despite all the similarities between both works, the intention of this proposal is not to overcome theirs but to note that the method presented here not only approaches the storage mechanism, but also gives an edge on printing the character itself.

7. Conclusion

Printing one character to the display is one of the most basic tasks one can perform when giving the first steps learning VHDL. Since it isn't that much of a simple task to both VHDL and VGA beginners, it holds great pleasure to those who fulfil its first display output. We proposed a simple way, one that does not require large mathematical knowledge, and can be used in a variety of circumstances. Understanding the equations aforementioned, thus realizing how easy it is to move, resize and alter the character, just by changing one parameter of the equation, it is expected that one will realize how useful this paper is.

Hence, we hope this paper serves both new and more experienced students of the VHDL language and VGA itself, by providing a simple solution to a, sometimes, complex problem.

Acknowledgment

This work was partially financed by Portuguese Agency "FCT - Fundação para a Ciência e a Tecnologia" in the framework of project Petri-Rig PTDC/EEI-AUT/2641/2012

References

[1] S. O. Popescu, A. S. Gontean, and G. Budura, "BPSK system on Spartan 3E FPGA," in *2012 IEEE 10th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2012, pp. 301–306.

[2] R. Wiśniewski, *Synthesis of compositional microprogram control units for programmable devices*. University of Zielona Góra, 2009.

[3] P. P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*. John Wiley & Sons, 2008.

[4] E. Christen and K. Bakalar, "VHDL-AMS-a hardware description language for analog and mixed-signal applications," *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, vol. 46, no. 10, pp. 1263–1272, Oct. 1999.

[5] D. Pellerin and D. Taylor, "VHDL made easy," *CERN Document Server*, 1997. [Online]. Available: http://cds.cern.ch/record/384827. [Accessed: 15-Nov-2014].

[6] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: the MEFISTO tool," in *, Twenty-Fourth International Symposium on Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers*, 1994, pp. 66–75.

[7] B. P. Knapp and C. H. Kraft, "Method and apparatus for combining video images on a pixel basis," US5532714 A, 02-Jul-1996.

[8] R. L. Mansfield, A. K. Spencer, and J. C. S. Clair, "Graphics function controller for a high performance video display system," US4916301 A, 10-Apr-1990.

[9] G. Wang, Y. Guan, and Y. Zhang, "Designing of VGA Character String Display Module Based on FPGA," in *2009 International Symposium on Intelligent Ubiquitous Computing and Education*, 2009, pp. 499–502.

An FPGA-embedded oscilloscope based on the IEEE1451.0 Std.

Ricardo J. Costa[1], Diogo Eloi Pinho[1], Gustavo R. Alves[1] and Mário Zenha-Rela[2]
ISEP/CIETI/LABORIS[1] FCTUC/CISUC[2]
rjc@isep.ipp.pt, diogoeloi@hotmail.com, gca@isep.ipp.pt, mzrela@dei.uc.pt

Abstract

Digital oscilloscopes are adopted in several areas of knowledge, in particular in electrical engineering, since they are fundamental for measuring and classifying electrical signals. Thanks to the proliferation of Field Programmable Gate Arrays (FPGAs), embedded instruments are currently an alternative solution to stand-alone and modular instruments, traditionally available in the laboratories. High performance, low cost and the huge flexibility to change functional characteristics, make embedded instruments an emerging solution for conducting electrical experiments. This paper describes the project and the implementation of a digital oscilloscope embedded in a FPGA. In order to facilitate their control, an innovative architecture is defined according to the IEEE1451.0 Std., which is typically used to develop the denominated smart transducers.

Keywords: Embedded instrumentation, FPGA, IEEE1451.0 Std., Oscilloscope.

1. Introduction

In the last decades technology evolution has been changing the adopted instrumentation in electrical engineering laboratories. At the beginning analogue instruments were the only available solutions to measure or generate electrical signals. These instruments were typically large and heavy, providing a reduced number of features, which difficult a more detailed analysis of the electrical signals. It was difficult to observe particular phenomena and to get specific signals' characteristics, e.g. overshoots, average or rout mean square values, etc. Visual observations supported, when possible, with pen-and paper calculations were required.

Since the appearance of digital processors, new digital processing techniques emerged. The old analogue instruments are being replaced by digital ones, with all the signals defined in the digital domain. Nowadays, digital instrumentation allows gathering measurements in digital formats, facilitating, this way, the use of personal computers to characterize the measured signals.

Instrumentation is a reality in every laboratory. Besides a division in analogue or digital instruments, these last can be divided in the three main groups represented in figure 1, namely: i) stand-alone; ii) modular and iii) embedded.

