Dimitrios Theodoropoulos

Custom Architecture for Immersive-Audio Applications

Custom Architecture for Immersive-Audio Applications

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op maandag 9 mei 2011 om 10:00 uur

door

Dimitrios THEODOROPOULOS

Master of Science in Computer Engineering
Technical University of Crete
geboren te Athene, Griekenland

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. H.J. Sips

Copromotor:
Dr. G.K. Kuzmanov

Samenstelling promotiecommissie:

Rector Magnificus,	voorzitter
Prof. dr. ir. H.J. Sips,	Technische Universiteit Delft, promotor
Dr. G.K. Kuzmanov,	Technische Universiteit Delft, copromotor
Prof. dr. W. Najjar,	University of California Riverside, USA
Prof. dr. D. Pnevmatikatos,	Technical University of Crete, GR
Prof. dr. E. Charbon,	Technische Universiteit Delft
Dr. ir. D. de Vries,	Technische Universiteit Delft
Dr. ir. G. N. Gaydadjiev,	Technische Universiteit Delft
Prof. dr. ir. P. M. Sarro,	Technische Universiteit Delft, reservelid

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Dimitrios Theodoropoulos

Custom Architecture for Immersive-Audio Applications

Delft: TU Delft, Faculty of Elektrotechniek, Wiskunde en Informatica - III

Ph.D. Thesis Technische Universiteit Delft.

Met samenvatting in het Nederlands.

ISBN 978-90-72298-16-4

Subject headings: reconfigurable, immersive-audio, GPGPU, multi-core processors.

Printed in The Netherlands

To my family back home...

Custom Architecture for Immersive-Audio Applications

Dimitrios Theodoropoulos

Abstract

In this dissertation, we propose a new approach for rapid development of multi-core immersive-audio systems. We study two popular immersive-audio techniques, namely the Beamforming and the Wave Field Synthesis (WFS). Beamforming utilizes microphone arrays to extract acoustic sources recorded in a noisy environment. WFS employs large loudspeaker arrays to render moving audio sources, thus providing outstanding audio perception and localization. Research on literature reveals that the majority of such experimental and commercial audio systems are based on standard PCs, due to their high-level programming support and potential of rapid system development. However, these approaches introduce performance bottlenecks, excessive power consumption and increased overall cost. Systems based on DSPs consume very low power, but performance is still limited. Custom-hardware solutions alleviate the aforementioned drawbacks, but designers primarily focus on performance optimization without providing a high-level interface for system control and test. To address the aforementioned problems, we propose a custom platform-independent architecture that supports immersive-audio technologies for high-quality sound acquisition and rendering. An important feature of the architecture is that it is based on a multi-core processing paradigm. This allows the design of scalable and reconfigurable micro-architectures, with respect to the available hardware resources, and customizable implementations targeting multi-core platforms. To evaluate our proposal we conducted two case studies: We implemented our architecture as a heterogeneous multi-core reconfigurable processor mapped onto FPGAs. Furthermore, we applied our architecture to a wide range of contemporary GPUs. Our approach combines the software flexibility of GPPs with the computational power of multi-core platforms. Results suggest that employing GPUs and FPGAs for building immersive-audio systems, leads to solutions that can achieve up to an order of magnitude improved performance and reduced power consumption, while also decrease the overall system cost, when compared to GPP-based approaches.

Preface

I still remember how it all started back in July 2006 when I was doing my military service at the Hellenic Air Force in Crete, Greece. It was during my midnight guarding shift, when my cell phone rung. Normally I was not supposed to pick it up, but a strange long number appeared on my phone's screen. It was my friend Christos Strydis from the Netherlands, who told me that soon there would be new Ph.D. positions available at the Computer Engineering laboratory of the Delft University of Technology, and encouraged me to apply. After a few days, I arranged to get an official permission and flew to the Netherlands to visit him. During this visit, I met for the first time Professor Dr. Stamatis Vassiliadis and had the one and only chat with him at his office. It didn't take long to convince me to apply...

Four and a half years later, where Professor Vassiliadis is not any more with us, still I would like to express my gratitude to him for accepting me as his Ph.D. student at the Computer Engineering laboratory. The fact that such great scientist gave me the opportunity to work at his group, always inspired and motivated me to push myself for the best.

The work presented in this dissertation was partially sponsored by "hArtes", a project (IST-035143) of the Sixth Framework Program of the European Community under the thematic area "Embedded Systems". From this point, I want to thank my supervisors Dr. Georgi Kuzmanov and Dr. Ir. Georgi Gaydadjiev who considerably helped and guided me during my Ph.D. research over the last four and a half years. As an original student of Professor Vassiliadis, Dr. Kuzmanov always tried to guide me based on his research principles, and I am grateful for that. I would like also to thank Professor Dr. Ir. Henk Sips for serving as a promotor, and all committee members for their valuable feedback and comments on this dissertation. Furthermore, I want to explicitly thank Lars Hörchens and Jasper van Dorp Schuitman from the Laboratory of Acoustical Imaging and Sound Control at the Delft University of Technology for providing valuable help to accomplish this work.

I would like to thank Lidwina Tromp and Monique Tromp for their administrative assistance, and Erik de Vries, Eef Hartman and Bert Meijs for their fast and reliable technical support. In addition, I am grateful to my officemates Yi Lu, Thomas Marconi and Fakhar Anjam for their help and all interesting discussions we had, and Roel Meeuws for translating the dissertation abstract in Dutch. Finally, I want to thank all my colleges at the Computer Engineering laboratory for making it an enjoyable working environment.

"*It's not only the place, but also the company that makes a moment unique*" they say, and I completely agree. I feel grateful to all my friends here in the beautiful city of Delft for the amazing time we had. I would like to thank (Dr. by now) Christos Strydis for his support and help in every aspect. We had an amazing time living next to each other inside the same almost-collapsed house in Vlamingstraat. Also, I want to thank Carlo Galuzzi and Niki Frantzeskaki for their true support and care. I will never forget the never-ending dinners at their house. The combination of Italian and Greek cuisine always made it a unique gastronomical experience. I really enjoyed also the time I had all these years with my friends Sebastian Isaza, Diomidis Katzourakis, Daniele Ludovici, Lotfi Mhamdi and Yannis Sourdis. Thanks to Sebastian and Aleja for always willing to help me improve my pathetic skills in speaking Spanish. Finally, a very special *thank you* goes to Kamana Sigdel.

This dissertation is dedicated to my parents Nikolaos and Artemis, and my brother George, who supported me all these years from when I left home for the first time in 1998 to study in Crete, Greece. From this point, I want to truly thank them for their unconditional love and care for me.

Dimitris Theodoropoulos Delft, The Netherlands, May 2011

Table of contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms and Symbols

ASIC	Application Specific Integrated Circuit
BF	BeamForming
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DSP	Digital Signal Processor
DMA	Direct Memory Access
DOA	Direction Of Arrival
IF	InterFace
ISA	Instruction Set Architecture
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSB	Front Side Bus
FSB	Filtered Samples Buffer
GPGPU	General Purpose Graphics Processor Unit
GPU	Graphics Processor Unit
GPP	General Purpose Processor
GFLOP	Giga Floating Point Operations
LFE	Low Frequency Enhancement
MADI	Multichannel Audio Digital Interface
MC − BFP	Multi-Core BeamForming Processor
MCP	Multi-Core Processors
MC − WFSP	Multi-Core Wave Field Synthesis Processor
nr − MCP	non-reconfigurable Multi-Core Processors
r − MCP	reconfigurable Multi-Core Processors
RISC	Reduced Instruction Set Computing
RF	Register File
SNR	Signal-to-Noise Ratio
SPR	Special Purpose Register
SDRAM	Synchronous Dynamic Random Access Memory
VLIW	Very Long Instruction Word
WFS	Wave Field Synthesis

1

Introduction

Recording and recreation of an accurate aural environment has been studied for many decades. The first stereophonic transmission was done by Clement Ader at the Paris Opera stage in 1881, while the first documented research on directional sound reproduction was done at AT & T Bell Labs in 1934 [28]. During 1938 and 1940, Walt Disney studio designed the Fantasound stereophonic sound technology, the first one that introduces surround loudspeakers, with audio channels derived from Left, Center and Right. In 1943, William Snow reported in one of his most famous papers regarding stereophonic sound [78] the fundamental principles of sound recording and stereophonic reproduction.

An improved audio rendering technology was designed in 1976 by Dolby Laboratories that introduced the quadraphonic surround sound system. It was called Dolby Stereo (or Dolby Analog) and consisted of four separate channels (left, center, right and mono surround) [52]. During the next two years the surround channel was split into two distinct channels (left surround and right surround), while the idea of a low frequency enhancement (LFE) was also established to properly convey special sound effects. In 1994 the International Telecommunication Union (ITU) specified the 775 standard regarding the loudspeaker layout and channel configuration for stereophonic sound systems [35]. Although most material is recorded and distributed based on this standard, many manufacturers produce loudspeaker setups consisting of more channels. Normally, such systems employ built-in effects-processing to generate all signals for the additional loudspeakers. In 2000 the THX company [19] introduced the 10.2 loudspeakers setup, which is the first one among the surround systems that adds height information on sound localization. An even more elaborated system, was proposed in 2003 by the NHK Science and Technical Research Laboratories in Japan, named 22.2 [36]. The latter consists of three loudspeaker layers positioned to different heights, thus it can deliver

elevation and depth information regarding the acoustic sources localization.

Over the last century, researchers from the audio domain have proposed and applied many different techniques for sound acquisition and rendering. This chapter aims to provide a short introduction to this research field and identify the challenges that arise, in order to build efficient audio systems. Moreover, we provide our research contributions that can help overcome such challenges, and assist to develop quality audio systems.

The chapter organization is as follows: Section 1.1 provides an overview to the sound acquisition and rendering techniques, and identifies their advantages and shortcomings. In Section 1.2 we discuss the major problems that prevent researchers and developers from implementing advanced audio systems on different processing platforms. Section 1.3 presents the key-research questions that we address in this thesis, while in Section 1.4 we present the goals of our research. Finally, Section 1.5 provides the dissertation overview.

1.1 Sound Acquisition and Rendering Techniques

Sound acquisition techniques: Nowadays, there are different techniques for sound acquisition. Efficient microphones placement has been well studied, because it directly affects the signal-to-noise ratio (SNR). In principle, sound recording techniques can be divided into four major approaches:

1. **Acquire the speech signal directly from the source.** This approach is suitable for applications where carrying a close-talk recording device is acceptable, like music concerts and live TV broadcastings.
2. **Surround recording.** This technique is followed when carrying recording devices is not acceptable solution. An exemplary case is the actors from movies, where microphones should not be visible.
3. **Recording of the signals that reach the ears (binaural signals)**. This method implies putting two microphones facing away from each other at a distance equal to human ears (approximately 18 cm). It is applicable in cases where the recorded signals will be rendered through headphones.
4. **Utilize microphone arrays to amplify the original acoustic source.** This solution is applicable in cases where distant speech signals need to be extracted and attenuate any ambient noise. An example application is surveyance systems inside public areas (like airports or public stations), where the security personnel can record and acquire the speech signals of suspects.

The first three techniques have been used for many decades, because they require the least complex hardware setup. However they introduce particular shortcomings. In the first technique for example, although it is well-established for performers and presenters to carry a wired recording device, still requires complex cable setups within the performance area. Even in the case of a wireless microphone, it is considered uncomfortable to constantly carry it. The second approach employs a small number of microphones to record "sound images" [78] of the area and not directly speech signals. Thus, there can be cases where the Signal-to-Noise Ratio (SNR) is low, leading to poor audio quality. The binaural recording method [52] offers high sound localization and perception quality, however it requires that the listener wears headphones. Although there are systems, called Ambiophonics [7], that address this shortcoming, still there are movement restrictions imposed within a small listening area [62].

The last technique is called beamforming (BF) [93] and has already been widely used for many decades in different application fields, like the SOund Navigation And Ranging (SONAR), RAdio Detection And Ranging (RADAR), telecommunications and ultra-sound imaging [96]. Over the last years, the BF technique has been also adopted by the audio research society, mostly to enhance speech recognition. The main advantage is that any stationary or moving audio source within a certain noisy area can be efficiently isolated and extracted with high SNR. Furthermore, there is no need for carrying any recording device. The BF technique requires the utilization of microphone arrays, which capture all emanating sounds. All incoming signals are then combined to amplify the primary source signal, while at the same time suppressing any environmental noise. However, due to the increased number of input channels compared to other approaches, its main shortcoming is that requires substantial signal computations, thus powerful processing platforms.

Sound rendering techniques: As it was mentioned in the beginning of the chapter, sound reproduction techniques have been studied for many decades. These approaches can be split into three fundamentally different categories [90]:

1. **Stereophony.** This is the oldest technique for audio rendering. Examples are the majority of home theater and cinema sound systems that utilize the ITU 5.1 or even more advanced loudspeaker setups.
2. **Generation of the signals that reach the ears (binaural rendering).** As it was mentioned before, this approach is suitable for applications that utilize headphones for sound reproduction. Contemporary binaural

products integrate noise cancellation and, in a few cases, head-rotation detectors, in order to realistically adjust the source location perceived by the listener.

3. **Wavefronts synthesis that are emitted from sound sources.** This approach is considered to be the most advanced among all sound rendering techniques, since it tries to synthesize the original wavefronts emitted from virtual sources.

Stereophony is the oldest and most widely used audio technology. The majority of home theater and cinema sound systems are nowadays based on the ITU 5.1 standard [37]. This is mainly caused by the fact that such systems are easy to be installed due to their rather small number of loudspeakers. However, the ITU 5.1 standard requires a specific loudspeaker configuration in the azimuthal plane, which unfortunately cannot be satisfied in most cases. Furthermore, various tests have shown that sound perception on the sides and behind the listener is poor, due to the large distance between the loudspeakers. Another important drawback of stereophony is that phantom sources cannot be rendered between the loudspeakers and the listener [8] [52]. Binaural systems can deliver a high quality of sound perception and localization, and are suitable only in applications where headphones are acceptable. However, this limitation has already been addressed by many researchers, who have proposed systems that render binaural signals through loudspeakers. These systems apply additional signal filtering to cancel the crosstalk between the left binaural signal reaching the right ear and vice versa [52]. Unfortunately, as it happens with the stereophonic systems, the listening area is size-constrained.

Finally, as we mentioned, an additional way of delivering a natural sound environment is audio technologies that can synthesize wavefronts from virtual sources. The most important benefit of these technologies is that they do not constrain the listening area to a small region, as it happens with stereophonic systems and binaural setups without headphones. On the contrary, a natural sound environment is provided within the entire room, where every listener experiences an outstanding sound perception and localization. However, their main drawback is that they require large amount of data to be processed and many loudspeakers to be driven simultaneously.

Two main technologies that try to synthesize the wavefronts of virtual sources are the Ambisonics and Wave Field Synthesis (WFS). The Ambisonics was proposed by the Oxford Mathematical Institute in 1970 [32]. Researchers focused on a new audio system that could recreate the original acoustic environment as convincingly as possible. In order to achieve this, they developed

a recording technique that utilizes a special surround microphone, called the Soundfield microphone [26]. The recording equipment generates a 4-channel format, called B-Format, that includes all the appropriate spacial information of the sound image. B-Format consists of left-right, front-back and up-down data, plus a pressure reference signal, providing the capability to deliver surround audio with height information. A major advantage of Ambisonics sound systems is that they can utilize an arbitrary number of loudspeakers that do not have to be placed rigidly.

The WFS acoustic algorithm was initially proposed by Berkhout [11] in 1993. It is based on Huygens' principle, which is applied by stating that a primary source wavefront can be created by secondary audio sources, i.e. plane of loudspeakers, that emit secondary wavefronts. The superposition of all secondary wavefronts creates the original one. However, some limitations arise in real world systems. For example, a plane of loudspeakers is not practical, so a linear loudspeaker array is used, which unavoidably introduces a finite distance between the loudspeakers. This fact introduces artifacts such as spatial aliasing, truncation effects, and amplitude and spectral errors of the emitted wavefront [24].

However, the WFS algorithm alleviates many problems that are inherent to other audio systems, like stereophony. For example, it allows the production of sources moving behind and up to a limited distance in front of the loudspeaker array [38]. Furthermore, it allows the production of plane waves which have a stable direction throughout the entire listening area. Finally, a major advantage is that there is no "sweet spot" area restriction. In contrast to stereophonic and Ambiophonic systems that require a fixed placement of the loudspeakers and the listeners remain at the center of the listening area, the WFS allows people to move freely inside the entire acoustic area and still experience an outstanding audio environment perception [89]. Unfortunately, due to the large number of loudspeakers, the WFS requires an excessive amount of signal computations compared to other approaches.

1.2 Problem Definition

As it can be observed from the previous section, over the last decades, researchers from the audio domain have developed new audio acquisition and rendering algorithms to significantly improve sound quality compared to previous methods. These technologies offer an immersive-aural experience to the audience compared to other approaches, thus called *immersive-audio* tech-

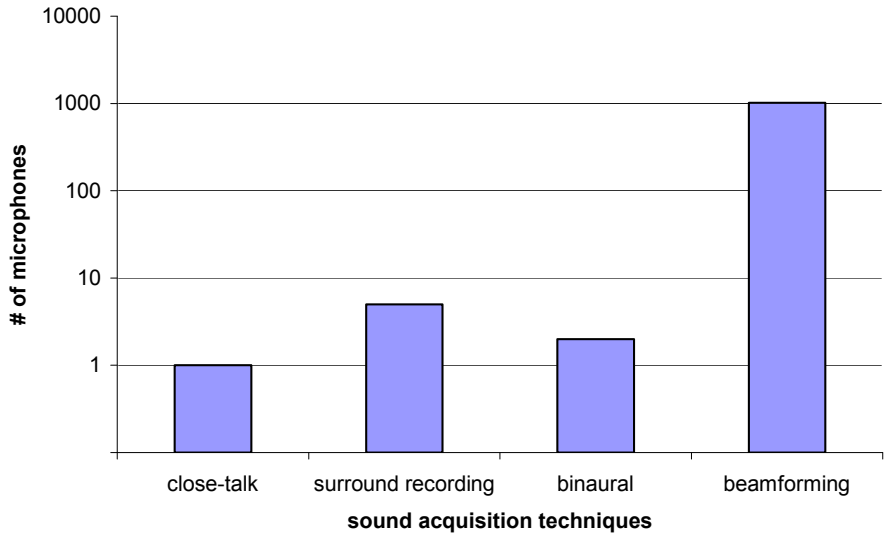

Figure 1.1: Maximum number of utilized microphones among different sound acquisition techniques.

nologies. A common specification among them is that they utilize microphone or loudspeaker arrays. For comparison reasons, Figure 1.1 shows the different number of microphones that each recording technique may require. As it is depicted, surround recording techniques employ no more than five microphones, one of each recorded channel [35]. Binaural recordings use only two microphones, one for each ear, while in the case of a close-talk recording, each speaker uses a single device. In contrast, nowadays there are commercial and experimental systems that utilize the BF technique and employ from tens to more than 1000 microphones [97].

Similarly, for the sake of comparison, Figure 1.2 indicates the number of loudspeakers that may be used under each of the aforementioned sound rendering techniques. Contemporary stereophonic surround systems employ up to 24 loudspeakers. Binaural recordings that are not reproduced through headphones, are normally rendered through two loudspeakers. Experimental Ambisonics-based systems have also been presented in the literature that employ up to 16 loudspeakers [56]. In contrast, as discussed in Section 2.3.2, over the last years, many WFS-based systems have been implemented that employ loudspeaker arrays that range from a few tens up to hundreds of elements [55].

As it was discussed in the previous section, the BF technology alleviates the majority of the shortcomings that other recording techniques introduce, at the

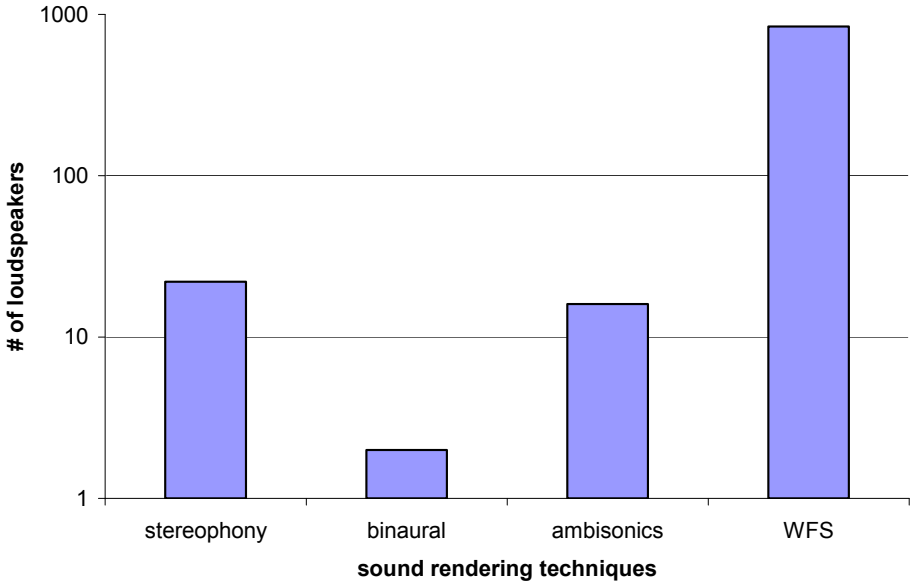

Figure 1.2: Maximum number of utilized loudspeakers among different sound rendering techniques.

expense of an increased number of input channels. At the same time, the WFS algorithm removes many problems that are inherent to stereophonic systems, at the cost of employing from small to very large loudspeaker arrays. Both technologies are highly scalable, thus can be applied to future consumer and professional multimedia and telecommunication products, ranging from portable devices and home theater systems, to high-quality teleconference systems and large cinema rooms. Consequently, because of their inherent parallelism, the most suitable implementation hardware platform domain is the one of Multi-Core Processors (MCPs), which integrate a large number of processing modules that can be either fixed or reconfigurable. We refer to them as non-reconfigurable Multi-Core Processors (nr-MCPs) and reconfigurable Multi-Core Processors (r-MCPs) respectively. Examples of the former are contemporary Graphic Processor Units (GPUs) or other multi-core solutions, that can be even heterogeneous, like the Cell Broadband Engine [34] [33], and of the latter custom multi-core reconfigurable processors that could scale according to the number of input/output channels.

However, research on literature reveals that the majority of experimental and commercial BF and WFS systems are based on standard Personal Computers (PCs), due to their high-level programming support and potential of rapid sys-

tem development. It is well accepted that today's software languages provide a very intuitive development environment that allows rapid systems prototyping and implementation. However, these approaches introduce the following drawbacks:

- *Performance bottlenecks*. General Purpose Processors (GPPs) provide limited computational power, thus in many cases additional PCs are required to efficiently drive all input/output channels.
- *Excessive power consumption*. Contemporary high-end GPPs consume tens to hundreds of Watts of power when they are fully utilized. Furthermore, when additional PCs are employed to drive all required channels, the total system power consumption may easily exceed the kWatt scale.
- *Increased overall system cost*. Utilization of many PCs leads to an approximately linear overall system cost increase, which constrains the employment of such systems only to professional applications or large academic projects.

To partially address the aforementioned problems, researchers have considered alternative hardware platforms to implement immersive audio systems. Regarding the BF technique, various systems have been developed based on Digital Signal Processors (DSPs), in order to reduce power consumption, however performance is still limited. In contrast, recent GPU-based BF approaches provide a significantly better performance compared to PC-based systems, however a considerable effort is required, in order to efficiently analyze and map the application onto the available processing resources. Custom-hardware solutions alleviate both of the aforementioned drawbacks. However, in the majority of cases, designers are primarily focused on just performing all required calculations faster than a GPP. Such approaches do not provide a high-level interface for testing the system that is under development. For example, in many cases, the designers should try what the SNR of an extracted source is under different filter sizes and coefficient sets. Such experiments can easily be conducted using a standard PC with a GPP and a high-level programming language, but they would take long time to be re-designed in hardware, and cannot be performed on the field at post-production time.

Regarding the WFS algorithm, research on literature reveals that all experimental and commercial WFS systems are implemented also using desktop PCs, again due to the support of very high-level software programming languages. However, as it is discussed in Section 2.3.2, GPPs can not cope with the processing requirements of WFS systems that utilize large loudspeaker arrays and render simultaneously many acoustic sources. Furthermore, up to

now, there are no GPU- or Field Programmable Gate Array (FPGA)-based WFS systems reported in the literature, rather only articles that present simulation results under different loudspeaker and source scenarios. As it was in the case of the BF technique, the lack of a high-level interface for the aforementioned hardware platforms, refrains researchers and developers from implementing systems to them, and thus choose mainstream GPPs.

Main research problem: Define a custom high-level and platform-independent architecture for immersive audio systems, which will allow performance and power efficient implementations on different contemporary multi-core technologies, such as FPGAs and GPUs.

1.3 Research Questions

To solve the above research problem, we have to address the following important research questions:

- *How to map rapidly and efficiently immersive-audio technologies onto Multi-Core Processors (MCPs)?* The main challenge is to provide a versatile architecture[1] to researchers, in order to enhance productivity and shorten testing and development time. This architecture should be at a high-level of abstraction, in order to make it applicable to different types of MCPs. Furthermore, such an approach would provide the benefit of portability and ease of application code reuse among the different hardware platforms.
- *Which instructions should be supported by the architecture for immersive-audio systems?* It is very important to provide a set of instructions that will allow easy customization of many vital system parameters, efficient audio-data processing, and system debugging through a high-level interface. Furthermore, they should be platform-independent and hide any platform-specific implementation details, thus allowing the same program to be executed to different hardware devices with minimal software changes.
- *How to enhance performance and efficiently support small- and large-scaled immersive-audio systems?* Nowadays, there are many different

[1] Throughout this dissertation, we adopt the terminology from [31], according to which, the computer architecture is termed as the conceptual view and functional behavior of a computer system as seen by its immediate viewer - the programmer. The underlying implementation, termed also as micro-architecture, defines how the control and the datapaths are organized to support the architecture functionality.

multi-core platforms. A key-issue is to choose the correct one, based on the application requirements. A direct selection of a powerful MCP for developing small-scaled systems, would lead to excessive power consumption and overall cost, while a cheap platform that integrates few processing cores could result to a poor solution that does not cope with the real-time constraints.

- *How to choose the most energy- and power-efficient approach for such complex systems?* As it was mentioned before, immersive-audio systems employ many input/output channels, thus requiring a lot of processing power. For example, contemporary WFS PC-based systems may utilize a PC-network to drive all loudspeakers, thus requiring many hundreds of Watts for powering only the GPPs. By choosing a suitable MCP to substitute the PC-network, future immersive-audio systems can consume orders of magnitude less power compared to current approaches.

Addressing the above questions would be an important step to achieve rapid development of immersive-audio systems based on MCPs. Furthermore, careful selection of the utilized processing platform would result to more efficient approaches that could support many real-time sources under a large number of input/output channels. Ultimately, an excessive amount of energy can be saved, since fewer, more efficient, processing units would consume less power.

1.4 Dissertation Contributions

In this dissertation, we addressed all research questions mentioned in the previous section. Our main contributions are the following:

- *High-level architecture for immersive-audio applications.* We propose a high-level architecture that consists of 14 instructions, which allow customization and control of BF and WFS immersive-audio systems implemented on MCPs. Our proposal considers a globally-shared, locally-distributed memory hierarchy and allows a high-level interoperability with different MCPs. This means that the same program, with slight modifications, can be mapped onto different platforms, thus providing a versatile and portable solution that is applicable to a wide range of immersive-audio systems.
- *Micro-architectural support for r-MCP- and nr-MCP-based immersive-audio algorithms.* The architecture implementation allows the utilization of a wide number of processing elements, thus making it suitable

for mapping onto r-MCPs and nr-MCPs. With respect to the available resources, different implementations with different performance characteristics are possible, where all of them use the same architecture and programming paradigm. In this dissertation we present two case studies of our architecture implementation, namely on a set of r-MCPs, and a wide range of off-the-shelf GPUs.

- *Extensive performance experiments under different input/output scenarios.* We conducted various tests for both BF and WFS applications, ranging from small- to large-scaled setups. Furthermore, we investigated the maximum number of real-time sources that each processing platform can support under different sizes of input/output channel arrays. Based on our experimental results, we propose the most suitable platform for each case, in order to build efficient immersive-audio systems.
- *Platform evaluation regarding energy consumption and system cost.* Based on the processing time and the power consumption of all platforms, we suggest their energy consumption. Immersive-audio systems utilize a large number of input/output channels, thus consume an excessive amount of energy. A good platform selection can help on reducing energy and consequently the overall system economic cost.

1.5 Dissertation Organization

The dissertation structure is organized as follows: In Chapter 2, we provide the theoretical background of the BF and WFS techniques. We also present many software and hardware implementations of them that are mapped onto different platforms, in order to build experimental and commercial immersive-audio systems. Furthermore, we provide an evaluation of many immersive-audio systems that utilize the BF and WFS techniques with respect to performance, power consumption and ease of use.

In Chapter 3, we present the proposed architecture for both BF and WFS algorithms that comprises custom memory hierarchy and instruction set. We describe its memory and register organization, and its application to r-MCPs and nr-MCPs. Moreover, we analyze each instruction and elaborate on the functionality of every input/output parameter. In addition, we demonstrate how our architecture can be used to develop programs for BF and WFS immersive-audio systems.

In Chapter 4, we present the underlying multi-core micro-architecture when

utilizing r-MCPs for both BF and WFS techniques. We also describe two custom-designed hardware accelerators for BF and WFS-oriented data processing. Furthermore, we show each instruction's micro-architecture implementation, in order to allow a high-level user interaction with the custom accelerators. Finally, we present the complete hardware prototypes of a Multi-Core Beamforming Processor (MC-BFP) and a Multi-Core WFS Processor (MC-WFSP) that were used to evaluate our proposal in Chapter 6.

In Chapter 5, we conduct a nr-MCPs case study for our architecture, by applying it to a wide range of GPUs. We provide a brief description of contemporary GPUs organization. We also describe how we implemented each high-level instruction by hiding all GPU-specific code annotations details from the user. Furthermore, we explain how we use important system parameters, like the number of input/output channels and filter sizes, to develop GPU BF and WFS kernels that are efficiently mapped onto the GPU processing cores.

In Chapter 6, we describe the experimental setup that we applied, in order to test our FPGA and GPU-based implementations regarding performance for the BF and WFS applications. We compare the results accuracy of our hardware approaches against a Core2 Duo approach, since the former employ a fixed-point format for all internal calculations. We also provide a comparison of the two multi-core systems against the Core2 Duo and related work. In addition, we investigate the architectural prospectives of high-end GPUs and latest generation FPGA families by comparing their execution times under many input/output channels and real-time sources scenarios. Finally, we discuss each system's energy consumption and overall cost.

Finally, in Chapter 7, we present our conclusions from our research. We also present a few open issues for future work. Such issues encounter the enhancement of our proposed architecture with more customizing options and additional immersive-audio technologies support, like the Ambisonics. Furthermore, an additional issue is its applicability to additional nr-MCPs, like the Cell Broadband Engine.

2

Background and Related Work

In this chapter we provide the theoretical background of the beamforming (BF) and Wave Field Synthesis (WFS) techniques in Section 2.1 and Section 2.2 respectively. In Section 2.3 we present many software and hardware implementations of them that are mapped onto different platforms, in order to build experimental and commercial immersive-audio systems. Section 2.4 provides an evaluation of many immersive-audio systems that utilize the BF and WFS techniques with respect to performance, power consumption and ease of use. Finally, in Section 2.5 we conclude the chapter.

2.1 Background of the Delay-and-Sum BF technique

The term of beamformer refers to a processor that performs spatial filtering, in order to estimate a signal arriving from a particular location. Thus, even in the case where two signals contain overlapping frequencies, a beamformer is able to distinguish each one of them, as long as they originate from different locations.

Generally, there are two different types of BF, non-adaptive (or time-invariant or non-blind) and adaptive (or blind) [9], [93]. Non-adaptive methods are based on the fact that the spatial environment is already known and tracking devices are used to enhance speech recognition. In contrast, adaptive approaches do not utilize tracking devices to locate the sound source. In fact, the received signals from the microphones are used to calibrate properly the beamformer, in order to improve the quality of the extracted source. In the audio domain, in the majority of the cases a non-adaptive delay-and-sum approach is utilized [93], due to its rather simple implementation and because a tracking device (such as a video camera) is almost always available.

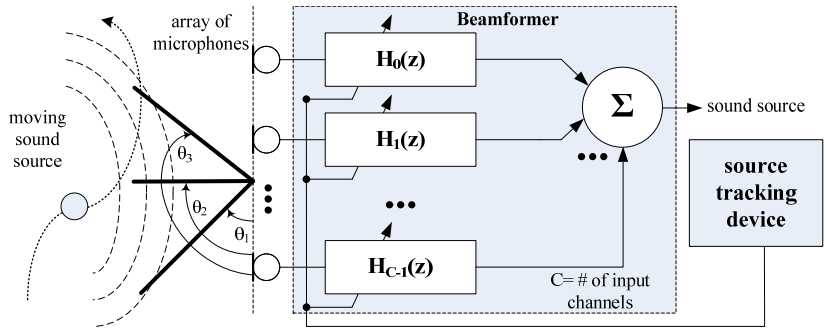

Figure 2.1: A filter-and-sum beamformer.

Figure 2.1 depicts a schematic overview of a beamformer utilizing the filter and sum approach [93]. As we can see, the system consists of an array of microphones sampling the propagating wavefronts. Each microphone is connected to a FIR filter $H_i(z)$, while all filtered signals are summed up to extract the desired audio source. In many cases, the input data channels are downsampled by a factor D in order to reduce the data rate:

$$xD_i[n] = x_i[n \cdot D] \tag{2.1}$$

where x_i is the input signal, xD_i is the downsampled signal, $i=0...C-1$ and C is the number of input channels (microphones). Each downsampled signal is filtered using a particular coefficient set based on the source location:

$$yD_i[n] = \sum_{j=0}^{H-1} h_i[j] \cdot xD_i[n-j] \tag{2.2}$$

where H is the number of filter taps and h are the filter coefficients. The beamformer output is given by the sum of all yD_i signals:

$$yD[n] = \sum_{i=0}^{C-1} yD_i[n] \tag{2.3}$$

where yD is the downsampled extracted source. Then, yD is upsampled by a factor L (normally $L=D$) according to equation (2.4) to acquire the upsampled extracted source y:

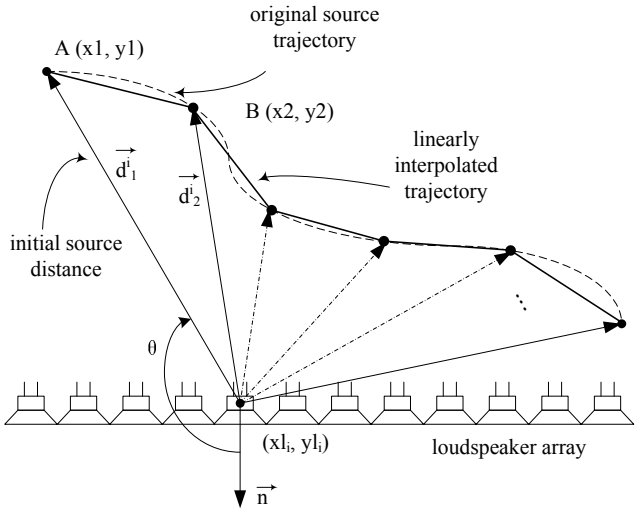

Figure 2.2: Linear interpolation of a moving sound source.

$$y[n] = \begin{cases} yD\left[\frac{n}{L}\right] & , if \frac{n}{L} \in Z \\ 0 & , otherwise \end{cases} \tag{2.4}$$

The idea behind this structure is to use the FIR filters as delay lines that compensate for the introduced delay of the wavefront arrival at all microphones [13]. The combination of all filtered signals will amplify the desired one, while all interfering signals will be attenuated. However, in order to extract a moving acoustic source, it is mandatory to reconfigure all filters coefficients according to the source current location. For example, as it is illustrated in Figure 2.1, a moving source is recorded for a certain time inside the aperture defined by the $\theta_2 - \theta_1$ angle. A source tracking device is used to follow the source trajectory. Based on its coordinates all filters are configured with the proper coefficients set. As soon as the moving source crosses to the aperture defined by the $\theta_3 - \theta_2$ angle, the source tracking device will provide the new coordinates, thus all filter coefficients must be updated with a new set. This process is normally referred to as "*beamsteering*".

