A Reconfigurable Audio Beamforming Multi-Core Processor

Dimitris Theodoropoulos, Georgi Kuzmanov, and Georgi Gaydadjiev
{D.Theodoropoulos, G.K.Kuzmanov, g.n.gaydadjiev}@tudelft.nl

Computer Engineering Laboratory, EEMCS, TU Delft, P.O. Box 5031,
2600 GA Delft, the Netherlands

Abstract. Over the last years, the Beamforming technique has been adopted by the audio engineering society to amplify the signal of an acoustic source, while attenuating any ambient noise. Existing software implementations provide a flexible customizing environment, however they introduce performance limitations and excessive power consumption overheads. On the other hand, hardware approaches achieve significantly better performance and lower power consumption compared to the software ones, but they lack the flexibility of a high-level versatile programming environment. To address these drawbacks, we have already proposed a minimalistic processor architecture tailoring audio Beamforming applications to configurable hardware. In this paper, we present its application as a multi-core reconfigurable Beamforming processor and describe our hardware prototype, which is mapped onto a Virtex4FX60 FPGA. Our approach combines software programming flexibility with improved hardware performance, low power consumption and compact program-executable memory footprint. Experimental results suggest that our FPGA-based processor, running at 100 MHz, can extract in real-time up to 14 acoustic sources 2.6 times faster than a 3.0 GHz Core2 Duo OpenMP-based implementation. Furthermore, it dissipates an order of magnitude less energy, compared to the general purpose processor software implementation.

1 Introduction

For many decades, the Beamforming technique has been used in telecommunications to strengthen incoming signals from a particular location. Over the last years, it has been adopted by audio engineers to develop systems that can extract acoustic sources. Usually, microphone arrays are used to record acoustic wavefronts originating from a certain area. All recorded signals are forwarded to processors that utilize Beamforming FIR filters [1] to suppress any ambient noise and strengthen all primary audio sources.

Research on literature reveals that the majority of experimental systems are based on standard PCs, due to their high-level programming support and potential of rapid system development. However, two major drawbacks of these approaches are performance bottlenecks and excessive power consumption. In order to reduce power consumption, many researchers have developed Beamforming systems based on Digital

This research is partially supported by Artemisia iFEST project (grant 100203), Artemisia SMECY (grant 100230), FP7 Reflect (grant 248976)

Signal Processors (DSPs), however performance is still limited. 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 General Purpose Processor (GPP). Such an approach does not provide a high-level programming environment for testing the system that is under development. For example, in many cases, the designers should evaluate the Signal-to-Noise-Ratio (SNR) of an extracted source under different filter sizes and coefficients sets. Such experiments can easily be conducted using a standard PC with a GPP, but they would take long time to be re-designed in hardware and cannot be performed on the field at post-production time. The latter problem can be alleviated by introducing reconfigurable hardware and appropriate means to control it.

In our previous paper [2], we have proposed a minimalistic architecture for embedded Beamforming. It supports nine high-level instructions that allow rapid system configuration and data processing control. Moreover, the presented model provides a versatile interface with a custom-hardware Beamforming processor, thus allowing faster system re-testing and evaluation.

The main contribution of this paper is the analytical presentation and evaluation of a Beamforming microarchitecture, which supports the high-level architecture proposed in our previous work [2]. Our microarchitecture is specifically tailored to reconfigurable multi-core implementations. We prove that our proposal combines the programming flexibility of software approaches with the high performance, low power consumption and limited memory requirements of reconfigurable hardware solutions. The architecture implementation allows utilization of various number of processing elements, therefore it is suitable for mapping on reconfigurable technology. 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. More specifically, the contributions of this paper are the following:

- We propose a microarchitecture of a Multi-Core BeamForming Processor (MC-BFP), which supports the high-level architecture, originally presented in our previous work. The new microarchitecture was mapped onto a Virtex4 FX60 FPGA. The program-executable memory footprint is approximately 40 kbytes, thus makes it a very attractive approach for embedded solutions with limited on-chip memory capabilities.
- We compared our FPGA-based approach against an OpenMP-annotated software implementation on a Core2 Duo processor running at 3.0 GHz. Experimental results suggest that our prototype can extract 14 acoustic sources 2.6 times faster than the Core2 Duo solution.
- Our hardware design provides a power-efficient solution, since it consumes approximately 6.0 Watts. This potentially dissipates an order of magnitude less energy,

Throughout this paper, we have adopted the terminology from [3]. According to the book, 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 microarchitecture, defines how the control and the datapaths are organized to support the architecture functionality.