Stand-alone instruments can accommodate both digital and analogue instruments, and have the entire architecture in a unique chassis that provides buttons and interfaces to control or observe the signals. Modular instruments are digital and they are mainly supported by a computer that controls dedicated hardware. The well none PXI system[1] can be integrated in this group, since the instruments (also named as virtual instruments) are provided by slot cards controlled by a computer, sometimes available in the same chassis. Embedded instruments also accommodate digital instruments and they can be seen as an alternative solution to stand-alone or modular. They comprise circuits implemented within chips for performing specific validation, test and debug functions of other electronic circuits in the same chip or circuit boards [1]. They are classified as a most cost-effective and flexible solution, since they are essentially supported by small devices such as µC, µP, FPAAs or FPGAs[2], this last well implemented in the market (Xilinx, Altera, etc.) and able to be easily reconfigured according to the measurement requirements of a particular circuit.

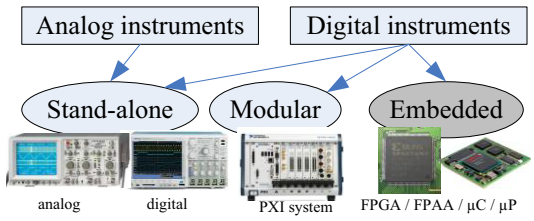


Figure 1: A possible classification of instruments.

It is precisely the enumerated advantages of embedded instrumentation with the flexibility provided by FPGAs, proved by the many projects

[1] PCI eXtensions for Instrumentation (PXI) is a modular instrumentation platform supported by the PXI Systems Alliance (http://www.pxisa.org/)

[2] µC (µController); µP (µProcessor); FPAA (Field Programmable Analog Array); FPGA (Field Programmable Gate Arrays).

using this type of devices (e.g. [2][3][4]), that incentivized the development of a digital oscilloscope embedded in a FPGA [5]. To complement some of the current research that is focusing on using the JTAG[3] interface, recently improved with the Internal JTAG interface (IJTAG)[4], to enable non-intrusive access and control of the embedded instruments, the described solution suggests the use of the IEEE1451.0 Std. A basic oscilloscope was therefore implemented in the core of an FPGA, following an architecture based on the IEEE1451.0 Std. The architecture uses a data structure named Transducer Electronic Data Sheet (TEDS) for describing and controlling all the features of the oscilloscope, and provides a set of IEEE1451.0 commands to enable its standard access and control.

Besides this introductory section, this paper has 5 other sections. Section 2 provides a brief overview about digital oscilloscopes, presenting their most common architectures. Section 3 suggests the use of FPGAs and the adoption of the IEEE1451.0 Std. for designing and accessing embedded instruments. Section 4 presents an implementation of a digital oscilloscope using an FPGA, and an architecture defined according to some issues of the IEEE1451.0 Std. Section 5 presents the verifications made to the implemented oscilloscope. Section 6 finalizes the paper with some conclusions.

2. Digital oscilloscopes

Digital oscilloscopes can be divided in: i) Mixed Signal Oscilloscopes (MSOs), ii) digital sampling oscilloscopes; iii) Digital Storage Oscilloscopes (DSOs) or; iv) Digital Phosphorus Oscilloscopes (DPOs) [7][8]. Since the last two are the most common in the market, it is reasonable to take a particular attention to them. As illustrated in figure 2, they use very similar architectures, differing in the adoption of a digital phosphor block.

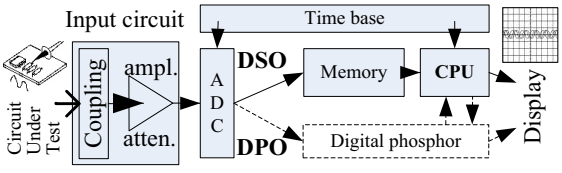


Figure 2: Block diagram of a digital oscilloscope according to the DSO and DPO architectures.

DSOs adopt serial architectures comprising an input circuit to acquire and handle analogue signals acquired from a Circuit Under Test (CUT). The input circuit is essentially implemented to remove (or not) the DC component, and to amplify/attenuate the amplitude of the measured signals according to the requirements defined by a particular user. The signals are then converted to the digital domain using an Analogue Digital Converter (ADC) using a sampling rate controlled by a time base block, which is defined by the users to adequately represent all signals in a display. Before that visual representation, the generated samples are processed using a Central Processing Unit (CPU) and gathered in a memory block, also controlled according to the defined time base. The CPU is the responsible to generate the samples of the measured signal to be visualized in the display.

DPOs follow the same architecture, but comprise an additional parallel block named digital phosphor. This block is essentially used to facilitate displaying the samples, freeing up the memory and the CPU for other computational tasks, namely for running specific digital calculations (digital filtering, interpolations, triggering, etc.). In comparison to the DSOs, this type of architecture brings additional advantages to the signal representation. It allows managing the intensity of all samples, providing more information about the signal, and provides higher bandwidth, since the sampling rate is faster, which facilitates the capture of sporadic and very fast events on a signal (e.g. signal glitches).

Based on the simpler DSO architecture, a digital oscilloscope described for an FPGA and defined according to some issues of the IEEE1451.0 Std., can be implemented as an embedded and "intelligent" oscilloscope.