2.2 Background of the WFS technique

As it was mentioned in Section 1.1, the WFS technique utilizes loudspeaker arrays, in order to generate the wavefronts of virtual sources. Figure 2.2 illus-

trates an example of a linear array loudspeaker setup. Each loudspeaker has its own unique coordinates (xl_i, yl_i) inside the listening area. In order to drive each one of them so as the rendered sound source location is at A(x1, y1), the so called *Rayleigh 2.5D operator* [91] needs to be calculated:

$$Q_m(\omega, |\overrightarrow{d_1^i}|) = S(\omega)\sqrt{\frac{jk}{2\pi}}\sqrt{\frac{Dz}{z+Dz}}\frac{z}{|\overrightarrow{d_1^i}|}\frac{exp(-jk|\overrightarrow{d_1^i}|)}{\sqrt{|\overrightarrow{d_1^i}|}} \tag{2.5}$$

where $k = \frac{\omega}{c}$ is the wave number, c is the sound velocity, z is the inner product between $\overrightarrow{n}$ and $\overrightarrow{d_1^i}$, Dz is reference distance, i.e. the distance where the Rayleigh 2.5D operator can give sources with correct amplitude, $S(\omega)$ is the acoustic source, $\sqrt{\frac{jk}{2\pi}}$ is a 3dB/octave correction filter, $\sqrt{\frac{Dz}{z+Dz}} \cdot \frac{z}{|\overrightarrow{d_1^i}|}$ is the source amplitude decay (AD) and e^{-jkr} is a time delay that has to be applied to the particular loudspeaker. According to Figure 2.2, since z is the inner product between $\overrightarrow{n}$ and $\overrightarrow{d_1^i}$ with angle θ, the *AD* can be calculated by the following formula:

$$AD = \sqrt{\frac{Dz}{(Dz+z)\cdot|\overrightarrow{d_1^i}|}} \cdot cos(\theta) \tag{2.6}$$

In order to render a moving source from a point A to a point B behind the loudspeaker array, a linearly interpolated trajectory is calculated [91]: Distance $|\overrightarrow{d_2}| - |\overrightarrow{d_1}|$ is divided by the samples buffer size *bs*, in order to calculate the distance difference (*DD*) in meters of the source from loudspeaker *i* between two consecutive audio samples:

$$DD = \frac{|\overrightarrow{d_2^i}| - |\overrightarrow{d_1^i}|}{bs} \tag{2.7}$$

Based on the *DD*, the source distance $|\overrightarrow{d_1^i}|$ from loudspeaker *i* with coordinates (xl_i, yl_i) is updated for every sample by the formula:

$$|\overrightarrow{d}| \Leftarrow |\overrightarrow{d_1^i}| + DD \tag{2.8}$$

According to the current distance $|\overrightarrow{d_1^i}|$ from loudspeaker *i*, an output sample is selected based on the formula:

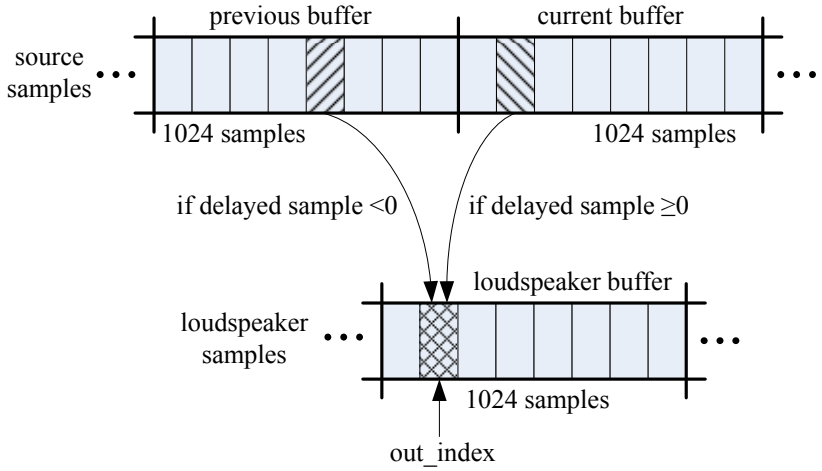

Figure 2.3: Proper choice of the delayed sample.

$$delayedsample = -(l + (df \cdot |\overrightarrow{d_1^i}|)) + (out\_index + +) \tag{2.9}$$

where $df = f_s/c$ is the distance factor (f_s is the sampling rate, c is the sound speed), *out_index* is the current output audio sample, and l is an artificially introduced latency, in order to allow sources to be rendered in front of the loudspeaker array. Finally, the selected delayed sample is multiplied by the *AD* and the system master volume.

Figure 2.3 illustrates how the delayed sample is calculated. The source samples are divided into *bs* source segments (for example *bs*=1024-sample segments). In each iteration a source segment is used to select the proper audio samples for each loudspeaker. However, there are cases where the evaluated delayed sample does not belong to the current source segment, but instead, to the previous one. Thus, in every iteration, two source segments are needed, the *current* and the *previous* one, to cover both cases where the evaluated delayed sample is positive or negative respectively. Further details can be found in [14], [16], [94], [38] and [91].

2.3 Commercial and Experimental Systems

2.3.1 Systems that utilize the BF technique

Over the last years, various systems that utilize GPUs under different application domains have been published in the literature. In [96] the authors describe a hybrid approach that utilizes 14 Virtex4 LX25 FPGAs [106] and a GPU connected to a desktop PC to perform 3D-parallel BF and scan conversion for real-time ultrasonic imaging. Input data are received from 288 channels that are connected to Analog-to-Digital Converters. Digitized data are forwarded to the FPGAs, which calculate the signal delay, interpolation and apodization. All processed data are transferred though the PCI from the FPGAs to the GPU. In [63], the authors utilize a GeForce 8800 GPU [65] to design a delay-and-sum beamformer in the time and frequency domain. To evaluate their designs they perform experiments under different number of input channel setups ranging from 79 to 1216. According to the results, a time-domain and a frequency-domain beamformer can achieve speedup up to 12x and 15x respectively compared to a Xeon Quad-core processor.

In the *audio* domain, the BF technique is widely used in handheld devices, like cell-phones and Personal Digital Assistants. Such embedded systems introduce many constraints regarding computational resources and power consumption. To alleviate these problems, the authors in [77] designed a time-invariant beamformer tailored to small devices that consists of two microphones. According to the paper, results suggest a signal to noise ratio (SNR) improvement of 14.95 dB when using two microphones, instead of one. A data driven beamformer for a binaural headset is presented in [47]. The authors integrate two microphones to the headphones and employ a Head and Torso Simulator to acquire the source signal for BF. The improvement of SNR is in the range between 4.4 and 6.88 dBC.

Commercial products for audio BF have been developed by various companies. For example, Squarehead [83] develops the Audioscope, a dual core PC-based system, that employs 300 omnidirectional microphones for audio capturing. Another company, called Acoustic Camera [2], develops PC-based BF systems, that utilize sound acquisition arrays ranging from few tens to more than hundred elements. Polycom and Microsoft presented the *CX5000* unified conference station [75], which is the latest version of the Roundcam, originally presented in [76]. Roundcam consists of five built-in cameras that offer a 360^{o} panoramic view of the conference room and eight microphones to capture the speech signals. It connects to a dual CPU 2.2 GHz Pentium 4 workstation

through a Firewire bus. All image and sound processing is done to the workstation. For computational efficiency and low latency, the authors utilize a delay-and-sum beamform approach. Lifesize is another company that produces high quality communication systems. For example the *LifeSize Focus* teleconferencing camera supports high definition video and uses two omni-directional microphones to capture audio sources using BF. A small set of these cameras is utilized in the company's advanced communication systems, like the *LifeSize Room* series, to record image and transmit it to the remote location. Sound sources are rendered to the remote location using high definition audio.

In [18], [71], the authors present the NIST Mark-III Microphone array that can be used for speech enhancement and recognition. The proposed platform utilizes 64 input channels that are connected to a Spartan II FPGA [105] via Analog-to-Digital converters. The FPGA is connected through Ethernet to a host desktop PC that runs the NIST Smart Flow II software platform [30] [27]. The latter employs a web-camera that identifies a speaker's face and steers accordingly the BF, in order to enhance the speech signal and attenuate any ambient noise.

The authors of [48] present a hardware accelerator that utilizes microphone array algorithms based on the use of calibrated signals together with subband processing. The proposed design utilizes a frequency domain modified recursive least squares adaptive algorithm and the SNR maximization of the BF algorithm. Up to 7 instances of the proposed design can fit in a Virtex4 SX55 FPGA [106], achieving a speedup of up to 41.7x compared to the software implementation.

A similar approach is chosen in [1] where a real-time beamformer mapped onto an FPGA platform is presented. The BF engine is based on the QR matrix decomposition (QRD). In each update of the beamformer, new input samples are generated by a Matlab [58] host application and forwarded to the FPGA, where the QRD engine processes them. Once processing is done, the new weight vector is returned back to the host processor and a new chunk of data is forwarded to the FPGA. The complete design occupies 3530 Virtex4 [106] slices and requires 56.76 μsec to decompose a 10x10 matrix at 250 MHz.

A Digital Signal Processor (DSP) implementation of an adaptive subband BF algorithm, is presented in [114], known as the Calibrated Weighted Recursive Least Squares (CWRLS) beamformer. The authors utilize an Analog Devices ADSP21262 DSP processor [5] to perform CWRLS-based BF over a two microphone array setup. According to the paper, results indicate that there is an up to 14 dB SNR improvement, but the computational load of the DSP proces-

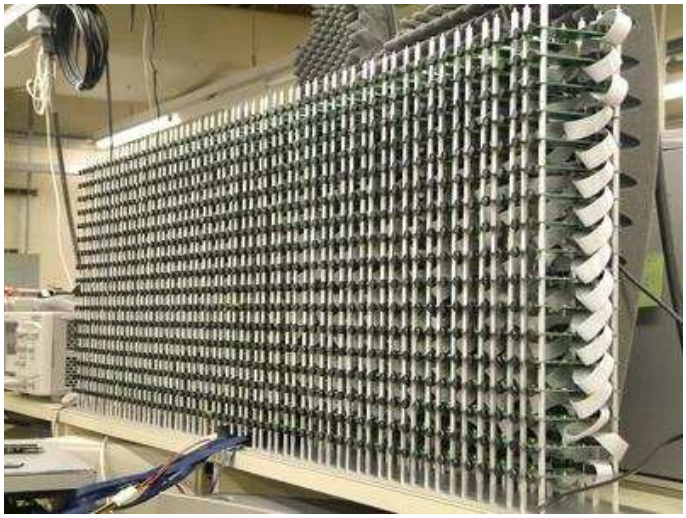

Figure 2.4: The MIT LOUD microphone array consisting of 1020 elements [97].

sor can be up to 50% with two input channels. The presented implementation is also energy efficient, since it was predicted to have an operation time of up to 20 hours, under the aforementioned processor utilization.

An experimental video teleconferencing system is presented in [57]. The authors combine an omnidirectional video camera and an audio BF array into a device that is placed in the center of a meeting table. Non-stationary participants are identified with computer vision algorithms and their speech is recorded from circular 16-microphone array. Audio processing is done using a TMS320C6201 DSP processor [88] at 11.025 kHz sampling rate.

Finally, nowadays there are many projects that utilize different microphone array sizes and setups. One of the most famous implementations is the Large AcOUstic Data (LOUD) [97], shown in Figure 2.4, which was part of the MIT Oxygen project [60]. The LOUD microphone array consist of 1020 elements arranged into a 2D planar setup and produce data at a rate of 50 MB/sec. All data are streamed to a custom-designed tiled parallel processor, based on the Raw ISA [85], [84], [86]. Experimental results suggest that utilizing such a large microphone array can dramatically improve the source recognition accuracy up to 90.6%.

2.3.2 Systems that utilize the WFS technique

One of the earliest and most important research efforts that exploited the WFS technology was the European project CARROUSO [15]. Its purpose was to provide a new technology that transfers a generated sound field to another remote location by exploiting the MPEG-4 standard. The sound field could be generated at a specific real or virtual space. The project also supported the combination of the spatial and perceptual properties of the sound field with visual data. During recording, only dry sources were captured, while room impulse response and source locations were also recorded separately. These data were encoded using the MPEG-4 standard and the encoded audio stream could optionally be multiplexed with video and transmitted to a remote location. The received data were de-multiplexed back to audio and video. The user had the option to further process the audio signals and then perform WFS rendering.

In [61] the authors describe a 840-loudspeaker channel setup that is installed to one of the lecture rooms at the Technical University in Berlin, Germany. Fifteen desktop PCs are utilized to drive the loudspeaker array. Moreover, in order to provide an efficient software platform that controls the WFS-based audio system, the authors presented the sWonder software in [55]. The latter was divided into submodules that can be mapped to multiple PCs, which exchange data using the OpenSoundControl communication protocol [6].

SonicEmotion [80] and Iosono [46] are two companies that produce audio systems based on the WFS technology. SonicEmotion deploys its unit on Intel Core2 Duo-based WFS setup, which requires a total power of 360 Watt for the entire system. It supports rendering up to 64 real-time sound sources, while driving a 24 loudspeaker array. Moreover, in [73], two employees of this company proposed a complete signal processing network for distributed WFS systems. Iosono also follows a standard PC approach that supports up to 64 real-time sources while driving 128 speakers. In both cases, when more loudspeakers are required, additional rendering units have to be cascaded.

An experimental WFS system has been developed at the Delft University of Technology [92]. The presented system is based on a desktop PC and utilizes 14 multi-actuator panels, thus acoustic sources are rendered through a 114 loudspeaker array. Another sound system that was built in IRT in Munich, called the Binaural Sky [59], actually combines both binaural [52] and WFS technologies. The Binaural Sky concept is based on the avoidance of Cross Talk Cancelation (CTC) filters calculation in real time, while the listener head is rotated. Instead of using two loudspeakers, the authors utilize a circular

Figure 2.5: Cinema in Ilmenau, Germany that utilizes the WFS technique equipped with 192 loudspeakers.

loudspeaker array that synthesizes focused sound sources around the listener. The system uses a head tracking device and, instead of real time CTC filter calculation, it adjusts the loudspeaker driving functions such as delay times and attenuations. The loudspeaker array consists of 22 broadband elements and a single low frequency driver. All real-time processing is done on a Linux PC with a 22 channel sound card.

A GPU-based WFS implementation that utilizes the NU-Tech software framework [64] is discussed in [53]. The authors have developed a NU-Tech plug-in that uses the CUDA libraries for the required data calculations, and run it on a GeForce GTX285 [66] and a Tesla C1060 GPU [67]. The presented implementation is compared against an Core i7-based approach based on the Intel Integrated Primitives (IIP) by the same authors. Results suggest that the GTX285-based solution can process data more than 3.5 times faster than the Core i7.

In [81], the authors apply the WFS technology to a multi tiled hardware architecture called ”Scalable Software Hardware computing Architecture for Embedded Systems” (SHAPES). Each of these tiles consists of a Distributed Network Processor for inter-tile communication, a RISC processor and one mAgicV VLIW floating point processor. According to the paper, a WFS system capable of supporting 32 sound sources while driving up to 128 speakers with the MADI interface [79], would require 64 such tiles.

In [3], the authors propose an immersive-audio environment for desktop applications. Their system also utilizes the WFS technology. Small loudspeakers are placed around the computer display, which allows the listener to move

freely inside the listening area. Again, the system is based on a standard 2 GHz PC.

In [82], the authors developed a system that combines WFS technology with a projection-based multi-viewer stereo display. The system hardware setup consists of 4 standard PCs and 32 loudspeakers. One PC is used to control 2 cameras that are tracking movements of a user. A second PC drives 4 LCD projectors that generate images on a perforated screen. A third PC is used as an audio player and is connected to a fourth PC, which is an older Iosono rendering unit [46]. The latter drives 32 loudspeakers divided into 4 8-loudspeaker panels.

Finally, in Ilmenau, Germany a cinema has already been equipped with 192 loudspeakers since 2003 [29], as illustrated in Figure 2.5. More specifically, the loudspeaker array consists of 24 panels, each equipped with 8 two-way loudspeakers. In order to efficiently drive all array elements, the cinema is also equipped with six rendering PCs.

2.3.3 Systems that utilize both BF and WFS techniques

In [10], the authors describe an immersive-audio system that consists of 12 linearly placed microphones. The sound source is tracked through audio and video tracking algorithms, while the beamformer is steered accordingly. The audio signal is extracted through BF and encoded using the MPEG2-AAC or G722 encoders. The encoded signal is received from a second remote PC and the audio signal is rendered using the WFS technology through a 10-loudspeaker array.

A similar system is presented in [87]. The authors describe a real-time immersive-audio system that exploits the BF technique and the WFS technology. The system performs sound recording from a remote location A, transmits it to another one B, and renders it through a loudspeaker array utilizing WFS. The complete system consists of 4 PCs out of which, one is used for the WFS rendering, one for BF, one for the source tracking and one as a beamsteering server.

Finally, the work presented in [17] addresses the problem of echo cancellation that is inherent to contemporary multimedia communication systems. The authors propose a strategy to reduce the impact of echo while transmitting the recorded signal to a remote location. The idea is to to apply the proposed acoustic echo cancellation (AEC) to the ”dry” source signals that will be rendered through the loudspeaker array. Then, the AEC output signals are sub-

tracted from the output signals of the beamformer's time invariant components. In order to test their approach, the authors develop a real-time implementation using a standard desktop PC that consists of 11 microphones and 24 loudspeakers.

2.4 Related Work Evaluation

Table 2.1 provides a summary of the majority of the references mentioned in Section 2.3. The *Technique* column provides the algorithm that each system utilizes, that is *BF* for beamforming and *WFS* for Wave Field Synthesis. The *Channels* column shows how many input / output channels each system supports. We should note that in [63], [53] and [81] the authors conducted only experiments to each underlying hardware platform, assuming different array setups. The *Platform* column indicates the hardware platform that each system utilizes to perform data calculations.

We evaluated each of the presented systems based on three major specifications, namely *performance*, *power consumption* and the ability to provide a *high-level interface* (IF) to the user / developer. Our evaluation for each of the aforementioned parameters to every system is represented by the following symbols, ✓ - good, ↕ - medium, and x - bad. We use the *Performance*, *Power* and *High-level IF* columns to grade each of the presented immersive-audio systems.

Lines 1 to 9 present the systems that utilize the BF technique. As we can observe, there is a variety of hardware platforms that have been employed over the last years, in order to build BF systems. The reason is because BF is a well established technique for many decades, thus researchers have presented various systems based on either off-the-shelf products or custom solutions. For example, in [48] and [1], the authors have used FPGAs to accommodate their systems, thus providing good application performance and low power consumption. On the other hand, such custom approaches do not provide a high level IF to the user, in order to parameterize the system based on the desired requirements, thus recustomizing usually take long time.

In contrast, DSP-based solutions almost always provide a high-level environment for application development, while at the same time they require very low power consumption. However, as it was described in Section 2.3, such hardware platforms usually lack of performance and they cannot be used to accommodate systems with high number of input channels.

Table 2.1: Related work summary for BF and WFS implementations.

Line	Reference	Technique	Channels	Platform	Performance	Power	High-level IF
1	[48]	BF	4	FPGA	✓	✓	x
2	[1]	BF	up to 10	FPGA	✓	✓	x
3	[114]	BF	2	DSP	x	✓	✓
4	[57]	BF	16	DSP	x	✓	✓
5	[63]	BF	79 to 1216*	GPU	✓	x	↕
6	[77]	BF	2	x86	x	x	✓
7	[47]	BF	2	x86	x	x	✓
8	[97]	BF	1020	Raw [84]	↕	↕	✓
9	[83]	BF	300	x86	↕	x	✓
10	[59]	WFS	22	x86	x	x	✓
11	[92]	WFS	114	x86	↕	x	✓
12	[53]	WFS	128 to 1024*	GPU	✓	x	✓
13	[29]	WFS	192	x86	↕	x	✓
14	[61]	WFS	832	x86	↕	x	✓
15	[81]	WFS	128*	RISC & DSP	↕	✓	✓
16	[3]	WFS	16	x86	x	x	✓
17	[82]	WFS	32	x86	↕	x	✓
18	[80]	WFS	24	x86	↕	x	✓
19	[46]	WFS	128	x86	↕	x	✓
20	[10]	BF, WFS	12, 10	x86	x	x	✓
21	[17]	BF, WFS	11, 24	x86	x	x	✓
22	[87]	BF, WFS	26, 24	x86	x	x	✓

* The reference provides only experimental results for these number of channels.

It is well accepted that GPU-based approaches provide very good application performance. Furthermore, as it was also mentioned in Chapter 1, they can be programmed using high-level languages that require certain extensions and code annotations, in order to efficiently map the most computationally intensive parts of an application to all available GPU resources. Unfortunately, GPUs, like CPUs, consume hundreds of Watts power [66]. Thus under certain scenarios where power consumption is constrained (e.g. handheld or battery-operated devices), GPUs are not a suitable solution.

The LOUD BF system is an ASIC-based approach and its primary objective is to provide a source signal quality that approximates the one of close-talking microphones. The Raw chip that performs all data calculations requires approximately 25 Watts of power, which makes it suitable only for stationary scenarios. However, the project researchers provide a high-level programming environment, in order to efficiently map the BF application to the Raw resources.

Finally, the solution from Squarehead presented in [83] is also based on a standard PC approach and can extract up to 5 acoustic sources. Its primary focus is on live events, like TV broadcasts or teleconferences. For this reason, the company provides a very intuitive configuration environment to the user. Un-

fortunately, since it is based on a 3.2 GHz quad-core PC, it requires excessive power consumption.

In lines 10 to 19 of Table 2.1 we provide many of the WFS systems that have been already developed from commercial companies and researchers in the audio domain. As it can be observed, the majority of these systems is based on standard PCs, which provide a high-level of programming and customizing environment, however their main drawbacks are the limited performance and excessive power consumption.

For example, six PCs are required to drive the 192 loudspeakers that have been installed inside the cinema listening area in Ilmenau, Germany. Moreover, as it is mentioned in [61] [55], the authors employed 15 WFS PCs to drive the 840 loudspeaker channels that are installed inside a lecture hall at the Technical University of Berlin, Germany. Consequently, such system setups require many kWatts of power to efficiently process data for every output channel.

In addition, as it can be observed from Table 2.1, there are very few non PC-based approaches to the literature that describe WFS systems. For example, in [53], the authors utilize the NU-Tech software framework [64], to experiment with loudspeaker arrays ranging from 128 to 1024 elements. The main benefit of this approach is that it combines the GPU performance with the NU-Tech high level environment, thus the user is shielded from the CUDA-specific software annotations [22]. However, the main drawback is the excessive power consumption of the GTX285 GPU that performs all required computations.

Similarly, the work presented in [81], does not use a desktop PC. Instead, it uses the SHAPES tiled hardware architecture approach, where each tile integrates an Atmel mAgic VLIW DSP [21] and a RISC processor. Unfortunately, the power consumption of a single tile is not reported, thus, in order to perform an estimation, we used the Atmel Diopsis D940HF datasheet [20], which couples an ARM926 processor with a mAgic DSP. Based on the specification, a single D940HF chip consumes P = Current $\cdot$ Voltage = 0.302 A $\cdot$ 1.2 V $\simeq$ 0.36 Watts. Thus, we can safely assume that a 16-tile implementation consumes approximately 16 Tiles $\cdot$ 0.36 $\frac{Watts}{Tile}$ = 5.76 Watts. An interesting observation is that, to the best of our knowledge, *there is no implementation in the literature* that maps the WFS technique to reconfigurable hardware.

Finally, in lines 20 to 22 of Table 2.1 we present three experimental multimedia communication systems that utilize immersive-audio technologies for source extraction and rendering. As we can observe, they are solely based on standard PC platforms, due to their ease of development and testing. However, there is a limitation on the employed number of input/output channels due to processing

bottlenecks. Furthermore, power consumption is also high, since in certain cases more than a single PC are connected through Ethernet to perform all data computations.

To summarize, as it can be observed, the majority of today's commercial and experimental immersive-audio systems utilize GPPs as their main processing platform, due to their high-level programmability. As it was mentioned in Section 1.2, these approaches introduce performance bottlenecks, excessive power consumption and increased overall system cost. Reported solutions that consider alternative processing platforms, like DSPs and FPGAs, address only part of the aforementioned problems. In contrast, with our proposal it is possible to combine the software flexibility of GPPs with the high performance of multi-core platforms, while at the same time providing to the developer the choice between a low-system-cost or reduced-power-consumption approach.

2.5 Conclusions

In this chapter we presented the theoretical background for both BF and WFS techniques. Moreover, we presented various commercial and experimental audio systems that utilize them, in order to efficiently extract acoustic sources using microphone arrays and/or render them through loudspeaker arrays consisting of hundreds of elements. Various software and hardware platforms have been proposed for both techniques, each having its own benefits and drawbacks.

The primary observation is that x86-based systems provide a high-level interaction environment to the user, but they also introduce performance limitations and excessive power consumption. DSP-based solutions alleviate the problem of power while keeping the advance of an intuitive user interface, however they lack of adequate processing resources that would allow a high number of sources to be extracted and/or rendered in real-time under complex input/output setups. On the other hand, GPU-based approaches provide also a rather versatile software platform and a much better performance than x86-based systems, however they require hundreds of Watts power. Regarding the BF technique, previous FPGA-based proposals are mainly micro-architectural ones, thus focusing on improving performance compared to x86 software implementations. Unfortunately, in case specific parameters need to be recustomized, like the number of input channels or any filter coefficients, the user needs to re-design the circuit and map it again onto the FPGA. In addition, there is no work that implements the WFS technique to reconfigurable hard-

ware. Consequently, based on the above observations, we can conclude that an ultimate solution would be an approach which combines the high-level IF and flexibility of GPP-based systems with the unmatched performance of GPUs and FPGAs, while at the same constraining power consumption.

3

Custom Architecture for Immersive-Audio Applications

In this chapter, we propose a processor architecture for BF and WFS applications. The supporting programming paradigm considers a distributed memory hierarchy and allows a high-level interaction with reconfigurable Multi-Core Processors (r-MCPs) and off-the-shelf non-reconfigurable Multi-Core Processors (nr-MCPs), like GPUs. This means that the same program, with slight modifications, can be mapped onto different platforms, thus providing a versatile solution that is applicable to various immersive-audio systems.

The chapter has been organized as follows. In Section 3.1 we present the proposed architecture. Section 3.2 describes the application of our architecture to reconfigurable devices, while in Section 3.3 we present its application to multi-core processors. Section 3.4 and Section 3.5 demonstrate how the proposed architecture can be used to develop programs using a high-level interface for BF and WFS applications mapped onto r-MCPs and nr-MCPs respectively. Finally, Section 3.6 concludes the chapter.

3.1 Instruction Set Architecture Definition

The design of BF and WFS systems requires various tests before their final implementation. Regarding the BF, based on the size of the recording area and the hardware costs limitations, the designer has to evaluate the SNR quality of the extracted sources under different number of microphones. Furthermore, internal signal calculations, except filtering, may also require decimation and interpolation. Based on the available hardware resources, the designer should carefully evaluate the size and the filtering coefficients of each one of these modules. For example, one coefficient set can provide a better filtered signal

than another set, under the same number of filter taps. It is important for the designer to have the option to rapidly change and evaluate each filter coefficients set. In addition, many tests should be conducted to decide the number of source apertures that the recording area should be divided into. Such tests, when developing a software beamformer, are easily applicable, however it is not the case when custom hardware solutions are required. In the latter case, the designers should also be able to perform easily tests under different source apertures.

Similarly, the design and implementation of a WFS-based audio system requires many tests before it can be properly used. Based on the size of the listening area, the designer has to conduct experiments under different loudspeaker setups and evaluate the proper localization of the rendered sources. A larger loudspeaker array can approximate more precisely the original sound waves than a smaller one, however it requires additional processing resources. Based on the available hardware resources, the designer should carefully decide the size of the loudspeaker array. Furthermore, a source signal quality is directly affected by the selected filter coefficients, thus the designer should have the option to rapidly perform trials under different filter sizes and coefficient sets.

The main goal of the proposed architecture is to provide a certain instruction set that will allow easy customization of many vital system parameters, efficient audio data processing, and system debugging through a high-level interface. Furthermore, these instructions should be platform-independent and hide any platform-specific implementation details, thus allowing the same program to be executed to different hardware devices with minimal software changes. After studying both BF and WFS applications, we concluded to the following requirements regarding the kind of instructions that the programmer should have access to.

1. *Enable or disable input/output channels.* In order to allow easy and fast input/output channels tests, we decided to provide instructions that would allow the programmer to disable/enable them in any arbitrary way. Consequently, they would assist on rapidly fine-tuning and deciding the number of required input/output channels for the entire system.
2. *System configuration.* It is very important to provide instructions to the user that would allow a high-level configuration of many key system parameters. This way, all implementation details for system customization can be hidden, thus assisting on easy and rapid development. Examples of these parameters can be the size and coefficient sets specification for

various digital filters, and the loudspeaker coordinates inside the listening area.

3. *Efficient audio data processing*. A specific instruction subset is required to control data processing of audio samples. Although all computations will be performed in parallel from many processing elements, the user should be completely isolated from any platform-specific details. Moreover, a simple high-level interface should provide all required parameters, which internally will initiate massively parallel audio data processing.
4. *Debugging capabilities*. Efficient system debugging considerably assists on rapid development. For this reason, an instruction that would allow the user to check important system parameters should be supported by our architecture.

Taking into account the aforementioned requirements, the proposed architecture consists of 13 high-level instructions that can be used to configure an immersive-audio system, and start, manage or stop processing of input and output data. Furthermore, there is an additional 14th instruction that can be used for debugging purposes. In order to support many system setups, we provide a versatile environment that allows the adjustment of various parameters. For example, an audio acquisition module that utilizes the BF technique, may have any number of source apertures that can be identified. Furthermore, the BF FIR filters can consist of any number of taps. Similarly, an audio rendering module that utilizes the WFS technique, may support different sizes of loudspeakers arrays. In addition, the WFS FIR filter can consist of any number of taps. For these reasons, the proposed programming architecture was designed in such a way, that the programmer can conveniently change these vital system parameters.

Table 3.1 shows the 14 instructions, divided into four categories, namely *I/O*, *system setup*, *data processing* and *debug*. The *I/O* instructions are used to enable or disable audio streaming to input/output processing units. The *system setup* instructions are used to customize system parameters and configure the filter coefficients. The *data processing* instructions are used to process input/output audio samples. Finally, the instruction that belongs to the *debug* category, provides an interface to the programmer to read all system parameters, which are stored to a Special Purpose Register File (SPR), as described in Section 3.2.

In the following, we describe each of the instructions. Throughout our description, we consider the utilization of multiple BF and WFS processing units that

Table 3.1: Instructions for BF and WFS applications.

Instruction type	Full name	Mnemonic	Platform	Algorithm
I/O	Input Stream Enable	InStreamEn	r-MCP, nr-MCP	BF
	Output Stream Enable	OutStreamEn	r-MCP, nr-MCP	WFS
System setup	Clear SPRs	ClrSPRs	r-MCP	BF, WFS
	Declare FIR Filter	DFirF	r-MCP, nr-MCP	BF, WFS
	Set Samples Addresses	SSA	r-MCP	BF, WFS
	Clear RU buffers	ClrRUBuf	r-MCP	WFS
	Store coordinates	StC	r-MCP, nr-MCP	WFS
	Buffer Coefficients	BufCoef	r-MCP	BF, WFS
	Load Coefficients	LdCoef	r-MCP, nr-MCP	BF, WFS
	Configure loudspeakers	ConfL	r-MCP, nr-MCP	WFS
	Configure microphones	ConfC	r-MCP, nr-MCP	BF
Data processing	Beamform Source	BFSrc	r-MCP, nr-MCP	BF
	Render Source	RenSrc	r-MCP, nr-MCP	WFS
Debug	Read SPR	RdSPR	r-MCP	BF, WFS

process data concurrently. The *Platform* column provides the platform that an instruction can be applied, namely r-MCPs and nr-MCPs. The *Algorithm* column shows the algorithm that each instruction is applicable, namely BF and WFS.

InStreamEn: Enables or disables streaming of audio samples from input channels to the BF processing units.

OutStreamEn: Enables or disables streaming of audio samples from the WFS processing units to the output channels.

ClrSPRs: Clears the contents of all Special Purpose Registers (SPRs) that are used in r-MCP-based approaches.

DFirF: Declares the size of a filter to the system.

SSA: Regarding the BF, it specifies the addresses from where the input samples are read. Regarding the WFS, it specifies the addresses from where the source samples are read and where the output data are written.

ClrRUBufs: Applicable to r-MCP-based approaches; clears the contents of all on-chip buffers that are currently configured to the system.

StC: Transfers the loudspeakers coordinates from external memory to local memory.

BufCoef: Applicable to r-MCP-based approaches; regarding the BF, it fetches all decimator and interpolator coefficients from external memory to on-chip buffers. Regarding the WFS, it fetches all WFS filter coefficients from the

external memory to an on-chip buffer.

LdCoef: Regarding the BF, it distributes all required coefficients to the corresponding filters in the system. Regarding the WFS, it loads the WFS filter coefficients to the 3dB/octave correction FIR filter of the system.

ConfL: Regarding the r-MCP-based approaches, it defines the number of loudspeakers that will be processed from a single WFS processing unit. Regarding the nr-MCP-based approaches, it defines the total number of loudspeaker channels.

ConfC: Defines the number of input channels that are available to the system.

BFSrc: Processes a 1024-sample chunk of streaming data from each input channel, in order to extract an audio source.

RenSrc: Processes a 1024-sample chunk of streaming source data and generates 1024 samples for each loudspeaker channel.

RdSPR: Applicable to r-MCP-based approaches; used for debugging purposes and allows the programmer to read any of the SPRs.

3.2 r-MCPs Implementation

The application of our proposal to reconfigurable devices comprises dedicated register organization and distributed memory buffers. An important feature of the architecture is that it is based on a multi-core processing paradigm. This allows the design of scalable micro-architectures, with respect to the available hardware resources, which makes the architecture suitable for reconfigurable implementations and multi-core hardware platforms.

Memory and registers organization for r-MCP-based applications: Figure 3.1 illustrates the logical organization of the memory and the registers of the proposed architecture when utilizing the BF technique. It is assumed that it operates as an architectural extension of a GPP in a co-processor paradigm. The architecture assumes multi-core processing, distributed among *C* processing modules that process data from *C* input channels. The *C* parameter can be determined both at design-time and at run-time. The latter option makes it suitable for implementations on platforms with partial configuration capabilities. The host GPP and the *Multi-Core BeamForming Processor* (MC-BFP) exchange synchronization parameters and memory addresses via a set of SPRs, shown in Table 3.3. Each *BeamFormer* module has an on-chip *BF buffer* and memory space for the decimator and H(z) filters coefficients. Furthermore,

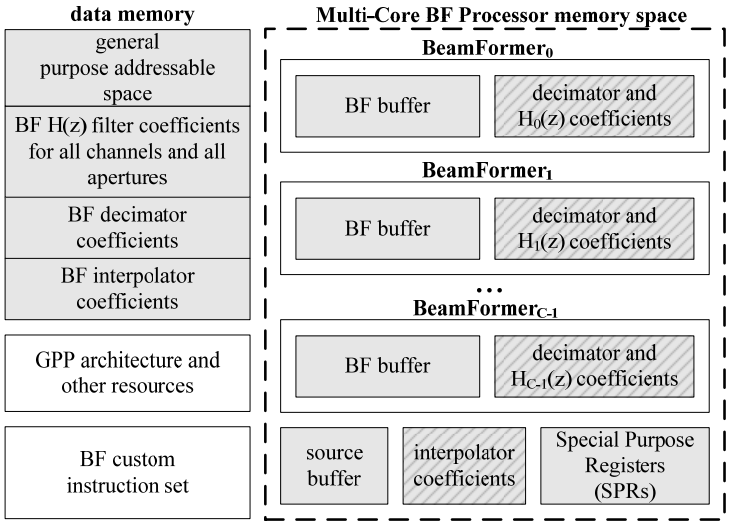

Figure 3.1: Memory organization for BF applications when utilizing r-MCPs.

there is also an on-chip *source buffer*, where samples of an extracted source are stored, and a memory space for the currently active coefficients set of the interpolator.

The globally-shared and locally-distributed memory organization that is considered by the proposed architecture is the user-accessible memory space, as illustrated in Figure 3.1. The non-user addressable space is annotated with the stripe pattern. In order to provide a high-level programming environment, the programmer has read and write access to the *BF buffers*, the *source buffer*, the external memory and the GPP on-chip memory. Furthermore, the programmer can only read from the SPRs for debugging purposes. There is no direct access to the memory space for the coefficients, since our architecture provides the functionality to reload all required coefficients from on-chip *BF buffers* to the decimators, H(z) filters and interpolator. This way the user avoids completely any low-level interaction with the hardware platform.