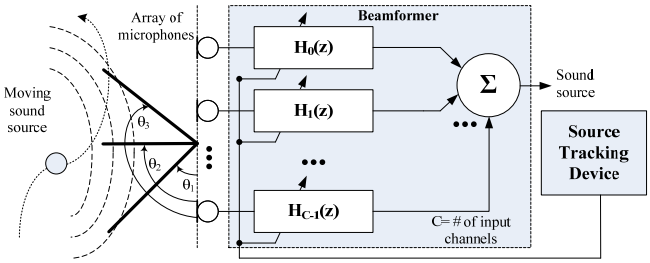

Fig. 1. A filter-and-sum beamformer.

compared to the Core2 Duo implementation that requires tens of Watts when fully utilized.

The rest of the paper is organized as follows: Section 2 provides a brief background on the Beamforming technique and references to various systems that utilize it. In Section 3, we shortly describe our previously proposed architecture. In Section 4 we elaborate on the proposed microarchitecture, while Section 5 presents our hardware prototype and compares it against a software approach and related work. Finally, Section 6 concludes the paper.

2 Background and Related Work

Background: Beamforming is used to perform spatial filtering over a specific recording area, in order to estimate an incoming signal's direction of arrival. Although there are many approaches to efficiently perform acoustic Beamforming [1], many systems utilize a filter-and-sum approach, which is illustrated in Figure 1. A microphone array of *C* elements samples the propagating wavefronts and each microphone is connected to an $H_i(z)$, FIR filter. All filtered signals are accumulated, in order to strengthen the extracted audio source and attenuate any ambient noise. The FIR filters are utilized as delay lines that compensate for the introduced delay of the wavefront arrival at all microphones [4]. As a result, the combination of all filtered signals will amplify the original audio source, while any interfering noise will be suppressed.

In order to extract a moving acoustic source, it is mandatory to reconfigure all filter coefficients according to the source current location. An example is illustrated in Figure 1, where a moving source is recorded for a certain time within the aperture defined by the $\theta_2 - \theta_1$ angle. A source tracking device is used to follow the trajectory of the source. 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. Finally, we should note that in practise, many systems perform signal decimation and interpolation before and after Beamforming respectively, in order to reduce the data size that requires processing.

Related work: The authors of [5] present a Beamforming hardware accelerator, where up to seven instances of the proposed design can fit in a Virtex4 SX55 FPGA, resulting in a 41.7x speedup compared to the software implementation. Another FPGA

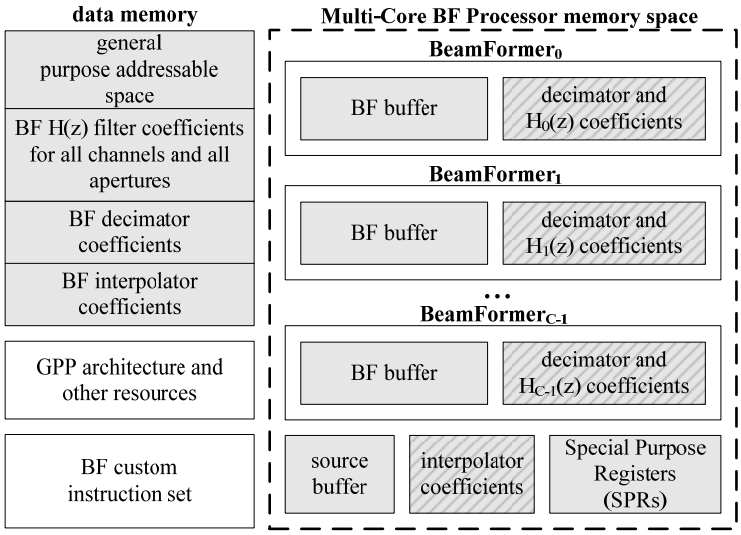

Fig. 2. Logical organization of the registers and the memory.

approach is presented in [6]. The authors have implemented an adaptive Beamforming engine, which is based on the QR matrix decomposition (QRD), and mapped it onto a Virtex4 FPGA.