3. Using FPGAs and the IEEE1451.0 for designing embedded instruments

The processing required to handle samples in digital instruments, in particular in an oscilloscope, can be implemented by several devices, such as µP, µC, FPAAs or FPGAs. Despite the last two allow the reconfiguration of hardware, FPAAs still have a limited number of analogue components in their core, not providing the same flexibility as FPGAs that comprise many digital blocks. Therefore, FPGAs are preferable for implementing part or the entire architecture of one or more instruments, since they are hardware reconfigurable using standard Hardware Description Languages (HDLs), and they can run several hardware blocks in parallel. This parallelism offered by FPGAs simplifies the implementation of specific digital algorithms required to handle the samples of a particular

[3] Joint Test Action Group (JTAG) is a common name for what later became the IEEE1149.1 Std. That stands for Standard Test Access Port and Boundary-Scan Architecture [6].

[4] Internal Joint Test Action Group (IJTAG) is an interface standard to instruments embedded in chips that defines a methodology for their access, automating their operations and analyzing their outputs. It is currently defined by the IEEEp1687 Std. (http://standards.ieee.org/develop/wg/IJTAG.html).

oscilloscope, and allows running more than one instrument in the same core.

Despite most of the processing is made in the digital domain (e.g. interpolation techniques, implementation of filters, etc.), providing flexibility in the design of a particular instrument, they must provide interfaces to the analogue domain. Therefore, analogue to digital conversions are required, as well as power drivers' interfaces to interconnect the CUT. For this purpose, the use of FPGA-based boards for accommodating the entire architecture of an oscilloscope, or any other instrument, is an interesting solution. The core of the instrument can be embedded into an FPGA, while other issues, such as memories, the amplification/attenuation of signals, among others, can be supported by the surrounding devices available on those boards, as represented in figure 3.

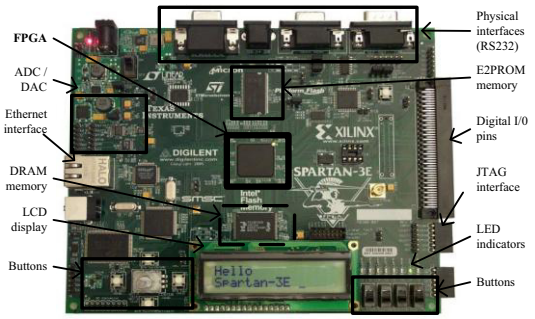


Figure 3: Example of a typical FPGA-based board.

Since the core of an FPGA is reconfigurable, it is easy to change the functionalities of a particular instrument by reconfiguring the hardware modules described in HDL. Unlike the use of µPs or µCs, whose programming depends on the assembly language provided by the manufacturer, FPGAs can use the same HDL code to reconfigure different instruments. Despite Verilog and VHDL are standard HDLs, they do not guarantee the standard access of a particular instrument embedded into an FPGA. For this purpose, using the IEEE1451.0 Std. is a solution, since it describes a particular architecture to manage the so-called smart transducers, whose architecture can be adopted to design an embedded instrument, and, in particular, an oscilloscope.

Defined in 2007, the IEEE1451.0 Std. [9][10] aims to network-interface transducers (sensors and actuators) and defines a set of operating modes, based on specifications provided by Transducer Electronic Data Sheets (TEDSs). This standard specifies operating modes controlled using commands that can be applied through a set of APIs. As represented in figure 4, it defines an architecture based on two modules that should be interconnected using an interface protocol: the Transducer Interface Module (TIM) and the Network Capable Application Processor (NCAP). This last provides remote access to the Transducer Channels (TCs) and to the TEDSs, both traditionally included in the TIM. A TIM may integrate several TCs (up to 65535), defined as the digital transducers. Each module is connected through an interface defined by another standard of the IEEE1451.x family, some already specified according to the IEEE1451.0 Std. (e.g. the IEEEp1451.6 Std. for the CANopen interface) and others intended to be modified in the future (e.g. IEEE1451.2 Std. which defines point-to-point interface).

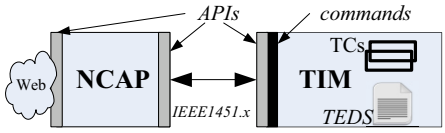


Figure 4: Overall architecture of the IEEE1451.0 Std.

Despite the relevance of the NCAP in the entire architecture, the TIM is the main module and the one that traditionally implements the hard-processing of any smart transducer. Therefore, after a careful analysis on the TIM's characteristics and on the requirements posed by a digital oscilloscope, it was decided to embed the oscilloscope within the FPGA as a TIM. The commands provided by the TIM allow the access and the control of the oscilloscope, whose definitions can be made through an internal TEDS constructed according to a particular structure. Using a simple computer interface can therefore enable the control of the oscilloscope and the representation of the measured signals.