A similar approach has been followed for the WFS memory organization. Figure 3.2 shows the logical organization of the memory and the registers. In this case, the architecture assumes multi-core processing, distributed among R Rendering Units (RUs), each processing data for L/R output channels, where L is the total number of loudspeakers. As it was in the case of BF, the R parameter can be also configured both at design-time and at run-time, thus makes it suitable for implementations on platforms with partial configuration capabili-

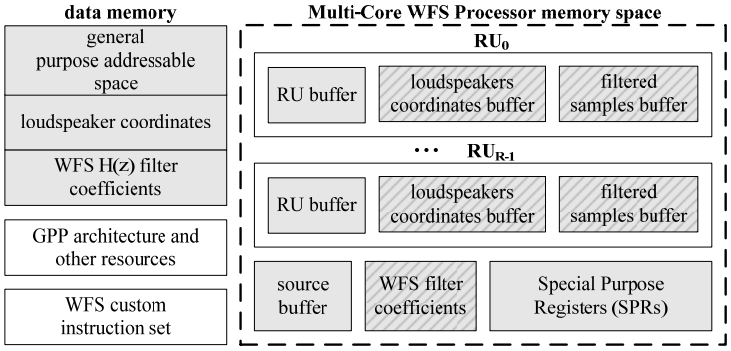

Figure 3.2: Memory organization for WFS applications when utilizing r-MCPs.

ties. The host GPP and the *Multi-Core WFS Processor* (MC-WFSP) exchange memory addresses and synchronization parameters using a set of SPRs, as depicted in Table 3.4. Each RU has its own on-chip *RU buffer* and memory space for the loudspeakers coordinates and filtered source samples, namely *loudspeakers coordinates buffer* (LCB) and *filtered samples buffer* (FSB) respectively. Furthermore, there is also an on-chip *source buffer*, where samples of an acoustic source are stored, and a memory space for the currently active coefficients set of the WFS filter.

Immersive-audio instruction set for r-MCPs: Table 3.2 shows the parameters of each instruction when applying our architecture to r-MCPs. In the following, we describe how these instruction parameters can be used to access the aforementioned memory organization.

InStreamEn: Its parameter is a binary mask *b_mask* equal to the number of input channels *C*. Within the mask, each bit can be used from the programmer to disable or enable channel streaming by setting 0 or 1 to its value respectively. The binary mask is stored in SPR0, as shown in Table 3.3.

OutStreamEn: Its parameter is a binary mask *b_mask* equal to the number of RUs *R*. Within the mask, each bit can be used from the programmer to disable or enable RU streaming by setting 0 or 1 to its value respectively. The binary mask is stored in SPR0, as shown in Table 3.4.

ClrSPRs: Does not require any parameters.

DFirF: Writes the size of a filter to the corresponding SPR. Its parameters are the filter size *FSize* and its type *FType*. The latter is used to distinguish among the four different filter types, which are decimator (*FType* = 1), interpolator (*FType* = 2), H(z) filter (*FType* = 3) and WFS filter (*FType* = 4). Based on

Table 3.2: Instructions parameters for architecture application on r-MCPs.

Instruction type	Mnemonic	Parameters	Algorithm
I/O	InStreamEn	*b_mask*	BF
	OutStreamEn	*b_mask*	WFS
System setup	ClrSPRs	NONE	BF, WFS
	DFirF	FSize, FType	BF, WFS
	SSA	buf_sam_addr	BF, WFS
	ClrRUBuf	RUs_addresses	WFS
	StC	xmem_spkr_coordinates, buf_spkr_coordinates	WFS
	BufCoef	xmem_coef_addr, buf_coef_addr	BF, WFS
	LdCoef	buf_coef_addr	BF, WFS
	ConfL	spkr_num	WFS
	ConfC	C	BF
Data processing	BFSrc	*aper*, xmem_read_addr, xmem_write_addr	BF
	RenSrc	Srcx1y1, Srcx2y2, RUs_addresses, source_id_num, xmem_read_addr, xmem_write_addr	WFS
Debug	RdSPR	SPR_num	BF, WFS

Table 3.3: Special Purpose Registers mapping for BF.

SPR	Description
SPR0	InStreamEn binary mask
SPR1	Decimators FIR filter size
SPR2	Interpolators FIR filter size
SPR3	H(z) FIR filter size
SPR4	LdCoef start/done flag
SPR5	aperture address offset
SPR6	BFSrc start/done flag
SPR7	source buffer address
SPR8	interpolator coefficients address
SPR9	number of input channels (C)
SPR10 - SPR[9+C]	channel i coefficients buffer address, i=0...*C*-1
SPR[10+C] - SPR[9+2·C]	channel i 1024 samples buffer address, i=0...*C*-1

the value of *FType*, this instruction writes the filter size to the appropriate SPR ranging from SPR1 to SPR3 for the BF and only to SPR1 for the WFS, as shown in Table 3.3 and Table 3.4 respectively.

SSA: Regarding the BF, it specifies the addresses from where the MC-BFP will read the input samples. Its parameter is an array of pointers *buf_sam_addr* to the starting address of all on-chip *BF buffers*. *SSA* writes from SPR[10+C] to SPR[9+2·C] the on-chip buffers starting addresses. Furthermore, it writes to SPR7 the *source buffer* address, where 1024 samples of the extracted source signal are stored, as shown in Table 3.3. Regarding the WFS, it specifies the ad-

Table 3.4: Special Purpose Registers mapping for WFS.

SPR	Description
SPR0	OutStreamEn binary mask
SPR1	WFS filter size
SPR2	Loudspeakers per RU
SPR3	StC start/done flag
SPR4	LdCoef start/done flag
SPR5	x1, y1 source coordinates
SPR6	x2, y2 source coordinates
SPR7	source buffer address
SPR8	WFS filter coefficients address
SPR9	RenSrc start/done flag
SPR10	source ID number
SPR11 - SPR[10+R]	loudspeakers coordinates address inside RU_i buffer, i=0,...,R-1
SPR[11+R] - SPR[10+2·R]	loudspeakers samples address inside RU_i buffer, i=0,...,R-1

dresses from where the MC-WFSP will read the source samples and write the output data. Its parameter is an array of pointers *buf_sam_addr* to the starting address of the *source buffer* and all *RU buffers*. *SSA* writes from SPR[11+R] to SPR[10+2·R] the *RU buffers* starting addresses. Also, it writes to SPR7 the *source buffer* address, where 1024 samples of the source signal are stored, as shown in Table 3.4.

ClrRUBufs: Clears the contents of all *RU buffers* that are currently configured to the system. Its parameter is an array *RUs_addresses* of pointers to each *RU buffer*. No SPR is modified during its execution.

StC: Reads the loudspeakers coordinates from the external memory, re-arranges their order based on the number of RUs at the system, and writes them to each *RU buffer*. Its parameters are the external memory address *xmem_spkr_coordinates* where the loudspeakers coordinates are stored, and an array of pointers *buf_spkr_coordinates* to the on-chip *RU buffers*. The instruction uses SPR3 to communicate with the MC-WFSP and writes to SPR11 - SPR[10+R] the address within each *RU buffer* where the arranged coordinates will be stored, as shown in Table 3.4.

BufCoef: Regarding the BF, it fetches all decimator and interpolator coefficients from external memory to on-chip *BF buffers*. Its parameters are an array *xmem_coef_addr* of pointers to the off-chip memory starting addresses of the coefficients sets, and an array *buf_coef_addr* of pointers within the on-chip *BF buffers*, where all coefficients will be stored. *BufCoef* does not write

any values to SPRs. Regarding the WFS, it fetches all WFS filter coefficients from the external memory to the *source buffer*. Its parameters are a pointer *xmem_coef_addr* to the off-chip memory starting addresses of the coefficients set, and a pointer *buf_coef_addr* within the *source buffer* where all coefficients will be stored. As before, *BufCoef* does not write any values to SPRs.

LdCoef: Regarding the BF, it distributes all decimator and interpolator coefficients to the corresponding filters in the system. Its parameter is an array *buf_coef_addr* of pointers within the on-chip buffers where all coefficients are stored. These addresses are written from SPR10 to SPR[9+C], as explained in Table 3.3. The instruction also writes to SPR8 the on-chip address of the interpolator coefficients from where the MC-BFP can read them. The coefficients distribution is initiated when a start flag is written to SPR4. Once all filter coefficients are transferred, *LdCoef* writes a done flag to SPR4, as shown in Table 3.3. Regarding the WFS, it loads the WFS filter coefficients to the 3dB/octave correction FIR filter of the system. Its parameter is a pointer *buf_coef_addr* within the *source buffer* where all coefficients are stored. This address is written to SPR8. As soon as all coefficients are transferred to the *source buffer*, their distribution is initiated when a start flag is written to SPR4. Once all coefficients are reloaded to the filter, *LdCoef* writes a done flag to SPR4, as shown in Table 3.4.

ConfL: Its parameter is the number of loudspeakers per RU *spkr_num* that will be enabled using *OutStreamEn*. The instruction writes the value of *spkr_num* to SPR2, as shown in Table 3.4.

ConfC: Its parameter is the number of active input channels C that will be enabled using *InStreamEn*. The instruction writes the value of C to SPR9, as shown in Table 3.3.

BFSrc: It requires as parameters the current source aperture *aper*, the starting read address from the external memory *xmem_read_addr* of the current chunk, and the write address to the external memory *xmem_write_addr*, where 1024 samples of the source signal will be stored. Based on *aper*, *BFSrc* writes to SPR5 an on-chip *BF buffer* address offset that allows the correct selection of $H_i(z)$ coefficients sets. In order to initiate processing, the instruction writes a start flag to SPR6. This flag is read by each $BeamFormer_i$ module, where i=0,..., C-1, thus channel processing is performed concurrently. Once all data calculations are finished, a done flag is written to SPR6, as shown in Table 3.3.

RenSrc: It requires as parameters the source coordinates (x1,y1) and (x2,y2), which designate the initial and final source location within the listening area respectively for a $\frac{1024}{f_s}$ time interval (f_s is the sampling frequency),

and are stored to variables *Srcx1y1* and *Srcx2y2*. Also it requires an array *RUs_addresses* of pointers to each *RU buffer*, the source identification number *source_id_num*, the starting read address from the external memory *xmem_read_addr* of the current chunk, and the write address to the external memory *xmem_write_addr*, where 1024·L output samples will be stored. In order to initiate processing, the instruction writes a start flag to SPR9. This flag is read by each RU, thus loudspeaker processing is performed concurrently. Once all data calculations are finished, a done flag is written to SPR9, as shown in Table 3.4.

RdSPR: It requires as parameter the number of SPR *SPR_num* that needs to be read.

3.3 nr-MCPs Implementation

In this section, we describe the logical memory organization and the instructions parameters that can be used, in order to apply our architecture to nr-MCPs. We assume that an nr-MCP is connected to a host GPP via a standard bus (e.g. PCI express [12]), and that it has its own memory hierarchy.

Memory and registers organization for nr-MCP-based applications: Figure 3.3 illustrates the logical organization of the memory when applying the proposed architecture to an nr-MCP-based platform. Again, the architecture assumes multi-core processing, distributed among a fixed number of processing modules that process data from *C* input channels and render audio through *L* loudspeakers. The host GPP and the nr-MCP exchange synchronization parameters and data via a standard bus. Furthermore, the host GPP and nr-MCP memory spaces are accessible by the programmer. However, the proposed architecture provides a high-level interaction with the nr-MCP, thus hiding any implementation details from the user on how to efficiently map the application onto the processing cores of the nr-MCP.

Immersive-audio instruction set for nr-MCPs: Table 3.5 shows the instruction parameters when applying our architecture to nr-MCPs. In the following, we describe each instruction parameter functionality.

InStreamEn: Its parameter is a mask *b_mask* equal to the number of input channels *C*. Within the mask, each digit can be used from the programmer to disable or enable input channel streaming by setting 0 or 1 to its value respectively.

OutStreamEn: Its parameter is a mask *b_mask* equal to the number of output channels. Within the mask, each digit can be used from the programmer to

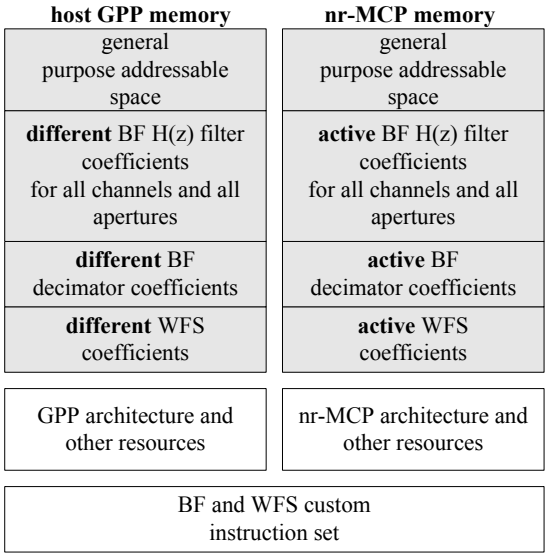

Figure 3.3: Memory organization for immersive-audio applications when utilizing an nr-MCP.

Table 3.5: Instructions parameters for architecture application on nr-MCPs.

Instruction type	Mnemonic	Parameters	Algorithm
I/O	InStreamEn	*b_mask*	BF
	OutStreamEn	*b_mask*	WFS
System setup	DFirF	FSize, FType	BF, WFS
	StC	host_spkr_coordinates, dev_spkr_coordinates	WFS
	LdCoef	coef_addr	BF, WFS
	ConfL	spkr_num	WFS
	ConfC	C	BF
Data processing	BFSrc	*aper*, coef_addr, dev_read_addr, dev_write_addr	BF
	RenSrc	source_coord, coef_addr, spkr_addresses, dev_read_addr, dev_write_addr	WFS

disable or enable output channel streaming by setting 0 or 1 to its value respectively.

DFirF: Stores the filters size to internal program variables. Its parameters are the filter size *FSize* and its type *FType*. The latter is used to distinguish among the four different filter types, which are decimator (*FType* = 1), interpolator (*FType* = 2), H(z) filter (*FType* = 3) and WFS filter (*FType* = 4).

StC: Reads the loudspeakers coordinates from the host GPP external memory and writes them to the nr-MCP external memory. Its parameters are

the GPP external memory address *host_spkr_coordinates* where the loudspeakers coordinates are stored, and a nr-MCP external memory address *dev_spkr_coordinates* to the nr-MCP external memory.

LdCoef: Regarding the BF, it transfers all decimator, H(z) and interpolator coefficients from the host GPP external memory to the nr-MCP external memory. Its parameter is an array *coef_addr* of pointers to the active coefficients sets that will be used by the nr-MCP. These addresses are used to transfer the coefficient sets to the external memory of the nr-MCP. Regarding the WFS, it transfers all WFS coefficients from the host GPP external memory to the nr-MCP external memory. Its parameter is an an array of pointers *coef_addr* to the host GPP memory that points to the active coefficients sets and to the destination address of the nr-MCP external memory.

ConfL: Its parameter is the number of loudspeakers *spkr_num* that are enabled using *OutStreamEn*.

ConfC: Its parameter is the number of active input channels *C* that are enabled using *InStreamEn*.

BFSrc: It requires as parameters the current sources aperture *aper*, the starting read address from the external memory *dev_read_addr* of the current chunk, the nr-MCP external memory coefficients addresses within the *coef_addr* array of pointers, and the write address to the external memory *dev_write_addr*, where 1024 samples of the source signal will be stored.

RenSrc: It requires as parameters an array *source_coord* with the sources coordinates, which designate the initial and final sources location within the listening area for every 1024-sample chunk. Also it requires the nr-MCP external memory coefficients address within the *coef_addr* array of pointers, the starting read address from the external memory *dev_read_addr* of the current chunk, and the write address to the external memory *dev_write_addr*, where 1024·L output samples will be stored.

3.4 Programming Paradigm for r-MCPs

BF programming paradigm: In Algorithm 3.1, we illustrate through pseudocode how to setup a BF system to extract an audio source when mapped onto r-MCPs. The *DISABLE_INPUTS_MASK* and *ENABLE_INPUTS_MASK* are binary masks that are used to disable or enable input channels, as described above. The *DECIMATOR_SIZE*, *H_SIZE* and *INTERPOLATOR_SIZE* variables are used to configure the decimator, H(z) and interpolator FIR fil-

ter sizes. Moreover, the *DECIMATOR_TYPE*, *H_TYPE* and *INTERPOLATOR_TYPE* variables are used to specify the filter type. *samples_addr* is an array of pointers to each on-chip *BF buffer*, where a 1024-sample chunk is stored. *coef_xmem_addr* is an array of pointers to the external memory where all required decimator, H(z) filters and interpolators coefficients are stored, and *buf_addr* is an array of destination pointers to on-chip *BF buffers*, where all coefficients will be transferred. *xmem_rd_addr* and *xmem_wr_addr* are pointers to the external memory that read input channels data and write source samples respectively. *INPUT_DATA_XMEM_ADDR* is an external memory address, where input channels data are stored, while *OUTPUT_DATA_XMEM_ADDR* is an external memory address, where samples of extracted sources are written back. Finally, *aper* is the current source aperture.

The pseudocode starts in line 2 by using the *ConfC* instruction to configure the number *C* of currently available input channels to the BF system. In line 4, all input channels are disabled from processing using the *InStreamEn*, since the system is not yet properly setup. In line 6 all SPRs are initialized by clearing their contents. In lines 8, 10 and 12, the *DFirF* instruction is used to configure the decimator, H(z) and interpolator filter sizes. In line 14, the address of all samples within the on-chip buffers are specified, using the *SSA* instruction. In line 16, the *BufCoef* distributes all required decimator and interpolators coefficients from the external memory to on-chip *BF buffers*. Once it is done, in line 18, the *LdCoef* instruction is used to configure the decimators and interpolator coefficients. In line 20, *xmem_rd_addr* is initialized pointing at the input data stored in the external memory. In line 21, *xmem_wr_addr* points to the external memory address, where the extracted source samples are stored. Once the system is properly configured, all BeamFormers are enabled in line 23. Finally, in each iteration of the while-loop in line 25, the current source aperture *aper* is used and 1024·C samples are read to extract 1024 source samples using the *BFSrc*, which are written to the external memory. The *xmem_rd_addr* and the *xmem_wr_addr* pointers are increased by 1024·C and 1024 entries respectively to properly point to the required input and output external memory locations for the next iteration.

It should be noted that any time, the designer can use the *RdSPR* instruction for debugging purposes. Also, if the user wants to perform additional experiments under different number of microphones, it can be done by reconfiguring the system using the *ConfC* instruction. Furthermore, in case the designer wishes during run-time to test different coefficients sets or increase/decrease the total number of source apertures, it can be done by just providing a new array of pointers to the *BufCoef* instruction. The *LdCoef* can then be used to reload

Algorithm 3.1 Pseudocode for BF when mapped onto r-MCPs.

```
1: {configure the number of input channels available}
2: ConfC (C);
3: {disable all BeamFormers until system is configured}
4: InStreamEn (DISABLE_INPUTS_MASK);
5: {clear the contents of all SPRs}
6: ClrSPRs ();
7: {configure decimators size}
8: DFirF (DECIMATOR_SIZE, DECIMATOR_TYPE);
9: {configure H(z) filters size}
10: DFirF (H_SIZE, H_TYPE);
11: {configure interpolator size}
12: DFirF (INTERPOLATOR_SIZE,
    INTERPOLATOR_TYPE);
13: {configure the samples addresses}
14: SSA (samples_addr);
15: {transfer all H(z) coefficients to on-chip buffers}
16: BufCoef (coef_xmem_addr, buf_addr);
17: {load the coefficients to all decimators and interpolator}
18: LdCoef (buf_addr);
19: {initialize external memory reading and writing pointers}
20: xmem_rd_addr=INPUT_DATA_XMEM_ADDR;
21: xmem_wr_addr=OUTPUT_DATA_XMEM_ADDR;
22: {enable BeamFormers}
23: InStreamEn (ENABLE_INPUTS_MASK);
24: {process streaming data}
25: while (1) do
26:     BFSrc (aper, coef_addr, xmem_rd_addr, xmem_wr_addr);
27:     {update external memory pointers}
28:     xmem_rd_addr=xmem_rd_addr+1024·C;
29:     xmem_wr_addr=xmem_wr_addr+1024;
30: end while
```

all decimators and interpolator with the new coefficients sets, while the *BFSrc* instruction will extract sources based on the new H(z) coefficients sets.

WFS programming paradigm: In Algorithm 3.2, we illustrate through pseudocode how to setup a WFS system to render an audio source when mapped onto r-MCPs. The *DISABLE_RUS_MASK* and *ENABLE_RUS_MASK* are bi-

nary masks that are used to disable and enable the RUs that are present to the system. The *WFS_FILTER_SIZE* parameter represents the size of the WFS FIR filter. The *samples_addr* array keeps the starting address of the source and all *RU buffers*. *coord_xmem_addr* is the external memory address where the loudspeakers coordinates are stored. *speakers_coord* is an array of pointers to each *RU buffer*. *coef_xmem_addr* is the external memory address where the filter coefficients are stored. *buf_addr* is a pointer to the *source buffer*. The *x1y1* and *x2y2* variables provide the source coordinates for a particular processing iteration. *RUs_addr* is an array of pointers to the *RU buffers*, while *source_id* is the acoustic source identification number. *xmem_rd_addr* and *xmem_wr_addr* are pointers to the external memory that read source data and write loudspeakers samples respectively. *INPUT_DATA_XMEM_ADDR* is an external memory address, where source data are stored, while *OUTPUT_DATA_XMEM_ADDR* is an external memory address, where loudspeakers samples are written back.

The pseudocode starts in line 2 by configuring the number of loudspeakers that will be processed per RU. In line 4, all RUs are disabled until the system is properly configured. In line 6, all SPRs are cleared, while in line 8 *DFirF* is used to customize the filter size. In line 10, *SSA* specifies from which address source samples are read and destination addresses where output data will be written. In line 12, all loudspeakers coordinates are read from the external memory and distributed accordingly to each *RU buffer*. In line 14, *BufCoef* is used to load all filter coefficients from the external memory to the *source buffer*, from the *LdCoef* in line 16 reads them to re-load the FIR filter. In line 18 and 19, both external memory read and write pointers are initialized. Once the system is properly configured, in line 21, all RUs are enabled, and in line 24, the *RenSrc* instruction renders an acoustic source based on its current coordinates. Both external memory read and write pointers are properly updated in lines 26 and 27, in order to process new incoming samples.

As it was mentioned before, the designer can again use the *RdSPR* instruction for debugging purposes. Also, if the user wants to perform additional experiments under different number of loudspeakers, it can be done by reconfiguring the system using the *ConfL* instruction. Furthermore, in case the designer wishes to test different coefficients sets, it can be done by just providing a new array of pointers to the *BufCoef* instruction. The *LdCoef* can then be used to reload the WFS filter with the new coefficients sets, while the *RenSrc* instruction will render sources based on the new coefficients set.

Algorithm 3.2 Pseudocode for WFS when mapped onto r-MCPs.

```
1: {configure the number of loudspeakers per RU}
2: ConfL (L/R);
3: {disable all RUs until system is configured}
4: OutStreamEn (DISABLE_RUS_MASK);
5: {clear the contents of all SPRs}
6: ClrSPRs ();
7: {configure WFS filter size}
8: DFirF (WFS_FILTER_SIZE);
9: {specify from where source samples are read and where output data will
   be written}
10: SSA (samples_addr);
11: {read the coordinates from SDRAM and distribute them to RU buffers}
12: StC (coord_xmem_addr,speakers_coord);
13: {transfer all WFS coefficients to source buffer}
14: BufCoef (coef_xmem_addr,buf_addr);
15: {load the coefficients to the WFS FIR filter}
16: LdCoef (buf_addr);
17: {initialize external memory reading and writing pointers}
18: xmem_rd_addr=INPUT_DATA_XMEM_ADDR;
19: xmem_wr_addr=OUTPUT_DATA_XMEM_ADDR;
20: {enable RUs}
21: OutStreamEn (ENABLE_RUS_MASK);
22: {process streaming data}
23: while (1) do
24:     RenSrc (x1y1, x2y2, RUs_addr, source_id,
        xmem_rd_addr, xmem_wr_addr);
25:     {update external memory pointers}
26:     xmem_rd_addr=xmem_rd_addr+1024;
27:     xmem_wr_addr=xmem_wr_addr+1024·R;
28: end while
```

3.5 Programming Paradigm for nr-MCPs

BF programming paradigm: In Algorithm 3.3, we illustrate through pseudocode how to setup a BF system to extract SOURCES audio sources when mapped onto nr-MPCs. The *DISABLE_INPUTS_MASK* and *ENABLE_INPUTS_MASK* are masks that are used to disable or enable input chan-

nels, as described above. The *DECIMATOR_SIZE*, *H_SIZE* and *INTERPOLATOR_SIZE* variables are used to configure the decimator, H(z) and interpolator FIR filter sizes. Moreover, the *DECIMATOR_TYPE*, *H_TYPE* and *INTERPOLATOR_TYPE* variables are used to specify the filter type. *coef_addr* is an array of pointers to the host GPP external memory where all required decimator, H(z) filters and interpolators coefficients are stored. Furthermore, it also includes the pointers where all aforementioned coefficient sets will be stored inside the nr-MCP external memory. *xmem_rd_addr* and *xmem_wr_addr* are pointers to the external memory that read input channels data and write source samples respectively. *INPUT_DATA_XMEM_ADDR* is an external memory address, where input channels data are stored, while *OUTPUT_DATA_XMEM_ADDR* is an external memory address, where samples of extracted sources are written back. Finally, *aper* are the current source apertures.

The pseudocode starts in line 2 by using the *ConfC* instruction to configure the number *C* of currently available input channels to the BF system. In line 4, all input channels are disabled from processing using the *InStreamEn*, since the system is not yet properly setup. In lines 6, 8 and 10, the *DFirF* instruction is used to configure the decimator, H(z) and interpolator filter sizes. In line 12, the *LdCoef* instruction is used to copy the decimators, H(z) and interpolator coefficients to the PCM external memory. In line 14, *xmem_rd_addr* is initialized pointing at the input data stored in the external memory. In line 15, *xmem_wr_addr* points to the external memory address, where all extracted source samples are stored. Once the system is properly configured, all BeamFormers are enabled in line 17. Finally, in each iteration of the while-loop in line 19, the current source apertures *aper* are used and 1024·C samples are read to extract 1024·SOURCES source samples using the *BFSrc*, which are written to the external memory. The *xmem_rd_addr* and the *xmem_wr_addr* pointers are increased by 1024·C and 1024·SOURCES entries respectively to properly point to the required input and output external memory locations for the next iteration.

WFS programming paradigm: In Algorithm 3.4, we demonstrate through pseudocode how to setup a WFS system to render SOURCES audio sources when mapped onto nr-MCPs. The *DISABLE_OUTPUTS_MASK* and *ENABLE_OUTPUTS_MASK* are masks that are used to disable and enable the system output channels. The *WFS_FILTER_SIZE* parameter represents the size of the WFS FIR filter. *spkr_addr* is the external memory address where the loudspeakers coordinates are stored. *speakers_coord* is a pointer where the loudspeakers coordinates will be stored in the nr-MCP external memory. *coef_addr*

Algorithm 3.3 Pseudocode for BF when mapped onto nr-MCPs.

```
1: {configure the number of input channels available}
2: ConfC (C);
3: {disable all BeamFormers until system is configured}
4: InStreamEn (DISABLE_INPUTS_MASK);
5: {configure decimators size}
6: DFirF (DECIMATOR_SIZE, DECIMATOR_TYPE);
7: {configure H(z) filters size}
8: DFirF (H_SIZE, H_TYPE);
9: {configure interpolator size}
10: DFirF (INTERPOLATOR_SIZE, INTERPOLATOR_TYPE);
11: {transfer the decimators, H(z) and interpolator coefficients to the nr-MCP
    external memory}
12: LdCoef (coef_addr);
13: {initialize external memory reading and writing pointers}
14: xmem_rd_addr=INPUT_DATA_XMEM_ADDR;
15: xmem_wr_addr=OUTPUT_DATA_XMEM_ADDR;
16: {enable BeamFormers}
17: InStreamEn (ENABLE_INPUTS_MASK);
18: {process streaming data}
19: while (1) do
20:     BFSrc (aper, coef_addr, xmem_rd_addr, xmem_wr_addr);
21:     {update external memory pointers}
22:     xmem_rd_addr=xmem_rd_addr+1024·C;
23:     xmem_wr_addr=xmem_wr_addr+1024·SOURCES;
24: end while
```

is any array of pointers inside the external memory address where the filter coefficients are stored, and the nr-MCP external memory address where the filter coefficients will be stored. The *source_coord* provides the sources coordinates for a particular processing iteration. *INPUT_DATA_XMEM_ADDR* is an external memory address, where source data are stored, while *OUTPUT_DATA_XMEM_ADDR* is an external memory address, where loudspeakers samples are written back.

The pseudocode starts in line 2 by declaring the available number L of loudspeakers to the system. In line 4, all output channels are disabled until the system is properly configured. In line 6, the *DFirF* is used to customize the filter size. In line 8, all loudspeakers coordinates are read from the external mem-

Algorithm 3.4 Pseudocode for WFS when mapped onto nr-MCPs.

```
1: {configure the number of loudspeakers of the system}
2: ConfL (L);
3: {disable all channels until system is configured}
4: OutStreamEn (DISABLE_OUTPUTS_MASK);
5: {configure WFS filter size}
6: DFirF (WFS_FILTER_SIZE);
7: {read the loudspeaker coordinates from the host GPP memory and copy
   them to the nr-MCP memory}
8: StC (spkr_addr, speakers_coord);
9: {transfer all WFS coefficients to source buffer}
10: LdCoef (coef_addr);
11: {initialize external memory reading and writing pointers}
12: xmem_rd_addr=INPUT_DATA_XMEM_ADDR;
13: xmem_wr_addr=OUTPUT_DATA_XMEM_ADDR;
14: {enable output channels}
15: OutStreamEn (ENABLE_OUTPUTS_MASK);
16: {process streaming data}
17: while (1) do
18:     RenSrc (source_coord, coef_addr, speakers_coord
        xmem_rd_addr, xmem_wr_addr);
19:     {update external memory pointers}
20:     xmem_rd_addr=xmem_rd_addr+1024·SOURCES;
21:     xmem_wr_addr=xmem_wr_addr+1024·L;
22: end while
```

ory and transferred to the nr-MCP external memory. In line 10, the *LdCoef* reads the WFS filter coefficients form the external memory and transfers them to the nr-MCP external memory. In line 12 and 13, both external memory read and write pointers are initialized. Once the system is properly configured, in line 15, all output channels are enabled, and in line 18, the *RenSrc* instruction renders all acoustic sources based on their current coordinates. Both external memory read and write pointers are properly updated in lines 20 and 21, in order to process new incoming samples.

3.6 Conclusions

In this chapter, we have introduced a custom architecture for immersive-audio applications. The proposed architecture consists of high-level instructions that allow the configuration of many vital parameters of immersive-audio systems. Moreover, it is applicable to both r-MCPs and off-the-shelf nr-MCPs, which nowadays are considered the most efficient platforms for mapping applications with inherent parallelism.

Furthermore, the major advantage of our proposal is that it hides any implementation details from the programmer on how to efficiently map the applications onto the considered hardware platforms. For example, one does not have to map the design to r-MCPs for every new testing input/output channels setup, since the proposed architecture provides a high-level interface to configure them. Furthermore, our proposal does not require any environment-specific code annotations to efficiently execute the software program to the available processing cores. These advantages can be translated to less error-prone audio applications and shorter immersive-audio systems development and implementation times, since the engineer can perform system tests easier and much faster.

This chapter is based on the the following papers:

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **Minimalistic Architecture for Reconfigurable Audio Beamforming**, International Conference on Field-Programmable Technology (FPT), pp. 503-506, Beijing, China, December 2010

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A Minimalistic Architecture for Reconfigurable WFS-Based Immersive-Audio**, International Conference on ReConFigurable Computing and FPGAs (ReConfig), pp. 1-6, Cancun, Mexico, December 2010

4

Immersive-Audio Reconfigurable Micro-Architectures

In this chapter, we present the underlying multi-core micro-architectures that support the proposed architectures for the BF and WFS techniques when mapped onto reconfigurable Multi-Core Processors (r-MCPs), organically presented in Chapter 3. The chapter is organized as follows. In Section 4.1, we present the micro-architectural support of our proposed BF architecture. Section 4.2 describes the micro-architecture of the proposed WFS architecture. Finally, Section 4.3 concludes the chapter.

4.1 Reconfigurable BF Micro-Architecture

4.1.1 Multi-Core BF Micro-Architecture

Figure 4.1 illustrates the multi-core implementation of the proposed BF architecture. As it was mentioned in Section 3.1, the proposed architecture is assumed for operation as an architectural extension of a GPP using the co-processor paradigm. The architecture assumes multi-core processing, distributed among *C* processing modules that process data from *C* input channels. A *GPP Bus* is used to connect the on-chip GPP memory and external SDRAM with the GPP via a standard bus interface (BUS-IF). Furthermore, in order to accelerate data transfer from the SDRAM to on-chip buffers, we employ a *Direct Memory Access (DMA) controller*, which is also connected to the same bus. A partial reconfiguration controller is employed to provide the option of reloading the correct bitstreams based on the currently available number of input channels. All user-addressable memory spaces inside the Multi-Core BF Processor (MC-BFP), like *SPRs*, *BF buffers* and the *source buffer*, are con-

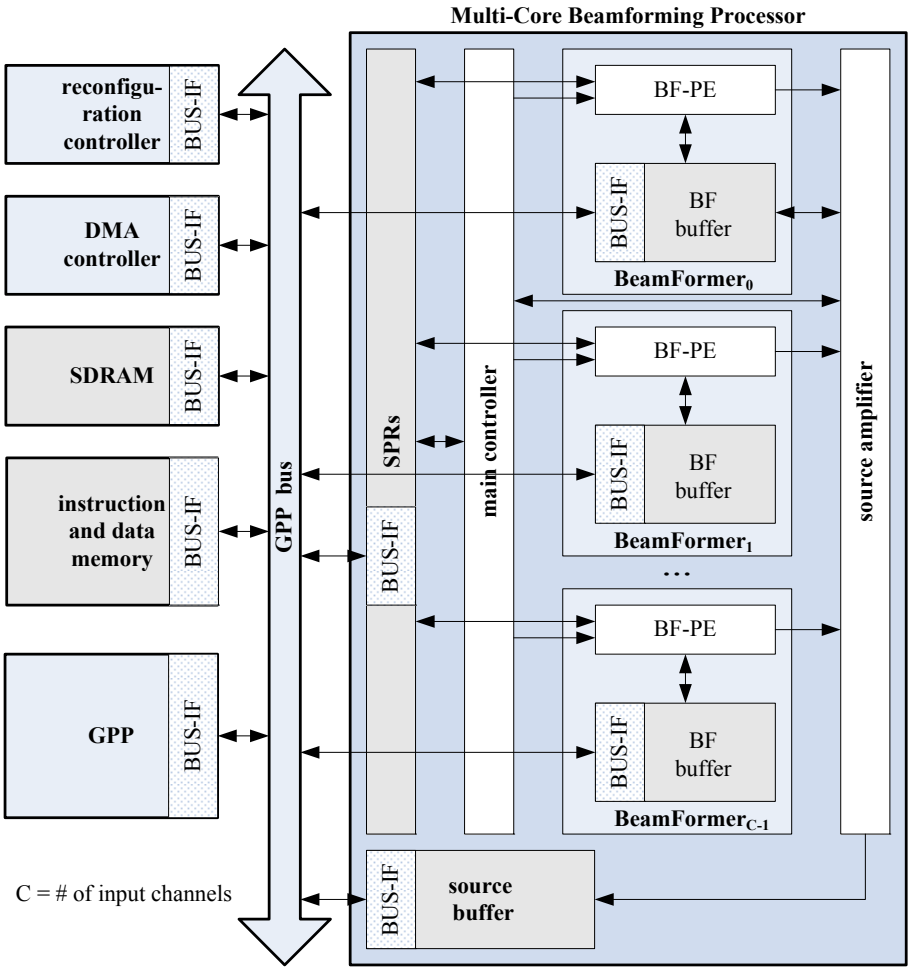

Figure 4.1: Multi-core implementation of the BF system.

nected to the *GPP Bus*. This fact enhances our architecture's flexibility, since they are directly accessible by the GPP. The *main controller* is responsible for initiating the coefficients reloading process to all decimators and the interpolator. Furthermore, it enables input data processing from all channels, and acknowledges the GPP as soon as all calculations are done.

Each *BeamFormer* module consist of a *BF buffer* and a *Beamforming Processing Element* (BF-PE), which is illustrated in Figure 4.2. As it can be seen, there is a *LdCoef controller* and a *BFSrc controller*. Based on the current source aperture, the former is responsible for reloading the required coefficient sets

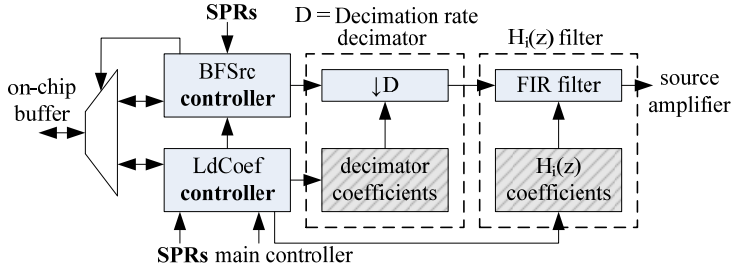

Figure 4.2: The Beamforming processing element (BF-PE) structure.

