# A Reconfigurable Beamformer for Audio Applications

Dimitris Theodoropoulos
*D.Theodoropoulos@tudelft.nl*

Georgi Kuzmanov
*G.K.Kuzmanov@tudelft.nl*

Georgi Gaydadjiev
*g.n.gaydadjiev@tudelft.nl*

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

## Abstract

*Beamforming is a signal processing technique that improves the signal strength received from a specific location. It is already used for many decades in telecommunications, while over the last years, it has been adopted by the audio research society, mostly to enhance speech recognition. In this paper, we propose a scalable organization for a hardware time-invariant beamformer that can be used in small handheld devices and complete 3D-audio systems. Our design can be configured according to the number of input channels. Furthermore, all critical internal modules, such as decimators, FIR filters and interpolators can be adjusted to support various input sampling rates. We developed a hardware prototype in VHDL targeting the Xilinx ML410 board incorporating Virtex4 FX60 FPGA. Following a constrained approach regarding FPGA resource utilization, our hardware prototype occupies 21% of the aforementioned FPGA when instantiating 16 beamforming modules, and consumes approximately 2 Watts of power. Furthermore, our design achieves a speedup of 28 compared to an OMP-annotated software implementation running on a Pentium D at 3.4 GHz. We also compared our design against prior related work. Results suggest that it can extract an audio source up to 11 times faster compared to a reconfigurable adaptive beamformer, and up to 19 times faster compared to DSP implementations.*

## 1. Introduction

Beamforming is a technique that is widely used over the last decades in many application fields, like the SOund Navigation And Ranging (SONAR), RAdio Detection And Ranging (RADAR), telecommunications and biomedical [1]. Over the last years, the beamforming technique has been also adopted from the audio research society, mostly to enhance speech recognition. As an example, NXP's "LifeVibes"[1] software solutions are utilized in various handheld devices to enhance voice communications. In the audio domain, the antennas array is replaced by a microphone array that captures the audio environment. Generally, there are two different types of beamforming, non-adaptive (or time-invariant or non-blind) and adaptive (or blind) [2], [1]. 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 most microphone array applications, there is a variable number of audio sources that move freely inside a certain area and they have to be tracked and identified, thus requiring a large amount of data to be processed. Related work on *audio systems* that exploit the beamforming techniques, reveals that many systems are implemented using desktop PCs [3], [4]. However, such approaches, when adaptive beamforming is applied to extract multiple audio sources, introduce processing bottlenecks because the algorithm may not converge fast enough [5]. Furthermore, embedded systems that apply the beamforming technique, require constrained resource utilization and reduced power budget. Thus, a PC-based approach cannot be considered as a suitable solution for embedded systems such as handheld devices. DSP-based approaches proposed in the literature provide a suitable solution regarding power consumption [6], but they also introduce processing bottlenecks when there is an increased number of input channels.

Related work on *audio systems* that utilize the beamforming technique mapped on hardware (e.g. [7], [8]) mainly focuses on designing hardware accelerators that calculate in real-time the impulse responses of the beamformer filters (adaptive beamformers). A major advantage of adaptive beamforming is that there is no need to take into account the acoustical properties of the environment [6]. However, in cases when there is no need for real-time calculation of the beamformer

1. http://www.software.nxp.com

filters (e.g. small conference rooms), such advanced approaches increase the hardware complexity and power consumption. Furthermore, they constrain the maximum number of modules that can be instantiated in parallel, because a considerable amount of hardware resources is occupied by the real-time filter coefficients recalculation circuit.

In this paper, we consider a design scenario, where adaptive beamforming is not needed. Thus, at the cost of small acceptable reduction of the signal quality, we present a hardware implementation of a non-adaptive beamformer where all possible filter coefficients are precalculated according to Parra's approach described in [9] and stored to on-chip memory. This way, we save valuable hardware resources that can be used to instantiate additional beamforming modules at the expense of extra on-chip memory. Parra's approach considers wideband frequency-invariant beamforming and the microphones array configuration can be arbitrary. Furthermore, it can produce better results even with fewer microphones comparing to other approaches, a fact that is very important especially for embedded systems.

The advantages of the design we propose are as follows:

- *Flexibility*: We propose a reconfigurable organization for a hardware beamformer that can be parameterized according to the number of input channels, the sampling rate and the number of source apertures;
- *A compact, real hardware prototype*: We designed a hardware prototype instantiating up to 16 beamforming modules and mapped it onto a Xilinx Virtex4 FX60 FPGA. In contrast to PC-based related work ([3], [4]) that requires in the order of 100 Watt power, our prototype, requires approximately 2 Watts, according to Xilinx XPower;
- *Improvements over related work*: We compared our prototype with an OMP-annotated C software implementation running on a Pentium D at 3.4 GHz. Experimental results suggest an application speedup of up to 28 compared to the software implementation. Furthermore, we compared our design against related work that utilizes adaptive beamforming approaches mapped onto a FPGA [8] and DSP ([10], [6]). Results suggest that our prototype can extract audio sources up to 12 and 19 times faster respectively.