4. The implemented oscilloscope

To prove the possibility of using the IEEE14151.0 Std. with FPGA devices to create an embedded instrument, a digital oscilloscope prototype was conceived and implemented according to a DSO architecture. Some specifications of the IEEE1451.0 Std. were adopted to define and control the entire behavior of the oscilloscope, namely the use of a TEDS and a set of commands. As illustrated in figure 5, the oscilloscope was implemented in an FPGA-based board from Xilinx (XC3S700AN starter kit) to acquire signals from a CUT (emulated by a traditional function generator), using an internal ADC. Through an RS-232 serial connection, the oscilloscope was attached to a computer running an interface developed in JAVA, represented in figure 6. This interface allows users to access and control all the functionalities of the oscilloscope.

The oscilloscope was implemented as the TC number 1 within a TIM running inside the FPGA. The TIM and the associated TC operate using modes defined in the IEEE1451.0 Std. The TIM can run in the *Initialization* or *active* modes, while the TC can run in the *Initialization*, *Inactive* or *Operational* modes. The transactions among the different operational modes are established through internal operations of the oscilloscope or through commands issued by the users.

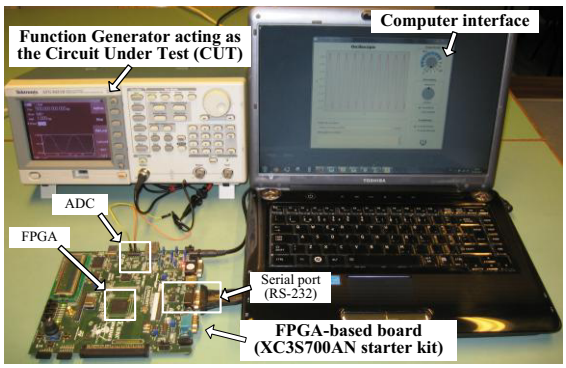


Figure 5: Picture of the implemented oscilloscope.

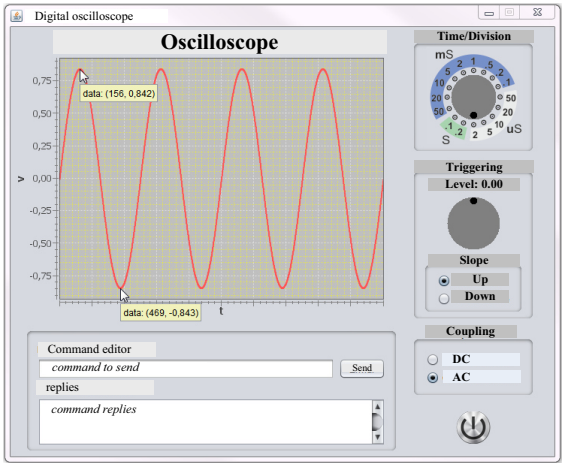


Figure 6: Computer interface used to control the digital oscilloscope.

The TIM is the main unit of the entire oscilloscope that was implemented in hardware blocks described in the Verilog HDL. As illustrated in the diagram of figure 7, it is divided three main modules: i) a Central Processing Unit (CPU); ii) a reformulated TC-TEDS and; iii) a Communication Module (CM).

The CPU is the unit that handles all samples used to display in the computer interface the measured signal. As represented in figure 8, after sampling the signal, the CPU defines the coupling mode (AC or DC) and implements interpolations and triggering methods to fill-in a data set with 2500 samples used to represent the signal. The number of samples was empirically selected, in order to get a good visual representation of the signal.

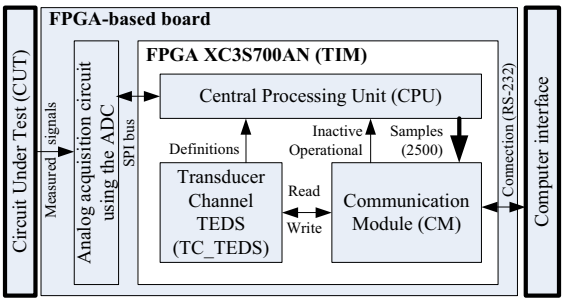


Figure 7: Architecture of the implemented oscilloscope.

The selected hardware to acquire the signals, namely the ADC and the FPGA maximum operation frequencies, limited the sampling rate up to approximately 1.5MHz[5], and to a bandwidth of 575 kHz calculated base on a step response of the oscilloscope, as it will be described in the next section 5 of this paper. Additionally, the amplitude of the measured signal is limited, going from 0.4 V up to 2.9 V due to hardware constrains posed by the FPGA-based board.

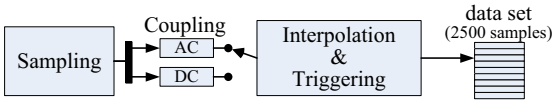


Figure 8: Picture of the implemented oscilloscope.

The coupling allows removing (or not) the continuous component of an analog signal by selecting the DC or AC modes. This functionality was implemented using a digital high pass band Finite Impulse Response (FIR) filter. This filter was projected using the Matlab software according to the Parks-McClellan algorithm[6] [11]. The filter was calculated with 318 coefficients with a normalized pass band frequency of 0.03; normalized cut-off frequency of 0; maximum deviation on the pass band of 0.001dB; and a minimum rejected band of -80dB. The normalized response of the implemented filter is represented in figure 9.