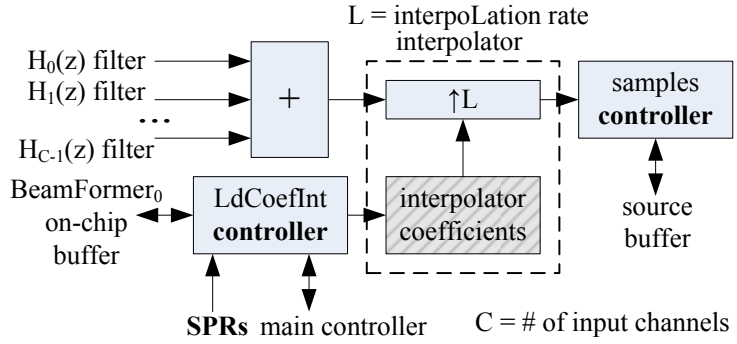

Figure 4.3: The source amplifier structure.

from the *BF buffer* to the decimator and H(z) filter. *BFSrc controller* reads 1024 input samples from the *BF buffer* and forwards them to the decimator and the H(z) filter.

All *BeamFormer* modules forward the filtered signals to the *source amplifier*, which is shown in Figure 4.3. The *LdCoefInt controller* is responsible for reloading the coefficients set to the interpolator. As we can see, all $H_i(z)$ signals, where i=0,...,C-1, are accumulated to strengthen the original acoustic source signal, which is then interpolated. Finally, the *samples controller* is responsible for writing back to the *source buffer* the interpolated source signal.

BF Data processing flow: Figure 4.4 illustrates how BF data processing is divided among *C BeamFormers*, under a *C*-sized microphone array setup. In each iteration, $1024 \cdot C$ samples are fetched from the SDRAM and stored to the on-chip *BF buffer* of each *BeamFormer*. Once data transfer is done, all *BeamFormers* start processing concurrently the audio samples. More specifically, each one of them downsamples the recorded signals by a factor *D*. The downsampled signals are forwarded to the H(z) BF filters, and all outputs are accumulated, in order to strengthen the original acoustic source. The latter

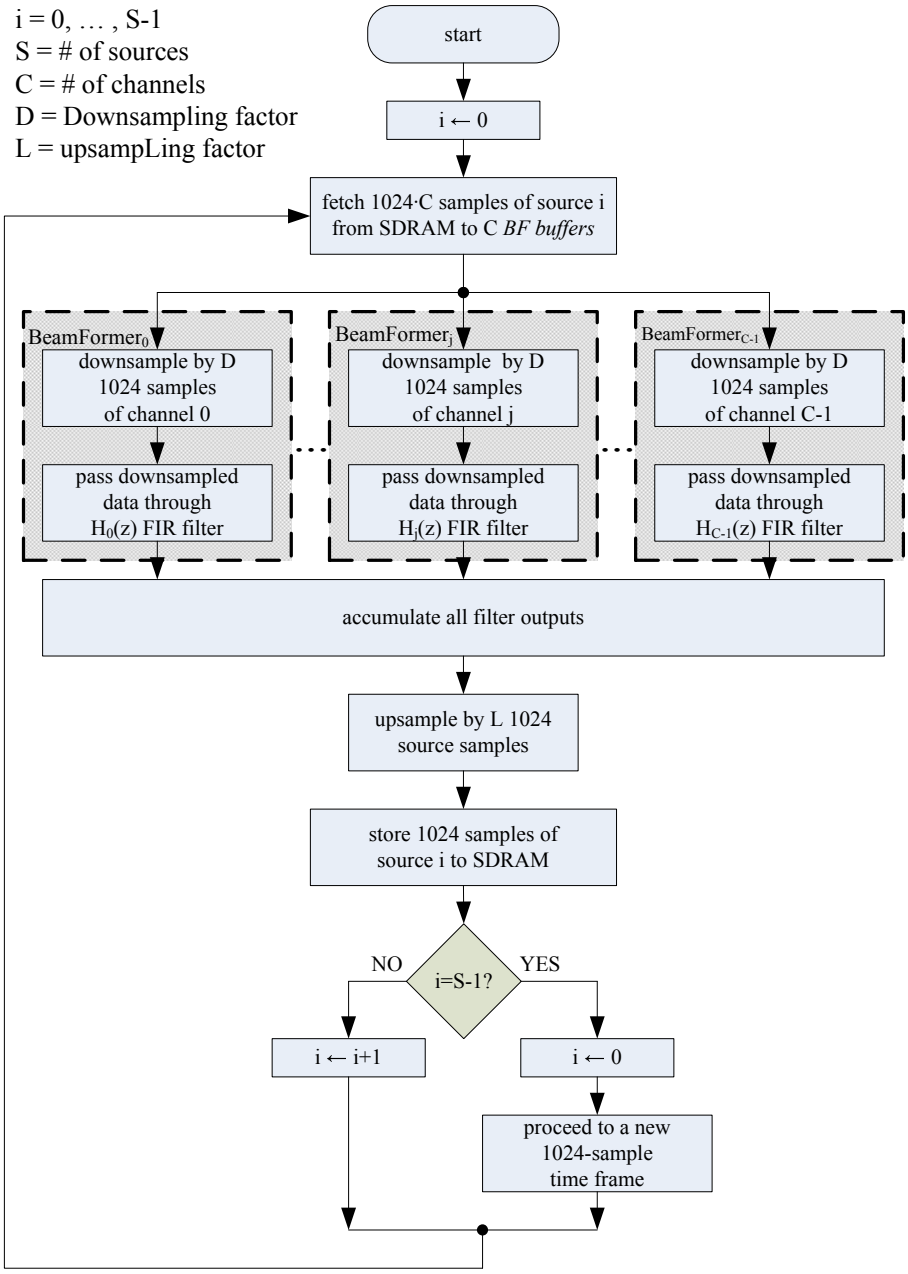

Figure 4.4: Flowchart of the BF data processing among all BeamFormers.

is upsampled by a factor L and the result is stored to the external memory.

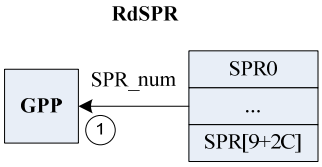

Figure 4.5: BF instruction where the GPP reads from SPRs.

The process is repeated for each i acoustic source within the recording area, where $i = 0, \ldots, S - 1$ and S is the total number of sources. As soon as a 1024-sample chunk is extracted for all S sources, the recorded data of a new 1024-sample time frame is loaded from the SDRAM to the *BF buffers* for further processing.

4.1.2 BF Instruction Implementation

All *SPRs* are accessible from the GPP, because they belong to its memory addressable range. Thus, the programmer can directly pass all customizing parameters to the *MC-BFP*. Each *SPR* is used for storing a system configuration parameter, a start/done flag or a pointer to an external/internal memory entry. For this reason, we have divided the instructions into four different categories, based on the way the GPP accesses the *SPRs*. The categories are: *GPP reads SPR*, *GPP writes to SPR*, *GPP reads and writes to SPR*, *GPP does not access any SPR*, and are illustrated in Figure 4.5, Figure 4.6, Figure 4.7, and Figure 4.8 respectively. In each figure, a number highlights the corresponding step that is taken during the entire instruction execution. All instruction categories are analyzed below:

GPP reads SPR: As illustrated in Figure 4.5, *RdSPR* is the only instruction that belongs to this category. The GPP initiates a *GPP Bus* read-transaction and, based on the *SPR_num* value (step 1), it calculates the proper SPR memory address.

GPP writes to SPR: As illustrated in Figure 4.6, the *InStreamEn*, *ClrSPRs*, *DFirF*, *ConfC* and *SSA* are the instructions that belong to this category. When the *InStream* instruction has to be executed, the GPP initiates a *GPP Bus* write-transaction and writes the *b_mask* value to SPR0 (step 1). Similarly, in *ClrSPRs* the GPP has to iterate through all *SPRs* and write the zero value to them (step 1). In *DFirF* instruction, the GPP uses the *Ftype* parameter to calculate the proper SPR address to write the *FSize* value (step 1). In *ConfC*, the GPP writes the *C* parameter to SPR9 (step 1), which is read from the partial

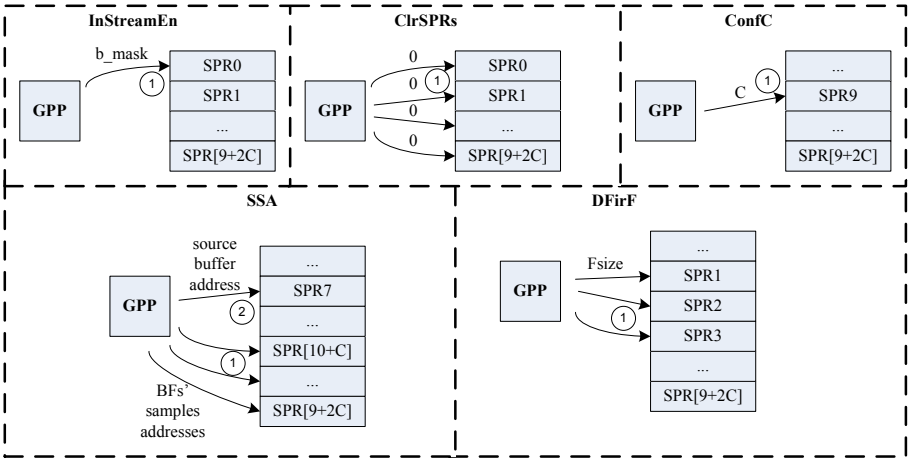

Figure 4.6: BF instructions where the GPP writes to SPRs.

reconfiguration controller, in order to load from the external memory the bitstream that includes *C BeamFormers*. Finally, in *SSA* instruction, the GPP iterates SPR[10+C] - SPR[9+2·C] and writes to them the on-chip *BF buffer* addresses (step 1), where 1024 input samples will be written, which are read from *buf_samp_addr*. Furthermore, it writes to SPR7 the *source buffer* address, where 1024 samples of the extracted source signal are stored (step 2).

GPP reads and writes to SPR: As illustrated in Figure 4.7, the *LdCoef* and *BFSrc* instructions belong to this category. In *LdCoef*, the GPP writes all decimator coefficients addresses to SPR10 - SPR[9+C] (step 1), and the interpolator coefficients address to SPR8 (step 2), which are read from *buf_coef_addr*. As soon as all addresses are written to the proper *SPRs*, the GPP writes a *LdCoef start flag* to SPR4 (step 3) and remains blocked until the *MC-BFP* writes a *LdCoef done flag* to the same SPR. As soon as a *LdCoef start flag* is written to SPR4, the *main controller* enables the *LdCoef controller* to start reloading the decimators coefficients (step 4). Once this step is finished, the *LdCoefInt controller* via the *main controller* initiates the interpolator coefficients reloading procedure (step 5). As soon as all coefficients are reloaded, the *LdCoefInt controller* acknowledges the *main controller*, which writes a *LdCoef done flag* to SPR4 (step 6). This unblocks the GPP, which can continue further processing.

In *BFSrc*, based on the source aperture *aper*, the GPP calculates a *BF buffer* address offset, called *aperture address offset*, in order to access the proper H(z) coefficients sets. The GPP writes the *aperture address offset* to SPR5 (step 1). Furthermore, it performs a DMA transaction, in order to read *C* 1024-sample

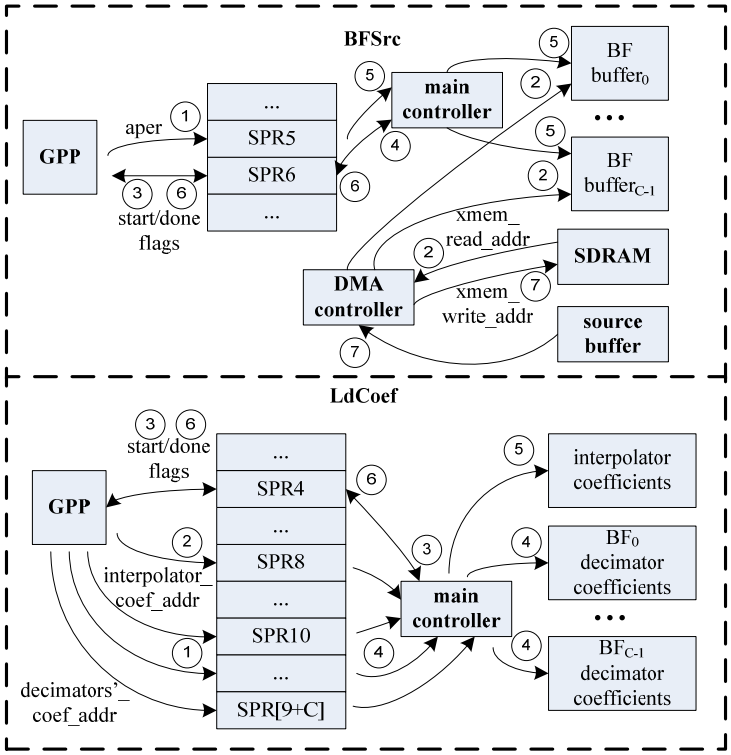

Figure 4.7: BF instructions where the GPP reads and writes to SPRs.

chunks from the *xmem_read_addr* memory location and distribute them to on-chip *BF buffers* of the *C BeamFormer* modules (step 2). As soon as all data are stored, the GPP writes a *BFSrc start flag* to SPR6 (step 3). The MC-BFP reads the start flag from SPR6 (step 4), while the GPP remains blocked until the MC-BFP writes a *BFSrc done flag* to the same *SPR*. Within each *BeamFormer* module, the *LdCoef controller* reads via *the main controller* the *aperture address offset* from SPR5 and reloads to the H(z) filter the proper coefficients set (step 5). Once all H(z) coefficients are reloaded, the *LdCoef controller* acknowledges the *BFSrc controller*, which enables processing of input data that are stored to the *BF buffers*. When all 1024 samples are processed, the *main controller* writes a *BFSrc done flag* to SPR6 (step 6), which unblocks the GPP. The latter performs again a DMA transaction, in order to transfer 1024 samples from the *source buffer* to the *xmem_write_addr* memory location (step 7).

GPP does not access any SPR: As illustrated in Figure 4.8, *BufCoef* is the only instruction that belongs to this category. The GPP reads all source and

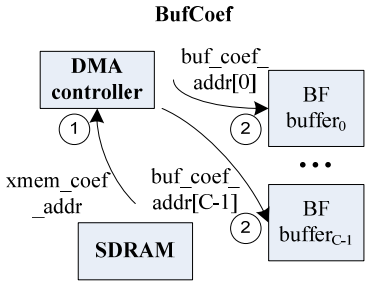

Figure 4.8: BF instruction where the GPP does not access any SPRs.

destination addresses from the *xmem_coef_addr* and *buf_coef_addr* arrays respectively. First, it performs a DMA transaction to transfer all decimator coefficients to the *BF buffers* (step 1). Next, based on the total number of source apertures to the system, it performs a second DMA transaction to load all H(z) coefficients and distribute them accordingly to the on-chip *BF buffers* (step 2). Finally, with a third DMA transaction, the GPP fetches the active interpolator coefficients set to the on-chip *BF buffer* of $\texttt{BeamFormer}_0$ module.

4.2 Reconfigurable WFS Micro-Architecture

4.2.1 Multi-Core WFS Micro-Architecture

Figure 4.9 illustrates the multi-core implementation of the proposed WFS architecture. A *GPP bus* is used to connect the on-chip GPP memory and external SDRAM with the GPP via a standard bus interface (BUS-IF). Furthermore, in order to accelerate data transfer from the SDRAM to *RU buffers*, a *Direct Memory Access (DMA) controller* is employed, which is also connected to the same bus. A partial reconfiguration controller is used to provide the option of reloading the correct bitstreams based on the currently available *RUs*. All user-addressable spaces inside the *MC-WFSP*, like *SPRs*, *RU buffers* and the *source buffer*, are connected to the *GPP bus*. This fact enhances our architecture's flexibility, since they are directly accessible by the GPP. The *main controller* is responsible for initiating the coefficients reloading process to the WFS filter and distributing the loudspeaker coordinates to all *RUs*. Furthermore, it broadcasts all filtered data to each *RU*, enables output data processing from the selected *RUs* and acknowledges the GPP as soon as all calculations are done.

Within each RU there is a *WFS Processing Element* (WFS-PE) module, illustrated in Figure 4.10. The *StCoord controller* is connected to the *main con-*

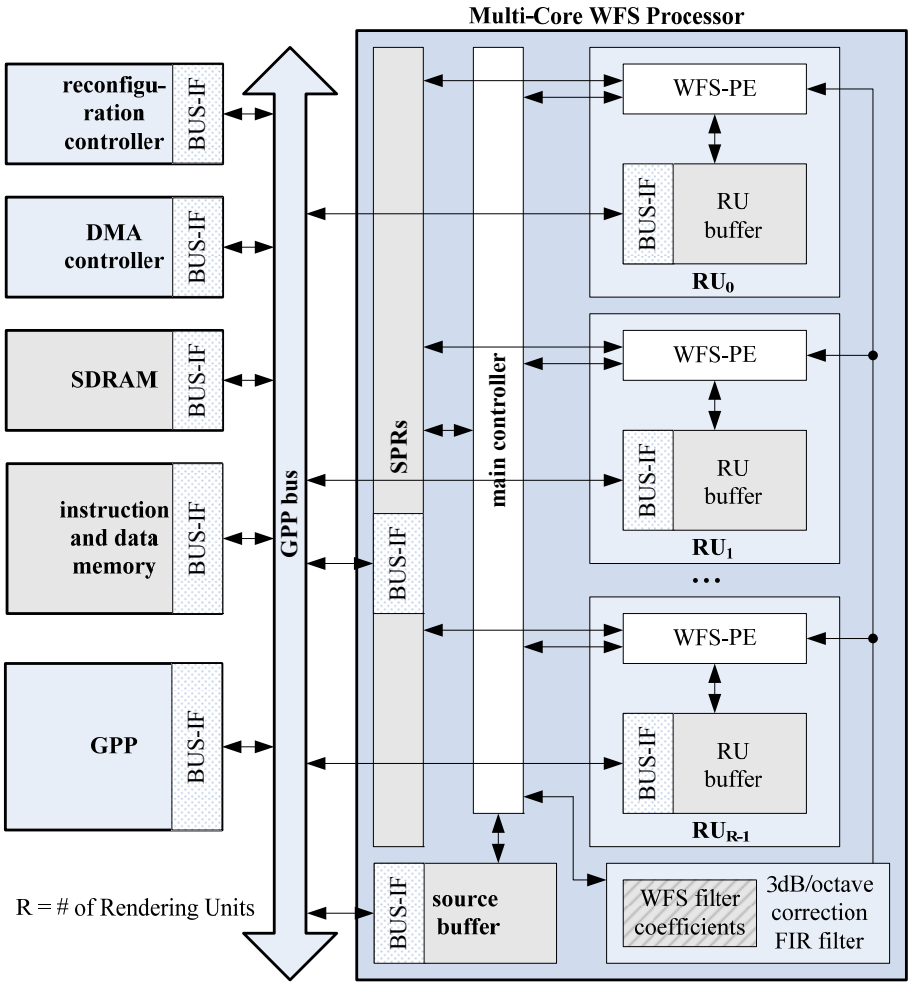

Figure 4.9: Detailed implementation of the WFS multi-core system.

troller and is responsible for transferring the loudspeaker coordinates from the *RU buffer* to the internal *Loudspeaker Coordinates Buffer* (LCB). The *Render controller* reads the coordinates of each loudspeaker from the *LCB* and forwards them to the *Preprocessor*. The latter reads the loudspeaker coordinates and calculates the amplitude decay, source velocity and source distance from a particular loudspeaker based on its current position inside the listening area. The *WFS Engine* module integrates the *Filtered Samples Buffer* (FSB) to store all filtered samples. Furthermore, it employs two cores that select the proper samples, based on the source distance from the same loudspeaker, and multi-

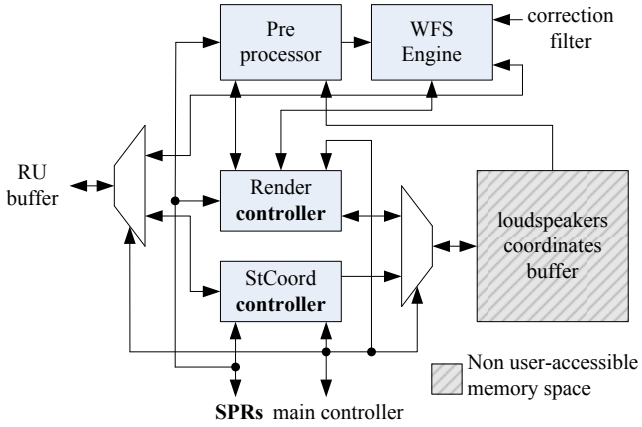

Figure 4.10: The WFS-PE structure.

ply them with the amplitude decay. All output samples are written back to the *RU buffer*.

WFS Preprocessor: Figure 4.11 illustrates the *WFS Preprocessor* organization. Targeting a minimalistic design, we decided to utilize only 1 adder/subtractor, 1 multiplier, 1 square root unit and 1 fractional divider. Furthermore, as mentioned before, the *WFS Preprocessor* always finishes execution before the *WFS Engine* does. Thus, spending additional resources to accelerate its execution, would eventually make the *WFS Preprocessor* just being idle for a longer time.

Current loudspeaker coordinates along with source header are stored into local registers. Since there is direct data dependency among many of these operations, the *WFS Preprocessor* controller issues them serially to the corresponding functional unit. Results are stored again to local registers and reused for further calculations. The *WFS Preprocessor* requires 142 clock cycles to complete data processing and the final results are forwarded to the *WFS Engine*.

WFS Engine: The *WFS engine* is the core computational part of the design, sketched in Figure 4.12. As stated above, once the *WFS Preprocessor* is done, it acknowledges the *Render controller*. The latter starts the *WFS Engine*, which reads from the *WFS Preprocessor* local registers the unit distance, amplitude decay and distance with respect to the current loudspeaker. These data are forwarded to two *Sample Selection Cores* (SSC), SSC_1 and SSC_2, which select the appropriate filtered sound samples from the *FSB*, according to equation (2.9). Each *SSC* consists of one multiplier, one subtractor, two accumulators and one adder, as illustrated in Figure 4.13.

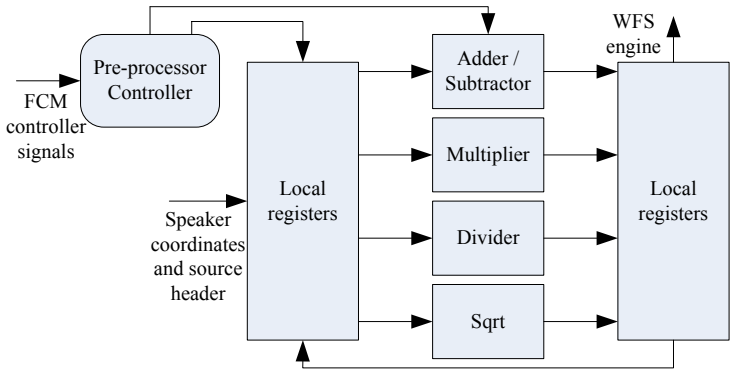

Figure 4.11: The WFS Preprocessor organization

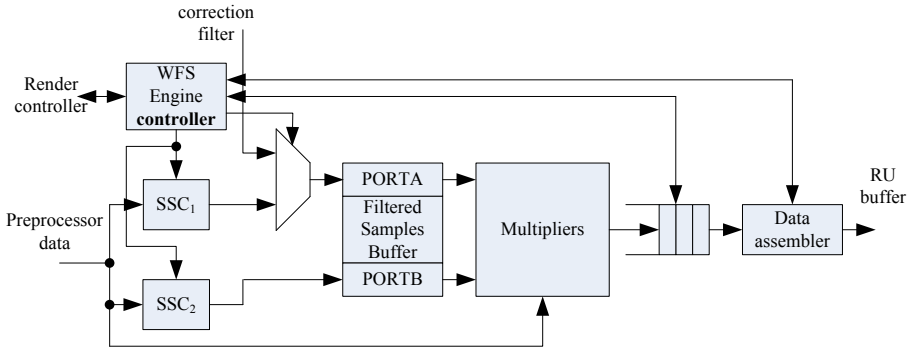

Figure 4.12: WFS Engine organization

All selected samples from SSC_1 and SSC_2 according to equation (2.9), are multiplied by the system master volume level and amplitude decay and forwarded through a FIFO queue to the *Data Assembler*. The latter checks whether the input samples belong to the source with ID=0, in order to perform a data accumulation with the previous samples stored to the *RU buffer* or not. Afterwards, it generates a 64-bit word consisting of four 16-bit audio samples that are written back to the *RU buffer*. The *WFS Engine* repeats the above process for 1024 samples, processing 2 samples per clock cycle, thus a total of 512 cycles. Also there are 11 more cycles spent on communication among internal modules, which results in a total of 523 required cycles for all samples.

The number of used *SSCs* was based on the tradeoff between performance and available resources. The *RU* performance versus the *SSCs* number for processing 1024 samples, is calculated according to the following formula:

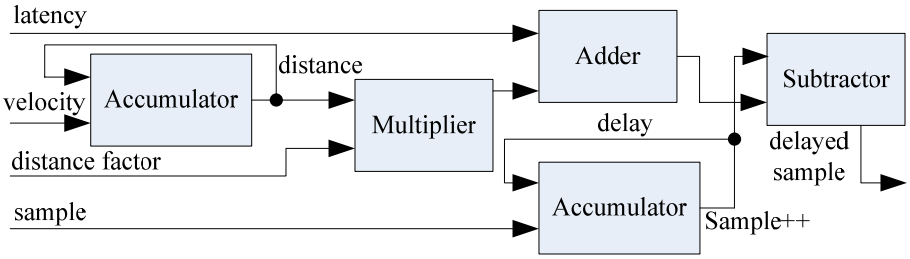

Figure 4.13: SSC organization

$$cc = 11 + 8 + \frac{buffersize}{SSC} \tag{4.1}$$

where 11 cycles are the aforementioned communication overhead among the *WFS Engine* internal modules, and 8 cycles are required for communication among the *WFS Engine*, the *WFS Preprocessor* and the *Render controller*. Formula (4.1) gives a performance of 1043, 531 and 275 clock cycles for 1, 2 and 4 *SSCs* respectively. Utilizing more *SSCs* would cause a BRAM write-back bottleneck, since its current width is 64 bits.

An approach of 2 and 4 *SSCs* would increase the *RU* performance 1043/531=1.96x and 1043/275=3.79x respectively compared to a single *SSC* approach, however, it would require 2x and 4x resources. Based on this analysis, we decided to utilize two *SSCs* which offer a good tradeoff between performance increase and occupied resources.

WFS Data processing flow: Figure 4.14 illustrates how WFS data processing is divided among *R RUs*, under a *L*-sized loudspeaker array setup. For each acoustic source, 1024 samples are fetched from the external memory to the *source buffer*. Once the samples transferring is finished, they are filtered through the 3dB/octave correction filter and stored inside the FSB of each *RU buffer*. Within each iteration i, where $i = 0, \ldots, \frac{L}{R} - 1$, the j-th *RU* processes all output samples of the $i \cdot R + j$ loudspeaker. If for the particular time frame that corresponds to $\frac{1024}{f_s}$ sec, the source with ID=0 is processed (i.e. is the first source), then data are written directly to the *RU buffers*. Otherwise, all data that were already stored to the *RU buffers* are accumulated with the new ones, and the results are stored back to the *RU buffers*. If $i \leq \frac{L}{R} - 1$ then the same process is repeated for the next set of loudspeakers, otherwise output data are stored back to SDRAM. If the source ID=S-1, where *S* is the number of acoustic sources, then data processing starts for a new 1024-sample time frame, otherwise 1024 samples of the next source, but for the current

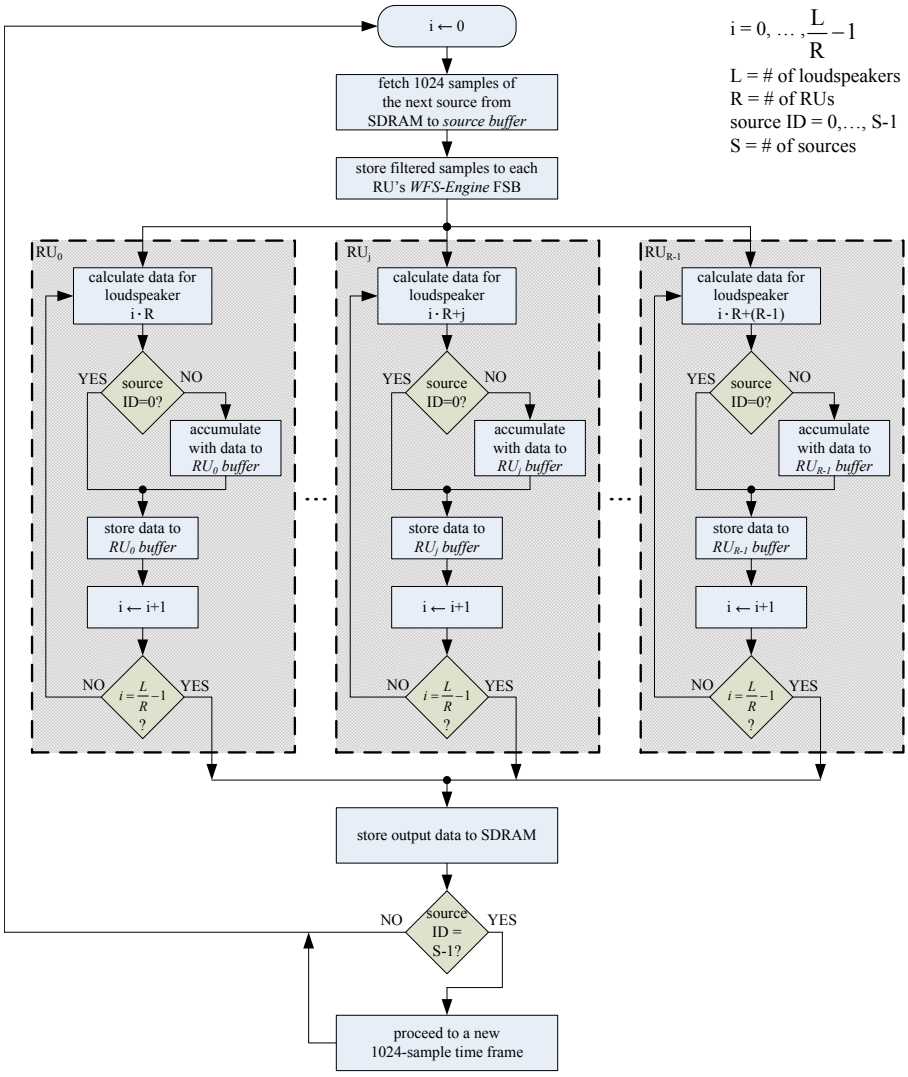

Figure 4.14: Flowchart of the WFS data processing among all RUs.

1024-sample time frame, are processed.

4.2.2 WFS Instruction Implementation

All *SPRs* are accessible from the GPP, because they belong to its memory addressable range. Thus, the programmer can directly pass all customizing pa-

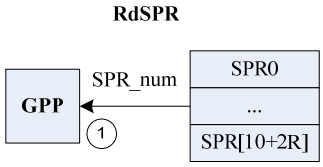

Figure 4.15: WFS instruction that the GPP reads from SPRs.

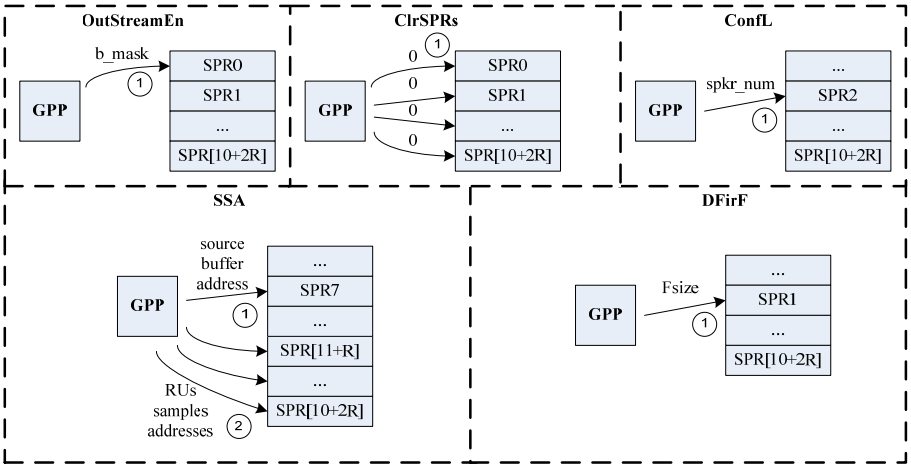

Figure 4.16: WFS instructions that the GPP writes to SPRs.

rameters to the *MC-WFSP*. Each *SPR* is used for storing a system configuration parameter, a start/done flag or a pointer to an external/internal memory entry. For this reason, we have divided the instructions into four different categories, based on the way the GPP accesses the *SPRs*. The categories are: *GPP reads from SPR*, *GPP writes to SPR*, *GPP reads and writes to SPR*, *GPP does not access any SPR* and are illustrated in Figure 4.15, Figure 4.16, Figure 4.17 and Figure 4.18 respectively. In each figure, a number highlights the corresponding step that is taken during the entire instruction execution. All instruction categories are analyzed below:

GPP reads from SPR: As illustrated in Figure 4.15, *RdSPR* is the only instruction that belongs to this category. The GPP initiates a *GPP bus* read-transaction and, based on the *SPR_num* value, it calculates the proper *SPR* memory address (step 1).

GPP writes to SPR: As illustrated in Figure 4.16, *OutStreamEn*, *ClrSPRs*, *ConfL*, *DFirF*, and *SSA* are the instructions that belong to this category. When the *OutStreamEn* instruction has to be executed, the GPP initiates a bus trans-

action and writes the binary mask *b_mask* to SPR0 (step 1). Similarly, in *Clr-SPRs* the GPP performs consecutive bus transactions to access all *SPRs* and writes the zero value to them (step 1). The *ConfL* instruction writes the *L/R* parameter to SPR2 (step 1) and also forwards the *R* parameter to the partial reconfiguration controller, in order to load from the external memory the bit-stream that includes *R RUs*. The *DFirF* also performs a bus transaction to write to SPR1 the WFS filter size. Finally, the *SSA* instruction accesses the *GPP bus* to write the *source buffer* address to SPR7 (step 1) and all sample addresses within each *RU buffer* to SPR[11+R] - SPR[10+2·R] (step 2).

GPP reads and writes to SPR: As illustrated in Figure 4.17, *StC*, *LdCoef* and *RenSrc* instructions belong to this category. When the *StC* instruction is executed, the GPP performs a DMA transaction to read all loudspeakers coordinates from the external memory address *xmem_spkr_coordinates* and store them the GPP on-chip memory. The loudspeaker coordinates are re-arranged based on the number of *RUs* of the system and stored to the *RU buffers* (step 1). The GPP writes a start flag to SPR3 and remains blocked until a done flag is written to the same register (step 2). The start flag is read by the *MC-WFSP* (step 3) and the *main controller* invokes the *StCoord controller* to load the coordinates from the *RU buffers* to the internal *LCBs* (step 4). As soon as all loudspeakers coordinates have been transferred, the *MC-WFSP* writes the done flag to SPR3 (step 5), which is read by the GPP to continue further processing.

When the *LdCoef* instruction is executed, the GPP performs a bus transaction to write to SPR8 the WFS coefficients address inside the *source buffer* (step 1). Furthermore, it writes to SPR4 the start flag and remains blocked until a done flag is written to the same SPR (step 2). The *MC-WFSP* reads the start flag (step 3) and the *main controller* starts the coefficients reloading to the FIR filter (step 4). Once all coefficients are loaded, the *MC-WFSP* writes a done flag to SPR4 (step 5), which is read by the GPP to continue further processing.

