Pyramic array

An FPGA based platform for many-channel audio acquisition

Juan Azcarreta Ortiz

Presented the 24 August 2016
at Laboratory of Processors
Architecture (LAP) and Laboratory of
Audiovisual Communications (LCAV)
École Polytechnique Fédérale de Lausanne

to obtain the Master's degree in Electronics
Engineering by

Juan Azcarreta Ortiz

suppervised by:

Prof René Beuchat, supervisor LAP
Robin Scheibler, supervisor LCAV
Prof Ramon Bragos, supervisor UPC
Lausanne, EPFL, 2016

"Microphones are to audio,
what colors are to painting."
— Stanley R. Alten

To my family. . .

Acknowledgements

First of all, I want to thank my supervisors, René Beuchat, expert professor at LAP and Robin Scheibler, PHD student at LCAV. Thanks for all your patiente and support, merci René for all the patiente and your help during the design, testing and building of the project. Thanks Robin for your great support and availability. I appreciate the interesting conversations sharing ideas we had during the project, arigatou. Special mention to my college, Sahand, who reviewed my project and helped a lot during design considerations. Without you this project would have not been the same, I am sure you will have a great success developing embeddeded systems. Thanks to Eric Bezzam and Balási Szabolcs, which collaborated in the project adding advanced knowledge and great times to it. Thanks to all of you.

I would like to thank UPC for giving me the chance to spend one year abroad at EPFL. Specially thanks to my supervisor, Ramón Bragos. Thanks to them, I was able to achieve my goal of performing my Master Thesis at EPFL.

I am greateful to my friends and colleagues I met during my Master and Bachelors at EPFL, UPC, University of Navarra and Cal Poly. Thanks to all the international friends I made during this period. I hope to see you again soon.

Last but not least, I want to thank my family, specially to my parents. They have always supported me, especially in the difficult times, thanks for your unconditional love and support.

Abstract

Microphone arrays techniques present compelling applications for robotic applications. Those techniques can allow robots to listen to their environment and infer clues from it. Such features might enable capabilities such as natural interaction with humans, interpreting spoken commands or the localization of victims during search and rescue tasks.

However, under noisy conditions robotic implementations of microphone arrays might degrade their precision when localizing sound sources. For practical applications, human hearing still leaves behind microphone arrays. Daniel Kisch is an example of how humans are able to efficiently perform echo-localization to recognize their environment, even in noisy and reverberant environments. For ubiquitous computing, another limitation of acoustic localization algorithms is within their capabilities of performing real-time Digital Signal Processing (DSP) operations. To tackle those problems, tradeoffs between size, weight, cost and power consumption compromise the design of acoustic sensors for practical applications. This works presents the design and operation of a large microphone array for DSP applications in realistic environments.

To address those problems this project introduces the Pyramic sound capture system designed at LAP in EPFL. Pyramic is a custom hardware which possesses 48 microphones distributed in the edges of a tetrahedron. The microphone arrays interact with a Terasic DE1-SoC board from Altera Cyclone V family devices, which combines a Hard Processor System (HPS) and a Field Programmable Gate Array (FPGA) in the same die. The HPS part integrates a dual-core ARM-based Cortex-A9 processor, which combined with the power of FPGA design suitable for processing multichannel microphone signals. This thesis explains the implementation of the Pyramic array. Moreover, FPGA-based hardware accelerators have been designed to implement a Master SPI communication with the array and a parallel 48 channels FIR filters cascade of the audio data for delay-and-sum beamforming applications. Additionally, the configuration of the HPS part allows the Pyramic array to be controlled through a Linux based OS. The main purpose of the project is to develop a flexible platform in which real-time echo-location algorithms can be implemented. The effectiveness of the Pyramic array design is illustrated by testing the recorded data with offline direction of arrival algorithms developed at LCAV in EPFL.

Key words: FPGA, HPS, microphone arrays, Altera Cyclone V, DE1-SoC, real-time DSP, beamforming, FIR filters.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Nowadays, listening and localizing sound sources is crucial in applications where the direction to the desired sound source is unknown. Apart of the number of available microphones to capture the audio signals, the precision of localization can be affected by the ambient noise and reverberation conditions.

This work presents the design, test, and operation of the Pyramic array, a large microphone array for real-time acoustic DSP applications. The proposed design is based on a new generation of digital microelectromechanical (MEMS) microphones. An Altera field-programmable gate array (FPGA) interfaces up to 48 such microphones to obtain samples representing the sound pressure and processes them using a cascade of FIR filters. Moreover, the data is stored and sent through the network to local storage, which can be used to perform offline beamforming algorithms.

Thanks to recent MEMS microphones in conjunction with reconfigurable logic, Pyramic array is a modular system which tackles ubiquitous computing for multi-channel audio applications. The system optimizes the trade-off between weight, size, power consumption and cost constraints of robotic systems for real-time audio applications.

1.2 Objective

The aim of the project is to build a novel 48 microphones modular array architecture called the Pyramic array. The system acquires multi-channel audio samples through a parallel SPI master to a host Altera Cyclone V device, which combines a Field Programmable Gate Array (FPGA) and an ARM-9 Hardcore Processor (HPS). The data is processed in real time by hardware accelerators FIR filters which implement a delay-and-sum beamformer. The ARM-9 processor runs on a *Ubuntu Core 14.04.4* operating system. The board has internet access thanks to an Ethernet port connection. Moreover, direction of arrival algorithms are tested offline in order

to evaluate the reliability of the acquired audio data.