A DSP implementation of an adaptive subband Beamforming algorithm, is presented in [7]. The authors utilize an Analog Devices DSP processor to perform Beamforming over a two microphone array setup. An experimental video teleconferencing system is presented in [8] that is based on a Texas Instruments DSP. The authors combine a video camera and a 16-microphone array into a device, which can be placed in the center of a meeting table.

Finally, in [9], the authors describe a PC-based system consisting of 12 microphones. The sound source is tracked through audio and video tracking algorithms. The extracted audio signal is received from a second remote PC that renders it through a loudspeakers array. A similar multiple PC-based system is presented in [10]. The authors have implemented a real-time audio software system that uses the Beamforming technique to record audio signals.

As it can be concluded, DSP and software implementations provide a high-level developing and testing environment, but they lack of adequate performance. Hardware approaches miss a versatile programming environment that would help developing Beamforming systems easier and faster. For this reason, we believe that our solution for reconfigurable Beamformers could combine high-level programmability with improved performance and low power consumption.

3 The Beamforming Architecture

As it was mentioned in Section 1, in our previous work [2], we presented a minimalistic high-level architecture tailored to embedded Beamforming applications, which is

Table 1. Programming Model for Beamforming applications

Instruction type	Full name	Mnemonic	Parameters	SPRs modified
I/O	Input Stream Enable	InStreamEn	*b_mask*	SPR0
System setup	Clear SPRs	ClrSPRs	NONE	SPR0 - SPR[9+2·C]
	Declare FIR Filter	DFirF	*FSize*, *FType*	SPR1, SPR2, SPR3
	Set Samples Addresses	SSA	**buf_sam_addr	SPR7, SPR[10+C] - SPR[9+2·C]
	Buffer Coefficients	BufCoef	**xmem_coef_addr, **buf_coef_addr	NONE
	Load Coefficients	LdCoef	**buf_coef_addr	SPR4, SPR8, SPR10 - SPR[9+C]
	Configure # of input channels	ConfC	C	SPR9
Data processing	Beamform Source	BFSrc	*aper*, *xmem_read_addr, *xmem_write_addr	SPR5, SPR6
Debug	Read SPR	RdSPR	*SPR_num*	NONE

Table 2. Special Purpose Registers

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

briefly described in this section. Figure 2 illustrates the logical organization of the registers and available memory space, assuming that processing is distributed among *C* *BeamFormer* modules receiving data from *C* input channels. Memory space is divided into user-addressable and non user-addressable, indicated by the stripe pattern. Off-chip memory is used for storing any required type of data, like various coefficients sets for the decimators, H(z) filters and interpolators, and any other general purpose variables. The host GPP and the Multi-Core BeamForming Processor (MC-BFP) exchange synchronization parameters and memory addresses via a set of Special Purpose Registers (SPRs). Each *BeamFormer* module has a *BF buffer* and memory space for the decimator and H(z) filters coefficients. Finally, the on-chip *source buffer* is used to store the audio samples of an extracted source.

Table 1 shows the nine instructions, originally presented in [2], divided into four categories. The *SPRs modified* column shows the SPR that is modified by the corresponding instruction, which are also provided in Table 2. The functionality of each instruction parameter is explained in Section 4 and in [2].

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

ClrSPRs: Clears the contents of all SPRs.

DFirF: Declares the size of a filter and writes it to the corresponding SPR.

SSA: Specifies the addresses from where the MC-BFP will read the input samples.

BufCoef: Fetches all decimator and interpolator coefficients from external memory to *BF buffers*.

LdCoef: Distributes all decimator and interpolator coefficients to the corresponding filters in the system.