The remainder of the paper is organized as follows: Section 2 provides a brief theoretical background on the beamforming technique and discusses some related work. In Section 3, we present our proposed design. In Section 4, we describe our hardware prototype and compare it against related work, while Section 5 concludes the paper.

## 2. Background, Related Work and Problem Statement

In this section, we provide a short theoretical background of the beamforming technique. Furthermore, we discuss some related work on the audio domain that utilizes beamforming.

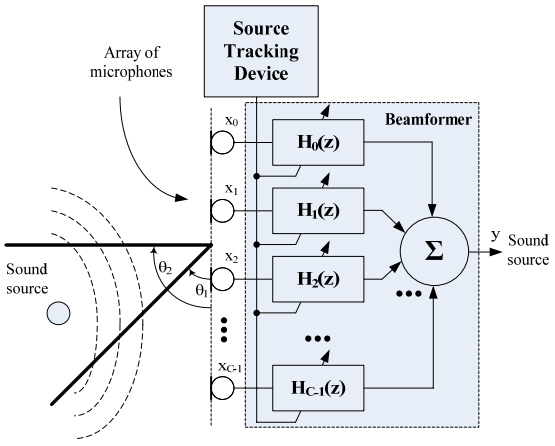


Figure 1. A filter-and-sum beamformer.

**Theoretical background:** 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. Figure 1 depicts a schematic overview of a beamformer utilizing the filter and sum approach [1]. 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. Normally, the input data channels are downsampled by a factor *D* in order to reduce the data rate:

$$xD_i[n] = x_i[n * D] \tag{1}$$

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 coefficients set based on the source location:

$$yD_i[n] = \sum_{j=0}^{L-1} h_i[j] * xD_i[n-j] \tag{2}$$

where *L* is the number of filter taps and *h* are the filter coefficients. The beamformer output is given by the summary of all $yD_i$ signals:

$$yD[n] = \sum_{i=0}^{C-1} yD_i[n] \tag{3}$$

where *yD* is the downsampled extracted source. Then, *yD* is upsampled by the same factor *D* according to equation (4) to acquire the upsampled extracted source *y*:

$$y[n] = \begin{cases} yD[\frac{n}{D}] & , if \frac{n}{D} \in Z \\ 0 & , otherwise \end{cases} \tag{4}$$

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 [11]. The combination of all filtered signals will amplify the desired one, while all interfering signals will be attenuated. However, in order to succeed on extracting the desired signal, we have to know a priori its direction-of-arrival (DOA). For example, in Figure 1, the desired sound source is in the aperture defined by the angle $\theta_2 - \theta_1$. For this reason, a tracking device can provide the source location and accordingly reconfigure the FIR filter coefficients (normally referred to as "*beamsteering*").

In the current C software implementation, all input channels are sampled at 48 KHz for 0.512 sec, thus in total there are 48000 samples/sec * 0.512sec = 24576 16-bit signed audio samples. However the audio streams are downsampled by 4 before they are forwarded to the beamsteering $H_i(z)$ FIR filters, in order to reduce the data volume that will be processed. Consequently, once the beamformer extracts the desired audio signal, the latter is again upsampled by 4. Both the decimator and the interpolator have 242 taps. Each one of the beamsteering $H_i(z)$ FIR filters has 128 taps. All calculations in the considered software implementation are done in IEEE 754 floating point single precision format.

**Related work:** In [3] the authors present a 3D-audio system oriented to future communication applications. It consists of 12 linearly placed microphones connected to an NI-4772 VXI board acquisition hardware board mounted on a standard PC. The sound source is tracked through audio and video tracking algorithms. Once its location is known, the beamformer is steered accordingly. The audio signal is extracted through beamforming 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 Wave Field Synthesis (WFS) technology through a 10 loudspeaker array.

Another 3D-Audio system is presented in [4] where the authors describe a real-time immersive audio system that exploits the beamforming 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 speaker array utilizing WFS. In order to preserve the original sound exact coordinates, face and acoustic tracking algorithms are employed. A beamformer records the sound source and steered according to the source location inside the listening area. The WFS rendering unit receives this information and the result is the same sound source being rendered exactly at the same position in the remote location B. The complete system consists of 4 PCs, out of which one used for the WFS rendering and one for the beamforming.