1.3 Contributions

In order to achieve our goals, we implemented the following solutions:

- A modular system with 6 linear arrays of 8 microphones each, which composes a total of 48 microphones distributed in the edges of a tetrahedron. Each array samples 8 audio channels in parallel through an ADC at 48 kHz and 16 bits resolution.
- A parallel SPI communication between the microphones arrays and an FPGA platform from Altera's Cyclone V family.
- An interface between the FPGA and a 925 MHz ARM-9 processor located in the Hard Processor System (HPS) of the Altera DE1-SoC board. The system runs on a Linux based operating system.
- A parallel bank of hardware accelerators FIR filters which allows real time processing of the microphones data for real-time beamforming applications.
- An Ethernet communication between the DE1-SoC host device and the Pyramic array, which allows internet sharing and control of the system through a WebServer.
- A test of the acquired data by applying offline processing algorithms using the direction of arrival algorithms developed at LCAV.

1.4 Structure of the thesis

The first main part of the thesis is composed by Chapters 2 and 3. Chapter 2 presents the main challenges of processing audio with microphone arrays. For those purposes, different array configurations are introcuded.

Chapter 3 explains the hardware used to build the Pyramic microphones array, which is a custom hardware of 48 microphone array placed on the sides of a tetrahedron.

The second part of the project is focused on embedded systems design for real-time applications and it is formed by Chapters 4, 5 and 6. Chapter 4 introduces FPGA technologies and explains Altera's Cyclone V family DE1-SoC system, which has been chosen as a processing unit for this project. Chapter 5 details the SPI communication and FIR filters Programmable Interfaces(PI) designed in the FPGA part of the DE1-SoC.

Chapter 6 describes the HPS part of the DE1-SoC device and how it communicates with the FPGA side to receive the data from the Pyramic array. Moreover, we will explain how the recorded data is stored and how the user can interface with the microphone array through

a WebServer, *ssh* and *scp* communication. Chapter 6 displays the results of different tests performed with the Pyramic system. Finally, offline beamforming algorithms developed at LCAV have been tested to analyze the reliability of the recorded data.

To conclude, the last part of the thesis summarizes the realized work and anticipates the future directions in which the project can be continued.

2 Microphone arrays sound capturing systems

In this chapter we will review the basis of sound capture and processing systems, emphasizing into microphone arrays applications. First, we will describe some areas in which speech enhancement algorithms might be critical. For those purposes, it is crucial to understand the behavior of noise. Then, the main approaches and geometries behind microphone arrays implementations will be explained [21].

2.1 Acoustic Noise

Captured sound is a mixture of wanted and unwanted signals. The speech we want to capture is the wanted signal, while unwanted signals include other speech signals, reverberation, and what in general we call noise. Noise sources usually degrade the desired signal, and might be correlated or uncorrelated to it. Consequently, prior to designing algorithms to combat the "enemy", we need to understand its properties and nature, the nature of noise.

2.1.1 Noise Sources

Wherever we go we are surrounded by noise. Listening to the environment is an exercise of realizing the way in which background noise affects our daily lives. For instance, cars passing by, PC fans noise, engine noise and people talking in nearby tables are some examples of common noise sources. Fortunately, the human brain is able to compensate this noise and even infer clues about the environment from it. However, this occurs below a certain noise level threshold, which depends on the environmental conditions and above which the perceived noise becomes annoying. An objective metric to benchmark audio intelligibility is the Signal-to-Noise-Ratio (SNR), which is defined as the ratio of the desired signal and noise energies and is usually measured in the logarithmic scale of Decibel (Db) [16].

In order to understand this double-edged nature of noise, it is important to recover its spectral and temporal characteristics and understand how it can affect the desired signal. Noise can be stationary, i.e., remains unchanged over time, such as the fan noise coming