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

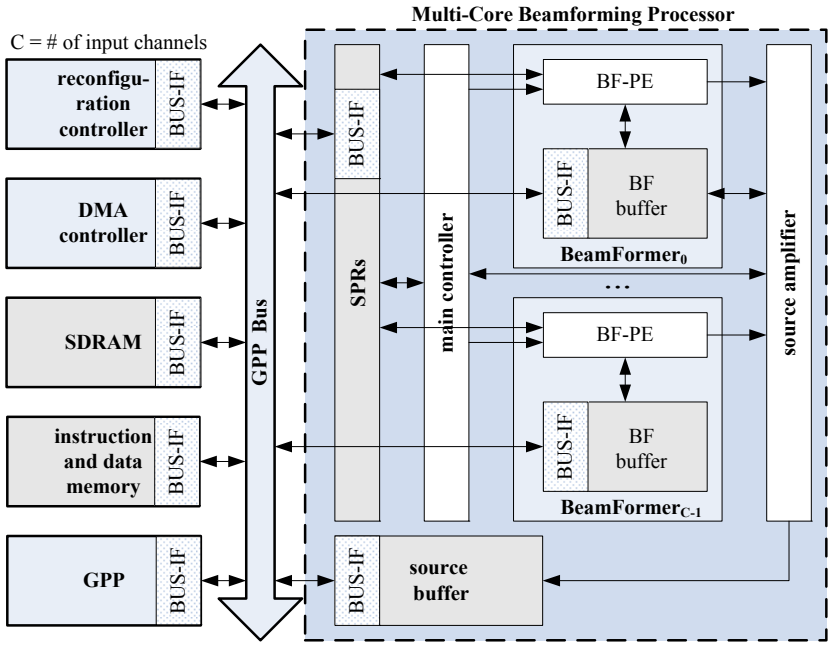

Fig. 3. Detailed implementation of the Beamforming system.

BFSrc: Processes a 1024-sample chunk of streaming data from each input channel that is enabled with the *InStreamEn* instruction, in order to extract an audio source.

RdSPR: Used for debugging purposes and allows the programmer to read any of the SPRs.

Note that the *ConfC* configures the number of input channels using a partial reconfiguration module (e.g. Xilinx Internal Communication Access Port). The reason we decided to provide this instruction is for the user to avoid performing the time-consuming implementation process when the number of microphones changes. By having already a set of reconfiguration files for different input setups (e.g. bitstreams), the user can quickly switch among them when conducting application tests.

4 The Reconfigurable Beamforming Microarchitecture

Beamforming system description: Figure 3 illustrates in more detail the Beamforming system implementation of the architecture from Figure 2. We should note that it is based on our previous Beamforming processor, originally presented in [11], however significant design improvements have been made, in order to support a high-level programming ability. In the following text, D and L represent the decimation and interpolation rates respectively.

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 *BF buffers*, we employ a *Direct Memory Access*

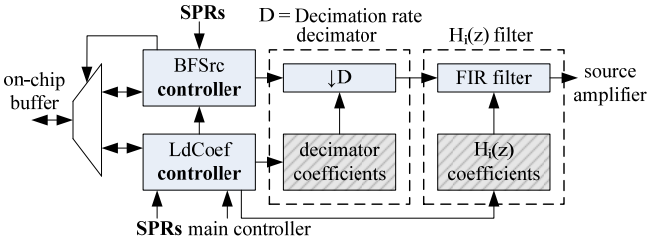

Fig. 4. The Beamforming processing element (BF-PE) structure.

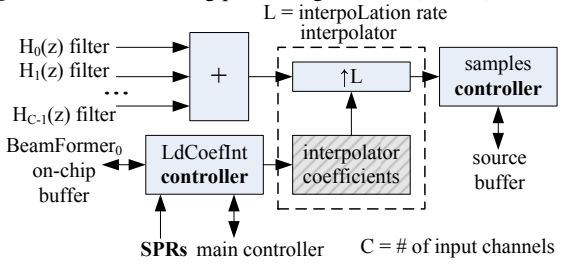

Fig. 5. The source amplifier structure.

(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 spaces inside the MC-BFP, like SPRs, *BF 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 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 consists of a *BF buffer* and a Beamforming Processing Element (BF-PE), which is illustrated in Figure 4. 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 coefficients sets from the *BF buffer* to the decimator and H(z) filter. The *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 5. 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.