Finally, when the *RenSrc* is executed, the GPP writes to SPR5 and SPR6 the (x1,y1) and (x2,y2) source coordinates (step 1). Furthermore, it writes to SPR10 the source identification number (step 2) and performs a DMA transaction to read a 1024-sample chunk from the external memory and store it to the *source buffer* (step 3). The GPP then writes to SPR9 a start flag and remains blocked until a done flag is written to the same register from the *MC-WFSP* (step 4). The *MC-WFSP main controller* reads the start flag (step 5) and invokes the *Render controller* within each *RU* to start data processing (step 6). For every loudspeaker that is processed within a specific *RU*, the *Render controller* reads its coordinates from the *LCB*, the source coordinates from SPR5

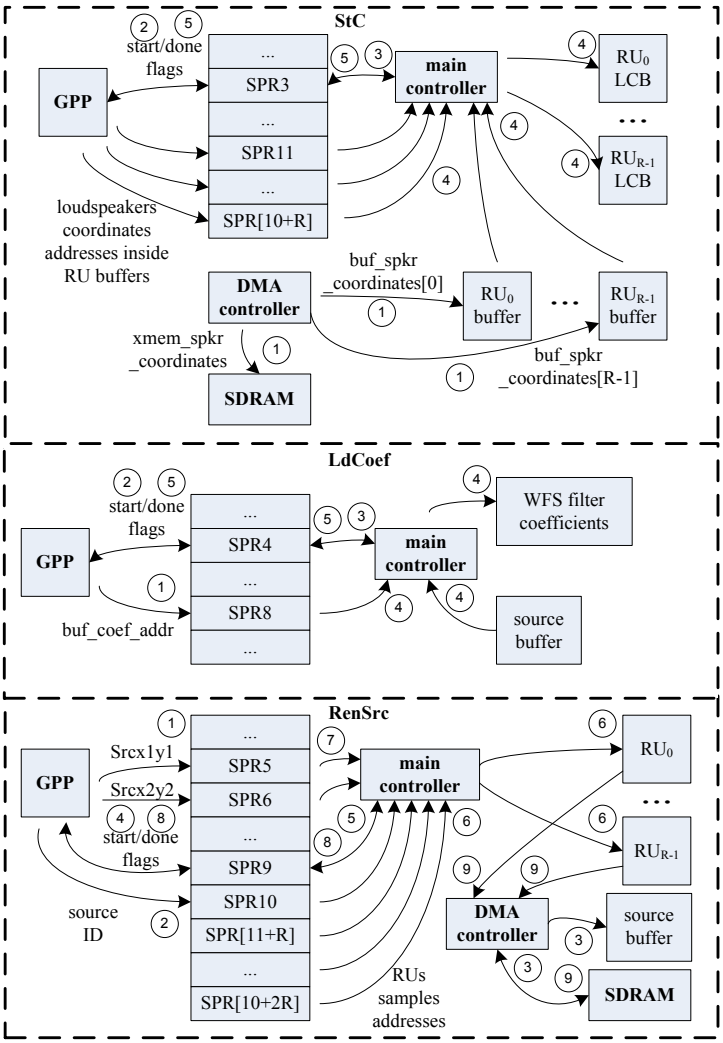

Figure 4.17: WFS instructions that the GPP reads and writes to SPRs.

and SPR6 (step 7) and 1024 samples from the *source buffer*, and forwards them to the *WFS Preprocessor* to calculate the amplitude decay, source velocity and source distance from the particular loudspeaker. Once these parameters are computed, the *Render controller* invokes the *SSCs* which select the proper audio samples, multiply them by the amplitude decay and store them back to the *RU buffers*. As soon as all assigned loudspeakers to all *RUs* are processed, the *main controller* writes a done flag to SPR9 (step 8), which is read by the GPP. The latter then performs DMA transactions to store back all output samples

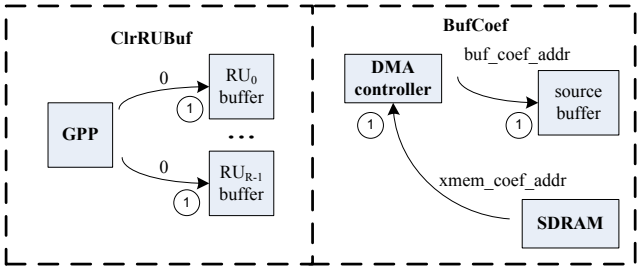

Figure 4.18: WFS instructions where the GPP does not access any SPRs.

from the *RU buffers* to the SDRAM (step 9).

GPP does not access any SPR: As illustrated in Figure 4.18, *BufCoef* and *ClrRUBufs* are the instructions that belong to this category. When the *BufCoef* instruction is executed, the GPP reads all source and destination addresses from the *xmem_coef_addr* and *buf_coef_addr* pointers respectively. Then it performs a DMA transaction to transfer all WFS filter coefficients from the SDRAM to the *source buffer* (step 1). The *ClrRUBufs* instruction performs DMA transactions to initialize all available *RU buffers* with zeros (step 1).

4.3 Conclusions

In this chapter we presented the underlying micro-architecture that supports the proposed instructions of both BF and WFS applications, originally presented in Chapter 2. A multi-core processing approach was chosen, in order to exploit the inherent parallelism that both immersive-audio techniques offer. The architecture implementation for both BF and WFS allows utilization of various number of processing elements, therefore it is suitable for mapping on reconfigurable technology. As it can be concluded, with respect to the available reconfigurable resources, different FPGA implementations with different performances are possible, where all of them use the same architecture and programming paradigm.

This chapter is based on the following papers:

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A Reconfigurable Audio Beamforming Multi-Core Processor**, International Symposium on Applied Reconfigurable Computing (ARC), pp. 3-15, Belfast, Ireland, March 2011

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **Minimalistic Architecture for Reconfigurable Audio Beamforming**, International Conference on Field-Programmable Technology (FPT), pp. 503-506, Beijing, China, December 2010

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A Minimalistic Architecture for Reconfigurable WFS-Based Immersive-Audio**, International Conference on ReConFigurable Computing and FPGAs (ReConfig), pp. 1-6, Cancun, Mexico, December 2010

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A Reconfigurable Beamformer for Audio Applications**, IEEE Symposium on Application Specific Processors (SASP), pp. 80-87, San Francisco, California, USA, July 2009

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **Reconfigurable Accelerator for WFS-Based 3D-Audio**, IEEE Reconfigurable Architectures Workshop (RAW), pp. 1-8, Rome, Italy, May 2009,

5

Architecture Implementation on nr-MCPs: Case Study on GPUs

Non-reconfigurable Multi-Core Processors (nr-MCPs) provide an attractive solution for mapping applications with inherent parallelism. Over the last years, a lot of research is conducted, in order to integrate as many processing cores as possible within a single chip. As a result, different multi-core architectures have emerged.

Figure 5.1 illustrates a chart with few examples of homogeneous and heterogenous hardware commercial platforms. For example, nowadays there are mainstream GPPs that integrate up to six processing cores, like the Phenom II X6 processor series from AMD [25]. Recently, Intel presented an experimental "Single-Chip Cloud Computer" (SCC) that integrates 48 cores [43]. Furthermore, there are various platforms that consider a heterogeneous approach, like the Sony Cell Broadband Engine, the Kilocore series from Rapport and IBM [44], and the picoChip PC200 series [74]. A third category of nr-MCPs is the GPUs, which over the last years have evolved to massively parallel processing platforms that offer a computational performance in the order of GFLOPs. As it can be observed from Figure 5.1, GPUs cover a wide range of implementations that integrate from a few tens up to hundreds of processing cores. Moreover, GPGPU computing is increasingly evolving due to the fact that major GPU companies like NVidia and AMD, provide high-level programming environments that help the developers to harness the potential processing power of GPUs. As a result, nowadays, GPUs can be utilized to build fast and computationally powerful hardware systems that can be applied to many scientific and commercial domains.

Based on these facts, we conducted a case study on the nr-MCP domain and applied our proposed architecture to GPUs. More specifically, in this chapter we preset how we mapped the instructions mentioned in Section 3.3 onto

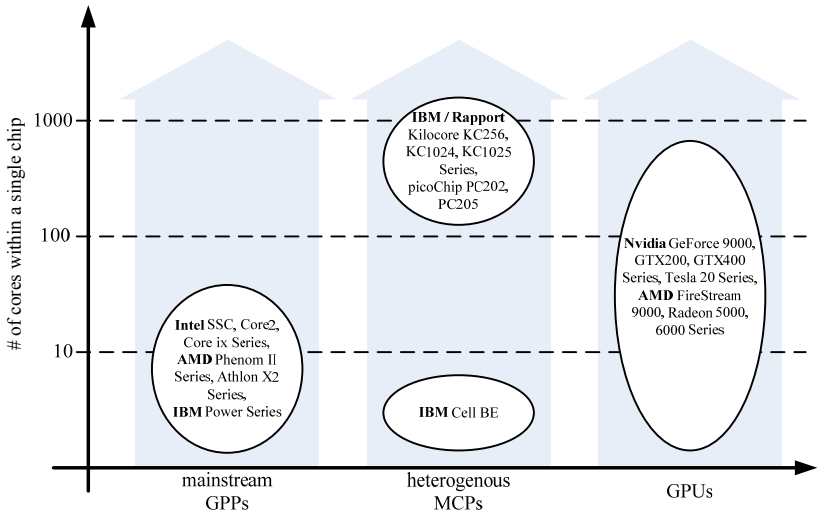

Figure 5.1: Number of processing cores integrated to contemporary nr-MCPs.

GPUs for both BF and WFS. Since contemporary GPUs consist of tens of on-chip multiprocessors [68], they provide a huge potential of concurrent thread execution, as long as the target application can be parallelized. Both BF and WFS applications can be considerably parallelized, thus making the GPUs a suitable target platform.

The rest of the chapter is organized as follows: In Section 5.1, we provide a brief description of contemporary GPUs organization. Section 5.2 and Section 5.3 describe how the BF and WFS instructions were mapped onto GPUs respectively. Finally, Section 5.4 concludes the chapter.

5.1 Contemporary GPUs organization

In order for the developers to efficiently map general purpose kernels on the GPU without using specific graphics terms, such as textures, vertices and fragments, NVidia launched the Compute Unified Device Architecture (CUDA) parallel software environment [68]. CUDA introduces a set of extensions to the C programming language that exposes to the developers the parallel processing capabilities of the GPU. Each kernel mapped on the GPU is executed concurrently by many threads mapped on the multiprocessors. CUDA defines a thread hierarchy based on a grid of thread blocks. Each block consists of up to 512 threads in a 1-, 2- or 3-dimensional order, while the maximum dimen-

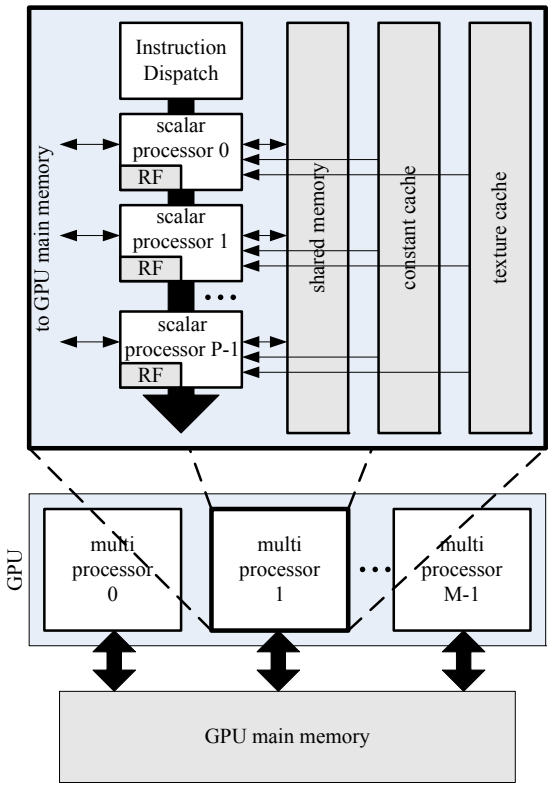

Figure 5.2: Contemporary NVidia GPUs organization.

sion size of a grid of thread blocks can be up to 2^{16}.

Contemporary NVidia GPUs architectures like the G80 and the GT200 [68] consist of M multiprocessors, as illustrated in Figure 5.2, that can process data concurrently. Each multiprocessor includes P scalar processors, each one consisting of two integer and one floating point units (FPU). Furthermore, a multiprocessor integrates special function units for transcendental functions, like sine and cosine, a multithreaded instruction unit and the following four different types of on-chip memory:

- A *shared memory* among all processors in a multiprocessor. The shared memory is used for data caching.
- A read-only *constant memory* used for caching reads of constants from the GPU main memory.
- Another read-only memory, called *texture memory*, that is used for caching textures read also from the GPU main memory.

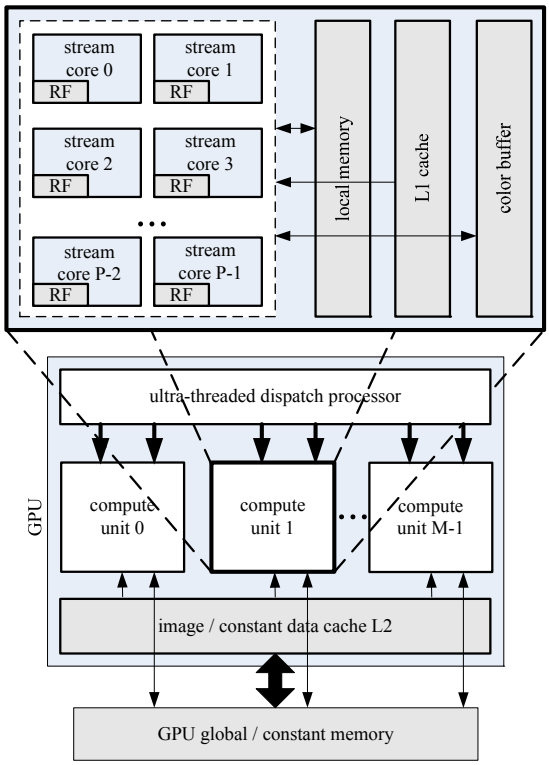

Figure 5.3: Contemporary AMD GPUs organization.

- *P* Register Files (RFs) distributed among all processors of a multiprocessor.

Recently, NVidia launched a new series of GPUs that incorporate the Fermi architecture [70]. One of its main innovations is that introduces a true cache hierarchy. More specifically, the shared memory among the CUDA processing cores can be partially configured as an L1 cache memory, in order to enhance applications performance, especially for the ones that do not exploit the shared memory [70]. In addition, the scalar processors, referred to the Fermi architecture context as CUDA cores, have been significantly improved. More precisely, the enhanced FPU supports the IEEE754 2008 standard [40] and a fuzed multiply-add floating point instruction. Furthermore, the integer unit is improved for multiply operations from 24 bits precision to 32 bits.

Similarly to NVidia, AMD has developed its own high-level parallel programming environment for GPUs, which supports the OpenCL framework [49], called Stream [4]. The latter can be used to map applications for parallel exe-

cution onto AMD GPUs. Figure 5.3 depicts the organization of contemporary AMD GPUs. As it can be observed, within a single GPU there are *M* compute units. Each one of them integrates *P* stream cores, while a single core includes five processing elements for single precision floating point and integer operations, and transcendental functions. Furthermore, there are the following types of on-chip memory:

- A *private memory*, i.e. a RF, that is used from each stream core.
- A *local memory* including the L1 and the *color buffer* that is used from each compute unit.
- A *global memory* that is accessible by all compute units.
- A *constant memory* that is accessible by all compute units and is used to initialize data that do not change during kernel execution.

As it can be concluded, both companies follow a rather similar approach to design GPUs. Their excessive computational power relies on a multiprocessor-based approach, where each processor comprises massive number of processing elements. In our case study, we have selected the NVidia GPU platforms to apply our proposed architecture, using the CUDA parallel programming environment. However, as it was mentioned in Chapter 3, all implementation details, like proper code annotation for mapping the application to the CUDA cores and threads scheduling of kernels are completely hidden from the user. In the following sections we describe the BF and WFS instructions implementation to NVidia GPUs using the CUDA programming environment.

5.2 BF Instructions Implementation to GPUs

As a case study of the BF technique, we have followed the specifications and processing steps of a BF application provided by the Fraunhofer Research Institute [41], in the context of the hArtes European project [51], [50], [39]. However, we should note that the proposed architecture implementation is applicable for any BF application that performs the same processing steps, regardless of any specifications, such as sampling frequency and different filters size. In the following we describe the implementation of each instruction presented in Section 3.3 using the CUDA parallel programming environment.

InStreamEn: The mask *b_mask* that controls the input channels streaming is stored to internal (i.e. non accessible to to the user) variable. The latter is used during the *BFSrc* execution, in order to enable or disable the corresponding input channels.

DFirF: Based on the value of the *FType* parameter, it stores the filter sizes to internal variables. The latter are used during the *BFSrc* execution, in order to configure efficiently the threads scheduling to the GPU.

LdCoef: Uses the *coef_addr* parameter and the decimator and interpolator sizes that are already stored to internal variables. It transfers the active decimators and interpolator coefficients sets to the GPU main memory. Since the coefficients number is a dynamic value, the instruction uses the *cudaMemcpyToSymbol* CUDA function [68] to transfer all data.

ConfC: Writes to an internal variable the number of system input channels.

BFSrc: As it was indicated in Section 2.1, BF is mainly based on FIR filters that compensate for the introduced delay of the sound wavefront arrival at all microphones [13]. The result is that the main sound source is strengthened while any ambient noise is attenuated. However, in order to reduce the data processing rate in this particular implementation, input signals are first decimated by a certain factor. As soon as all decimated input signals are processed, the extracted sound source is interpolated by the same factor that was used during the decimation process, thus restoring the initial sampling frequency. To summarize, the BF application consists of the following four phases:

1. Decimation of the input signal by a certain factor.
2. FIR filtering of each input signal with a specific set of coefficients.
3. Accumulation of all filtered signals to strengthen the main sound source.
4. Interpolation of the extracted source signal by the same factor that was used during the decimation process.

It is observed that in three out of four of the above phases FIR filtering is required, thus it is the dominant operation of the BF application. FIR filtering has the potential to be considerably accelerated when mapped onto GPUs, since the latter provide hundreds of processing elements that can calculate data concurrently. Based on this fact, we decided to develop a flexible computational kernel that could be reused with minor changes to efficiently map the decimation, FIR filtering and interpolation processes onto the GPU. Furthermore, an additional GPU kernel was developed to perform the accumulation of all filtered signals. Algorithm 5.1 illustrates how the *BFSrc* instruction is mapped onto the GPU. Variable *C* represents the total number of input channels (microphones) available to the sound system, which can be configured using the *ConfC* instruction. *SOURCES* is the number of present sources that need to be extracted. We use the *GPU* annotation to designate the parts of the application that are executed by the GPU.

Algorithm 5.1 Beamforming implementation to GPU

Require: Input signals from C microphones
Ensure: Extracted $SOURCES$ audio sources

1: Move input data to GPU main memory
2: Store H(z) filter coefficients to GPU main memory
3: **for** $c = 0$ to $C - 1$ **do**
4: GPU: Decimate channel c
5: **end for**
6: **for** $s = 0$ to $SOURCES - 1$ **do**
7: **for** $c = 0$ to $C - 1$ **do**
8: GPU: Extract source s from channel c
9: **end for**
10: GPU: Accumulate signals from all channels
11: GPU: Upsample source s signal
12: **end for**
13: Move all extracted sources signals back to CPU main memory

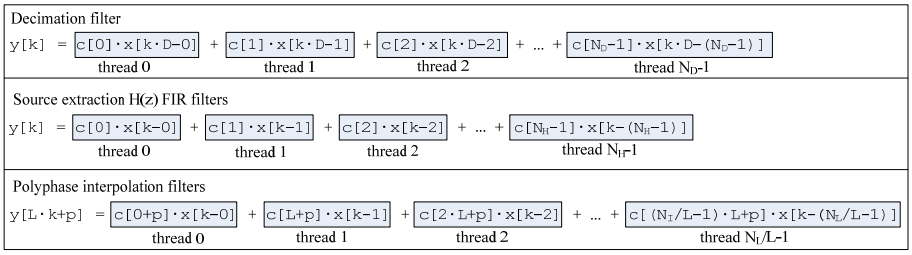

Figure 5.4: Decimation, source extraction and interpolation filters onto GPU threads.

Input signal decimation: The initial sampling frequency f_s in the specific BF application is 48 kHz and a downsampling factor D of 4 is used to decimate the signal. However, before downsampling any signal, it is mandatory to filter it in order to avoid any aliasing effects. Since $D = 4$, the resulting signal will have $f_s = 12$ kHz, which is its new sampling frequency. This means that if the applied filter before downsampling eliminates all components above $\frac{12}{2}$ kHz = 6 kHz, which is the Nyquist frequency, then the final decimated signal will have no aliasing effects. Therefore, a decimation FIR filter with size of 242 taps is applied to cut off any components above 6 kHz.

In order to efficiently filter and downsample a signal, the following formula is used:

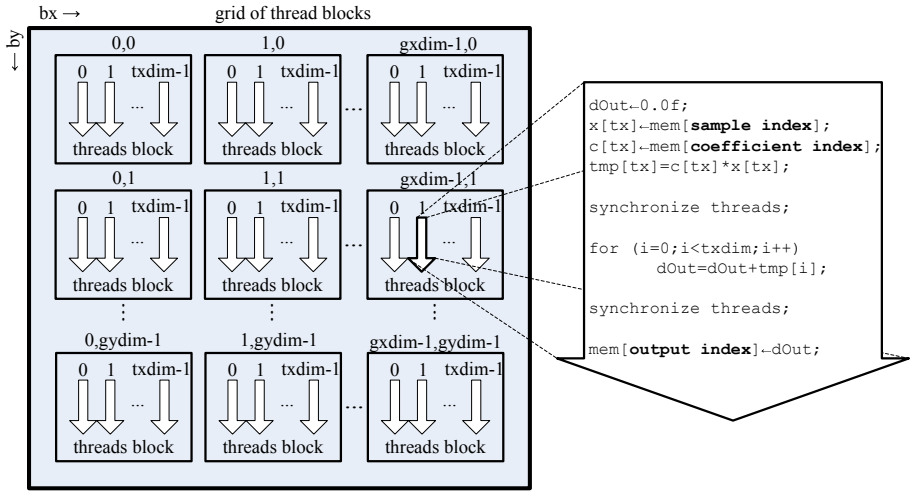

Figure 5.5: Grid of thread blocks that are dispatched during the FIR filter calculations onto the GPU.

Table 5.1: Sample, coefficient and output indices for the BF application.

GPU kernel	sample index	coefficient index	output index
decimation	bx·D+by·gxdim-tx	tx	bx+by·gxdim
H(z) filter	bx+by·gxdim-tx	angle·N_H·C+C_cur·N_H	bx+by·gxdim
interpolation	$\frac{bx}{L}$+by·gxdim-tx	tx	bx+by·gxdim

$$y[k] = \sum_{tx=0}^{N_D-1} c[tx] \cdot x[k \cdot D - tx] \tag{5.1}$$

where k is the current filter output sample, N_D is the decimator filter size provided by the *DFirF* instruction, tx=0...N_D-1, *c[]* represents the filter coefficients, *x[]* represents the current status of the filter delay line, and D is the downsampling factor. The first row of Figure 5.4 illustrates how (5.1) is mapped onto different GPU threads, in order to calculate a single output sample of the decimated signal. Each multiplication is mapped onto a different GPU thread and all of them are concurrently executed. As soon as all multiplications are performed, all results are accumulated and stored back to the GPU main memory. The second and third lines of Figure 5.4 are discussed in the "Source extraction" and "Source signal interpolation" subsections.

Figure 5.5 depicts the grid of thread blocks that was designed to efficiently map the calculation of all decimated samples onto a GPU kernel. Input signals

are divided into 1024-sample chunks. Since the decimator FIR filter size is 242, we launch the same number of GPU threads per block, that is *txdim* = 242 in Figure 5.5. This way we can map the calculation of each output sample of the decimated signal to 1 thread block. Within each thread block variable *dOut* is used to store the corresponding output sample. Each thread calculates a proper sample and coefficient index to load the respective sample and filter coefficient from the GPU external memory to *x[]* and *c[]* arrays inside the shared memory of a GPU multiprocessor. We use the variables *bx* and *by* as coordinates of every thread block within the grid, thus *bx*=0...*gxdim*-1 and *by*=0...*gydim*-1. Since each thread block is responsible for calculating a single sample of the decimated signal, the illustrated grid in Figure 5.5 can process up to *gxdim*·*gydim* samples. Based on these variables, each thread calculates the correct sample, coefficient and output indices that are shown in the second row of Table 5.1. As soon as the two values are multiplied, all threads are synchronized and data are accumulated to the *dOut* variable. Threads are then again synchronized and the final result is stored to the main GPU memory based on the output index that each thread calculates.

Source extraction: In order to efficiently map the FIR filter operations onto the GPU, we used the same approach as in decimation. Each H(z) FIR filter is represented by:

$$y[k] = \sum_{tx=0}^{N_H-1} c[tx] \cdot x[k - tx] \tag{5.2}$$

where k is the current filter output sample, N_H is the H(z) filter size provided by the *DFirF* instruction, *tx*=0...N_H-1, *c[]* represents the filter coefficients, and *x[]* represents the current status of the filter delay line. The second row in Figure 5.4 illustrates how (5.2) can be separated into different GPU threads, where each thread block is responsible for calculating a single output sample. As it was in the case of decimation, each multiplication is assigned to a unique GPU thread and all of them are executed concurrently. All results are accumulated, and the final value is stored back to the GPU main memory.

We designed again a *gxdim* by *gydim* grid of thread blocks, as illustrated in Figure 5.5, where *bx*=0...*gxdim*-1 and *by*=0...*gydim*-1 are the coordinates of each thread block within the grid. Each FIR filter is 128 taps long. Therefore, we use 128 GPU threads per block, thus *txdim* = 128 in Figure 5.5 for the source extraction. Finally, the third row of Table 5.1 illustrates how the new sample and coefficient indices are calculated within each GPU thread. The *angle* variable designates the source aperture, *C* is the total number of input

channels, and *C_cur* is the current channel that is being processed.

It has been noted in Section 2.1 that each decimated signal is filtered by an H(z) FIR filter. In the case considered, the BF application can recognize 19 source apertures. Furthermore, since each FIR filter consists of 128 taps, we need to store 128 $\frac{coefficients}{aperture} \cdot 19 \frac{apertures}{channel} \cdot$C $channels = 2432 \cdot$C $coefficients$, where each coefficient is in single precision floating point format. Since current GPUs constant memory is not enough to fit all coefficients, we decided to store them to the GPU main memory.

Source signal interpolation: Signal interpolation consists of two consecutive stages: signal upsampling and filtering. Upsampling a signal by a factor L introduces undesired spectral images at multiples of the initial sampling frequency, and for this reason, the upsampled signal needs to be filtered [54]. We upsampled the extracted source signal by inserting L-1 zeros between its samples. Following a polyphase filter approach [54], we can use the following equation to describe all L polyphase filters that are utilized to efficiently upsample and filter the input signal:

$$y[k \cdot L + p] = \sum_{tx=0}^{\frac{N_L}{L}-1} c[tx \cdot L + p] \cdot x[k - tx] \tag{5.3}$$

where k is the current filter output sample, N_L is the interpolator filter size provided by the *DFirF* instruction, tx=0...$\frac{N_L}{L}$-1, *c[]* represents the filter coefficients, *x[]* represents the current status of the filter delay line, L is the upsampling factor, and p=0...L-1 is the corresponding polyphase filter.

The third row in Figure 5.4 illustrates how (5.3) can be efficiently mapped to different GPU threads. As it was in both cases of decimation and source extraction, each multiplication is assigned to a different GPU thread, while all threads are executed concurrently. Based on the polyphase filter approach [54], the interpolation FIR filter length should be a multiple of L. The BF application that we used during our experiments utilizes a FIR filter with N_L=242 taps long, while the upsampling factor $L = D = 4$. In order to make the FIR filter length multiple of 4 we added two more taps at the end of its delay line with both coefficients being equal to 0, thus increasing the FIR filter to 244 taps. Since $L = 4$, we use $p = 4$ polyphase filters to upsample the extracted source signal, each one being $\frac{244}{4}$=61 taps long. The multiplication results are accumulated, and the computed value is stored back to the GPU main memory. As in the case of decimation and source extraction, we designed a *gxdim* by *gydim* grid of thread blocks, as depicted in Figure 5.5, to calculate *gxdim·gydim* output samples.

We use again the variables *bx* and *by* as coordinates of every thread block within the grid, thus *bx*=0...*gxdim*-1 and *by*=0...*gydim*-1. Since each polyphase filter is 61 taps long, we launch 61 GPU threads per block, thus *txdim* = 61 in Figure 5.5 for the interpolation. The fourth row of Table 5.1 suggests how the GPU kernel for the signal interpolation calculates the corresponding sample, coefficient and output indices.

5.3 WFS Instructions Implementation to GPUs

As a case study of the WFS technique, we have followed the specifications and processing steps of a WFS application provided by the TU Delft SoundControl department [92]. However, we should note that the proposed architecture implementation is applicable for any WFS application that performs similar processing steps, regardless of any specifications, such as sampling frequency and different filters size. In the following we describe the implementation of each instruction presented in Section 3.3 using the CUDA parallel programming environment.

OutStreamEn: The mask *b_mask* that controls the output channels streaming is stored to internal (i.e. non accessible to to the user) variable. The latter is used during the *RenSrc* execution, in order to enable or disable the corresponding output channels.

DFirF: It stores the WFS filter size to internal variables. The latter is used during the *RenSrc* execution, in order to configure efficiently the threads scheduling to the GPU.

LdCoef: Uses the *coef_addr* parameter and the WFS filter sizes that are already stored to internal variables. It transfers the active coefficient sets to the GPU main memory. Since the coefficients number is dynamic, the instruction uses the *cudaMemcpyToSymbol* CUDA function [68] to transfer all data.

ConfL: Writes to an internal variable the number of output channels to the system.

RenSrc: The WFS algorithm can be considerably parallelized. Once the source distance from a loudspeaker and the source amplitude decay have been calculated, each sample for this particular loudspeaker can be processed concurrently (eq. 2.9). Furthermore, all calculations regarding each loudspeaker are also independent. While mapping the WFS kernel onto the GPU, special attention was paid on the data transfers between the GPU main memory and the on-chip shared memory. As recommended in [68], while a GPU kernel is running,

the GPU main memory accesses by the threads should be minimized, because each one requires hundreds of cycles. For this reason, source and loudspeaker coordinates are stored in the constant memory, because it is cachable and thus will be faster to read them, instead of accessing the GPU main memory each time they are required. When the WFS kernel starts execution, its first task is to load all required data from the GPU main memory to the shared memory. When data loading is done, all threads within a block are synchronized to make sure that further shared memory readings will be valid. Once all threads are synchronized, each one of them processes two loudspeaker samples as mentioned earlier. All results are temporarily stored in the shared memory and, as soon as all threads are done, the kernel copies them back to the GPU main memory. Following this approach regarding the data transfers, we managed to minimize the memory access cost impact on the overall kernel execution time.

Overall, the processing steps of the WFS kernel for a source *i* rendered through loudspeaker *j* can be summarized as follows:

1. *Copy from the GPU main memory the next 2048 source samples to the shared memory.* For every 1024 loudspeaker samples to be calculated, the algorithm requires to access the *previous* source buffer when the delayed sample is negative, or the *current* source buffer when the delayed sample is positive (eq. 2.9).
2. *Calculate the amplitude decay and distance of source i from loudspeaker j.* A single thread per block calculates the source i distance from loudspeaker j and the corresponding amplitude decay. The reason that we use a single thread is because the distance and the amplitude decay are common for a specific 1024-sample segment.
3. *Process loudspeaker samples 0 to 511.* Each thread processes one sample, and stores the results back to the shared memory.
4. *Process loudspeaker samples 512 to 1023.* Again each thread processes a second sample and stores the results back to the shared memory.
5. *Copy processed loudspeaker samples to the GPU main memory.* As soon as all loudspeaker samples are processed, the threads are again synchronized and each one stores two loudspeaker samples back to the main GPU memory.

Algorithm 5.2 sketches how the WFS is mapped onto the GPU. The variable L represents the number of loudspeakers of the audio system. The variable *SOURCES* is the number of audio sources that need to be rendered through the L loudspeakers. Again, we use the *GPU* annotation to designate the GPU

Algorithm 5.2 Wave Field Synthesis implementation to GPU

Require: *SOURCES* audio sources to be rendered to L loudspeakers
Ensure: L audio signals to drive all loudspeakers

1: Move input data to GPU main memory
2: **for** $s = 0$ to $SOURCES - 1$ **do**
3: GPU: Apply FIR filtering to source s
4: GPU: Calculate all L signals for source s and accumulate with previous ones
5: **end for**
6: Move all L signals back to CPU main memory

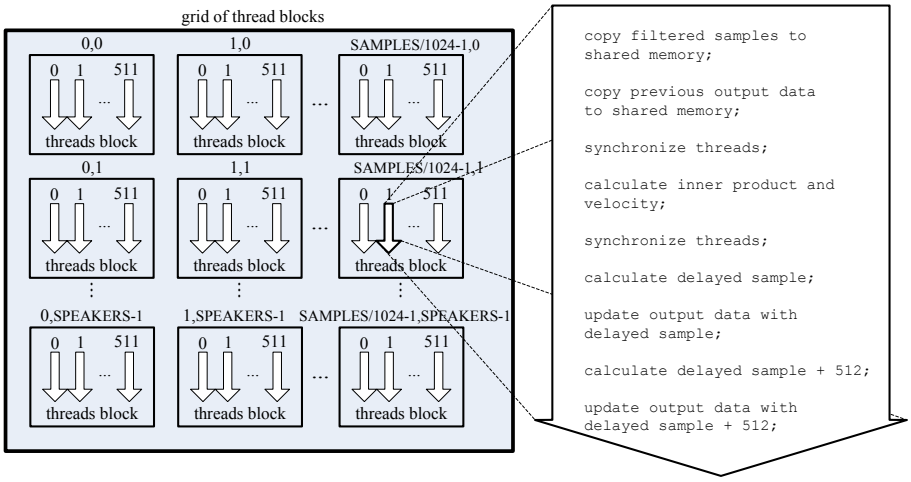

Figure 5.6: Grid of thread blocks that are dispatched during the WFS calculations to the GPU.

kernels developed. In order to filter all audio sources, we used the same FIR filter GPU kernel that was described in BF. However, in this case the FIR filter consists of 64 taps, so 64 threads are launched within each thread block. Figure 5.6 illustrates the WFS implementation to the GPU. Input signals are divided into chunks of 1024 samples. We designed a 2D-grid of thread blocks, where each block consists of 512 GPU threads. Each thread within a block is responsible for calculating 2 output samples, thus when a thread block is executed, 1024 samples are calculated. Assuming there are *SAMPLES* input samples to be processed and rendered through L loudspeakers, we need $\frac{SAMPLES}{1024} \cdot L$ thread blocks for each loudspeaker. Therefore, we designed the 2D-grid with dimensions $\frac{SAMPLES}{1024}$ by L.

5.4 Conclusions

In this chapter we conducted a case study on mapping the proposed immersive-audio instructions onto GPUs, one of the most popular contemporary nr-MCP platforms. Although we used NVidia's CUDA parallel programming environment to develop the instructions implementation, as it was mentioned in Chapter 5.1, all CUDA-specific code annotation details are hidden from the user. More specifically, our architecture implementation uses efficiently the high-level parameters that are provided by the programmer using the instructions presented in Section 3.3, in order to schedule the grid size of the thread-blocks, when a kernel is executed to the GPU. Finally, specific attention was paid to preserve a dynamic GPU memory allocation, in order to support various filter sizes and coefficients sets.

This chapter is based on the following paper:

D. Theodoropoulos, CB Ciobanu, G. Kuzmanov, **Wave Field Synthesis for 3D Audio: Architectural Prospectives**, ACM International Conference on Computing Frontiers, pp. 127-136, Ischia, Italy, May 2009

6

Experimental Results

In this chapter, we present our experimental results for both beamforming (BF) and Wave Field Synthesis (WFS) architectures when mapped onto FPGAs, GPUs and compared against GPP-based approaches. As it was mentioned in the previous chapter, the software BF application was provided to us by the Fraunhofer Research Institute, Germany. Moreover, the WFS implementation was provided to us by the Sound Control Department of the Delft University of Technology, The Netherlands. The BF algorithm has been already used as an evaluation application for a custom-made hardware platform under the hArtes European project [51]. The WFS algorithm has been already used to develop and test a real WFS-based audio system that is presented in [92]. We selected the Xilinx Virtex4 and Virtex6 families for our multi-core reconfigurable hardware prototypes. Regarding the GPUs that were used during our experiments, we chose NVidia chips that represent a wide range of the market, namely a low-end Quadro FX1700, a slightly faster GeForce 8600GT [65], a high-end GeForce GTX275 [66], and the newest GTX460 [69] that utilizes the Fermi architecture [68]. Finally, we employed an Intel Core2 Duo processor at 3.0 GHz to run both OpenMP-annotated [72] BF and WFS applications, which are considered as the two baseline implementations.

The rest of the chapter is organized as follows: In Section 6.1 we describe the experimental setup that we applied, in order to test our FPGA and GPU-based implementations regarding performance for the BF application. Furthermore, we compare the results accuracy of our hardware approach against the Core2 Duo, since the former employs a fixed-point format for all internal calculations. In the same section, we provide a comparison of the two multi-core systems against the Core2 Duo and related work. In addition, we discuss each system energy consumption and overall cost. The same structure is also applied in Section 6.2, where we report our experimental results for the WFS application. Finally, Section 6.3 concludes the chapter.