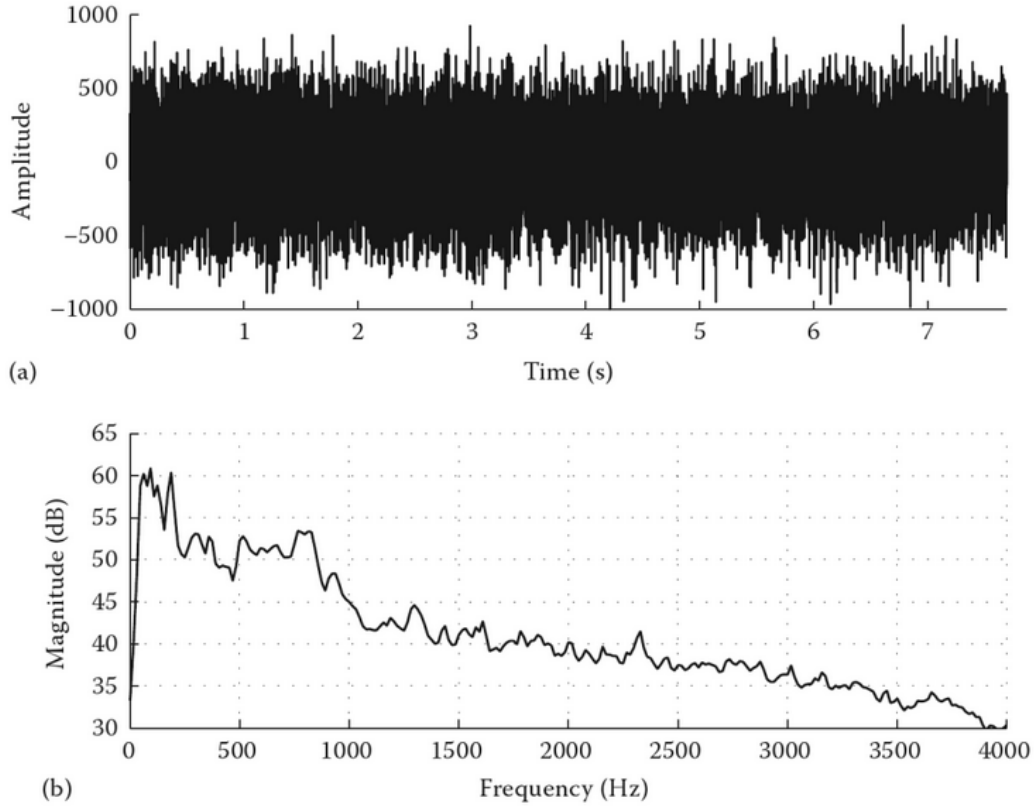


Figure 2.1 – Top panel shows noise from a car, and the bottom panel shows its long-term average spectrum. [16, p. 3].

from PCs, or nonstationary, such as restaurant noise, i.e., multiple people speaking in the background. Figure 2.1 and 2.2 display time waveform examples of a car engine and restaurant noise respectively, along with its corresponding long-term average spectra. The difference between stationary (car engine noise) and non-stationary (restaurant noise) is clearer in the frequency domain than in the time domain representation. While for the car noise most of the energy is concentrated in the low frequencies, restaurant noise is more broadband as it occupies a wider frequency range.

Sound Pressure Level (SPL) is presented as a common metric to measure the relative sound pressure in reference to 20 µPa, which corresponds to the barely audible pressure level for an average listener. Thanks to SPL, a compreshensive analysis and measurement of speech and noise levels in real-world evironments can be done. In the same manner as the SNR, it is expressed in decibels(dB):

$$A_{SPL} = 20log_{10}\left(\frac{P}{P_{ref}}\right). \tag{2.1}$$

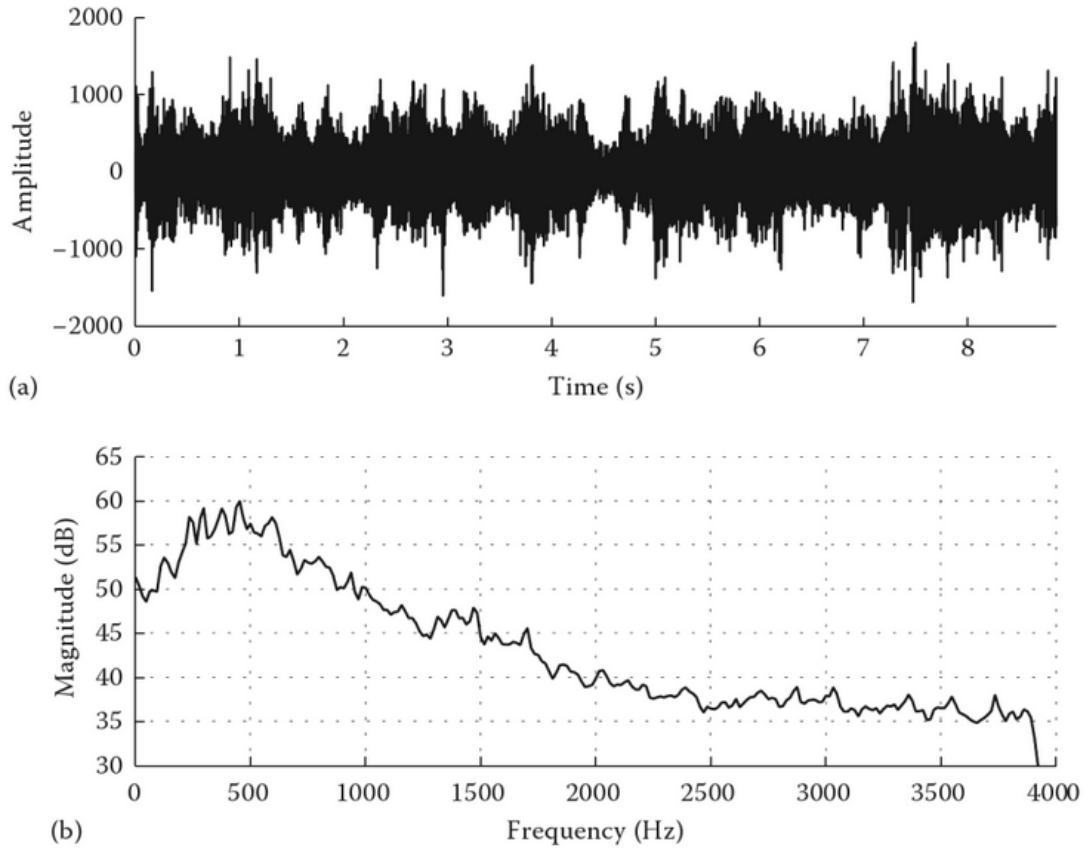


Figure 2.2 – Top pannel shows noise from a restaurant, and the bottom panel shows its long-term average spectrum. [16, p. 4].