Instructions 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*

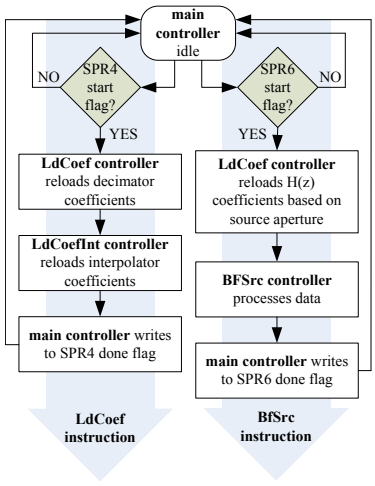

Fig. 6. Implementation of the *LdCoef* and *BFSrc* instructions.

to SPR, *GPP reads and writes to SPR*, *GPP does not access any SPR*, and they are analyzed below:

GPP reads SPR: *RdSPR* is the only instruction that belongs to this category. The GPP initiates a *GPP bus* read-transaction and, based on the SPR number *SPR_num*, it calculates the proper SPR memory address.

GPP writes to SPR: *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 (a binary mask, where each bit indicates if a channel is enabled or not) to SPR0. Similarly, in *ClrSPRs* the GPP has to iterate through all SPRs and write the zero value to them. In *DFirF* instruction, the GPP uses the filter type *Ftype* parameter to distinguish among the three different filter types (decimator, H(z) filter and interpolator), and calculate the proper SPR address to write the filter size *FSize* to SPR1, SPR3 or SPR2. In *ConfC*, the GPP writes the *C* parameter to SPR9, which is read from the partial reconfiguration controller, in order to load from the external memory the bitstream that includes *C* BF-PEs. Finally, in *SSA* instruction, the GPP iterates SPR[10+C] - SPR[9+2·C] and writes to them the *BF buffer* addresses, where 1024 input samples will be written, which are read from an array of pointers to the starting addresses of all *BF buffers*, called *buf_sam_addr*.

GPP reads and writes to SPR: *LdCoef* and *BFSrc* instructions belong to this category. In *LdCoef*, the GPP writes all decimators coefficients addresses to SPR10 - SPR[9+C], and the interpolator coefficients address to SPR8, which are read from an array of pointers within the *BF buffers*, where all coefficients are stored, called *buf_coef_addr*. As soon as all addresses are written to the proper SPRs, the GPP writes a *LdCoef start flag* to SPR4 and remains blocked until the MC-BFP writes a *LdCoef done flag* to the same SPR. Figure 6 illustrates the next processing steps within the MC-BFP, in order to reload all decimators and interpolator with the proper coefficients sets. As soon as *LdCoef start flag* is written to SPR4, the *main controller* enables the *LdCoef controller* to start reloading the decimators coefficients. Once this step is finished,

Table 3. 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

the *LdCoefInt controller* initiates the interpolator coefficients reloading procedure. As soon as all coefficients are reloaded, the latter acknowledges the *main controller*, which writes a *LdCoef done flag* to SPR4. 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* (*AAO*), in order to access the proper H(z) coefficients sets. The GPP writes the *AAO* to SPR5. Furthermore, it performs a DMA transaction, in order to read *C* 1024-sample chunks from the *xmem_read_addr* memory location and distribute them to *BF buffers* of the *C BeamFormer* modules. As soon as all data are stored, the GPP writes a *BFSrc start flag* to SPR6 and remains blocked until the MC-BFP writes a *BFSrc done flag* to the same SPR. Figure 6 shows the next processing steps inside the MC-BFP. Within each *BeamFormer* module, the *LdCoef controller* reads the *AAO* from SPR5 and reloads to the H(z) filter the proper coefficients set. 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*. As soon as 1024 samples are processed, the *main controller* writes a *BFSrc done flag* to SPR6, 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.

GPP does not access any SPR: *BufCoef* is the only instruction that belongs to this category. The GPP reads all source addresses from an array of pointers to the off-chip memory starting addresses of the coefficients sets, called *xmem_coef_addr*. Also, the GPP reads all destination addresses from *buf_coef_addr*. First, it performs a DMA transaction to transfer all decimator coefficients to the *BF buffers*. 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 *BF buffers*. Finally, with a third DMA transaction, the GPP fetches the active interpolator coefficients set to the *BF buffer* of $BeamFormer_0$ module.