Table 6.1: Resource utilization of each module

Module	Slices	DSP Slices	Memory(bytes)
Single BeamFormer	598	2	8192
Source Amplifier	2870	0	2048
MC-BFP	14165	32	133120
System infrastructure	6650	0	317440
Complete system with C=16	20815	32	450560

Table 6.2: Maximum number of *BeamFormers* that can fit in different FPGAs

FPGA	# of BeamFormers fit
V4FX60	19
V4FX100	54
V4FX140	89
6VLX75T	78
6VLX760	360
6VSX315T	352
6VSX475T	532
6VHX250T	252
6VHX565T	432

6.1 BF Experimental Results

FPGA prototype: We used the Xilinx ISE 9.2 [103] and EDK 9.2 [101] CAD tools to develop a VHDL hardware prototype of our Multi-Core Beamforming Processor (MC-BFP). The latter was implemented on a Xilinx ML410 board [102] with a Virtex4 FX60 FPGA [106] and 256 MB of DDR2 SDRAM. As host GPP processor, we used one of the two integrated PowerPC processors [100]. Furthermore, we used the Processor Local Bus (PLB) [110] to connect all peripherals, which are all on-chip *BF buffers*, the *source buffer*, all SPRs, and the DMA [104] and SDRAM controllers. For the partial reconfiguration we have used the Xilinx Internal Communication Access Port (ICAP) [111], which is also connected to the PLB. The PowerPC runs at 200 MHz, while the rest of the system is clocked at 100 MHz when mapped onto a Virtex4 chip. When the design is mapped onto a Virtex6 FPGA, we utilize the Microblaze soft-core processor and the hardware is clocked at 200 MHz. Our prototype is configured with C=16 *BeamFormer* modules, thus it can process up 16 input channels concurrently. Also, within each BF-PE and the *source amplifier*, all decimators, H(z) filters and the interpolator were generated using the Xilinx Core Generator [99] [98].

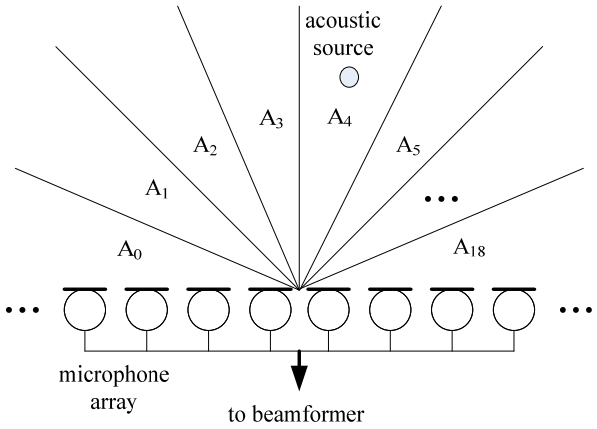

Figure 6.1: Microphone array setup and source position inside aperture A_4.

Table 6.1 displays the resource utilization of each module. The first two lines provide the required resources for a single *BeamFormer* and the *source amplifier* modules. The third line shows all hardware resources occupied by the MC-BFP. In the fourth line, we show the resources required to implement the PLB, DMA, ICAP and all memory controllers with their corresponding BRAMs. Finally, the fifth line provides all required resources for the entire BF system.

As it can be observed, a single *BeamFormer* requires less than 600 slices, 2 DSP slices [107] and 8 Kbytes of Block RAM (BRAM), which makes it feasible to integrate many such modules within a single chip. Table 6.2 shows how many *BeamFormers* could fit into different V4FX FPGA chips. Moreover, even a medium-sized FPGA can support up to 19 channels, while larger chips, like the V4FX100 and V4FX140, could accommodate up to 54 and 89 input channels respectively. Of course, newer FPGA families, like the Xilinx Virtex6 [108], integrate much more resources, thus can fit more *BeamFormer* modules. In order to investigate how many input channels a single Virtex6 could accommodate, we used the Xilinx ISE 11.4 [109] and mapped our MC-BFP onto different chips of the FPGA family. As we can observe from Table 6.2, a 6VLX75T FPGA chip, which is the smallest of the Virtex6 family, could fit up to 78 input channels. Moreover, the 6VSX475T chip, which is the largest one, could support setups that consist up to 532 microphones. We should note that during our calculations we took into account the required area to also map a Microblaze processor [112] onto the reconfigurable hardware, since the Virtex6 families do not integrate any hard-core processor.

Data accuracy: In order to evaluate the data accuracy of our GPU and FPGA

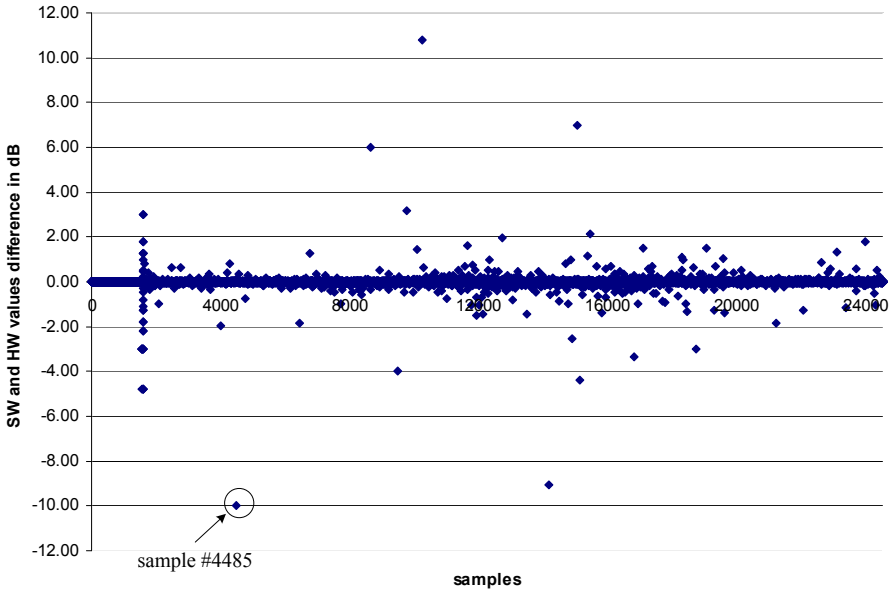

Figure 6.2: Difference between software and hardware values for an acoustic source in dBs inside aperture A_4.

implementations, we compared their output results against the ones from a Core2 Duo-based approach. As stimulation input, we used recorded signals from up to 16 microphones that consist of 24576 samples in a 16-bit signed format. However, the Core2 Duo BF application performs all calculations using the IEEE 754 standard single precision floating point format, which is also the case with all GPUs that were used for our experiments. In contrast, our MC-BFP follows a fixed-point format approach, in order to reduce area utilization and increase performance. However, due to the fixed-point format, a calculation error is introduced, which results in a reduced output accuracy compared to the IEEE 754 single precision floating point format.

In order to verify that the reduced accuracy does not affect the extracted source quality, we compared the software and hardware sample values, in the exemplary case of a source being located within source aperture A_4, as illustrated in Figure 6.1. We used the following formula to estimate the introduced error for each calculated signal sample:

$$DdB = 10 \cdot \log\left(\frac{s_{SW}}{s_{HW}}\right) dBs \tag{6.1}$$

Table 6.3: GPUs specifications for all experiments.

GPU chip	Core (Mhz)	CUDA core (Mhz)	Memory BW (GB/s)	# of CUDA cores	FF-XIV Score
FX1700	207	414	12.8	32	694
8600GT	540	1190	22.4	32	898
GTX275	633	1404	127	240	3817
GTX460	675	1350	115.2	336	3716

where s_{SW} and s_{HW} are the software and hardware sample values respectively. Figure 6.2 shows the introduced error for an extracted source signal consisting of 24576 16-bit samples. In the ideal case, the difference between the two values should be zero. As it can be observed, almost all introduced errors do exceed a +/-0.01 decibels (dBs) boundary. The few exceptional cases where the difference is large, are because the absolute sample value is very low. For example, as depicted in Figure 6.1, the sample #4485 has a value difference of 10 dBs. This happens because the correct value s_{SW} is 1, however the s_{HW} is 10. In practise however, this would not introduce any loss in quality, since both values are very close to 0. In total, by taking into account these exceptional cases, we measured that the hardware output extracted source signal of our MC-BFP is 99.6% accurate to the software one.

Hardware prototypes performance evaluation: Regarding the GPUs that were used during our experiments, we chose chips that would represent a wide range of the market. Table 6.3 presents the specifications of the GPUs considered. It can be observed that the 8600GT has the same number of CUDA cores comparing to the FX1700 chip. However, the 8600GT has higher memory bandwidth (BW) and faster Core and CUDA core clocks compared to the FX1700. Furthermore, the GTX275 and the GTX460 consist of 240 and 336 CUDA cores respectively, an order of magnitude higher memory bandwidth, and faster clocks compared to the other two GPUs. In order to test the processing capabilities of each GPU, we installed each one of them as a secondary GPU to our system and ran the official Final Fantasy XIV (FFXIV) benchmark for NVidia GPUs[1] on low resolution. The achieved benchmark score of each GPU is shown to the rightmost column of Table 6.3. As we can observe, the 8600GT scored $\sim$30% higher than the FX1700, while the GTX275 performed 2.7% better than the GTX460. We should note than, while running the FFXIV benchmark and our experiments, no over-clocking was applied to any of the GPUs considered.

In order to test the performance of all platforms, we provided input signals

[1] http://www.finalfantasyxiv.com/media/benchmark/na/#1

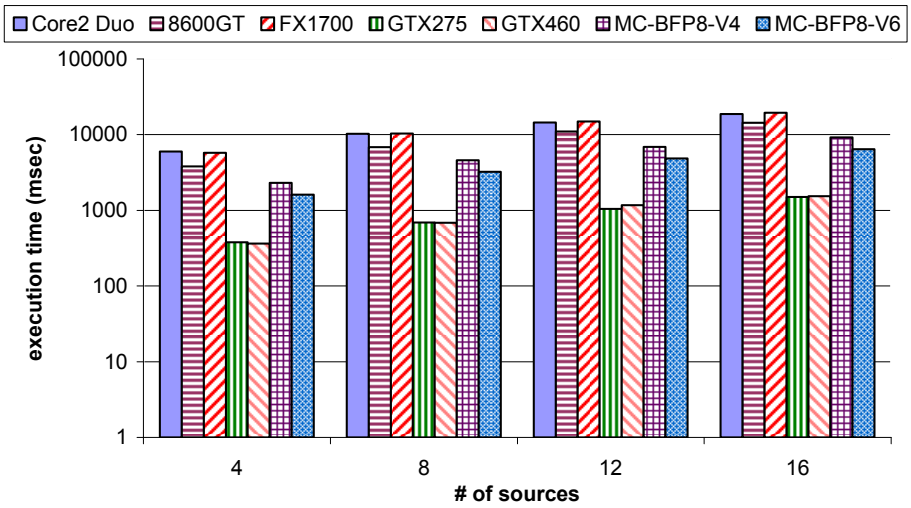

Figure 6.3: Execution time on all platforms under an 8-microphone setup.

consisting of 540672 audio samples, divided into 528 1024-sample chunks. Assuming a sampling frequency of 48000 kHz, each iteration should be calculated in $\frac{1024}{48000}\ \frac{samples}{samples/sec} \approx 21.33$ msec. Since there are in total 528 iterations, every hardware platform that is considered for a real-time audio system must complete all data calculations within $528\ iterations \cdot 21.33\ \frac{msec}{iteration} \approx 11.264$ sec.

To evaluate our approach, we conducted experiments for all hardware platforms with 8 and 16 channels and up to 16 sources. Regarding the different FPGA implementations, we use the "MC-BFPx-Vy" naming rule, where x defines the number of input channels the design uses and y refers to the utilized Virtex FPGA family, that is y=4 for Virtex4 and y=6 for Virtex6. Figure 6.3 depicts the execution times for setups consisting of 8 input channels, mapped onto all platforms considered. It can be observed that all platforms can process all data in real time, thus within 11.264 sec, for up to 8 sources. In the case of 12 sources, the Core2 Duo and the FX1700, which is the simplest of all GPUs, fail to process data faster than the actual source length, thus making these devices not suitable for such real-time implementations. The 8600GT cannot also process all data fast enough when there are 16 sources. In contrast, the Virtex4-based MC-BFP with 8 *BeamFormer* modules (MC-BFP8-V4), the Virtex6-based MC-BFP with 8 *BeamFormer* modules (MC-BFP8-V6), and the GTX275 and GTX460 implementations could be used to build real-time systems that are capable to extract up to 16 sources. An interesting observation

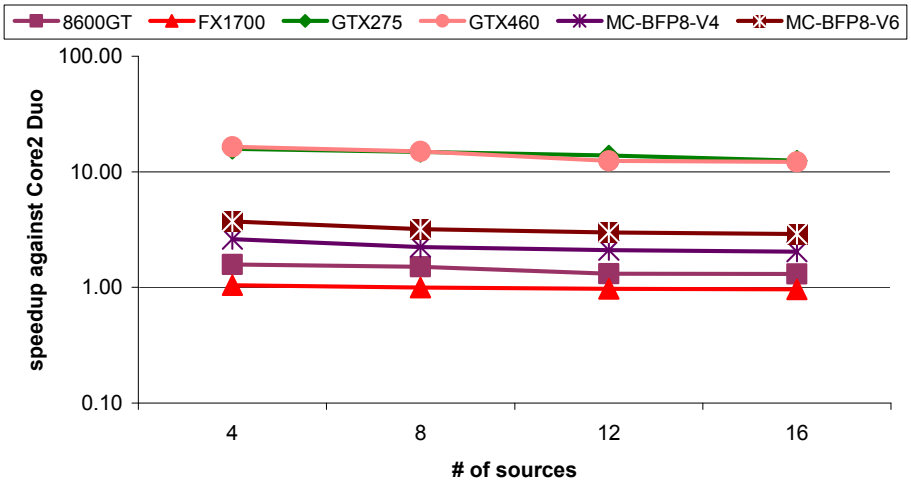

Figure 6.4: Execution speedup of all platforms against the Core2 Duo under an 8-microphone setup.

that can be made from the execution times shown in Figure 6.3, is that the GTX275 performs equally well to the GTX460.

Figure 6.4 illustrates the obtained speedup of each platform against the Core2 Duo. As we can observe, under an 8-source scenario the FX1700 GPU is actually slower than the software solution. In contrast, although the 8600GT integrates the same number of CUDA cores as the FX1700, it achieves a speedup up to 1.58 against the Core2 Duo. Regarding the FPGA implementation, we investigated the ratio of the overall execution time that was spent on accessing the external memory, which was up to 40%. The reason is because all peripherals are connected to the same PLB, thus leading to a slow external memory access time. Moreover, as we can observe from Figure 6.4, although the MC-BFP8-V6 implementation functions at double frequency compared to the one of MC-BFP8-V4, the overall execution time is improved by approximately 42%.

In Figure 6.5 we provide the results of our experiments when utilizing a 16-microphone setup. Apparently the FX1700 fails to process data in real-time for all cases. The Core2 Duo and 8600GT could be used for a real-time BF system when there are up to 4 sources to be extracted. The Virtex4-based MC-BFP with 16 *BeamFormer* modules (MC-BFP16-V4), could be used for a BF system that supports up to 14 sources, while the Virtex6-based MC-BFP with 16 *BeamFormer* modules (MC-BFP16-V6) can support up to 16 sources. As it was in the case of 8 microphones, the slow SDRAM interface through

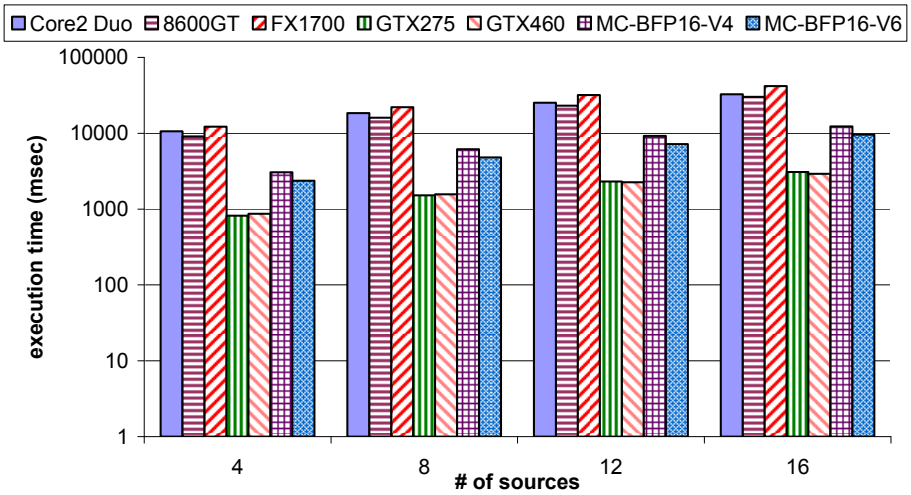

Figure 6.5: Execution time on all platforms under a 16-microphone setup.

the PLB affects significantly the achieved FPGA performance, since its access time occupies up to 55% of the overall application execution time. Finally, the GTX275 and GTX460 implementations could be used to build a real-time system that is capable to extract up to 16 sources.

In addition, Figure 6.6 suggests the achieved speedup of all platforms against the Core2 Duo implementation. As it was in the case of the 8-microphone setup, the FX1700 processes data slower than the software approach. Furthermore, the 8600GT again is slightly faster than the Core2 Duo under every source scenario. The MC-BFP16-V4 and MC-BFP16-V6 FPGA prototypes performs only up to 3.5 and 4.5 times better than the software implementation, because of the currently used slow external memory interface. Finally, the two high-end GPUs are an order of magnitude more efficient compared to the Core2 Duo processor.

From the experiments conducted, we can draw the main conclusion that the external memory interface through the PLB can significantly affect the performance of reconfigurable BF systems. Since there are many microphone input signals to be read from the external memory, a slow interface severely affects the overall performance. Figure 6.7 shows the required and actual MC-BFP16-V4 memory bandwidth in each processing iteration under different source scenarios, when there are 8 and 16 input channels. As we can observe, the MC-BFP16-V4 provides the required memory bandwidth in every source scenario under an 8-channel setup. However, when there are 16 input channels and 16

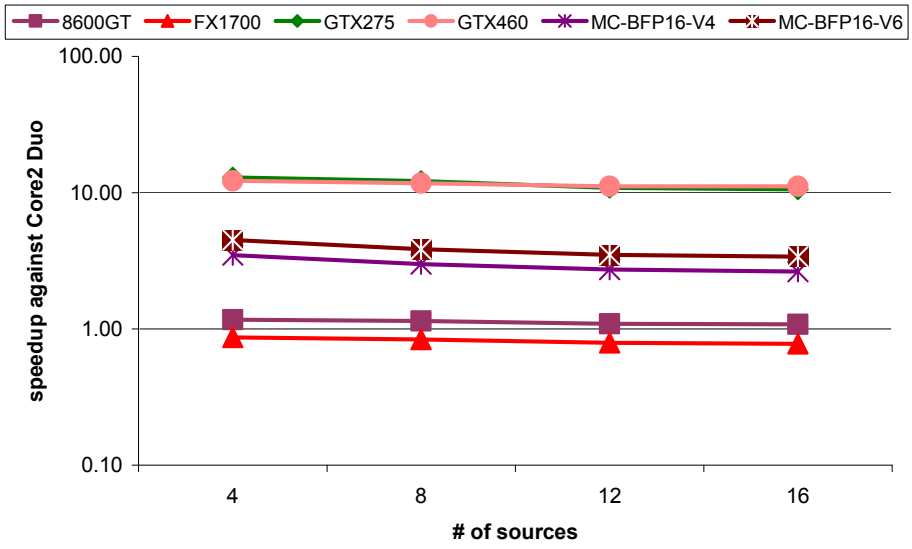

Figure 6.6: Execution speedup of all platforms against the Core2 Duo under a 16-microphone setup.

sources, the MC-BFP16-V4 does not meet the external memory requirements, thus failing to successfully extract all sources in real-time.

GPU and FPGA BF architectural prospectives: All execution times reported above, include the external memory access delays. However, each platform has its own memory hierarchy, thus memory overhead impacts the overall execution time differently. For example, the Core2 Duo has to access the external memory through the Front Side Bus (FSB) and the motherboard chipset (usually referred to as North Bridge), while GPUs and FPGAs have direct connection with their own external memory. Since the GTX275, the GTX460 and the FPGA implementations performed much better than the other platforms, we decided to measure the memory impact, exclude it from the total execution time, and compare them in terms of processing speed. However, each platform is fabricated using a different technology. More specifically, the Virtex4 FPGA families are fabricated using a 90-nm technology, while the GTX275 and GTX460 use a 55-nm and 40-nm one respectively. In order to use a common comparison technology, we decided to use the advanced 40-nm one. Moreover, in the context of the platform evaluation, we used our MC-BFP16-V6 prototype with its maximum operating frequency at 285 MHz.

In order to normalize the technology factor for the GTX275, we referred to the International Technology Roadmap for Semiconductors (ITRS) 2005 re-

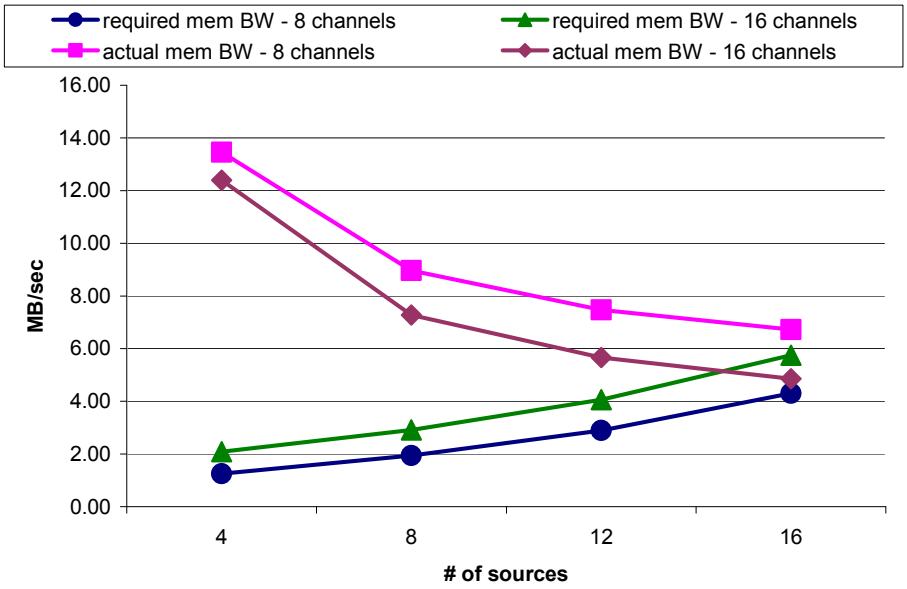

Figure 6.7: Required and actual memory bandwidth achieved by the MC-BFP16-V4 design.

port [45] to calculate the percentage ratio of the GPU processing time reduction, assuming that the chip would be fabricated using a 40-nm technology. Throughout this thesis, we refer to it as *optimized GTX275*. Based on the ITRS report and aware of all the odds of such estimation, we could safely assume that the 40-nm technology has potentials for 30% lower execution time than the 55-nm one. Furthermore, since we know exactly how much data needs to be accessed during the GPU kernel execution, we used the CUDA Visual Profiler 3.1 to measure the achieved memory throughput, and based on that, subtract the external memory access time from the total execution time. For the GTX275, we calculated that an average of 54.1% of the total execution time throughout our experiments was spent on accessing the GPU external memory. The remaining time, which is the *actual* GPU processing time, was multiplied by a factor of 0.7, in order to be normalized to the 40-nm hypothetical GPU fabrication. Regarding the GTX460, the CUDA Profiler 3.1 unfortunately provides the L1 cache hits % per kernel, and not the achieved memory throughput. Thus, the aforementioned GPU has been excluded from our comparison of the processing times.

Figure 6.8 summarizes the comparison between the optimized GTX275 and the MC-BFP-V6 processing times. On the Y-axis, if the ratio is smaller than

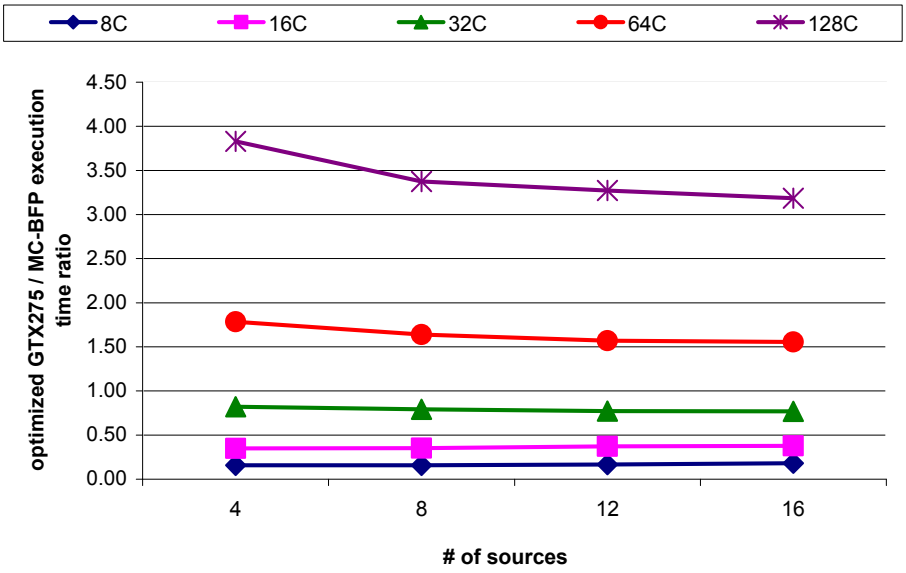

Figure 6.8: Processing time comparison between the optimized GTX275 and MC-BFP approaches for the BF.

1.0, the GPU is faster, otherwise the FPGA performs better. We performed additional experiments with up to 128 channels. The reason that we can fit so many *BeamFormer* modules is because a minimal resource utilization strategy was followed in the current BF implementation. As it was already indicated in Table 6.1, because each *BeamFormer* module occupies very few resources, we can accommodate systems that utilize microphone arrays consisting of hundreds of elements. In Figure 6.8, it can be observed that, when there are up to 32 input channels, the optimized GTX275 performs all data calculations faster than the FPGA. However, when the input channels are more, the FPGA processes data faster than the optimized GTX275, because all *BeamFormer* modules work concurrently on all input channels. In other words, when there are more than 32 input channels, the GTX275 processing resources are saturated, while the MC-BFP can still process concurrently all input channels up to 3.8 times faster than the GPU-based approach. This observation, leads us to the conclusion that high-end GPUs can support more efficiently microphone setups up to a certain number of input channels. When this number of channels is exceeded, a MC-BFP-V6 approach has the potential to process data more efficiently than high-end GPUs.

Energy consumption: Performance is not the only parameter to be considered when choosing a hardware platform for a BF system. Energy consumption is

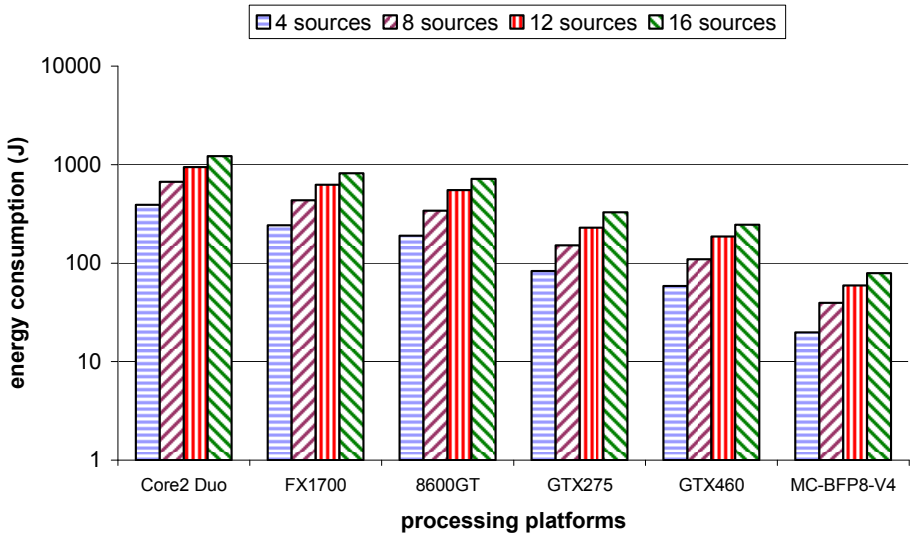

Figure 6.9: Energy consumption of all platforms under an 8-microphone setup.

an additional parameter that should also be taken into account. In order to evaluate the consumed energy from each platform, we used the formula:

$$P = \frac{E}{t} \Leftrightarrow E = P \cdot t \tag{6.2}$$

where E is the energy consumed during the time t, while applying P power. The primary Y-axis in Figures 6.9 and 6.10 suggest the energy consumption by each processing platform under an 8- and 16-microphone setup for different number of sources. As it can be observed, in every case the Core2 Duo-based system consumes the most energy. Even though its peak power consumption is 65 Watts [42], the fact that it requires much more time to process all data, results to the maximum energy consumed among all tested platforms. The FX1700-based system consumes less energy than the Core2 Duo, because it requires only 42 Watts of power. Moreover, the 8600GT-based approach, not only calculates data faster than the Core2 Duo, but it also consumes less energy. This observation, leads to the conclusion that utilizing a 8600GT GPU as processing platform for simple BF with few acoustic sources, would result to a more energy-efficient BF solution than employing a Core2 Duo GPP. Even better solutions can be provided by high-end GPUs. Although the GTX275 and GT460 consume 219 and 160 Watts respectively, they can process data very fast, thus consume lower energy than middle-ranged GPUs and the Core2

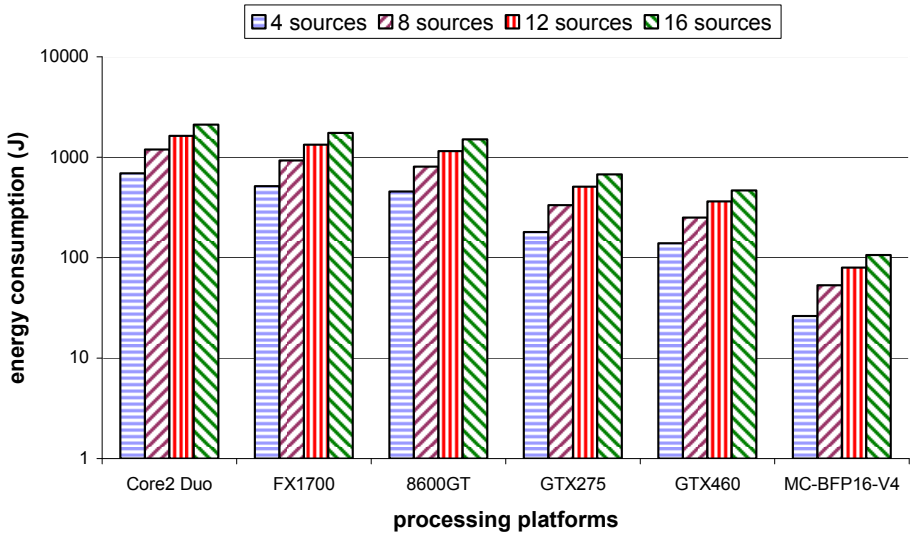

Figure 6.10: Energy consumption of all platforms under a 16-microphone setup.

Table 6.4: Platform costs in Euros.

Platform	Cost (Euros - Nov. 2010)
Core2 Duo	116
FX1700	165
8600GT	55
GTX275	117
GTX460	114
ML410	2198

Duo GPP. Ultimately, the MC-BFP16-V4 prototype, according to the Xilinx XPower utility [113], requires approximately 8.6 Watts, thus resulting to the lowest energy consumption among all tested platforms.

Platform cost: The platform cost is also another parameter that should be considered when building a BF system. Table 6.4 shows the unit price of each platform in Euros (November 2010). As we can see, the FX1700 is more expensive than the Core2 Duo GPP. Consequently, by taking also into account its poor performance, we can conclude that the FX1700 is not an effective platform for implementing a BF system. In contrast, the 8600GT is cheaper, more energy- and performance-efficient than the Core2 Duo and the FX1700, which makes it an attractive solution for implementing simple BF systems. Regarding the high-end GPUs, the GTX275 and GTX460 cost slightly more than the Core2 Duo. However, provided that they process data very fast, they make an

Table 6.5: GPU- and FPGA-based implementations comparison against related work.

Design	# of channels	# of supported sources	Sampling frequency (kHz)
[114]	2	4	48
[48]	4	19	16
optimized GTX275 GPU	16	174	48
MC-BFP16-V6	16	65	48

efficient solution for building energy- and performance-efficient complex BF systems, which could not be supported by the Core2 Duo and middle-ranged GPUs. Finally, since FPGA platforms are targeted only by a much more limited group of people compared to off-the-shelf GPPs and GPUs, they have a high cost. Thus, an MC-BFP-based approach can be considered for ultra-low energy consumption solutions that can also accommodate complex BF systems.

Comparison against related work: Table 6.5 provides a comparison of the optimized GTX275 approach and the MC-BFP16-V6 design against related work. In both cases we assume a 16-microphone setup and an ideal memory bandwidth utilization, which is 127 GB/sec and 6.4 GB/sec for the GPU- and FPGA-based BF systems respectively. Direct comparison against related work is not straightforward, since each system has its own design specifications. Moreover, to our best knowledge, we provide the first architectural proposal for reconfigurable BF. Previous proposals are mainly micro-architectural ones. In [114], the authors utilize an ADSP21262 DSP, which consumes up to 250 mA. Furthermore, the voltage supply of ADSP21262 is 3.3 V [5], thus we can assume that the design requires approximately 3.3 V·0.25 A = 0.825 W. In addition, according to the paper, the ADSP21262 is 50% utilized when processing data from a two-microphones array at 48 KHz sampling rate, or alternatively 48000 samples/sec/input·2 inputs = 96000 samples/sec. Based on this, we can assume that 192000 samples/sec can be processed in real-time with 100% processor utilization, which means $\lfloor 192000/48000 \rfloor = 4$ sources can be extracted in real-time. Finally, in [48] the authors use four microphones to record sound and perform Beamforming using an FPGA. They have mapped their design onto a V4SX55 FPGA and, according to the paper, every instance of the proposed beamformer can process 43463 samples/sec, with up to seven instances fitting into the largest V4SX FPGA family. Since the sampling frequency is 16 KHz, $\lfloor (43463 \cdot 7)/16000 \rfloor = 19$ sources could be extracted in real-time.

Table 6.6: Resource utilization of each module

Module	Slices	DSP Slices	Memory(bytes)
Single RU (100 MHz)	3566	26	36864
Common modules among all RUs	6734	0	2048
MC-WFSP with 4 RUs	20998	104	149504
System infrastructure	3213	0	227328
Complete system with R=4 RUs	24211	104	376832

Table 6.7: Maximum number of *RUs* that can fit in different FPGAs

FPGA	# of *RUs* fit
V4FX40	1
V4FX60	4
V4FX100	6
V4FX140	7
6VLX75T	9
6VLX760	33
6VSX315T	50
6VSX475T	77
6VHX250T	22
6VHX565T	33

6.2 WFS Experimental Results

FPGA prototype: As it was in the case of the BF application, we used the Xilinx ISE 9.2 and EDK 9.2 CAD tools to develop a VHDL hardware prototype of our Multi-Core WFS Processor (MC-WFSP), and implement it on a Xilinx ML410 board. Again, as host GPP processor, we used one of the two integrated PowerPC processors and the PLB to connect all peripherals, which are all *RU buffers*, the *source buffer*, all SPRs, and the DMA and SDRAM controllers. For the partial reconfiguration we have used the Xilinx ICAP, which is also connected to the PLB. Regarding the Virtex4-based prototype, the PowerPC runs at 200 MHz, while the rest of the system is clocked at 100 MHz. The Virtex6-based implementations utilize a Microblaze soft-core processor and the hardware is clocked at 200 MHz. Our prototypes were configured with R=4 Rendering Units (RUs).

Table 6.6 displays the resource utilization of each module. The first line provides the required resources for a single *RU*. The second line shows all required resources to implement the common modules among all *RUs*. The third line shows all hardware resources occupied by the MC-WFSP. In the fourth line, we show the resources required to implement the PLB, DMA, ICAP and all memory controllers with their corresponding BRAMs. Finally, the fifth line

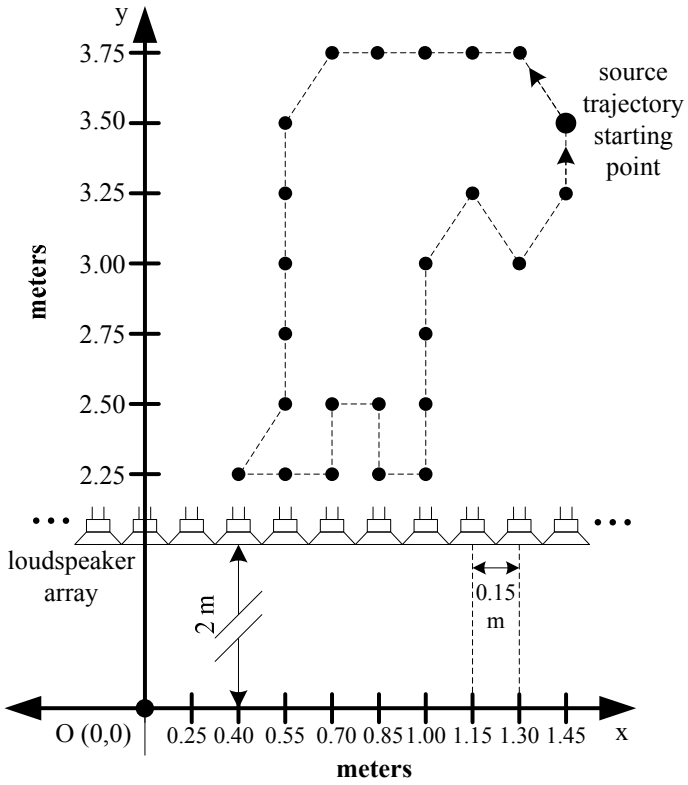

Figure 6.11: Loudspeaker array setup and source trajectory behind the array.

provides all required resources from the entire WFS system.