2.1.2 Speech Signal

In this section, we will describe the wanted part of the mixture of sounds captured from the microphone, i.e., a signal. In the context of this project by signal, we will mean human speech. Despite the fact that in the signal processing domain human speech is a complex signal, it can be characterized in the following segments [21]:

- *Voiced segments* are characterized by their fundamental frequency, called "pitch" and its harmonics. For instance, for male voices, the pitch frequencies vary from 50 to 250 Hz, while for female voices it ranges between 120 to 500 Hz.

- *Unvoiced segments* might take place simultaneously with voiced segment. However, those segments are noise-like, usually characterized by the shape of their spectral envelope.

- *Silence segments* are used to separate words and phonemes and are an integral part of human speech.

2.2 Microphone arrays

Microphone arrays take advantage of multipath propagation, in which multipath components are added coherently in order to increase the effective signal-to-noise (SNR) ratio. This concept is inspired in antenna arrays for directional radio transmission and receiving applications, which have been developed since World War I. Initially, sensor arrays were extensively developed as a tool for radar-based tracking of objects [19]. Therefore, the initial steps were based on established working algorithms from antenna arrays. However, with those first attempts, non-negligible differences between antenna and microphone arrays became visible.

Comparing with electromagnetic waves, sound is an extremely wideband signal, ranging from 20 Hz to 20 kHz. In addition, construction constraints limit the size of the microphone array - for instance, the length of the upper bezel of the laptop screen where we want to integrate the microphone array. Moreover, the performance of microphone arrays can be limited by microphone's self-noise described in Section 2.1.1. The microphone parameters manufacturing variations are higher than the ones of a passive antenna. Computing power is another limiting factor for microphones array, which constrains the amount of microphones that can be placed on the array, especially for real-time applications.

In despite of all the challenges, microphone arrays have been extensively developed for applications such as room shape recognition [14] and direction of arrival (DoA) [15] detection. Moreover, microphone array architectures can be a suitable solution for speech enhancement and noise suppression applications. Till the moment, the largest planar array developed is the LOUD array at MIT [10] composed of 1024 microphone transducers. However, we believe placing fewer numbers of microphones in a 3D arrangement can be more efficient localizing sound sources. [12] starts combining microphone arrays with FPGA architectures, showing promising results for real-time application.

2.2.1 Types of microphone arrays

Linear microphone Arrays

In this configuration microphones are positioned in one line, therefore the work area of these arrays is a half plane. It cannot be distinguished sound waves arriving under the same body angle to the microphone array axis, which forms what is known as the "cone of uncertainty". This occurs due to the fact that under these circumstances the sound wave reaches the microphones in the same order and with the same delay. The work area is usually in the range of $\pm 45^{o}$ from the line perpendicular to the microphone array axis. These arrays can change the listening direction to capture the voices of several people, or to track the movements of one of them. The number of microphones varies from one to eight, but it is typically two or four.

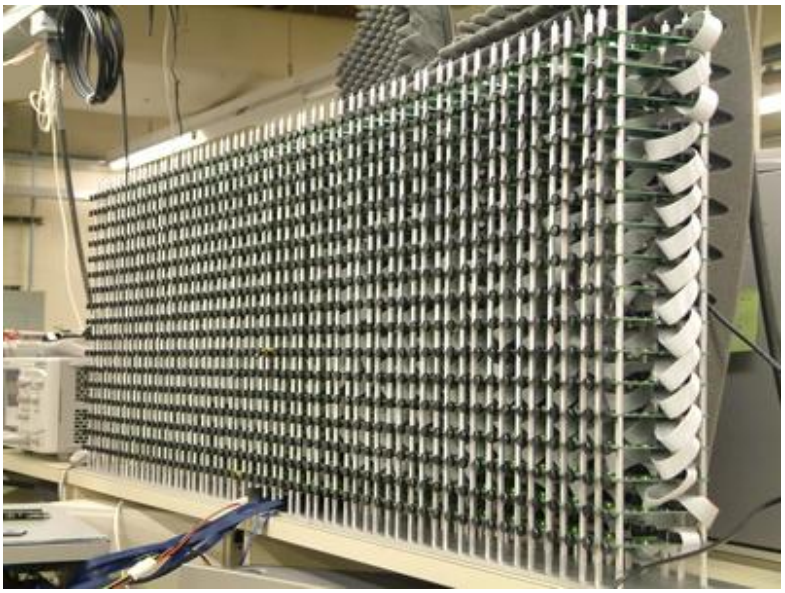


Figure 2.3 – LOUD 1020-microphone array developed at MIT [10]

Figure 2.4 – Linear microphone array [21, p. 171]

Circular microphone arrays

The microphones are positioned in a circle and they work mainly in one plane. They can change listening direction slightly above the plane of the circle. For those reasons, they are typically placed in the center of conference room tables.

Planar microphone arrays

Those microphones are positioned in one plane. The desired sound sources are usually towards the direction perpendicular to the microphone array plane. Those arrays can be mounted on the wall or on the ceiling of a conference room and can capture sound sources in a half sphere. [10] enters in this category of microphone arrays.