5 Hardware Prototype

FPGA prototype: Based on the above study, we used the Xilinx ISE 9.2 and EDK 9.2 CAD tools to develop a VHDL hardware prototype. The latter was implemented on a Xilinx ML410 board with a Virtex4 FX60 FPGA and 256 MB of DDR2 SDRAM. As host GPP processor, we used one of the two integrated PowerPC processors, which executes the program-executable that requires only 40 kbytes of BRAM. Furthermore, we used the Processor Local Bus (PLB) to connect all peripherals, which are all *BF 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. The PowerPC runs at 200 MHz, while the rest of the system is clocked at 100 MHz. Our prototype is configured with *C*=16 *BeamFormer* modules, thus it can process 16 input channels concurrently. Also, within each BF-PE and the *source amplifier*, all decimators, H(z) filters and the interpolator were generated with the Xilinx Core Generator.

Table 3 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 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 from the entire Beamforming system. As it can be observed, a single *BeamFormer* requires less than 600 slices, 2 DSP slices and 8 Kbytes of BRAM, which makes it feasible to integrate many such modules within a single chip. Based on the data of Table 3, we estimated that a V4FX60 could accommodate up to 19 channels, while larger chips, such as the V4FX100 and V4FX140, could be used for up to 54 and 89 microphone arrays setups respectively.

Performance evaluation: We conducted a performance comparison of our hardware prototype against an OpenMP optimized software implementation on a Core2 Duo processor at 3.0 GHz. In both cases, *C*=16 input channels were tested, with sampling frequency of 48000 Hz. The decimation and interpolation rates were *D*=*L*=4. Figure 7 shows the results of our experiments. We conducted tests with up to 14 acoustic sources placed inside the same recording area. Recorded audio data of 11264 msec duration, divided into 528 iterations each consisting of 1024 samples, were stored to external memories and used as input to both implementations. Thus, in order to support real-time data processing, all iterations should be done within the aforementioned time window. Furthermore, for all the tests conducted to our hardware prototype, we have included the time spent on SDRAM transactions (HW SDRAM), on the Beamforming calculations (HW BF) and on the PPC (SW-PPC). Thus, the achieved speedup is for the complete application execution.

As we can see from Figure 7, both PC-based (SW-OMP) and FPGA-based systems can support real-time processing when there are present up to four sources. However, when there are five or more acoustic sources, the PC-based system fails to meet the timing constraints, since it takes more than 11264 msec to perform all required calculations. Regarding the hardware prototype, approximately 55% of the execution time is spent on SDRAM accesses, since in each processing iteration, input data from all channels have to be stored to *BF buffers*. This performance degradation could be improved if a faster SDRAM module is available. Even though, our FPGA-based approach can still support up to 14 sources extracted in real-time.

Energy and power consumption: Based on the Xilinx XPower utility, our FPGA prototype consumes approximately 6.0 Watts of power. This is an order of magnitude lower compared to a contemporary Core2 Duo processor, since the latter consumes up to 65 Watts of power when fully utilized [12]. Based on the power consumption and the execution time of both systems, we calculated the energy consumption using the well-known formula $E = P \cdot t$, where E is the energy consumed during the time t, while applying P power. During the software execution, the Core2 Duo was approxi-

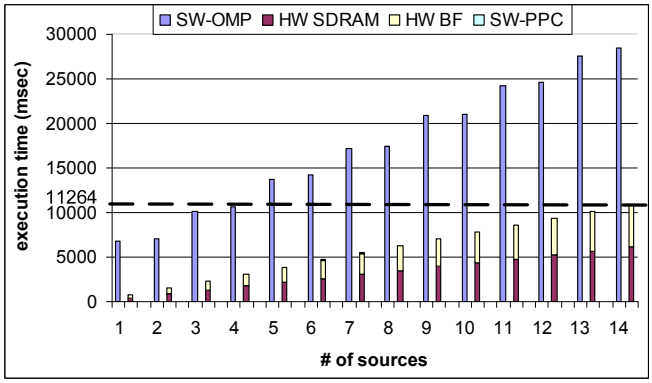

Fig. 7. Software and hardware systems execution time.