[5] The ADC available in the FPGA-based board is the LTC 1407A-1, which requires a minimum of 34 clock cycles to acquire a sample. Since it is controlled by the FPGA available in the board, that runs using a digital clock of 20ns, the maximum acquisition frequency (fa) is approximately 1.5 MHz [fa_{max}=1/(20ns x 34)=1.47MHz].

[6] http://www.mathworks.com/help/signal/ref/firpm.html; http://www.mathworks.com/help/signal/ref/firpmord.html

 ISBN: 978-989-98875-2-7

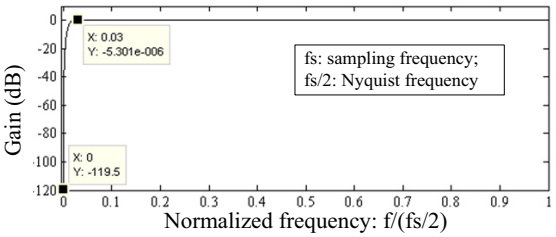


Figure 9: Response of the implemented filter to cut-off the direct current of the measured signal.

Independently of the selected coupling mode (AC or DC), the remaining circuit must generate 2500 samples to represent the measured signal in the computer interface. Those samples are generated by the CPU using an interpolation factor (L) and by the level and slope defined for triggering the signal. The interpolation factor is calculated according to several interrelated parameters defined in table 1, namely: i) the measured time (mt), which may be seen as a time window used to capture a signal defined by the user in the computer interface through the time/division knob button; ii) the number of acquired samples (as), internally determined by the CPU; iii) the selected sampling frequency of the ADC (fs), and; iv) the required frequency (fr) to get the indicated 2500 samples. For better understanding the entire generation process of the 2500 samples, figure 10 and equations 1, 2 and 3 represent these variables and their inter-reliance.

Table 1: Parameters that influence the process of filling-in the data set with 2500 samples.

Measure time (mt)	Required freq. (fr)	Samp. freq. (fs)	Samples acq. (as)	Interpol. factor (L)
20 µs	125 MHz	1.25 MHz	25	100
50 µs	50 MHz	1.25 MHz	62.5	40
100 µs	25 MHz	1.25 MHz	125	20
200 µs	12.5 MHz	1.25 MHz	250	10
500 µs	5 MHz	500 kHz	250	10
1 ms	2.5 MHz	250 kHz	250	10
2 ms	1.25 MHz	125 kHz	250	10
5 ms	500 kHz	50 kHz	250	10
10 ms	250 kHz	25 kHz	250	10
20 ms	125 kHz	12.5 kHz	250	10
50 ms	50 kHz	5 kHz	250	10
100 ms	25 kHz	2.5 kHz	250	10
200 ms	12.5 kHz	1.25 kHz	250	10
500 ms	5 kHz	500 Hz	250	10
1 s	2.5 kHz	250 Hz	250	10
2 s	1.25 kHz	125 Hz	250	10

The interpolation was simulated in Matlab and implemented using Cascaded Integrator Comb (CIC) interpolators balanced with FIR filters [12][13]. The result was two interpolation factors (L_{FIR} and L_{CIC}) whose product results in the referred L factor previously indicated to calculate the required frequency (fr) to generate the 2500 samples. The selected triggering is used to define the time when the first sample is acquired in the time window defined by the mt parameter.

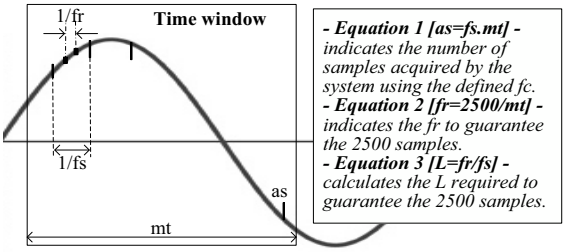


Figure 10: Example of sampling and interpolating a signal and associated equations.

Besides all these digital processing methods that may be encountered in any digital oscilloscope, the innovation of the implemented solution focuses on the use of the IEEE1451.0 Std. Therefore, to enable users to get real time information about the oscilloscope, and to control its functionalities, a TEDS was defined and implemented within a memory block of the FPGA. This TEDS was designed based on the structure of a TC TEDS defined by the IEEE1451.0 Std. It is divided in different blocks and fields of eight bits (octets), organized according to a Type-Length-Value (TLV) structure, as represented in figure 11. The TEDS's structure comprises three distinct blocks: i) length (2 octets), indicating the number of octets available in the remaining blocks; ii) data block, providing the main information about the oscilloscope in 12 TLV structures with a total of 107 octets and; iii) a checksum field used to verify the data integrity.

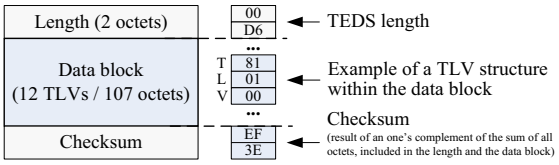


Figure 11: Implemented TEDS structure.