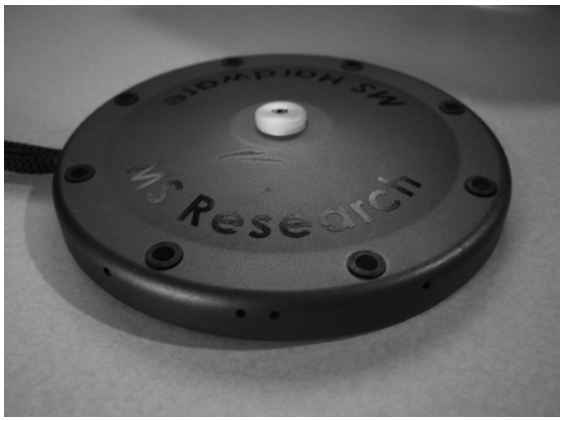


Figure 2.5 – Circular microphone array[21, p. 173]

Volumetric (3D) Microphone Arrays

The Pyramic microphone array developed through this project is based on this type of microphone distribution. Three-dimensional microphone arrays can capture sound sources from any direction in a 3D space; they do not have areas of confusion. The minimum number of microphones is four, not lying in one plan, and under the condition of placing them just hanging in the air.

2.3 Sound Capture model

2.3.1 Coordinate system

The Pyramic array designed through this project is a three-dimensional microphone array, thus we will define a 3-D coordinate system. The position of each point *c* can be represented by three coordinates in cartesian, *c=(x,y,z)*, or polar, $c = (\rho , \phi , \theta)$ coordinates.

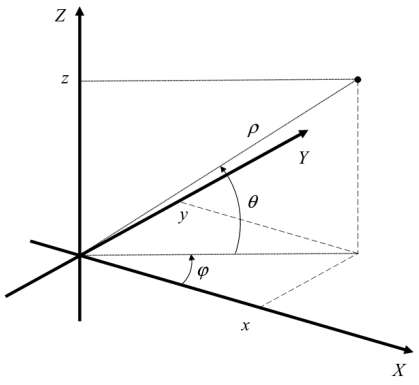


Figure 2.6 – 3D Coordinate System [21, p. 175]

2.3.2 Far-field Model

The vector $\mathbf{p} = (p_m, m = 0, 1, ..., M-1)$ denotes the positions of the M microphones in the array, where $p_m = (x_m, y_m, z_m)$ [21]. For a sound source at location c under non-reverberant conditions, the captured signal from each microphone is

$$X_m(f, p_m) = D_m(f, c)S(f) + N(f), \tag{2.2}$$

where the term on the right-hand side represents the phase rotation and the decay due to the distance to the microphone $\| c - p_m \|$ and v is the speed of sound. Through this project we will assume that the sources are in the far field, i.e., far enough away that the sound arrives approximately as a plane wave front. This assumption is reasonable for distances 5-times the size of the microphone array, where

$$\frac{1}{\|c - p_i\|} \approx \frac{1}{\|c - p_i\|} \forall i, j \tag{2.3}$$

Then the sound source can be described with direction ϕ and distance ρ from the center of the coordinate system. In two dimensions the captured sound can be expressed with the following expression:

$$D_m(f, c) = \frac{1}{\rho} e^{-j2\pi f \frac{\|p_m\| \cos((p_m) - \phi)}{v}} A_m(f) U_m(f, c), \tag{2.4}$$

where $U_m(f)$ represents the microphone directional response, $A_m(f)$ the frequency response of the system preamplifier/ADC system and c is the speed of sound equal to 343 m/s.

2.3.3 Spatial Aliasing and Ambiguity

If we have equal signal delay in two microphones within the same microphone array for at least two different directions spatial aliasing may occur. Imagine we placed symmetrically two microphones with a distance between them $d = \|p\|$ as in Figure 2.7. For this geometry, looking to 2.3 (ignoring the terms $U_m(f)$ and $A_m(f)$) it can be seen the signals coming from ϕ_1 will reach the microphones with the same delay, leading to:

$$\pi \frac{d}{\lambda} sin(\phi_1)\pi = \frac{d}{\lambda} sin(\phi_2) \pm 2\pi n, \quad \phi_1 \neq \phi_2 \tag{2.5}$$

As -1 ≤ sin(ϕ)≤ 1 we can have more than one solution when $d > \lambda/2$. This means that, to prevent spatial aliasing and direction ambiguity, we have to distance the microphones closer than half a wavelength of the highest frequency in the work band. Consequently, for a distance d between microphones, the maximum frequency we can sample without spatial aliasing can calculated from the following expression:

$$f_{max} = \frac{1}{\tau_{max}} = \frac{v}{d}, \tag{2.6}$$

Otherwise, there will also be other directions equal to ϕ_1 and $\pi - \phi_1$ where the delayed microphone signals are coherent, i.e, there will be beams at non-desired directions.

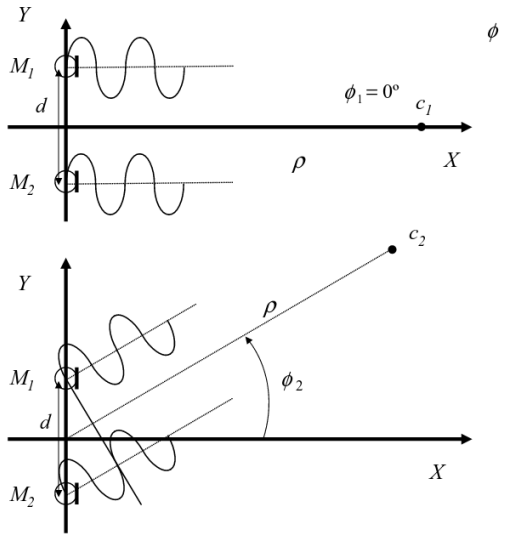


Figure 2.7 – Spatial aliasing for two-element microphone array [21, p. 179]

2.4 Summary

In this chapter, we distinguished between the wanted and unwanted parts of a signal. We introduced important properties of noise signals and how they can degrade speech signals by reverberation or cross-talking. To address this problem, microphone arrays have been introduced, which can perform algorithms to detect the direction of arrival of the wanted signals.