However, non-adaptive beamformers are mostly used for handheld devices, like cell-phones and Personal Digital Assistants (PDAs). Such embedded systems introduce many constraints regarding computational resources and power consumption, so intensive adaptive beamforming approaches that utilize an increased number of inputs cannot be considered. As an example, in [12], the authors design a time-invariant beamformer tailored to small devices that consist of two microphones. According to the paper, results suggest a SNR improvement of 14.95 dB when using two microphones, instead of one.

A DSP implementation of an adaptive subband beamforming algorithm, known as the Calibrated Weighted Recursive Least Squares (CWRLS) beamformer, is presented in [6]. The authors utilize an Analog Devices ADSP21262 DSP processor to perform CWRLS-based beamforming 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 processor can be up to 50% with two input channels. The presented implementation is also very energy efficient, since it was predicted to have an operation time of up to 20 hours.

The authors of [8] present a hardware accelerator that utilizes microphones 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 maximization of signal-to-noise ratio beamforming algorithm. The FPGA implementation runs up to 175 MHz and achieves a speedup of up to 6x compared to their software implementation on a Pentium 4 at 3.2 GHz. Furthermore, up to 7 instances of the proposed design can fit in a Virtex4 SX55 FPGA, achieving a speedup of up to 41.7x compared to the software implementation.

**Problem statement:** As we can see, time-invariant beamforming approaches that are implemented on embedded systems, have a *limited number of input channels* due to the lack of sufficient computational resources. We address these problems, by proposing a hardware design that requires limited power and provides enough processing resources, so even *compact devices* can integrate an increased number of input channels. Furthermore, we aim at a minimal design that can be configured to support different sampling rates and angular regions. As we will show in Section 4, a prototype with 16 input channels and 19 aperture regions has a memory footprint of approximately 270 KBytes and consumes only 2 W of power.

## 3. Proposed Design

This section presents our complete BeamForming Fabric Coprocessor Module (BF_FCM)[2] that accelerates the beamforming algorithm considered, following a bottom-up description. Figure 2 presents the BF_FCM internal organization; it consists of the *primary controller* and two modules, the *buffers-decimators* and the *beamformer-interpolator*. The primary controller is a Finite State Machine (FSM) , explained in detail in Section 4. Our design is parameterizable regarding the number of input channels *C* that can process and the number *N* of source apertures. Although the software implementation employs the

2. We follow Xilinx terminology.

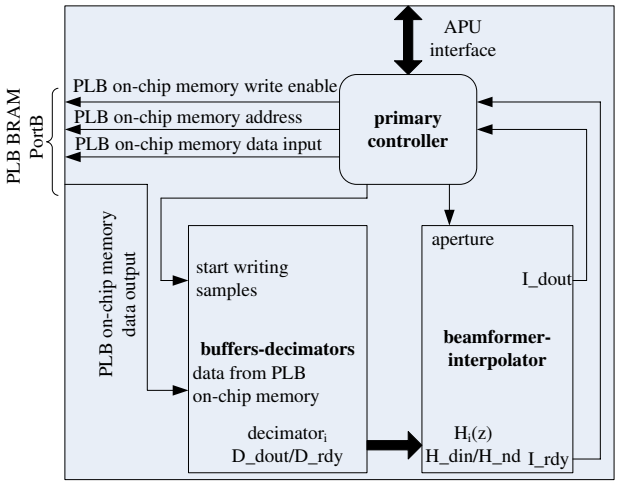


Figure 2. BF_FCM organization.

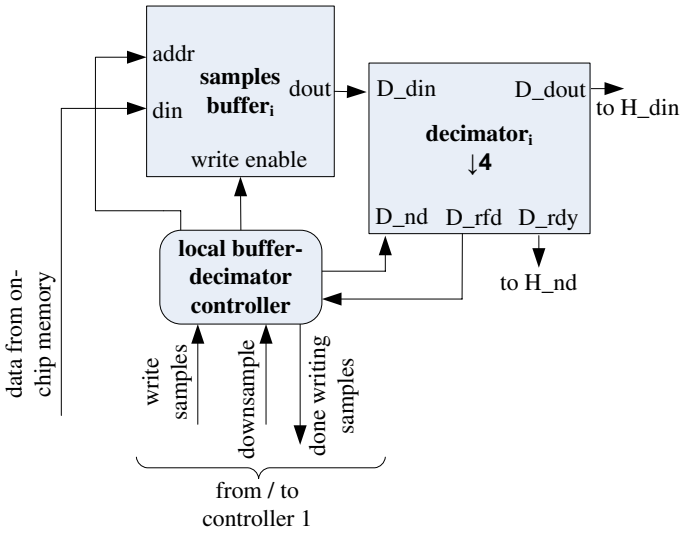


Figure 3. Buffer-decimator module.