As we can observe, a single *RU* requires 3566 slices, which makes it feasible to integrate more such modules within a single chip. Table 6.7 suggests how many *RUs* could fit into different V4FX FPGA chips. Based on our estimations, even a medium-sized FPGA, like the V4FX40, can host a complete WFS system with 1 *RU*. Moreover, larger Virtex4 chips, like the V4FX60, V4FX100 and V4FX140, could accommodate up to 4, 6 and 7 *RUs* respectively. We also investigated how many *RUs* could fit in newer FPGA chips, like the Virtex6 family. Again, we used the Xilinx ISE 11.4 to map our MC-WFSP to different chips. Table 6.7 suggests the number of *RUs* that can fit into different FPGAs. As we can observe, the 6VSX475T chip, which is the largest one, can fit up to 77 *RUs*, thus allowing the realization of very fast and complex WFS systems. As before, we should note that during our calculations we took into account the required area to also map a Microblaze processor onto the reconfigurable hardware, since the Virtex6 families do not integrate any hard-core processor.

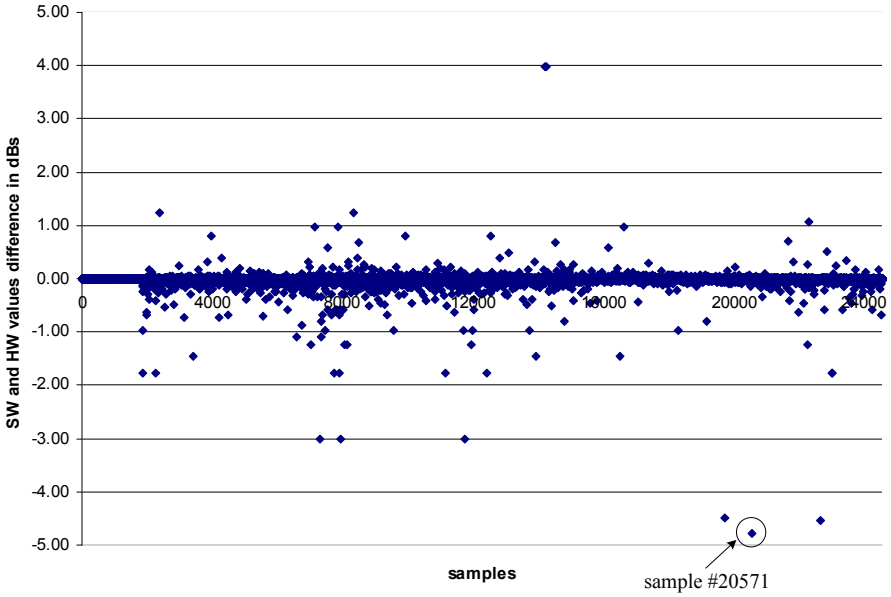

Figure 6.12: Difference between software and hardware values for a loudspeaker signal in dBs.

Data accuracy: In order to evaluate the data accuracy of our GPU and FPGA implementations, we compared their output results against the ones from a Core2 Duo-based approach, which is considered the baseline. As stimulation input to each implementation, we used the source signal that was extracted from the corresponding platform by using the BF technique. For example, for the GPUs we used the BF output source signal as input to the WFS implementation. Consequently, the input signal consisted of 24576 source samples in a 16-bit signed format and all calculations were done using the IEEE 754 single precision floating point format. Regarding the MC-WFSP implementations, we used a fixed-point format, in order to save area utilization and increase performance compared to a floating-point approach. However, due to the fixed-point format, a calculation error is introduced, which results in a reduced output accuracy compared to the IEEE 754 floating point format.

In order to verify that the reduced accuracy does not affect the loudspeaker signals quality, we compared the software and hardware sample values, in the exemplary case of a source moving behind a loudspeaker array, as illustrated in Figure 6.11. The loudspeaker array consisted of 32 elements, each having a 0.15 m distance between them. The array is located 2 m away from the listening area center. We applied the formula (6.1) to estimate the introduced

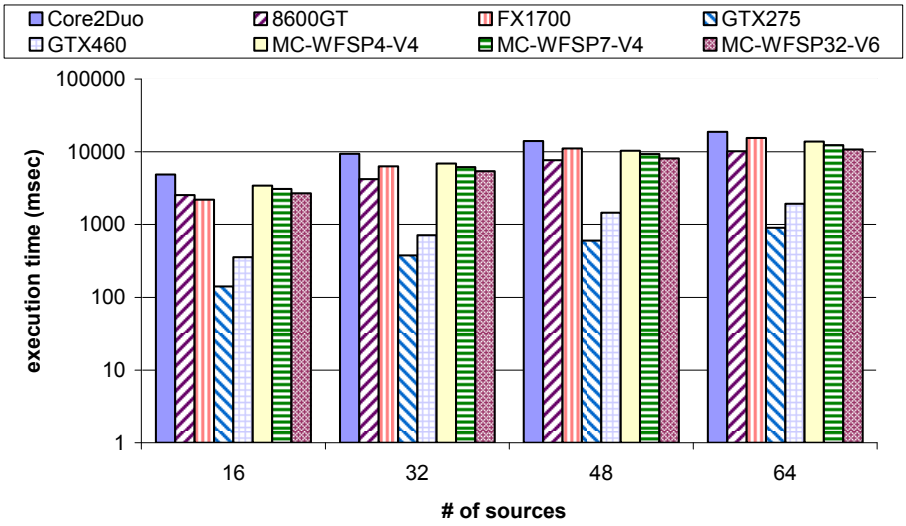

Figure 6.13: Execution time on all platforms under a 32-loudspeaker setup.

error for each loudspeaker output signal. Figure 6.12 shows as an example, the introduced calculation error to the rightmost loudspeaker signal that consists of 24576 16-bit samples. In the ideal case, the difference between the two values should be zero. As we can observe, almost all introduced errors do exceed a +/-0.01 decibels (dBs) boundary. In the few exceptional cases where the difference is large, is because the absolute sample value is very low. For example, as depicted in Figure 6.12, the sample #20571 has a value difference of -4.7 dBs. This happens because the correct value s_{SW} is -1, however the s_{HW} is -3. In practise however, this would not introduce any loss in quality, since both values are very close to 0. In total, by taking into account these exceptional cases, we measured that the hardware output loudspeaker signal of our MC-WFSP is 99.4% accurate to the software one.

Hardware prototypes performance evaluation: For the purposes of our GPU experiments, we used again the chips that have already been discussed in Section 6.1 and presented in Table 6.3. Regarding the MC-WFSP, we considered an improved implementation from the one presented Table 6.6. The enhanced prototype utilizes a 128-bit wide PLB, the PowerPC runs at 300 MHz and the MC-WFSP uses 150 MHz clock. Moreover, based on the fact that our MC-WFSP can integrate different number of RUs, for our experiments we considered the largest number that each FPGA family could fit.

In order to test the performance of all platforms, we provided source signals

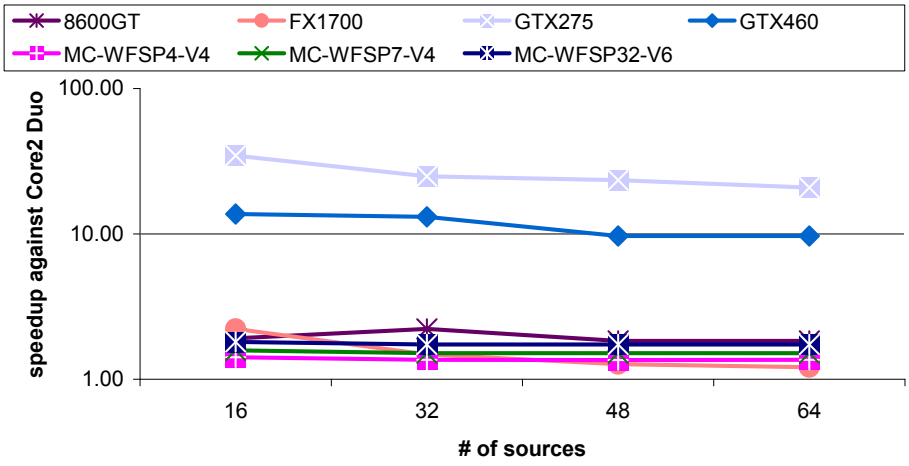

Figure 6.14: Execution speedup of all platforms against the Core2 Duo under a 32-loudspeaker setup.

consisting of 540672 audio samples, divided into 528 1024-sample chunks. Assuming a sampling frequency of 48000 kHz, each iteration should be calculated in $\frac{1024}{48000}\ \frac{samples}{samples/sec} \approx 21.33$ msec. Since there are in total 528 iterations, every hardware platform that is considered for a real-time audio system must complete all data calculations within $528\ iterations \cdot 21.33\ \frac{msec}{iteration} \approx 11.264$ sec. Regarding the different FPGA implementations, we use the "MC-WFSPx-Vy" naming rule, where x defines the number of RUs the design uses and y refers to the utilized Virtex FPGA family, that is y=4 for Virtex4 and y=6 for Virtex6.

Figure 6.13 shows the execution times for all considered platforms under a 32-loudspeaker setup. As we can observe, all systems can support rendering up to 32 real-time sources when driving 32 loudspeakers. However, the Core2 Duo implementation fails to meet the real-time constraints when the number of real-time sources increases to 48. When there are 64 sources to be rendered the FX1700 fails to meet the timing constraints, while the 8600GT, the MC-WFSP32-V6 and the two high-end GPUs can support such complex WFS systems. Regarding the FPGA implementations, the Virtex4-based designs with 4 RUs (MC-WFSP4-V4) and 7 RUs (MC-WFSP7-V4) can support up 60 sources, and the reason is again the slow external memory interface. The Virtex6-based design with 32 RUs (MC-WFSP32-V6) also suffers the same low memory access time, however due to the increased functioning frequency can support up to 64 sources.

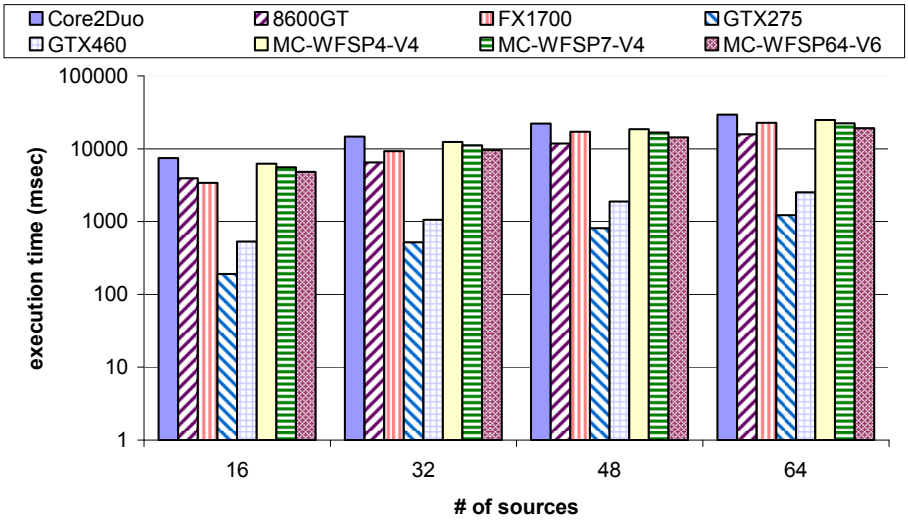

Figure 6.15: Execution time on all platforms under a 64-loudspeaker setup.

Figure 6.14 compares the WFS application execution speedup on all platforms against the Core2 Duo baseline approach. As we can observe, the MC-WFSP7-V4 and MC-WFSP32-V6 approaches are up to 57% and 81% faster than the Core2 Duo. In addition, the FX1700 and 8600GT GPUs achieve an application execution speedup up to 2.2 times, while the high-end GPUs process data at least an order of magnitude faster than the GPP-based system.

Figure 6.15 shows the execution times of every considered platform under a 64-loudspeaker setup. As it can be observed, when there are 16 sources to be rendered, all platforms meet the timing constraints, thus can be used for implementing real-time WFS systems. When the number of sources is increased to 32, the Core2 Duo and the MC-WFSP4-V4 exceed the 11.264 sec time limit. In contrast, the MC-WFSP7-V4, the Virtex6-based design with 64 RUs (MC-WFSP64-V6) and all GPUs perform all calculations within the required time window. Under a 48-source scenario, the FX1700 also fails to process all data fast enough, while when there are 64 sources only the GTX275 and GTX460 could be used to implement a real-time WFS system. As it was explained before, the slow SDRAM interface severely affects the overall WFS execution time of reconfigurable systems, resulting to a poorer performance compared to the high-end GPU-based approaches.

Finally, regarding the application speedup against the Core2 Duo baseline system, we observed the same platform classification as in the case of a 32-

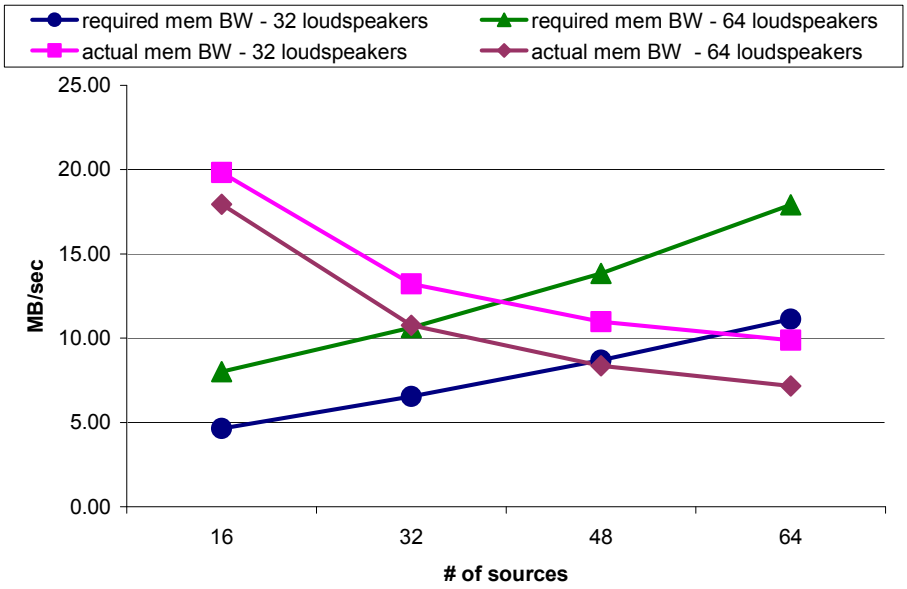

Figure 6.16: Required and actual memory bandwidth achieved by the MC-WFSP7-V4 design.

loudspeaker setup. Figure 6.17 suggests the application speedup for every platform against the software approach. As it can be seen all platforms are always faster than the Core2 Duo under every real-time source scenario. Moreover, the two high-end GPUs are again more than an order of magnitude faster than the GPP-based approach.

From the above experiments, we can draw the conclusion that a slow external memory interface can considerably affect the performance of reconfigurable WFS systems. Since there are many source input signals and loudspeaker output signals to be read and written to the external memory respectively, a slow interface severely affects the overall performance. Figure 6.16 shows the required and actual MC-WFSP7-V4 memory bandwidth in each processing iteration under different source scenarios, when there are 32 and 64 output channels. As we can observe, the MC-WFSP7-V4 provides the required memory bandwidth when there are up to 60 sources under an 32-channel setup. However, when there are 64 loudspeakers and more than 32 sources, the MC-BFP16-V4 does not meet the external memory requirements, thus failing to successfully render all sources in real-time.

GPU and FPGA WFS architectural prospectives: In order to have a more accurate evaluation of each platform's processing potentials, we followed the

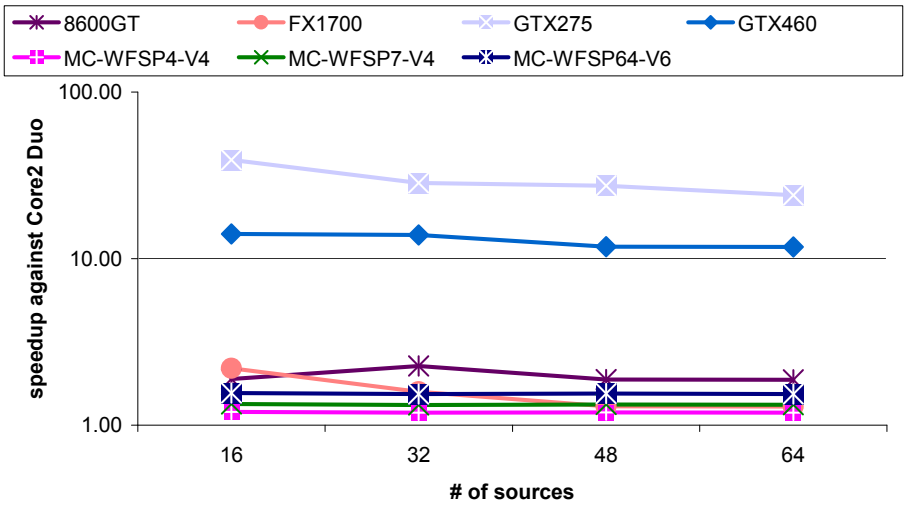

Figure 6.17: Execution speedup of all platforms against the Core2 Duo under a 64-loudspeaker setup.

same approach that was described in Section 6.1. More specifically, we decided to measure the memory impact, exclude it from the total execution time, and compare the platforms in terms of processing speed. Once again, we took also into account the fact that each chip is fabricated under a different technology, thus we decided to use the 40 nm as common comparison technology. In the context of the platform evaluation, we used our Virtex6-based prototype that utilizes the maximum available number of RUs and its operating frequency at 235 MHz.

Regarding the GPUs, we selected the GTX275, since in most of the cases achieved the best performance among all tested GPU chips. We used again the ITRS 2005 report to cancel out the difference between the original and hypothetical fabrication technologies. Thus for the GPU architectural prospectives evaluation, we used the *optimized GTX275* version that was described in Section 6.1. We should note that, as it was mentioned before, we did not consider the GTX460, because the CUDA Profiler 3.1 unfortunately provides the L1 cache hits % per kernel, and not the achieved memory throughput. Thus, the aforementioned GPU was excluded from our comparison of the processing times.

In Figure 6.18 we summarize the comparison between the optimized GTX275 and the FPGA processing times. On the Y-axis, if the ratio is smaller than 1.0, the GPU is faster, otherwise the FPGA performs better. We performed addi-

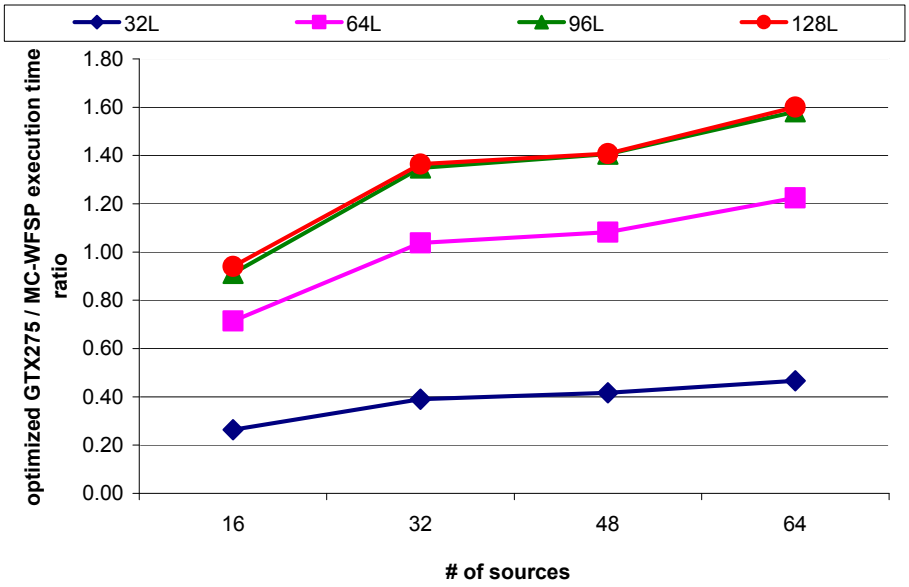

Figure 6.18: Processing time comparison between the optimized GTX275 and MC-WFSP approaches for the WFS.

tional experiments with up to 128 loudspeakers. As it was already indicated in Table 6.6, each *RU* occupies very few resources, thus we can accommodate systems using a single FPGA that can drive loudspeaker arrays consisting of hundreds of elements. In the ideal case, each *RU* can drive a single loudspeaker, and all *RUs* can process data concurrently. However, in case where there are more loudspeakers than *RUs*, output signals processing is distributed among them.

As it can be observed from Figure 6.18, the optimized GTX275 processes data up to 3.8 times faster compared to the MC-WFS32-V6 implementation, when there are 32 loudspeakers to be driven. The same applies even under a 64-element array with less than 32 sources, where the GTX275 is up to 39% faster than the MC-WFSP64. However, when there are 32 or more sources to be rendered under the same loudspeaker setup, the MC-WFSP64-V6 achieves a better performance than the GPU, because its processing resources are saturated. In contrast, the FPGA chip fits as many *RUs* as the number of loudspeakers, thus all output signals can be concurrently processed. When the loudspeakers array is increased to 96 elements, a Virtex6-based implementation with 77 RUs (MC-WFSP77-V6) still processes data 40% more efficiently than the GTX275. However, a performance saturation is observed, since the number

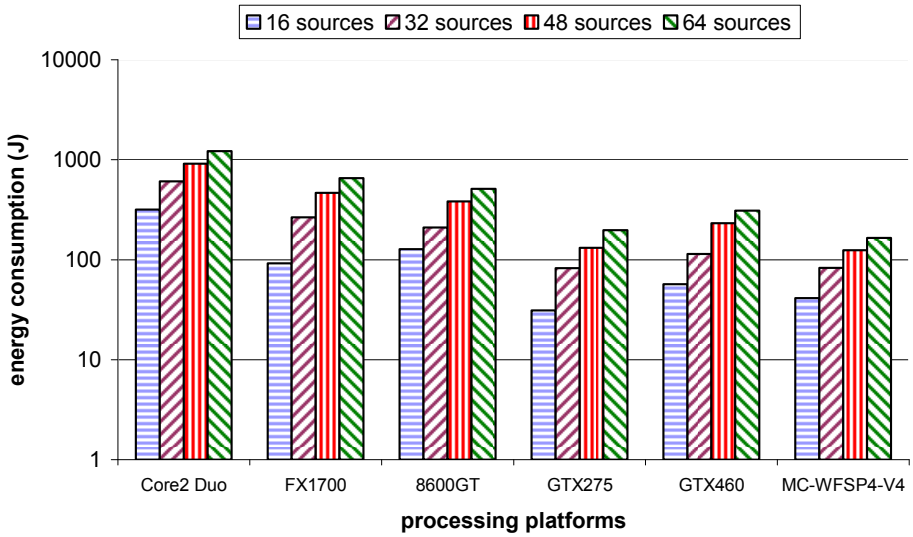

Figure 6.19: Energy consumption of all platforms under a 32-loudspeaker setup.

of *RUs* is less than the loudspeakers, thus not all of the output signals can be processed concurrently. This performance saturation is more clear when there are 128 loudspeakers to be driven, where the ratio between the GTX275 and MC-WFSP77-V6 execution times remains approximately the same with the ratio of a 96-element array. This observation, leads us to the conclusion that high-end GPUs can support more efficiently loudspeaker setups up to a certain number of output channels. When this number of channels is exceeded, a MC-WFSP-based approach can process data more efficiently than high-end GPUs.

Energy consumption: As it was mentioned before, performance is not the only parameter to be considered when choosing a hardware platform for a WFS system. Energy consumption is an important parameter that should also be taken into account. We used again formula (6.2) to evaluate the consumed energy from each platform. Figures 6.19 and 6.20 suggest the energy consumption by each processing platform under a 32- and 64-loudspeaker setup for different number of sources. As we can be observe, in every source scenario the Core2 Duo GPP consumes the most energy, thus making it unsuitable for energy-efficient solutions. In contrast, the middle-ranged GPUs consume less energy because they require less power and process data faster than the Core2 Duo. The GTX275 and GTX460 require 219 W and 160 W respectively, but since they process data very efficiently, they consume less energy

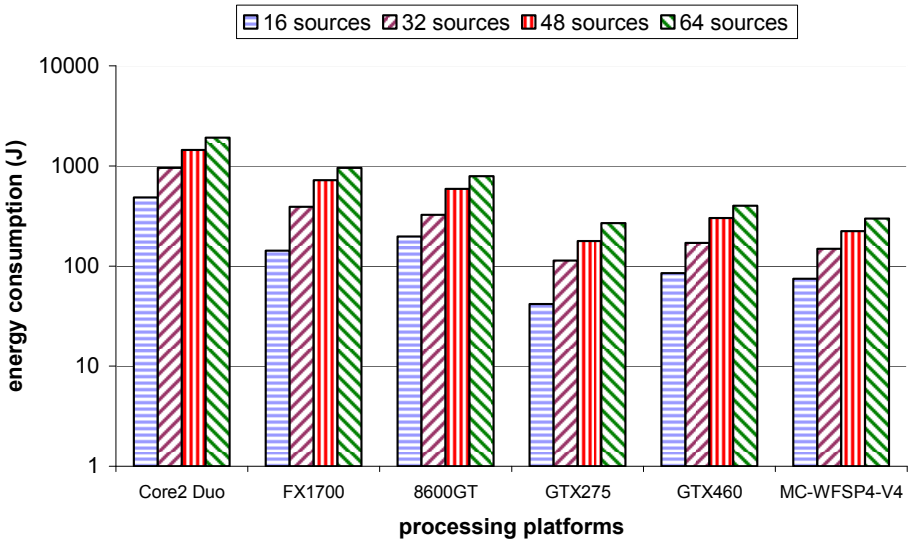

Figure 6.20: Energy consumption of all platforms under a 64-loudspeaker setup.

even than the FX1700 and 8600GT GPUs. Regarding the FPGA-based approach, according to the Xilinx XPower utility, our MC-WFSP4-V4 prototype requires approximately 12 Watts, thus resulting in most of the cases to the lowest energy consumption among all tested platforms.

Platform cost: The platform cost is also another parameter that should be considered when building a WFS system. As it was mentioned in Section 6.1, Table 6.4 shows the unit price of each platform in Euros (November 2010). As we can see, the FX1700 is more expensive than the Core2 Duo GPP. However, by taking into account its better performance, we can conclude that the FX1700 can provide an attractive platform for implementing small-scaled WFS systems. Even better choice for building such WFS systems would be the 8600GT, since it is cheaper, more energy- and performance-efficient than the Core2 Duo and the FX1700. Regarding the high-end GPUs, as we can see from Table 6.4, the GTX275 and GTX460 cost slightly more than the Core2 Duo. However, provided that they process data very efficiently, they make an attractive solution for building energy- and performance-efficient large-scaled WFS systems, which could not be supported by the Core2 Duo and middle-ranged GPUs. Finally, FPGA platforms cost higher, because they are targeted only by a much more limited group of people, compared to off-the-shelf GPPs and GPUs. Thus, an MC-WFSP-based approach can be considered for low energy consumption solutions that can also accommodate large-scaled WFS

Table 6.8: GPU- and FPGA-based implementations comparison against commercial products under a 128-loudspeaker setup

Platform	maximum # of real-time sources
Iosono Spatial Audio Processor	64
6 Sonicemotion Sonic Wave I	64
MC-WFSP77-V6	1335
optimized GTX275 GPU	871

systems.

Comparison against related work: As it was mentioned in Section 2.3.2, there are very few commercial WFS products that are based only on GPPs. Furthermore, all experimental WFS systems reported in the literature also utilize GPPs. For example, in [59], the authors employ an array of 22 loudspeakers and all data processing is done by a standard Linux PC. The paper mostly focuses on experiments regarding the correct sound localization and does not provide data regarding the maximum number of real-time sources that can be rendered. Two commercial systems are built from SonicEmotion and Iosono companies and based also on GPPs. A single SonicEmotion system support up to 64 sources when driving 24 loudspeakers. A single Iosono system supports up to 64 sources when driving 128 loudspeakers. In case a larger loudspeaker array is required, then additional such units must be cascaded. For example, as it shown in Table 6.8, six SonicEmotion systems are required to drive a 128-loudspeaker array and support 64 real-time sources.

Regarding the optimized GPU- and FPGA-based solutions, in Table 6.8 we show the maximum number of real-time sources that each one would support under a 128-loudspeaker setup. We note that in both cases the data memory access time is included, assuming a peak bandwidth of 127 GB/sec and 6.4 GB/sec for the GPU and FPGA systems respectively. Moreover, regarding the GPU implementation, we exclude the time spent on data transfer between the GPP and the GPU memories, since we want to focus only on the GPU performance evaluation and not the complete PC infrastructure. As it can be concluded, these platforms have the potential to drive equal or larger loudspeaker arrays than desktop PCs, while rendering more real-time sources. Thus, the total number of processing platforms would be much less, leading to more energy- and performance-efficient solutions.

Regarding power consumption, as indicated in Section 2.3.2, in the cinema of Ilmenau, Germany, six desktop PCs are employed to drive a 192-loudspeaker array. Since a Core2 Duo-based PC consumes approximately 65 Watts of

power when fully utilized, following the aforementioned example, a system with 6 such PCs would require approximately 6·65 = 390 Watts of power just for the processors. In contrast, a single GTX460 consumes a maximum power of 160 Watts. However, in this case, a less powerful host processor would suffice, since it would not be used for data processing of the algorithms. Based on this fact, the host GPP would require much less power compared to a Core2 Duo-based system. Thus a GTX460-based platform has the potential to reduce power consumption by $\sim$2.4 times. Furthermore, the entire system cost would be also reduced, since it would substitute six PCs with one GPU-based unit. Ultimately, employing an FPGA-based system would reduce power consumption even further, thus providing the best solution regarding performance and power consumption at the expense of a higher system cost.

6.3 Conclusions

In this chapter we presented results of the BF and WFS applications from experiments that were conducted using a GPP, middle-ranged and high-end GPUs, and FPGA-based implementations. We evaluated each processing platform based on various parameters, namely performance, energy consumption and cost. Base on the results, we can draw the following conclusions:

- **GPP-based approaches are not suitable for building cost- and energy-efficient immersive-audio systems.** As we observed from the results reported in the previous sections, the Core2 Duo-based approaches were in most cases performing data calculations slower than any other tested platform. This fact leads to the utilization of additional processing elements, when there are large-scaled BF or WFS systems, thus increasing the overall system cost. Furthermore, they consume the most energy among all considered platforms, which makes them unsuitable for energy-efficient solutions. Their main benefit is the support of high-level programming environments, thus allowing the rapid development of immersive-audio prototypes.
- **Middle-ranged GPUs are suitable for small-scaled immersive-audio systems.** Based on the performance figures presented in Section 6.1 and Section 6.2, we can conclude that the 8600GT GPU processed data faster than the Core2 Duo. Furthermore, it consumes less energy, while it costs half the price of a contemporary GPP. These facts lead to the conclusion that small-scaled immersive-audio systems are realizable using middle-ranged GPUs, thus leading to energy- and cost-efficient solutions.

- **High-end GPUs can support large-scaled immersive-audio systems.** According to our experiments, high-end GPUs can be used to build very large-scaled immersive-audio systems, thus resulting to the overall system cost reduction. Furthermore, although they require the highest power, efficient BF and WFS implementations lead to very fast execution times, thus requiring less energy even than contemporary GPPs to process the same amount of data.
- **FPGA-based approaches provide performance- and low energy- and power-efficient solutions.** Contemporary FPGA chips require the lowest power than any of the considered processing platforms. Combing the fact that a large number of processing elements can fit within a single chip that process data concurrently, it results to immersive-audio systems that are very energy-efficient. The Virtex4- and Virtex6-based presented prototypes process data slower than the GTX275 and GTX460, due to the high-latency external memory interface. However, based on our architectural prospectives evaluation, we concluded that newer FPGA families can perform computations more efficiently than high-end GPUs, because they can fit a large number of parallel processing elements. Thus, by employing an efficient external memory connection strategy, FPGA-based systems can support very large-scaled immersive-audio systems with the lowest power and energy consumption, at the price of a higher overall system cost.

To summarize, based on the presented experiments, we can conclude that it is very important to choose the correct implementation platform for immersive-audio systems. Although both GPUs and FPGAs can efficiently support large-scaled input/output channel arrays, it is very important to choose the most suitable platform based on the current power and economic cost budgets. High-end GPUs have the benefit that cost much less compared to contemporary large FPGAs, but their main drawback is that they require at least an order of magnitude more power. In case multiple GPUs are cascaded to drive complex channel setups, the total power consumption increases to the kWatt range. On the other hand, high-end FPGAs can also support large input/output channel arrays, while consuming much less power compared to GPU- and GPP-based immersive-audio systems. Consequently, an FPGA-based approach that utilizes many chips would require much less power compared to a multi-GPU solution, at the expense of a higher overall system cost.

This chapter is based on the following papers:

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A Reconfigurable Audio Beamforming Multi-Core Processor**, International Symposium on Applied Reconfigurable Computing (ARC), pp. 3-15, Belfast, Ireland, March 2011

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **Multi-Core Platforms for Beamforming and Wave Field Synthesis**, IEEE Transactions on Multimedia, pp. 235-245, Volume 13, No. 2, April 2011

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **Minimalistic Architecture for Reconfigurable Audio Beamforming**, International Conference on Field-Programmable Technology (FPT), pp. 503-506, Beijing, China, December 2010

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A Minimalistic Architecture for Reconfigurable WFS-Based Immersive-Audio**, International Conference on ReConFigurable Computing and FPGAs (ReConfig), pp. 1-6, Cancun, Mexico, December 2010

D. Theodoropoulos, G. Kuzmanov, G. N. Gaydadjiev, **A 3D-Audio Reconfigurable Processor**, ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 107-110, Monterey, California, USA, February 2010

7 Conclusions and Future Work

In this dissertation, we proposed a new custom architecture for Beamforming (BF) and Wave Field Synthesis (WFS) immersive-audio applications targeting contemporary Multi-Core Processors (MCPs). The proposed architecture consists of 14 application specific instructions, while the supporting programming paradigm employs a distributed memory organization. The instructions allow customization and control of many system parameters, like the number of input/output channels and the filter sizes. We proved the architecture applicability on MCPs by conducting two case studies, namely on multi-core BF and WFS reconfigurable processors and a wide range of off-the-shelf GPUs. Furthermore, we compared our FPGA- and GPU-based hardware prototypes against OpenMP-annotated approaches running on a Core2 Duo processor. Results suggested that contemporary GPUs can be used to support more complex immersive-audio setups compared to traditional GPPs, while at the same time consume less energy. Furthermore, in certain cases, the latest FPGA families can be used for more performance- and energy-efficient solutions compared to high-end GPUs and GPPs.

The rest of the chapter is organized as follows: In Section 7.1 we provide an overview of each chapter. Section 7.2 outlines all major conclusions of this dissertation and addresses the research questions discussed in Section 1.3. Finally, Section 7.3 provides a few open issues for future work.

7.1 Outlook

In Chapter 2, we provided the theoretical background of the BF and WFS techniques. Furthermore, we presented various software and hardware implementations of them mapped onto different platforms, in order to build experimental and commercial immersive-audio systems. As it was presented, nowadays

there are BF experimental systems that utilize from few tens to more than 1000 microphones. In addition, there are commercial products that also employ up to hundreds of input channels for efficient source extraction. Regarding the WFS algorithm, there are few experimental and commercial systems that are based only on GPPs. Finally, we provided an evaluation of many immersive-audio systems that utilize the BF and WFS techniques with respect to performance, power consumption and ease of use.

In Chapter 3, we proposed our architecture that supports both BF and WFS algorithms. As it was presented, our proposal considers a distributed memory hierarchy and allows a high-level interaction with reconfigurable (r-MCPs) and non-reconfigurable Multi-Core Processors (nr-MCPs). Moreover, we analyzed the functionality of each instruction and demonstrated how our architecture could be used to develop programs for BF and WFS immersive-audio systems.