3 Pyramic Microphone array

3.1 Introduction

This chapter presents the core of the project: The Pyramic microphone array, which is a custom hardware for real-time audio processing developed at the Laboratory of Processors Architecture (LAP) in EPFL. This array is formed by 48 microphone channels and is connected to an FPGA device which performs real-time Digital Signal Processing (DSP) algorithms.

At the beginning of the chapter we will present a general overview of a DSP system chain. After that, the Pyramic array will be introduced, presenting its main components and their functionalities in detail.

3.2 Digital Signal Processing

Digital Signal Processing (DSP) is a mature technology used to transform or manipulate analog or digital signals [17]. Nowadays, fundamental operations such as sampling, filtering or Fourier Transform can be easily implemented as DSP algorithms. Discrete-time processing techniques have shifted the importance of traditional analog signal processing systems in many applications, ranging from data communication systems to biomedical signal processing and robotics. DSP techniques yield to more flexible, low-power and low-cost designs. Moreover, DSP is less sensitive to noise when comparing with equivalent analog signal processing algorithms.

Figure 3.1 boils down a general application used to implement an analog system by means of a digital signal processing (DSP) system. The recorded analog signal can be any imaginable physical measurement like a temperature value or a variation in air pressure (i.e. an acoustic signal). A sensor records the desired analog signal *x(t)*, which is fed through an analog anti-aliasing filter whose stopband usually starts at half the frequency (in order to fill the Nyquist criteria) of the sampling frequency f_s. The sampling frequency is given by the Analog-to-Digital Converter (ADC), which usually implements a sample-and-hold circuit followed by a quantizer and encoder modules. The ADC transforms the signal from the analog (*x(t)*) to the digital

domain(*x[k]*) by a process called *sampling* or *discretization*. The magic of sampling allows us to represent signals in the digital domain, i.e., a numerical value of the signal based on a binary representation. Finally, a discrete signal *x[k]* is sent to a host DSP system where digital signal processing techniques are applied. A Digital-to-Analog-Converter (DAC) at the output of the DSP block allows recovering the processed data into the analog domain again. This architecture served as inspiration to design the DSP platform developed through this project, the Pyramic array.

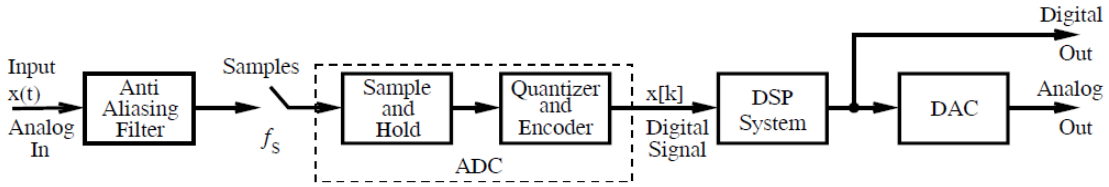


Figure 3.1 – A general DSP application [17, p. 2]

3.3 Pyramic Array

3.3.1 Overview

The Pyramic array is a custom hardware developed at the Laboratory of Processors Architecture (LAP) in EPFL. The overall architecture is an embedded system specifically designed for multi-channel acquisition and real-time processing of audio signals. The Pyramic array is composed of 48 MEMS microphones distributed on the edges of a tetrahedron. It is a modular system based on six linear arrays with eight microphone channels each. Figure 3.2 shows a perspective of the Pyramic array. The inverted pyramid geometry served as an inspiration to address the system as the 'Pyramic array'. This geometry allows detecting sound sources in both azimuth and zenith angles, leading to a 3D representation of the surrounding sound field.

The six microphone arrays are connected in pairs by daisy-chain configuration. Each array has an ADC which samples and transfers the data through a parallel Master SPI interface programmed in an FPGA device. FPGAs are reprogrammable chips which allow parallelization of algorithms, presenting very interesting for real-time applications. The chosen device is Altera's DE1-SoC board (Chapter 4). This board contains a single chip featuring both an FPGA and a Hard Processor System (HPS), which contains an ARM Cortex-A9 processor. The processor system runs on a Linux OS.

The recorded data is stored locally in an off-chip DDR3 memory and can be sent through the network to a host computer, which can access the board by Ethernet and UART connections. The HPS part loads the FPGA design when the system is powered on. The HPS reads all the image files and devices trees from a *fat32* partition in a microSD plugged in the DE1-SoC board. The microphones data is stored as *.wav* files in the microSD memory as well.

Summarizing, Pyramic is a flexible and powerful device. On the one side, it is flexible due to

its modularity, which allows the designer to test algorithms upon different conditions such as different microphone number or geometries. On the other side, it is powerful because the FPGA allows performing computationally expensive algorithms with minimum latency, which presents very interesting when dealing with the huge amount of information provided by an array of 48 microphones.