IEEE 754 single precision format, we followed a fixed point approach for all of our calculations. Accuracy evaluation, presented in Section 4, suggests that such an approach guarantees sufficient signal quality.

**The buffer-decimator module:** As we mentioned in Section 2, each microphone signal (input channel) is downsampled from 48 KHz to 12 KHz (eq. 1). For this reason, the *buffer-decimator* (BD) module consists of the *samples buffer* (SB), a decimator by 4 and the *local buffer-decimator controller* (BDC), as illustrated in Figure 3. The latter is responsible for storing 1024 16-bit signed audio samples to the *SB* that were captured by the corresponding microphone. The external *controller1* (C1) initiates copying samples to the *SB* through the *write samples* signal. Once all samples are copied, the *BDC* acknowledges the *C1* through the *done writing samples* signal. Furthermore, the *C1* instructs the *BD* module to start downsampling through the *downsample* signal. The *BDC* controls the samples streaming to the decimator using the *D_nd* (new data) and *D_rfd* (ready for data) signals. Once the decimator generates an output, it is forwarded to the corresponding *H(z)* beamsteering FIR filter, which is acknowledged through the *D_rdy* (ready) signal. We assumed that all captured samples from the microphones are already stored in an on-chip memory and can be read from the *data from on-chip memory* signal.

**The buffers-decimators top module:** As we mentioned in Section 1, our design can be reconfigured to support variable number of input *C* channels. In order to exploit the fact that all captured signals can be downsampled concurrently, we designed the *controller1* (C1) that can efficiently connect *C* number of *BD* modules, as illustrated in Figure 4. As soon as the external *primary controller* instructs the *C1* to start copying samples, the latter initiates the data transfer to each one of the *BD* modules through the *write samples* signal. Depending on the available memory bandwidth, the *C1* can be configured to initiate the

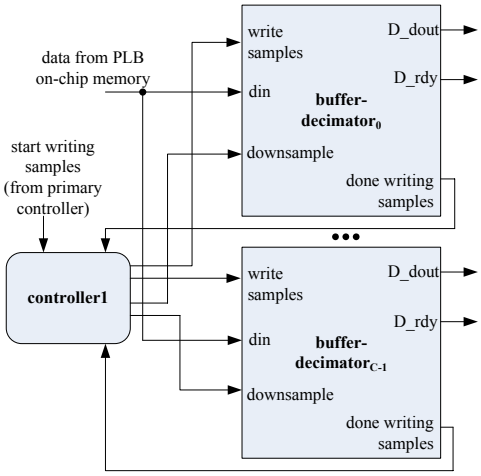


Figure 4. *Controller1* and buffer-decimator (BD) modules.

data transfer serially in each *BD* module or concurrently to all of them. In either case, once all *BD* modules acknowledge the *C1* (through the *done samples writing* signal) that all samples are written, the latter concurrently instructs all *BD* modules through the *downsample* signal to start downsampling. Each *BD* module forwards the downsampled signal to the corresponding beamsteering FIR filter *H(z)*.

**The H(z) beamsteering FIR filter:** The *H(z)* FIR filter is based on the traditional multiply and add approach (eq. 2). As depicted in Figure 5, its inputs are the source aperture that is recorded from the source tracking device and the *D_out* and *D_rdy* signals from the corresponding decimator. Once there is an output generated from the decimator, it is forwarded to the *H(z)* filter, which is acknowledged through the *D_rdy* signal. Based on the recorded aperture, the *H(z)* uses the

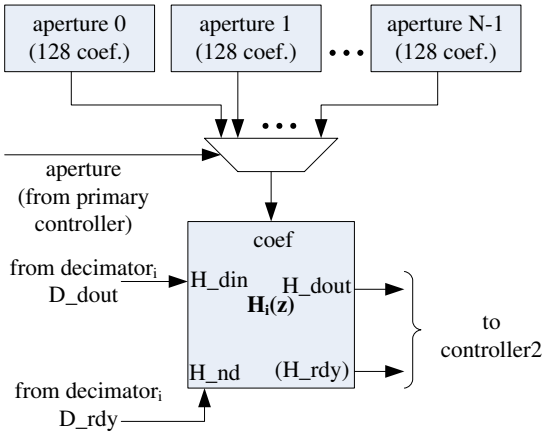


Figure 5. A beamsteering FIR filter.