Each TLV indicates different issues, such as: i) defines the oscilloscope as a sensor; ii) the units retrieved from the oscilloscope (volts); iii) the minimum and maximum amplitude of a signal able to measure (0.4 V to 2.9 V); iv) measurement resolution (0.15 mV defined according to the ADC); v) the number of samples available in the data set (2500) and their representation format (float - 4 octets); vi) trigger level and slope; vii) the way samples are transmitted to the computer interface, which was defined in an *immediate mode*, indicating that the samples in the data set are immediately transmitted to the computer when it is full and; viii) scale definitions for representing the samples in the

computer interface (the size of the provided display window).

Using the information provided within the TEDS data block, the CPU controls all the oscilloscope operations. When in the *active mode* it reads the TEDS's fields and, based on its values, controls all the operations of the oscilloscope to fill-in the data set with the 2500 samples. When it is filled-in, all samples are sent to the computer interface to represent the measured signal, according to a specific field defined in the TEDS that indicates the adoption of an immediate transmission of all data to the computer.

Therefore, the control and characteristics of the oscilloscope are defined within the TEDS's fields and managed by the computer interface through some IEEE1451.0 commands. These commands, and associated replies, are defined according to the IEEE1451.0 standard message structures divided in the octets illustrated in figure 12.

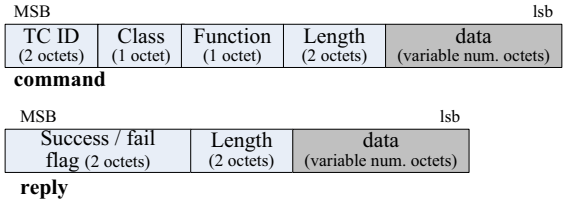


Figure 12: Message structures used for commands and associated replies.

As discriminated in table 2, four commands were implemented: i) *TCoperate*; ii) *TCidle*; iii) *readTEDS* and; iv) *writeTEDS*. All these commands follow the same structure defined in the standard. The first two are adopted to place the oscilloscope in the *operation* or *idle* modes, which means turning it on or off. The *read/write TEDS* commands are adopted to control the oscilloscope. They are sent through the computer interface when a user changes a specific button or when they manually define a particular command using the command editor. Every issued command to the TIM generates a particular reply described in table 3, indicating a successful or a faulty operation. The replies generated follow the same structure of the IEEE1451.0 Std., but most of them add particular codes´ responses, enumerated in table 4, for better management of the oscilloscope using the computer interface. Internally, it is the CM that decodes commands received from the computer interface and creates and sends the associated replies. This way, it is possible to control the oscilloscope in a standardized way (using IEEE1451.0-based commands) and to provide information about it.

Table 2: Implemented commands' data structures.

command	TC	class	func.	length	data	
					TEDS	data
TCoperate	01	04	01	00.00	-	
TCidle	01	04	02	00.00	-	
ReadTEDS	01	01	02	00.05	03	p
WriteTEDS	01	01	03	Δ	03	p+d

Δ-variable length; p- offset position (octet location to read/write from/to the TEDS); d-data (data to write in the TEDS).

All the functionalities of the oscilloscope and some specific aspects were verified using a particular scenario that includes monitoring the messages and associated replies between the computer interface and the TIM, and measuring some representative signals, namely a step signal and some sinusoidal waves with different frequencies.

Table 3: Data structures of the commands' replies and the data transmitted to the computer interface.

Response to	Succe. (01) Fail (00)	Length (2 octets)	Data response
Error message	00	00.01	C[00-06]
WriteTEDS; TCoperate/idle	01	00.01	C07
ReadTEDS	01	Δ	C08 + Δd
Data sent to the computer interface	01	27.10 (hex)	10000*

Δ-variable length of the retrieved TEDS (depends on the op defined in table 2); Δd-data retrieved from the TEDS, Cx- indicates reply codes indicated in table 4; *in float format (4 octets per each sample, totalizing 10000/4=2500 samples).

Table 4: Codes' responses to issued commands.

Lost message header	C00
Invalid TC	C01
Unknown class/function ID	C02
Lost command	C03
Unknown TEDS ID	C04
Invalid TEDS position	C05
Checksum error	C06
Successful write operation	C07
Read response	C08

(note: the values in the table are in the hexadecimal format)

5. Verification of the oscilloscope

To verify the correct operation of the implemented oscilloscope, a scenario was defined according to the schematic of figure 13. It uses the implemented oscilloscope (FPGA-based board connected to a computer), and two stand-alone instruments, namely a function generator[7] and a digital oscilloscope[8]. The function generator was adopted to emulate a common CUT, generating different signals able to be measured by the implemented oscilloscope. The stand-alone oscilloscope was adopted for comparison purposes, i.e. to verify if the implemented oscilloscope behaves in a similar way as the commercial one.