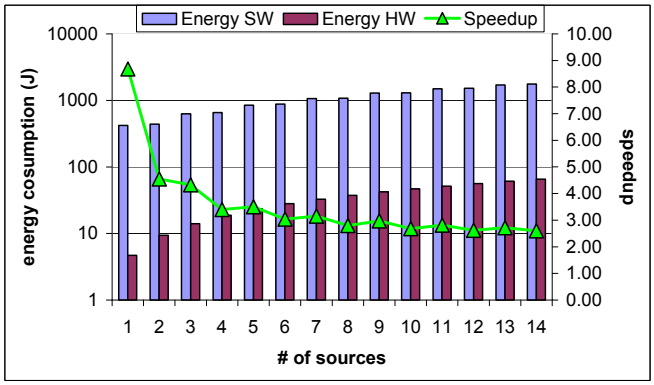

Fig. 8. Energy consumption and speedup comparison.

mately 95% utilized. Thus, we safely assumed that its power consumption during the program execution is approximately 0.95·65 Watts = 61.75 Watts. Figure 8 suggests the energy consumption for the PC-based system (Energy SW), the hardware system (Energy HW) and the achieved speedup against the Core2 Duo software implementation for all conducted tests. As we can observe, because the hardware solution is faster and also requires much less power, it consumes more than an order of magnitude less energy compared to the PC-based system.

Related work: Direct comparison against related work is difficult, since each system has its own design specifications. Moreover, to our best knowledge, we provide the first architectural proposal for a reconfigurable Beamforming computer. Previous proposals are mainly micro-architectural ones. In [7], the authors utilize an ADSP21262 DSP, which consumes up to 250 mA. Furthermore, its voltage supply is 3.3 V [13], 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-microphone 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 [5] 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.

6 Conclusions

In this paper, we implemented our previously proposed minimal architecture as a multi-core processor for Beamforming applications. Our FPGA prototype at 100 MHz can extract in real-time up to 14 acoustic sources 2.6x faster than a Core2 Duo solution. Power consumption is an order of magnitude lower than the software approach on a modern GPP. Ultimately, our solution combines high-level programmability with improved performance, better energy efficiency and limited on-chip memory requirements.

References

1. B. V. Veen et. al., "Beamforming: a versatile approach to spatial filtering," in *IEEE ASSP Magazine*, vol. 5, April 1988, pp. 4–24.
2. Dimitris Theodoropoulos et. al., "Minimalistic Architecture for Reconfigurable Audio Beamforming," in *International Conference on Field-Programmable Technology*, December 2010, pp. 503–506.
3. Gerrit Blaauw and Frederick Brooks, "Computer Architecture: Concepts and Evolution," February 1997.
4. Bill Kapralos et. al., "Audio-visual localization of multiple speakers in a video teleconferencing setting," in *International Journal of Imaging Systems and Technology*, vol. 13(1), October 2003, pp. 95–105.
5. Ka-Fai Cedric Yiu et. al., "Reconfigurable acceleration of microphone array algorithms for speech enhancement," in *Application-specific Systems, Architectures and Processors*, 2008, pp. 203–208.
6. "Implementing a Real-Time Beamformer on an FPGA Platform," in *XCell journal*, Second Quarter 2007, pp. 36–40.
7. Zohra Yermeche et. al., "Real-time implementation of a subband beamforming algorithm for dual microphone speech enhancement," in *IEEE International Symposium on Circuits and Systems*, May 2007, pp. 353–356.
8. Mark Fiala et. al., "A panoramic video and acoustic beamforming sensor for videoconferencing," in *IEEE International Conference on Haptic, Audio and Visual Environments and their Applications*, October 2004, pp. 47–52.
9. J. Beracoechea, et. al., "On building Immersive Audio Applications Using Robust Adaptive Beamforming and Joint Audio-Video Source Localization," in *EURASIP Journal on Applied Signal Processing*, June 2006, pp. 1–12.
10. H. Teutsch, et. al., "An Integrated Real-Time System For Immersive Audio Applications," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, October 2003, pp. 67–70.
11. D. Theodoropoulos, et. al., "A Reconfigurable Beamformer for Audio Applications," in *IEEE Symposium on Application Specific Processors*, pp. 80–87.
12. Intel Corporation, "http://ark.intel.com/Product.aspx?id=33910."
13. Analog Devices Inc, "SHARC Processor ADSP-21262," May 2004.