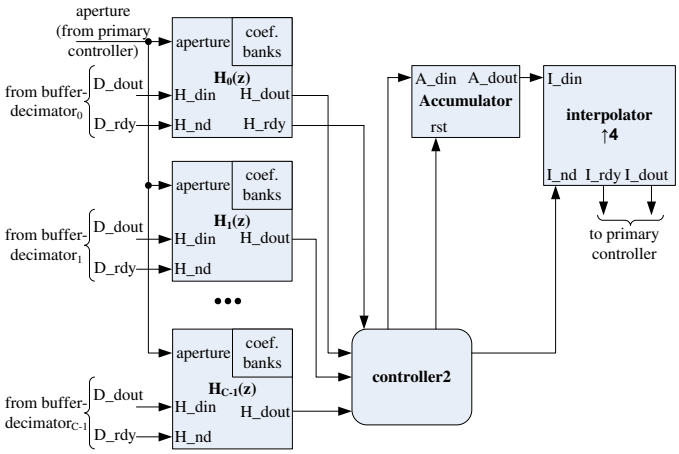


Figure 6. *Controller2* and the accumulator.

appropriate set among the *N* available 128-coefficient sets to filter the downsampled signal. The filtered signal is forwarded to the external *controller2* (C2). The *H_rdy* signal is used to acknowledge the *C2* that there are new available data.

**The beamformer-interpolator module:** As we mentioned in Section 2, once the source is extracted from the filtered signals, it is again upsampled from 12 KHz to 48 KHz. Based on that, the *beamformer-interpolator* (BI) module integrates all *H(z)* beamsteering FIR filters, an accumulator (eq. 3) and an interpolator by 4 (eq. 4), as is illustrated in Figure 6. The *controller2* (C2) can efficiently connect *C* number of *H(z)* modules that process concurrently the downsamples signals. Since all of them are structurally identical, it suffices to have only one *H_rdy* signal to acknowledge the *C2* each time there is a new sample ready. All *H(z)* outputs are forwarded to the accumulator, in order to calculate the extracted source. Once all samples are accumulated, the *C2* acknowledges the interpolator through the *I_nd* signal and the result is forwarded to it. Finally, as soon as the interpolator generates new data, it acknowledges the external *primary controller* through the *I_rdy* signal. The upsampled signal *I_dout* is stored back to the on-chip memory through the external *primary controller*.

## 4. Experimental Results

In this section we describe a hardware prototype that was built based on the design organization described in Section 3. To build the complete system, we used a Xilinx ML410 board with a V4FX60 FPGA on it, which integrates two PowerPC processors. We utilized one of them just to transfer audio samples between the SDRAM and the on-chip PLB memory. Our beamforming accelerator was designed in VHDL and synthesized using the Xilinx Integrated Synthesis Environment (ISE) 10.1.03 and the Xilinx Synthesis Tool (XST). The complete embedded system was designed using the Xilinx Embedded Development Kit (EDK) 10.1.03. All decimators, beamsteering FIR filters and the interpolator were designed using the Xilinx LogiCore generator.

**Experimental setup:** We utilize a PowerPC as host GPP for our beamforming application. The PowerPC utilizes a 32-Kbytes instruction memory connected to the Processor Local Bus (PLB) through its PORTA. A second memory of 128 Kbytes is used by the PowerPC for temporal storage of on-chip data through its PORTA. For this reason the latter is connected to the PLB, while PORTB is connected directly to the BF_FCM. This shared memory implementation allows the BF_FCM to access memory more efficiently compared to accessing it through the PLB. A 64-Mbytes DDR SDRAM is used to store audio samples, which can be accessed from PowerPC again through the PLB. The BF_FCM is connected directly to the PowerPC through its Auxilary Procesor Unit (APU) interface [13]. In our case, we configured it to decode one User Defined Instruction (UDI) that would start the BF_FCM. In order also to monitor the correct functionality of our system, we connected the FPGA board through an RS232 module to a standard PC.

**Beamforming Hardware Accelerator:** In each loop, the PowerPC copies 1024 16-bit audio samples (from each channel) from the SDRAM to the on-chip PLB memory. When samples storing is done, the BF_FCM execution is initiated via our UDI, as shown in the following pseudocode snippet:

```
For all audio samples in SDRAM
{
    copy 1024 samples from SDRAM to BRAM;
    UDI (source angle, samples address);
    copy samples from BRAM to SDRAM;
}
```

As soon as a UDI is detected, the primary controller acknowledges the PowerPC through the APU interface. It also reads the source aperture and the starting address of the audio samples stored in the on-chip PLB memory. Both parameters are stored

Table 1. FPGA resource utilization of decimator, H(z) FIR filter and interpolator.