[7] AFG 3021B from Tektronix.
[8] TDS 3114b from Tektronix

The correct operation and some characteristics of the implemented oscilloscope was verified in different stages, namely: i) by monitoring commands exchanged between the FPGA-based board and the computer interface using a serial monitor application installed in the computer and; ii) by measuring different signals generated by the stand-alone function generator.

Figure 14 exemplifies the observation made to the command messages and associated replies when a specific button is changed or a particular command is edited in the computer interface (in the particular case the *WriteTEDS* command).

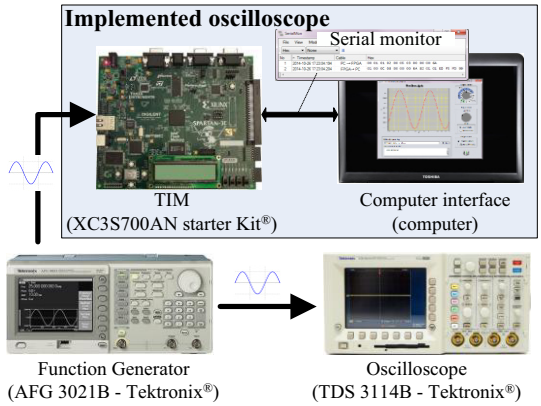


Figure 13: Adopted scenario to verify the oscilloscope operation.

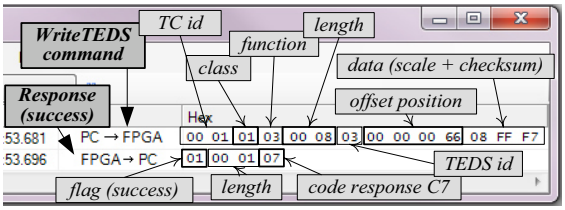


Figure 14: Example of a WriteTEDS command and associated reply monitored by the serial monitor.

To get particular characteristics of the oscilloscope, in particular its bandwidth, a step signal was applied and analyzed by the oscilloscope, as represented in figure 15. The step was defined with 1V of amplitude (from 1V to 2V) and a rise time of 1.8ns. A scale of 2µs that is associated to the scale of 20 µs in table 1[9], was selected in the control interface, which means a sampling frequency of 125MHz (fr). The time between two consecutive samples is therefore 8ns, which means a rising time (tr) of 0.608µs that defines a bandwidth of 575kHz, calculated according to the indicated equation retrieved from [14].

[9] Each time/division in the computer interface is associated to the scales indicated in table 1 attenuated by a factor of 10.

While the step was essentially adopted to understand the dynamic response of the oscilloscope, sinusoidal signals were also applied to both oscilloscopes for comparison purposes. It was selected three sinusoidal signals with different amplitudes, frequencies and DC components, as detailed in table 5. The range values of the selected signals were defined according to the characteristics of the implemented oscilloscope, namely the bandwidth and the amplitudes it can measure.

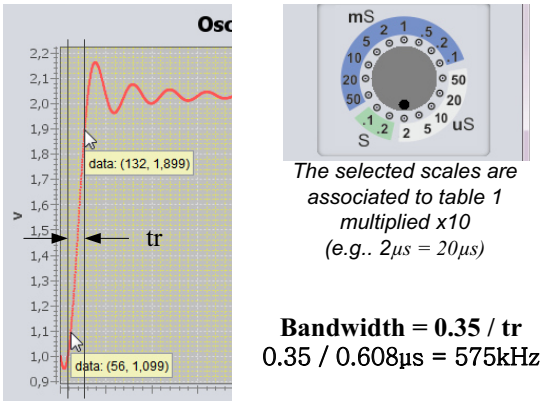


Figure 15: Step response of the oscilloscope.

Table 5: Sinusoidal signals applied to the oscilloscopes.

frequency	amplitude	DC component
500kHz	2V	1.65V
100kHz	1V	1V
1Hz	2.4V	1.65V

A detailed comparison of all signals in both oscilloscopes' displays was made, leading to conclude the correct operation of the implemented oscilloscope, since all signals were represented similarly in both displays, as exemplified in figure 16 for the signal with 100kHz.

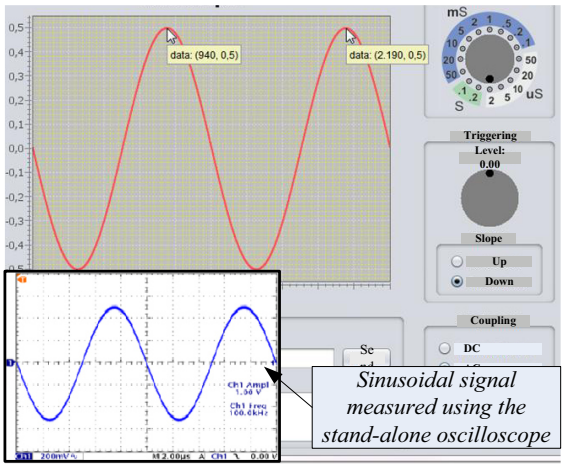


Figure 16: Example of a measurement of the sinusoidal signal with 100kHz with the implemented and the stand-alone oscilloscopes.