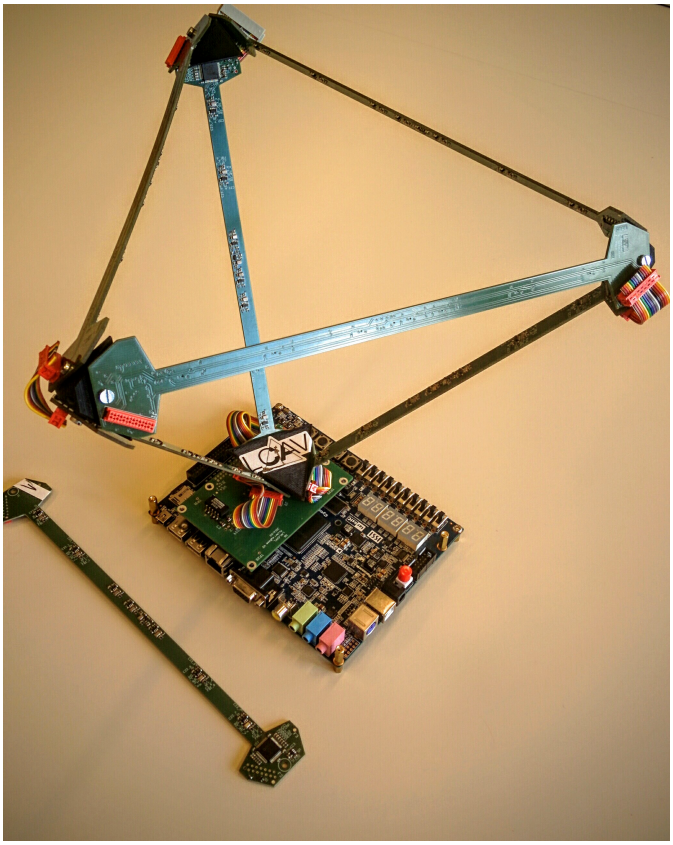


Figure 3.2 – Pyramic microphone array developed at LAP, EPFL

3.3.2 Pyramic array layout

The Pyramic array of Figure 3.2 is composed of six PCBs connected in pairs by daisy-chain connection. Figure 3.3 displays one PCB array, which main components are listed below:

- Eight INMP540 MEMS microphone transducers [13].
- Eight OPA170 operational amplifiers [25].
- One TPS780 power converter LDO from 5V to 3,3V [24].
- One AD7606 converter to sample the eight microphone signals [4].
- One NC7SZ32 OR gate for the daisy-chain connection of the BUSY signals [11].

Appendix A summarizes the main subsystems and schematic of each PCB array. Besides that, it includes an explanation of the power system and some design configuration regarding the

Figure 3.3 – 3-D representation of a single array PCB, bearing eight microphone channels.

components of the array. The microphones distribution in each board is displayed in Figure 3.4. The distance between microphones has been calculated in order to attain the highest frequency resolution and avoid spatial aliasing (Section 2.3.3). On the one hand, the shortest distance between two microphones is 8 mm, according with Equation 2.6 the f_{max} recorded without spatial ambiguity is equal to 42,5 KHz. On the other hand, the maximum distance is 200 mm, corresponding to a f_{max} of 1,7 KHz. Therefore, with this logarithmic microphone distribution the system is able to span a wide frequency range while maintaining a reasonable size for the microphone array design.

Figure 3.4 – Eight mics placement in one Pyramic PCB array, where distances are in mm

3.3.3 INMP504 microphone

The transducer chosen for sensing the acoustic analog inputs is the INMP540 microphone. A transducer transforms one form of energy to another. In our implementation, we are transforming mechanical energy (a longitudinal wave which modifies air pressure) into electrical

energy (a voltage value).

Those microphones belong to a new category of low-power, miniature MEMS microphones, which reduced size allows integrating them massively in a small area, suitable for many channel audio processing applications. Its omnidirectional behavior ensures uniform frequency response in all directions. Figure 3.5 displays the frequency response of the INMP504 transducer, which possesses characteristics of a high-pass filter with a cut-off frequency around 100 Hz. Note that the microphone bears a non-homogeneous frequency response between 2 kHz and 11 kHz. This high-frequency peak can be compensated in further post-processing. For an extensive information about the INMP5404 characteristics refer to its Datasheet [13].

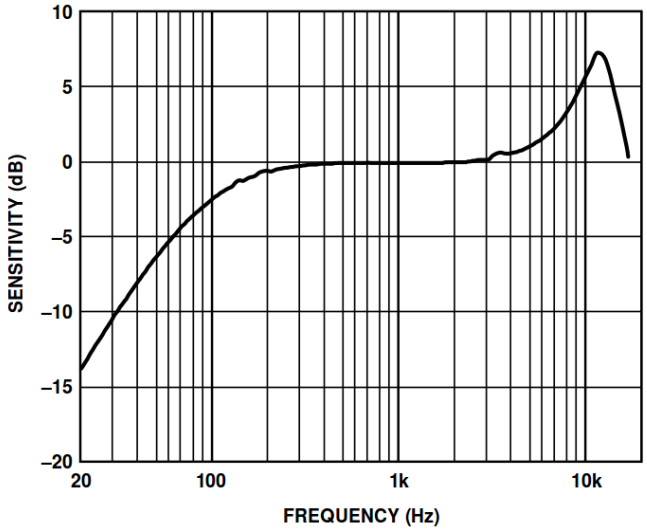


Figure 3.5 – INMP5404 frequency response [13]

3.3.4 Signal conditioning

The anti-aliasing filter mentioned in Section 3.2 is in charge of preparing the acquired electrical signal for the ADC at the next stage. It is composed of the following elements: a coupling capacitor to remove DC offset from the input, a non-inverting operational amplifier (OPA170) circuit to adapt the signal to the resolution of the ADC and a ferrite bead to reduce Power Supply Rejection Ratio (PSRR) from the microphone supply of 3,3 V. The circuit schematic and the calculations of its main parameters can be found in Appendix A.1.