| | Decimator | H(z) | Interpolator |
|---|---|---|---|
| **rdy** (cc) | 268 | 268 | 31 |
| **rfd** (cc) | 67 | 268 | 268 |
| **DSP48/MEM** | 1/2 | 1/5 | 2/1 |

in local registers. The source *aperture* parameter is used to select the appropriate coefficients set for the beamsteering filters $H_i(z)$, where i=0,...,C-1. The primary controller initiates copying the first 1024 samples from all *C* channels (using the *start writing samples* signal) to local samples buffers, instantiated in the buffers-decimators module. Once all audio samples are transferred, the latter downsamples the signal from all channels by 4. The downsampled signal $decimator_i$ *dout* from decimator *i* is forwarded to the beamformer-interpolator module, where it is filtered from the FIR filter $H_i(z)$. In the same module, all filtered signals are accumulated and the result is upsampled by 4. The primary controller stores back the upsampled signal *dout* to the on-chip PLB memory, whenever the *rdy* signal indicates that a new sample is processed. Once the BF_FCM has finished, all processed samples are written back to the SDRAM and 1024 new audio samples (again from all *C* channels) are fetched from the SDRAM to the on-chip PLB memory for processing. We should note that, since there are many data transfers between the SDRAM and the on-chip PLB memory, a Direct Memory Access (DMA) controller and double buffering can be employed to improve the data-transfer rate.

**FPGA resource utilization:** In our current prototype, we cascaded as many *BD* and *H(z)* modules as the maximum number of available input channels (*C*=16). Furthermore, the *H(z)* coefficient sets support *N*=19 different source apertures. However, we followed a constrained approach regarding the decimator, *H(z)* and interpolator implementation. More specifically, we choose to map all of them onto DSP48 slices and utilize the minimum required amount of them. Table 1 shows the specifications of each module. *Rdy* indicates after how many cycles a sample is ready, while *rfd* indicates after how many cycles the corresponding module can accept new data to process. We have to clarify however that the proposed design is independent from these specifications, meaning that if these modules are substituted by others with different timings (but of course with the same interface), the BF_FCM will be again completely functional. Table 2 shows the Virtex4 FX60 FPGA resource utilization from the BF_FCM. As it is expected, more than 50% of the available on-chip memory is dedicated to store all coefficient sets for the decimators, *H(z)* filters and the interpolator.

**Performance evaluation:** As we mentioned in Section 3, in each iteration 1024 audio samples captured from each channel are copied to the *SBs* inside the *BD* modules. In our current

Table 2. BF_FCM resource utilization.

| | | |
|---|---|---|
| **XtremeDSP Slices** | 35 | 27% |
| **RAMB16s** | 130 | 56% |
| **Total Slices** | 5384 | 21% |
| **Maximum frequency (MHz)** | 250 | N/A |

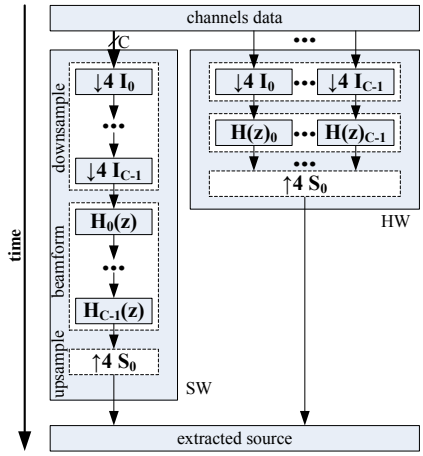


Figure 7. Tasks scheduling in SW and HW implementations of the beamforming application.

prototype, the memory bandwidth supports reading up to 4 16-bit audio samples per cycle. Thus, *C1* spends 256 cycles per channel to store all audio samples in each *SB*. In total, we need 256*16+2=4134 clock cycles to copy all data to the *SBs* (the 2 additional cycles are spent on starting and stopping the samples reading). Once all data are stored in the *SBs*, the BF_FCM requires 68871 clock cycles to process all 1024 samples for all channels, since all of them are processed concurrently. In order to estimate the number of sources that can be extracted in real-time, we use the following formula:

$$\#ofSources = \lfloor \frac{samples\_segment}{cc * clk\_period * f_s} \rfloor \quad (5)$$

where *cc*=256*C+2+68871 (the number of clock cycles required to process all data), *C* is the number of input channels, *samples_segment* is the number of samples that are processed in each iteration (1024 in our implementation), *clk_period* is the clock period of the design and $f_s$ is the sampling rate. We consider the Virtex4 FX60 FPGA, which has in total 522 Kbytes of on-chip memory. If we subtract 160 Kbytes that are used from the PowerPC as data and instruction memories respectively, there are 362 Kbytes left available to our design. The memory footprint for C=21 input channels is 355 Kbytes, thus this is the maximum number of inputs that can be employed when using the current FPGA. Furthermore, since $f_s$=48000 samples/sec, by substituting these data in eq. (5), we calculate that up to 71 sources can be extracted in real-time from our design when employing 21 input channels.