6. Conclusions

The technology evolution of the last two decades and the emergence of new and sustainable digital processing techniques, incentivized a shift on the type of adopted instruments in any laboratory. The old analogue instruments are being replaced by digital ones, since they provide more reliable results and offer the possibility of handling data using personal computers. The traditional and modular instrumentation are very common today, providing different types of computer interfaces. However, their adoption to measure/generate particular signals from/to a CUT can be prohibitive, due to development costs. The evolution of small devices, namely μP, μC, FPAAs and in particular FPGAs, are incentivizing the appearance of the so-called embedded instrumentation. Despite these instruments are essentially adopted to test internal circuits within the same board or chip, they can also be seen as traditional instruments able to control using a personal computer. The advantage is the possibility of easily reconfiguring their functionalities without changing the entire platform. This way, the proliferation of FPGAs in the market, and the capability of reconfiguring their internal hardware blocks with modules developed using standard HDLs, such as Verilog or VHDL, make this type of devices the most appropriated for developing embedded instrumentation.

The project described in this paper proposed the use of a FPGA to embed a digital oscilloscope described in Verilog and externally controlled through a personal computer. Additionally, since there is a lack of a standard for controlling and accessing a similar solution, it was suggested the use of some issues of the IEEE1451.0 Std., namely a TEDS and a set of commands. Although the implemented oscilloscope is a prototype, which would require some improvements concerning its robust operation, the developments described and the obtained results, led to conclude the real advantage of using the IEEE1451.0 Std. and FPGA technology for developing embedded instruments able to be externally controlled as traditional stand-alone or modular instruments. The use of standard commands to control their operation and to get real-time information from their associated TEDS, was seen as an interesting and promising solution to take into consideration for the development of other instruments embedded in FPGAs. The design of embedded instruments based on reconfigurable technology, such as FPGAs, and supported on the IEEE1451.0 Std., may contribute for the proliferation of those instruments to run the traditional experiments in laboratories, replacing the traditional stand-alone and modular instrumentation.

Note: This paper resumes a MSc. thesis submitted by the second author to the Polytechnic Institute of Porto – School of Engineering (IPP/ISEP) - Portugal.

References

[1] Frost & Sullivan - White Paper, "Embedded Instrumentation Its Importance and Adoption in the Test & Measurement Marketplace." http://www.frost.com, 12-May-2012.

[2] Fernando Pereira, Luís Gomes and Luís Redondo, "FPGA Controller for Power Converters with integrated Oscilloscope and Graphical User Interface," in *Proceedings of the 2011 International Conference on Power Engineering, Energy and Electrical Drives*, 2014, p. 6.

[3] B.Hemalatha, V.R.Ravi, S.Divya and M.Uma, "Embedded FPGA Controller for Robot Arm in Material Handling Using Reconfigurable NICompact RIO," in *2 nd International Conference on Current Trends in Engin eering and Technology, ICCTET'14*, 2014, p. 7.

[4] Fernando Machado et al., "FPGA-Based Instrument for Satellite Beacon Monitoring on Propagation Experiments," *IEEE Trans. Instrum. Meas.*, vol. 63, no. 12, p. 10, Dec. 2014.

[5] Diogo Pinho, "MSc. Thesis - Projeto e implementação de osciloscópio digital baseado na norma IEEE1451.0," *Inst. Super. Eng. Porto ISEP*, p. 152, 2014.

[6] IEEE1149.1-2013™, "IEEE Standard for Test Access Port and Boundary-Scan Architecture," *Inst. Electr. Electron. Eng. Inc*, p. 442, May 2013.

[7] A. Campilho, *Instrumentação electrónica: métodos e técnicas de medição, 1ª ed.* FEUP edições, 2000.

[8] Tektronix, *XYZs of Oscilloscopes*. available in: http://info.tek.com/rs/tektronix/images/03W_8605_6.pdf, 2011.

[9] IEEE Std. 1451.0™, "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats," *Inst. Electr. Electron. Eng. Inc*, p. 335, Sep. 2007.

[10] Eugene Y. Song and Kang Lee, "Understanding IEEE 1451-Networked smart transducer interface standard What is a smart transducer," *IEEE Instrum. Meas. Mag.*, pp. 11–17, Apr. 2008.

[11] J. H. McClellan and T. W. Parks, "A personal history of the Parks-McClellan algorithm," *IEEE Signal Process. Mag.*, vol. 22, no. 2, pp. 82–86, Mar. 2005.

[12] L. Milic, *Multirate Filtering for Digital Signal Processing: MATLAB Applications*. IGI Global, 2009.

[13] E. Hogenauer, "An economical class of digital filters for decimation and interpolation," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 29, no. 2, pp. 155–162, Apr. 1981.

[14] Tektronix, "Technical brief: Understanding Oscilloscope Bandwidth, Rise Time and Signal Fidelity." 2008.

 ISBN: 978-989-98875-2-7