In Chapter 4, we presented the underlying multi-core micro-architecture when utilizing r-MCPs for both BF and WFS techniques. We also described our custom-designed hardware accelerators for sound acquisition and rendering based on the BF and WFS respectively. Furthermore, we showed each instruction's micro-architecture implementation, which allows a high-level user interaction with the custom accelerators. Finally, we presented the complete MC-BFP and MC-WFSP hardware prototypes that were used to evaluate our proposal in Chapter 6.

In Chapter 5, we conducted a nr-MCP case study for our architecture, by applying it to middle-ranged and high-end GPUs. First, we provided a brief description of contemporary GPUs organization. We also presented how we implemented each high-level instruction, in order to hide all GPU-specific code annotations details from the user. Furthermore, we explained how we used important system parameters, like the number of input/output channels and filter sizes, to develop BF and WFS GPU kernels that are efficiently mapped onto the processing cores.

Finally, in Chapter 6, we described our experimental setup that we applied, in order to test our FPGA- and GPU-based implementations regarding performance for the BF and WFS applications. We compared the results accuracy of our fixed-point format hardware approaches against a Core2 Duo floating point implementation. We also provided a comparison among the two multi-core systems, the Core2 Duo and related work. Furthermore, we investigated the architectural prospectives of high-end GPUs and latest generation FPGA families; we compared their execution times under many input/output channels and real-time sources scenarios. Finally, we discussed each system's energy

consumption and overall cost.

7.2 Conclusions

Below we summarize the general conclusions that are drawn from our work:

- GPP-based approaches perform data calculations slower than any other tested platform. This fact leads to developers on employing a large number of processing units, when there are complex BF or WFS systems, thus increasing the overall system cost. Furthermore, they consume the most energy among all considered platforms. However, their main benefit is the support of high-level programming environments, thus allowing the rapid development of immersive-audio prototypes.
- Middle-ranged GPUs can efficiently support small-scaled immersive-audio systems. Our tests suggest that a 8600GT GPU can process data faster than the Core2 Duo when implementing immersive-audio systems. Furthermore, it consumes less energy, while it costs half the price of contemporary high-performance GPPs. These facts lead to the conclusion that small-scaled immersive-audio systems are realizable using middle-ranged GPUs, thus leading to energy- and cost-efficient solutions.
- High-end GPUs are suitable for large-scaled immersive-audio systems. Thus, for the same number of input/output channels, less overall number of processing units are required compared to the case of employing GPP-based modules. In addition, although high-end GPUs require more power compared to GPPs, they can process the same amount of data an order of magnitude faster, thus requiring less energy.
- FPGA-based approaches provide performance- and power-efficient solutions. Contemporary FPGA chips require the lowest power than any of the considered processing platforms. Combing the fact that a large number of processing elements can fit within a single chip that process data concurrently, it results to immersive-audio systems that are the most energy-efficient. Based on our architectural prospectives evaluation, we concluded that newer FPGA families can perform computations for the BF up to 3.8 times and for the WFS up to 60% faster than high-end GPUs, because they can fit a large number of parallel processing elements. Thus, by employing an efficient external memory interface, FPGA-based systems have the potential to support very

large-scaled immersive-audio systems with very low energy and power consumption, at the price of a higher overall system cost.

Based on the above conclusions we can answer the research questions posed in Section 1.3:

- *How to map rapidly and efficiently immersive-audio technologies onto Multi-Core Processors (MCPs)?* We addressed the difficulty of rapid immersive-audio systems development that are based on MCPs, by proposing a custom architecture for the BF and WFS technologies. Our proposal is completely platform-independent, thus can be applied in a variety of multi-core processors.
- *Which instructions should be supported by the architecture for immersive-audio systems?* We proposed an instruction set that allows easy customization of many vital system parameters, efficient audio data processing, and system debugging through a high-level interface. This way, the programmer can easily develop and test a wide range of immersive-audio systems that can be mapped onto different processing platforms.
- *How to enhance performance and efficiently support small- and large-scaled immersive-audio systems?* We implemented the micro-architectural support of our architecture that allows utilization of various number of processing elements, therefore it is suitable for mapping onto MCPs. With respect to the available resources, different r-MCP and nr-MCP implementations with different performances are possible, where all of them use the same architecture and programming paradigm. In this dissertation, we presented two case studies of our architecture implementation, namely on a set of multi-core reconfigurable processors and a wide range of off-the-shelf GPUs. We used our reconfigurable and GPU-based prototypes to conduct various tests for both BF and WFS applications, ranging from small- to large-scaled setups. Furthermore, we investigated the maximum number of real-time sources that each processing platform can support under different sizes of input/output channel arrays. Based on our experimental results, we proposed the most suitable platform for each case, in order to build efficient immersive-audio systems.
- *How to choose the most energy- and power-efficient approach for such complex systems?* Based on the processing time and the power consumption of all platforms, we suggested their energy consumption under a variety of setups. Since immersive-audio systems utilize a large

number of input/output channels, it is important to select the most suitable processing platform that complies to any power constraints and cost limitations. Moreover, developers can consider our results to select the most energy- and cost-efficient solution based on their requirements.

7.3 Open Issues and Future Directions

In this dissertation, we presented a custom architecture for immersive-audio applications. In the current section, we provide a few future research directions to improve it.

- **Additional instructions for improved system configuration.** New instructions that would allow a more fine-grained system configuration can significantly enhance the entire development. For example, in the current version of our immersive-audio architecture, the decimation/interpolation rates have to be configured off-line. Especially in the case of reconfigurable processors, this limitation requires the synthesis and implementation of the design based on the new resampling rates. New instructions that support updating the resampling rates, would enhance even more the rapid development of reconfigurable immersive-audio systems. These instructions require the design of custom FIR filters that can have customizable resampling rates.
- **Unified BF and WFS systems.** Many applications require real-time sound acquisition and rendering. For example, during video teleconferencing, all locations require equipment that records the local speaker and also renders the voice of any remote meeting participant. Another example is music concerts where the performers and/or their music instruments are recorded using a microphone array. At the same time, all extracted signals are rendered to the audience through large loudspeaker arrays. In the first example, the BF and WFS processors can be integrated within a single system, where they work in parallel. The output of the BF processor is sent to any remote location, where the local WFS processor renders every source signal. However, in the second scenario, the output signals of the BF processor are used as input to the WFS processor, which renders the source signal through the loudspeaker array.
- **Faster external memory interface.** As it was discussed in Sections 6.1 and 6.2, a considerable amount of the MC-BFP and MC-WFSP prototypes total execution time was spent on accessing the external memory. In order to improve this processing bottleneck, a more efficient interface

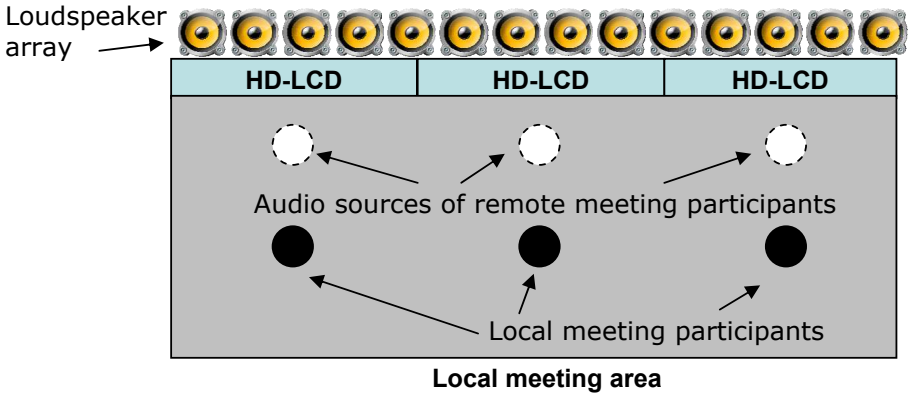

Figure 7.1: Teleconference scenario using the WFS technology.

is required. A possible solution could be the avoidance of the central bus and the design of a controller that would connect all *BeamFormers* and *RUs* directly to the off-chip memory.

- **Tests with real microphone and loudspeaker setups.** Finally, a very interesting and important research direction would be the implementation of actual BF and WFS systems that employ different number of input/output channels. This fact would help on evaluating more precisely the performance and sound quality of every hardware platform considered.

Providing to the audio engineering community an architecture for rapid development of immersive-audio systems that utilize MCPs, introduces their potential application to new fields and every-day activities:

- **Next generation consumer and professional audio products**. In the majority of the cases, contemporary sound equipment utilizes stereophonic or surround approaches. The latter suffer from poor audio quality thus decreasing the source localization accuracy and the overall acoustic experience. Our approach has the potential to provide scalable and efficient immersive-audio audio systems that can be applicable to consumer products, like home theater systems and computer gaming industry, and professional installations, like cinemas, amusement parks and concert halls. These days 3D-vision becomes very popular not only into cinemas, but also to home/office equipment like TVs, computer screens and game consoles. Consequently, it would be very interesting to combine them with immersive-audio solutions and provide a 3D audio-visual out-

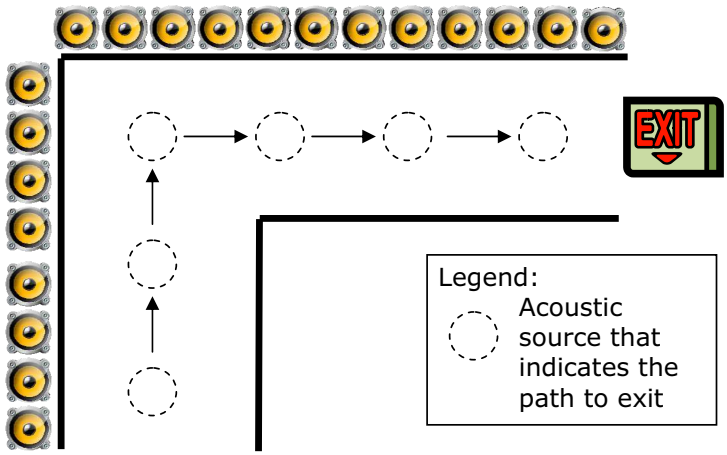

Figure 7.2: Guidance to emergency exit using virtual acoustic sources.

standing experience.

- **High-quality teleconference systems**. Nowadays, people perform teleconference meetings more than ever before. Unfortunately, in most cases, the utilized equipment is regular phone devices or computers with small and low quality loudspeakers. Professional video teleconference systems, although they provide High Definition (HD) video quality through Liquid Crystal Displays (LCDs), still employ stereophonic approaches for the audio reproduction, which has a significant impact on the overall teleconference experience. In contrast, a WFS sound system will provide an outstanding audio perception, since it would be able to position within the audience area a virtual acoustic source of every remote speaker, thus local meeting participants would literary perceive a ”tele-presence” of the person, as illustrated in Figure 7.1. It is a well-established fact that major companies spend annually hundreds of thousands of dollars for their employees to travel around the globe for meeting purposes. However, a WFS teleconference system with such realistic audio experience, would make feasible excellent-quality remote meetings, thus significantly reducing a company's travel expenses.
- **Improved safety to critical environments**. Over the last years, researchers have proposed techniques that are based on auditory cues, in order to improve warnings in commercial aircraft and military aviation [95]. In the majority of the cases, they utilize monaural or binaural approaches, which do not always provide the best solutions [23]. We be-

lieve that medium-scaled WFS sound systems can significantly enhance the sound alarms perception from the pilots, by alleviating the necessity for them to wear headphones to hear the sound cues, thus actively contribute to improved overall safety during the flight. A similar approach can also be applied to other critical environments like hospitals and nuclear power plants. For example, in case of fire, the smoke limits visibility considerably, thus the paths to emergency exits are extremely difficult to be found. As illustrated in Figure 7.2, a solution would be to render specific audio sounds at certain locations that would form an ”audio-path” that people can follow to the nearest emergency exit.

Bibliography

[1] Implementing a Real-Time Beamformer on an FPGA Platform. In *XCell journal*, pages 36–40, Second Quarter 2007.

[2] Acoustic Camera. http://www.acoustic-camera.com.

[3] Robert Höldrich Alois Sontacchi, Michael Strauß. Audio Interface for Immersive 3D-Audio Desktop Applications. In *International Symposium on Virtual Environments, Human-Computer Interfaces, and Measurement Systems*, pages 179–182, July 2003.

[4] AMD Corporation. ATI Stream Computing OpenCL Programming Guide. August 2010.

[5] Analog Devices Inc. SHARC Processor ADSP-21262. May 2004.

[6] Andrew Schmeder, Adrian Freed, David Wessel. Best Practices for Open Sound Control. In *Linux Audio Conference*, 2010.

[7] Angelo Farina, Ralph Glasgal, Enrico Armelloni, Anders Torger. Ambiophonic Principles for the Recording and Reproduction of Surround. In *AES 19th International Conference*, June 2001.

[8] Enrico Armelloni, Paolo Martignon, and Angelo Farina. Comparison Between Different Surround Reproduction Systems: ITU 5.1 vs PanAmbio 4.1. In *118th Convention of Audio Engineering Society*, May 2005.

[9] Benny Sallberg and Mikael Swartling and Nedelko Grbic and Ingvar Claesson. Real-time implementation of a blind beamformer for subband speech enhancement using kurtosis maximization. In *International Workshop on Acoustic Echo and Noise Control*, pages 485–489, September 2006.

[10] J.A. Beracoechea, S. Torres-Guijarro, L. García, and F.J. Casajús-Quirós. On building Immersive Audio Applications Using Robust Adaptive Beamforming and Joint Audio-Video Source Localization. In *EURASIP Journal on Applied Signal Processing*, pages 1–12, June 2006.

[11] A.J. Berkhout, D. de Vries, and P. Vogel. Acoustic Control by Wave Field Synthesis. In *Journal of the Acoustical Society of America*, volume 93, pages 2764–2778, May 1993.

[12] Ajay V. Bhatt. Creating a PCI Express Interconnect. Intel Corporation.

[13] Bill Kapralos and Michael Jenkin and Evangelos Milios. Audio-visual

localization of multiple speakers in a video teleconferencing setting. In *International Journal of Imaging Systems and Technology*, volume 13(1), pages 95–105, October 2003.

[14] M.M. Boone, E.N.G. Verheijen, and P.F. van Tol. Spatial Sound Field Reproduction by Wave Field Synthesis. In *Journal of the Audio Engineering Society*, volume 43, pages 1003–1012, December 1995.

[15] S. Brix, T. Sporer, and J. Plogsties. CARROUSO - An European approach to 3D-audio. In *110th AES Convention. Audio Engineering Society*, May 2001.

[16] Werner Paulus Josephus De Bruijn. *Application of Wave Field Synthesis in Videoconferencing*. PhD thesis, TU Delft, The Netherlands, October 2004.

[17] H. Buchner, S. Spors, W. Kellermann, and R. Rabenstein. Full-duplex communication systems using loudspeaker arrays and microphone arrays. In *IEEE International Conference on Multimedia and Expo*, pages 509–512, November 2002.

[18] Rochet Cedrick. Documentation of the Microphone Array Mark III. 2005.

[19] THX company. http://www.thx.com/.

[20] Atmel Corporation. DIOPSIS 940HF. July 2008.

[21] Atmel Corporation. mAgic DSP architecture document. December 2008.

[22] NVidia Corporation. CUDA programming guide version 3.2. August 2010.

[23] C.W Johnson and W. Dell. The Limitations of 3D Audio to Improve Auditory Cues in Aircraft Cockpits. In *International Systems Safety Conference*, pages 990–999, 2003.

[24] Jérome Daniel, Rozenn Nicol, and Sébastien Moreau. Further Investigations of High Order Ambisonics and Wave Field Synthesis for Holophonic Sound Imaging. In *114th Convention of Audio Engineering Society*, pages 58–70, March 2003.

[25] Advanced Micro Devices. http://www.amd.com.

[26] Angelo Farina and Emanuele Ugolotti. Software Implementation of B-Format Encoding and Decoding. In *104rd AES Convention*, 1998.

[27] Antoine Fillinger, Lukas Diduch, Imad Hamchi, Stephane Degre, and

Vincent Stanford. Nist smart data flow system ii: speaker localization. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 549–550, 2007.

[28] H. Fletcher. Auditory perspectiveBasic requirements. In *Electrical Engineering*, volume 53, pages 12–17, 1934.

[29] Fraunhofer Institute for Digital Media Technology. http://www.idmt.fraunhofer.de/eng/about_us/facts_figures.htm.

[30] Gang Mei, Roger Xu, Debang Lao, Chiman Kwan. Real-Time Speaker Verification with a Microphone Array. Technical report.

[31] Gerrit Blaauw and Frederick Brooks. Computer Architecture: Concepts and Evolution. February 1997.

[32] Michael A. Gerzon. Periphony: With-Height Sound Reproduction. In *Journal of the Audio Engineering Society*, volume 21, pages 2–10, 1973.

[33] Michael Gschwind. The cell broadband engine: exploiting multiple levels of parallelism in a chip multiprocessor. *International Journal of Parallel Programming*, pages 233–262, June 2007.

[34] Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe, and Takeshi Yamazaki. Synergistic processing in cell's multicore architecture. pages 10–24, March 2006.

[35] Günther Theile. Multichannel Natural Music Recording Based on Pychoacoustics Principles. In *AES 19th International Conference*, May 2001.

[36] Kimio Hamasaki, Wataru Hatano, Koichiro Hiyama, Setsu Komiyama, and Hiroyuki Okubo. 5.1 and 22.2 Multichannel Sound Productions Using an Integrated Surround Sound Panning System. In *Audio Engineering Society Convention 117*, October 2004.

[37] T. Holman. *5.1 Surround Sound Up and Running*. Focal Press, December 1999.

[38] Edo Hulsebos. *Auralization using Wave Field Synthesis*. PhD thesis, TU Delft, The Netherlands, October 2004.

[39] R. Tripiccione I. Colacicco, G. Marchiori. The hardware application platform of the hartes project. In *Field Programmable Logic and Applications*, pages 439–442, September 2008.

[40] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. August 2008.

[41] Fraunohfer IGD. http://www.igd.fraunhofer.de/.

[42] Intel Corporation. http://ark.intel.com/Product.aspx?id=33910.

[43] Intel Labs. The SCC Platform Overview. Technical report, Intel Corporation, 2010.

[44] International Business Machines Corporation. http://www-03.ibm.com/press/us/en/pressrelease/19508.wss.

[45] International Technology Roadmap for Semiconductors. Process Itegration, Devices and Structures. 2005.

[46] Iosono Company. http://www.iosono-sound.com.

[47] Ivan Tashev and Michael L. Seltzer. Data driven beamformer design for binaural headset. In *International Conference on Acoustice, Echo and Noise Control*, September 2008.

[48] Ka-Fai Cedric Yiu and Chan Hok Ho and Yao Lu and Xiaoxiang Shi and Wayne Luk. Reconfigurable acceleration of microphone array algorithms for speech enhancement. In *Application-specific Systems, Architectures and Processors*, pages 203–208, 2008.

[49] Khronos OpenCL Working Group. The OpenCL Specification v1.10. June 2010.

[50] E. Moscu Panainte G. N. Gaydadjiev Y. D. Yankova V.M. Sima K Sigdel R. J. Meeuws S. Vassiliadis K.L.M. Bertels, G. Kuzmanov. Hartes toolchain early evaluation: Profiling, compilation and hdl generation. In *Proceedings of 17th International Conference on Field Programmable Logic and Applications*, pages 402–408, August 2007.

[51] K.L.M. Bertels, G. Kuzmanov, E. Moscu Panainte, G. N. Gaydadjiev, Y. D. Yankova, V.M. Sima, K Sigdel, R. J. Meeuws, S. Vassiliadis. Profiling, Compilation, and HDL Generation within the hArtes Project. In *Design Test and Automation in Europe Workshop*.

[52] Chris Kyriakakis. Fundamental and Technological Limitations of Immersive Audio Systems. In *Proceedings of the IEEE*, volume 86, pages 941–951, May 1998.

[53] A. Lattanzi, E. Ciavattini, S. Cecchi, L. Romoli, and F. Ferrandi. Real-Time Implementation of Wave Field Synthesis on NU-Tech Framework Using CUDA Technology. In *Audio Engineering Society Convention 128*, May 2010.

[54] Richard G. Lyons. *Understanding Digital Signal Processing*. Pearson

Education, November 1996.

[55] Marije Baalman and Simon Schampijer and Torben Hohn and Thilo Koch and Daniel Plewe and Eddie Mond. Renewed architecture of the sWONDER software for Wave Field Synthesis on large scale systems. In *5th International Linux Audio Conference*, pages 76–83, 2007.

[56] Mark F. ODwyer, Guillaume Potard, Ian Burnett. A 16-speaker 3d audio-visual display interface and control system. In *International Conference on Auditory Display*, July 2004.

[57] Mark Fiala and David Green and Gerhard Roth. A panoramic video and acoustic beamforming sensor for videoconferencing. In *IEEE International Conference on Haptic, Audio and Visual Environments and their Applications*, pages 47–52, October 2004.

[58] Mathworks Corporation. http://www.mathworks.com/.

[59] Daniel Menzel, Helmut Wittek, Gnther Theile, and Hugo Fast. The Binaural Sky: A Virtual Headphone for Binaural Room Synthesis. In *International Tonmeister Symposium*, October 2005.

[60] MIT CSAIL: MIT Project Oxygen. http://oxygen.lcs.mit.edu/. 2004.

[61] Moldrzyk C, Goertz A, Makarski M, Feistel S, Ahnert W, Weinzierl S. Wellenfeldsynthese für einen groSSen hörsaal. In *Fortschritte der Akustik, DAGA Stuttgart*, 2007.

[62] Athanasios Mouchtaris, Panagiotis Reveliotis, and Chris Kyriakakis. Inverse of Filter Design for Immersive Audio Rendering Over Loudspeakers. In *IEEE Transactions on Multimedia*, volume 2, pages 77–87, June 2000.

[63] Carl-Inge Colombo Nilsen and Ines Hafizovic. Digital Beamforming using a GPU. In *IEEE International Conference on Acoustics, Speech and Signal processing*, pages 609–612, May 2009.

[64] NU-Tech Framework. http://www.leaff.it/content.php?id=31.

[65] NVidia Corporation. http://www.nvidia.com/page/geforce_8800.html.

[66] NVidia Corporation. NVIDIA GeForce GTX 200 GPU Datasheet.

[67] NVidia Corporation. Tesla C1060 Computing Processor Board. 2008.

[68] NVidia Corporation. CUDA programming guide version 3.1.1. Technical report, July 2010.

[69] NVidia Corporation. NVIDIA GeForce GTX 400 GPU Datasheet. 2010.

[70] NVidia Corporation. NVIDIAs Next Generation CUDA Compute Architecture: Fermi. 2010.

[71] National Institude of Standards and Technology. http://www.nist.gov/smartspace/mk3_presentation.html.

[72] OpenMP Architecture Review Board. OpenMP Application Program Interface v3.0. May 2008.

[73] Renato S. Pelegrinni and Matthias Rosenthal. Wave Field Synthesis With Synchronous Distributed Signal Processing. In *6th IEEE Workshop on Multimedia Signal Processing*, pages 227–230, 2004.

[74] picoChip. http://www.picochip.com/.

[75] Polycom Inc. Polycom CX5000 Unified Conference Station. March 2009.

[76] Ross Cutler and Yong Rui and Anoop Gupta and JJ Cadiz and Ivan Tashev and Li-wei He and Alex Colburn and Zhengyou Zhang and Zicheng Liu and Steve Silverberg. Distributed meetings: A meeting capture and broadcasting system. In *International Conference on Multimedia*, pages 503–512, December 2002.

[77] Slavy G. Mihov and Tyler Gleghorn and Ivan Tashev. Enhanced sound capture system for small devices. In *International Conference of Information, Communication and Energy Systems*, June 2008.

[78] William Snow. Basic principles of stereophonic sound. pages 42 – 53, March 1955.

[79] Audio Engineering Society. AES10-2003: AES Recommended Practice for Digital Audio Engineering – Serial Multichannel Audio Digital Interface (MADI). In *Rev 2003*, May 2003.

[80] SonicEmotion Company. http://www.sonicemotion.com.

[81] Thomas Sporer, Michael Beckinger, Andreas Franck, Iuliana Bacivarov, Wolfgang Haid, Kai Huang, Lothar Thiele, Pier S. Paoloucci, Piergiovanni Bazzana, Piero Vicini, Jianjiang Ceng, Stefan Kraemer, and Rainer Leupers. SHAPES - a Scalable Parallel HW/SW Architecture Applied to Wave Field Synthesis. In *International Conference of Audio Engineering Society*, pages 175–187, September 2007.

[82] Jan P. Springer, Christoph Sladeczek, Martin Scheffler, Jan Hochstrate, Frank Melchior, and Bernd Fröhlich. Combining Wave Field Synthesis And Multi-viewer Stereo Displays. In *IEEE Virtual Reality Conference*, pages 237–240, 2006.

[83] Squarehead Technology. Audio Scope Zoom Audio.

[84] M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, Jae-Wook Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The raw microprocessor: a computational fabric for software circuits and general-purpose programs. In *Micro, IEEE*, pages 25–35, March 2002.

[85] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. The Raw Processor, A Composeable 32-Bit Fabric for Embedded and General Purpose Computing. In *MIT Student Oxygen Workshop*, 2001.

[86] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *International Symposium on Computer Architecture*, pages 2–13, 2004.

[87] H. Teutsch, S. Spors, W. Herbordt, W. Kellermann, and R. Rabenstein. An Integrated Real-Time System For Immersive Audio Applications. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 67–70, October 2003.

[88] Texas Instruments Inc. TMS320C62x/C67x Power Consumption Summary. July 2002.

[89] Günther Theile and Helmut Wittek. Wave field synthesis: A promising spatial audio rendering concept. In *Acoustical Science and Technology*, pages 393–399, 2004.

[90] Gunther Theile, Helmut Wittek, and Markus Reisinger. Potential Wavefield Synthesis Applications in the Multichannel Stereophonic World. In *24th International Conference on Multichannel Audio, the New Reality*, pages 43–57, May 2003.

[91] Jasper van Dorp Schuitman. The Rayleigh 2.5D Operator Explained. Technical report, Laboratory of Acoustical Imaging and Sound Control, TU Delft, The Netherlands, June 2007.

[92] Jasper van Dorp Schuitman, Lars Hörchens, and Diemer de Vries. The

MAP-based wave field synthesis system at TU Delft (NL). In *1st DEGA symposium on wave field synthesis*, September 2007.

[93] B.D. Van Veen and K.M. Buckley. Beamforming: a versatile approach to spatial filtering. In *IEEE ASSP Magazine*, volume 5, pages 4–24, April 1988.

[94] Peter Vogel. *Application of Wave Field Synthesis in Room Acoustics*. PhD thesis, TU Delft, The Netherlands, 1993.

[95] W. Dell. The Use of 3D Audio to Improve Auditory Cues in Aircraft. Technical report, Department of Computing Science, Univeristy of Glasgow, 2000.

[96] Kieran Wall and Geoffrey R. Lockwood. Modern implementation of a realtime 3d beamformer and scan converter system. In *2005 IEEE Ultrasonics Symposium*, pages 1400–1403, September 2005.

[97] Eugene Weinstein, Kenneth Steele, Anant Agarwal, and James Glass. LOUD: A 1020-Node Modular Microphone Array and Beamformer for Intelligent Computing Spaces. In *MIT/LCS Technical Memo MIT-LCS-TM-642*, April 2004.

[98] Xilinx Inc. Distributed Arithmetic FIR Filter v9.0. April 2005.

[99] Xilinx Inc. MAC FIR v5.1. April 2005.

[100] Xilinx Inc. PowerPC 405 Processor Block Reference Guide. July 2005.

[101] Xilinx Inc. Embedded System Tools Reference Manual, EDK 9.2i. 2007.

[102] Xilinx Inc. ML410 Embedded Development Platform. September 2007.

[103] Xilinx Inc. Xilinx ISE 9.2i Software Manuals and Help. 2007.

[104] Xilinx Inc. XPS Central DMA Controller. September 2007.

[105] Xilinx Inc. Spartan-II FPGA Family Data Sheet. June 2008.

[106] Xilinx Inc. Virtex-4 FPGA User Guide. December 2008.

[107] Xilinx Inc. XtremeDSP for Virtex-4 FPGAs. 2008.

[108] Xilinx Inc. Virtex-6 Family Overview. February 2009.

[109] Xilinx Inc. Xilinx ISE 11.1i Software Manuals and Help. 2009.

[110] Xilinx Inc. LogiCORE IP Processor Local Bus (PLB) v4.6. 2010.

[111] Xilinx Inc. LogiCORE IP XPS HWICAP. 2010.

[112] Xilinx Inc. The Simple MicroBlaze Microcontroller Concept. 2010.

[113] Xilinx Inc. XPower Estimator User Guide. October 2010.

[114] Zohra Yermeche and Benny Sallberg and Nedelko Grbic and Ingvar Claesson. Real-time implementation of a subband beamforming algorithm for dual microphone speech enhancement. In *IEEE International Symposium on Circuits and Systems*, pages 353–356, May 2007.

List of Publications

International Journals (this thesis)

1. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **Multi-Core Platforms for Beamforming and Wave Field Synthesis**, IEEE Transactions on Multimedia, Volume 13, No. 2, April 2011

International Conferences (this thesis)

1. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **A Reconfigurable Audio Beamforming Multi-Core Processor**, International Symposium on Applied Reconfigurable Computing (ARC), pp. 3-15, Belfast, Ireland, March 2011
2. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **Minimalistic Architecture for Reconfigurable Audio Beamforming**, International Conference on Field-Programmable Technology (FPT), pp. 503-506, Beijing, China, December 2010
3. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **A Minimalistic Architecture for Reconfigurable WFS-Based Immersive-Audio**, International Conference on ReConFigurable Computing and FPGAs (ReConfig), pp. 1-6, Cancun, Mexico, December 2010
4. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **A 3D-Audio Reconfigurable Processor**, ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 107-110, Monterey, California, USA, February 2010
5. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **A Reconfigurable Beamformer for Audio Applications**, IEEE Symposium on Application Specific Processors (SASP), pp. 80-87, San Francisco, California, USA, July 2009
6. *D. Theodoropoulos*, G. Kuzmanov, G. N. Gaydadjiev, **Reconfigurable Accelerator for WFS-Based 3D-Audio**, IEEE Reconfigurable Architectures Workshop (RAW), pp. 1-8, Rome, Italy, May 2009,
7. *D. Theodoropoulos*, CB Ciobanu, G. Kuzmanov, **Wave Field Synthesis for 3D Audio: Architectural Prospectives**, ACM International Conference on Computing Frontiers, pp. 127-136, Ischia, Italy, May 2009

International Conferences (not related to this thesis)

1. T. Marconi, *D. Theodoropoulos*, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel HDL Coding Style to Reduce Power Consumption for Reconfigurable Devices**, International Conference on Field-Programmable Technology (FPT), pp. 295-299, Beijing, China, December 2010

2. C. Galuzzi, *D. Theodoropoulos*, R. J. Meeuws, K.L.M. Bertels, **Algorithms for the Automatic Extension of an Instruction-Set**, Design, Automation and Test in Europe (DATE), pp. 548-553, Nice, France, April 2009

3. *D. Theodoropoulos*, A Siskos, D.N. Pnevmatikatos, **CCproc: A custom VLIW cryptography co-processor for symmetric-key ciphers**, International Workshop on Applied Reconfigurable Computing (ARC), pp. 318-323, Karlsruhe, Germany, February 2009

4. C. Galuzzi, *D. Theodoropoulos*, R. J. Meeuws, K.L.M. Bertels, **Automatic Instruction-Set Extensions with the Linear Complexity Spiral Search**, IEEE International Conference on ReConFigurable Computing and FPGAs (ReConfig) , pp. 31-36, Cancun, Mexico, December 2008

5. *D. Theodoropoulos*, I Papaefstathiou, D.N. Pnevmatikatos, **CCproc: An Efficient Cryptographic Coprocessor**, IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), pp. 160-163, Rhodes, Greece, October 2008

6. C. Galuzzi, *D. Theodoropoulos*, K.L.M. Bertels, **A Clustering Method for the Identification of convex Disconnected Multiple Input Multiple Output Instructions**, International Conference IC-SAMOS VIII (SAMOS), pp. 65-73, Samos, Greece, July 2008

7. A Dollas, K Papadimitriou, E Sotiriadis, *D. Theodoropoulos*, I Koidis, G Vernardos, **A Case Study on Rapid Prototyping of Hardware Systems: the Effect of CAD Tool Capabilities, Design Flows, and Design Styles**, International IEEE Workshop on Rapid Prototyping (RSP), pp. 180-186, Geneva, Switzerland, June 2004

Local Conferences

1. *D. Theodoropoulos*, Y. D. Yankova, G. Kuzmanov, K.L.M. Bertels, **Automatic hardware generation for the Molen reconfigurable architecture: a G721 case study**, ProRisc Conference, pp. 380-387, Veldhoven, The Netherlands, November 2007

Reports

1. T. Marconi, *D. Theodoropoulos*, K.L.M. Bertels, G. N. Gaydadjiev, **A Novel HDL Coding Style for Power Reduction in FPGAs**, CE-TR-2010-02, Computer Engineering Lab, TU Delft, January 2010

Samenvatting

In dit proefschrift presenteren we een nieuwe aanpak voor de snelle ontwikkeling van multi-core immersive-audio systemen. We bestuderen twee populaire immersive-audio technieken, namelijk Beamforming en Wave Field Synthesis (WFS). Beamforming maakt gebruik van microfoon-arrays om akoestische bronnen te extraheren die opgenomen zijn in een lawaaierige omgeving. WFS past grote luidspreker-arrays toe om bewegende geluidsbronnen na te bootsen, zodoende vertstrekt het uitstekende geluidswaarneming en -lokalisatie. Uit literatuuronderzoek blijkt dat de meerderheid van dergelijke experimentele en commerciele audio-systemen zijn gebaseerd op standaard PC's, vanwege hun high-level programmeerondersteuning en de mogelijkheden voor snelle systeemontwikkeling. Deze benaderingen introduceren echter prestatieknelpunten, overmatig energieverbruik en hogere totale kosten. Systemen op basis van DSP's verbruiken zeer weinig stroom, maar hebben nog steeds beperkte prestaties. Custom-hardware oplossingen verlichten de bovengenoemde nadelen, maar ontwerpers zijn vooral gericht op het optimaliseren van de prestaties, zonder een high-level interface voor systeem-controle en testen te bieden. Om de bovengenoemde problemen aan te pakken, stellen wij een aangepaste platform-onafhankelijke architectuur voor, die immersive-audio technologieen voor hoge-kwaliteit geluidopname en -rendering ondersteunt. Een belangrijk kenmerk van de architectuur is dat het is gebaseerd op een multi-core processing paradigma. Dit maakt het ontwerp van schaalbare en herconfigureerbare micro-architecturen mogelijk, met betrekking tot de beschikbare hardware resources, en aanpasbare implementaties gericht op multi-core platforms. Om ons voorstel te kunnen evalueren, voerden we twee case studies uit: We hebben onze architectuur geimplementeerd als een heterogene multi-core herconfigureerbare processor gemapped op FPGA's. Verder hebben we onze architectuur toegepast op een breed scala van hedendaagse GPU's. Onze aanpak combineert de flexibiliteit van software inherent aan GPP's met de rekenkracht van multi-core platforms. De resultaten suggereren dat het gebruik van GPU's en FPGA's voor het bouwen van immersive-audio systemen leidt tot oplossingen die verbeterde prestaties kunnen verwezenlijken van wel een orde van grootte, en een laag stroomverbruik, als ook een daling van de totale kosten voor het systeem in vergelijking met GPP-gebaseerde benaderingen.

Curriculum Vitae

Dimitris Theodoropoulos (S'06) was born in Athens, Greece. In 2003 and 2006 he obtained his Diploma (5-year degree) and M.Sc degree respectively from the Electronic and Computer Engineering department at the Technical University of Crete, Greece. In 2007 he joined the Computer Engineering department of the Delft University of Technology, where he worked towards his Ph.D with scientific advisors dr. Georgi Kuzmanov and dr. Georgi Gaydadjiev.

During his M.Sc. he worked for the "InMoreTech" spin-off company on designing a 4-layer PCB prototype of an input device for disabled people. Furthermore, he was a teacher in the fields of computer usage, network access and e-commerce for the Go-Online action line of the Operational Programmes, Information Society and Competitiveness funded by the 3rd European Union Support Framework. Also, he was a teaching assistant for the "Real-time Systems Development" and "Computer Architecture" courses of the Computer Engineering Department bachelor program in Technical University of Crete. In addition, he worked as private teacher for the 3rd class high school course "Applications development using programming languages".

During his Ph.D, he worked as a researcher for the "hArtes", a project (IST-035143) of the Sixth Framework Programme of the European Community under the thematic area "Embedded Systems". In addition, he served as a peer reviewer for many international conferences and journals. He also provided teaching assistance for many courses of the Computer Engineering M.Sc. program at the Delft University of Technology.

Dimitris Theodoropoulos is a member the Technical Chamber of Greece from 2003. His current research interests include: reconfigurable computing, immersive-audio applications, embedded systems, cryptography and computer architecture.