Table 3. Comparison among all designs.

| design | channels | speedup | sources | $f_s$(Khz) | power(W) |
|---|---|---|---|---|---|
| $SW_{OMP}$ | 16 | 28x | 3 | 48 | ∼100 |
| [10] | 16 | 19x | 16 | 11.025 | ∼1.7 |
| BF_FCM-16 | 16 | 1x | 73 | 48 | 2 |
| [6] | 2 | 19x | 4 | 48 | ∼0.825 |
| BF_FCM-2 | 2 | 1x | 76 | 48 | 0.89 |
| [8] | 4 | 11x | 19 | 16 | N/A |
| BF_FCM-4 | 4 | 1x | 76 | 48 | 0.9 |

Table 4. Software and hardware comparison.

| channels | SW(ms) | $SW_{OMP}$(ms) | HW(ms) | SPEEDUP | $SPEEDUP_{OMP}$ |
|---|---|---|---|---|---|
| 8 | 661 | 383 | 14 | 47x | 27x |
| 12 | 904 | 580 | 21 | 43x | 28x |
| 16 | 1152 | 774 | 28 | 41x | 28x |

Table 3 summarizes the comparison of our proposed design against other related work. We categorize the comparison based on the input channels of each system and compare it with a prototype of our design (BF_FCM) that employs the same number of input channels. We compared our design against a software version of the beamforming algorithm running on a Pentium D at 3.4 GHz. As we mentioned in Section 2, the software processes 24576 samples/channel and there are 4 different sources. It is also written in a way to exploit OMP pragma annotations for a more efficient execution in more than one processors. Figure 7 illustrates the different tasks (downsampling, beamforming and upsampling) between the software and hardware implementations. The PC-based implementation processes serially each channels data, thus leading in an increased total execution time compared to the parallel hardware implementation. Table 4 shows the comparison of the proposed design against the software with and without the OMP pragma annotations under different number of input channels. Values in the second, third and fourth columns indicate times of processing in msec. With the OMP pragma annotations, the software performance was increased by 1.5x comparing to when they are disabled. The last two columns in Table 4 show the achieved speedup under the different number of input channels. As we can see, even with the OMP annotations, our hardware prototype can accelerate the application up to 28x. The time for the software implementation to extract 0.512 sec of a source signal when there are 16 input channels is 0.154 sec. Thus, up to $\lfloor 0.512/0.154 \rfloor$=3 sources can be extracted in real-time. In contrast, by using eq. (5), we calculate that our hardware prototype can support up to 73 real-time sources for 16 input channels.

We also compared our prototype against the designs presented in [8], [6] and [10]. As we mentioned in Section 2, in contrast to our approach, the proposed design of [8] calculates in real-time the beamforming coefficients. In order to perform a fair comparison between the two designs, we consider the number of input audio samples that can be processed per second. According to the paper, up to 7 instances of the proposed beamformer can fit into the largest Virtex4 SX family. Since each one of them can process 43463 samples/sec, we assume that the largest amount of data that can be processed are 43463*7=304241 samples/sec at 175 MHz. Consequently, since $f_s = 16$ KHz, it means that up to $\lfloor 304241/16000 \rfloor$=19 sources can be extracted in real-time. Assuming C=7 also for our design, our prototype will require cc=256*7+2+68871=70663 clock cycles to process 1024 samples, or 3662532 samples/sec at 250 MHz, which is 11 times faster than the design proposed in [8]. Furthermore, according to eq. (5) our proposed design can support up to 76 sources for 16 input channels.

Regarding the real-time DSP implementation in [6], according to the paper, the proposed adaptive beamformer utilizes up to 50% of the DSP processor, which uses a $f_s = 48$ KHz with two input channels. Based on these data, we can assume that up to 48000 samples/sec/input * 2 inputs = 96000 samples/sec can be processed when the DSP utilization is 50%. Thus, the proposed beamformer can roughly process 192000 samples/sec when the DSP is utilized at 100%. On the other hand, assuming also a prototype that utilizes our proposed design with C=2, it would require 256*2+2+68871=69385 clock cycles to process 1024 samples, or 3689558 samples/sec, which is 19 times faster. Consequently, the design described in [6] could support up to $\lfloor 192000/48000 \rfloor$=4 sources at $f_s = 48$ KHz, while our prototype could support up to 76 sources being extracted in real-time at the same $f_s$ for 4 input channels. We should note that the proposed beamformer of [6] does not consider moving sound sources.

In [10] the authors utilize 16 input channels at $f_s = 11.025$ KHz. Thus, we assume that their design can process up to 11025*16=176400 samples/sec, which is approximately 19 times slower comparing to our prototype that utilizes also 16 input channels. Consequently, the beamformer of [10] could support up to $\lfloor 176400/11025 \rfloor$=16 sources at $f_s = 11.025KHz$, while our prototype could support up to 73 sources being extracted in real-time at $f_s = 48$ KHz for 16 input channels.

As we can see, the number of sources that can be extracted from our design in real-time, changes only slightly, despite the different number of input channels that are employed. Thus, it provides a versatile solution that can be applied under different scenarios. For example, small devices could employ an increased number of microphones comparing to the one or two that have up to now, to enhance speech recognition. Furthermore, our design could also be integrated to high-end video-teleconferencing systems. Such systems can exploit the increased processing power and number of available inputs, since under these circumstances there are many speakers inside large rooms that require an increased number of microphones to be employed.

**Energy efficiency:** Another benefit of our FPGA design is that it requires significantly less power than systems based on high-end CPUs. We used Xilinx XPower to analyze the

complete system power consumption, which is shown in the last column of Table 3. In contrast, high-end CPUs that are utilized in similar approaches like [3] and [4], when not in idle mode, normally require tens of Watts, which in essence is an order or two of magnitude difference in favor of our design. Moreover, our design approaches the power figures of DSPs, which are typically designed as low power devices. In order to make an estimation of power consumption of the design proposed in [10], we referred to the TMS320C6201 power summary [14], in which it is stated that a typical power requirement is 1.7 W. In [6] the authors mention that their design does not consume more than 250 mA. Thus, since the voltage supply of ADSP21262 is 3.3 V [15], we can assume that the design power requirement is approximately P=3.3*0.25=0.825 W.

**Results accuracy:** In order to evaluate our hardware prototype, we used the same input audio data with the ones that were used with the software version of the beamforming algorithm. Results suggest that the signals generated from our hardware prototype could follow the software ones with an accuracy of up to three decimal digits.

## 5. Conclusions

In this paper, we proposed a reconfigurable architecture for a hardware beamformer. Our design is parameterizable regarding the supported source apertures, while it can be easily cascaded in order to process an arbitrary number of input channels. We built a hardware prototype using VHDL and mapped it onto a Virtex4 FX60 FPGA, providing a speedup of 28 compared to the OMP-enabled software implementation in a Pentium D. We also compared our design against related work that utilizes the beamforming technique and maps in onto DSPs and FPGA. Results suggested that our proposed design can extract sources in real time up to 11 and 19 times faster respectively. Furthermore, our prototype requires approximately 2 Watts of power which is two orders of magnitude less than PC-based solutions. Ultimately, if a larger FPGA is available, then additional BF_FCMs can be cascaded to concurrently extract more than one sound sources.

## 6. Acknowledgment

This work was partially sponsored by hArtes, a project (IST-035143) of the Sixth Framework Programme of the European Community under the thematic area ”Embedded Systems”; and the Dutch Technology Foundation STW, applied science division of NWO and the Technology Program of the Dutch Ministry of Economic Affairs (project DCS.7533).

## References

[1] B. V. Veen and K. Buckley, “Beamforming: a versatile approach to spatial filtering,” in *IEEE ASSP Magazine*, vol. 5, April 1988, pp. 4–24.

[2] 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*, September 2006, pp. 485–489.

[3] J. A. Beracoechea, S. Torres-Guijarro, L. Garcia, 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*, 2006, pp. 1–12.

[4] 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*, October 2003, pp. 67–70.

[5] Ivan Tashev and Henrique S. Malvar, “A new beamformer design algorithm for micrpohone arrays,” in *International Conference of Acoustic, Speech and Signal Processing*, March 2005, pp. iii/101–iii/104.

[6] 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*, May 2007, pp. 353–356.

[7] “Implementing a Real-Time Beamformer on an FPGA Platform,” in *XCell journal*, Second Quarter 2007, pp. 36–40.

[8] 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*, 2008, pp. 203–208.

[9] L. Parra, “Steerable frequency-invariant beamforming for arbitrary arrays,” in *Journal of the Acoustical Society of America*, June 2006, pp. 3839–3847.

[10] 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*, October 2004, pp. 47–52.

[11] 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*, vol. 13(1), October 2003, pp. 95–105.

[12] 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.

[13] Xilinx Inc., “PowerPC 405 Processor Block Reference Guide,” July 2005.

[14] Texas Instruments Inc, “TMS320C62x/C67x Power Consumption Summary,” July 2002.

[15] Analog Devices Inc, “SHARC Processor ADSP-21262,” May 2004.