3.3.5 AD7606 Analog to Digital Converter

Usually DSP algorithms are limited to handle a finite amount of data, otherwise, a computing system able to perform an infinite number of calculations would be necessary. Unfortunately, nowadays even most advanced computers need numerical representation of continuous values in order to perform meaningful operations. To address those problems, an Analog-to-

Table 3.1 – Oversample bit decoding [4].

OS[2..0]	SNR 10 V Range (dB)	3 dB BW 10 V Range (kHz)	Maximum Throughput CONVST Frequency (kHz)
000	90	22	200
001	91.292	22	100
010	93.6	18.5	50
011	95	11.9	25
100	96	6	12.5
101	96.7	3	6.25
110	97	1.5	3.125
111	Invalid		

Digital Converter (ADC) is presented as a device capable of converting an analog representation of a signal into a discrete value. It samples an analog input, holds its value for a specific amount of time (called the sampling period T_s) and generates a digital representation of the data. This value can be sent to a digital system such as a processor, a memory device or an FPGA.

ADCs are core components in any digital application. For our implementation AD7606 device from Analog Devices was chosen. AD7606 is a 16-bit, Successive Approximation ADC component which can receive up to eight multiplexed channel inputs and allows either parallel or serial mode output reading. This allows to convert up to eight microphone signals from each array in parallel. Moreover, an intermediate digital filter allows to vary f_s from 3,125 kHz to 200 kHz. Figure 3.6 presents the block diagram of the system and presents the pin connections of the ADC. For instance, depending on the values of **OS[0..2]** pins the maximum sampling frequency and SNR can be controlled as represented in Table 3.1. In our design **OS[2..0] = 010** was chosen, which corresponds to a maximum throughput of 20 μs (f_s = 50 kHz) and SNR of 93,6 dB. For a broader description of the characteristics of the AD7606 refer to the Datasheet provided by Analog Devices [4]. Furthermore, Appendix A.2 summarizes the connections diagram designed for the AD7606 in the Pyramic system. For this application, the ADC was configured with the following characteristics:

- 8 analog parallel input channels.
- Serial reading through D_{outA}.
- 16 bits resolution.
- 48 kHz sampling frequency (f_s).
- $\pm$10 V bipolar output (LSB = 305μm).
- Signal to Noise Ratio (SNR) of 93,6 dB.

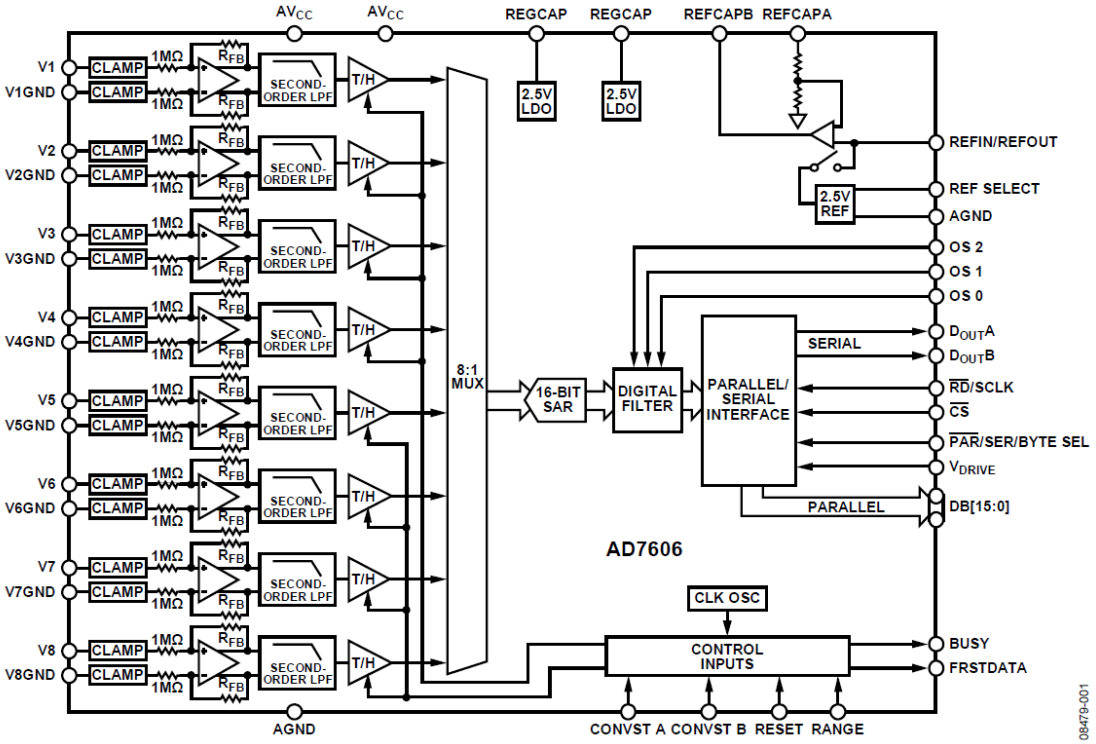


Figure 3.6 – AD7606 Block Diagram by Analog Devices [4].

Signed-Magnitude (SM)

The output data from the ADC is 16-bits mono signed-magnitude audio data representation. Signed-Magnitude (SM) represents an integer as a N-bit signal where the sign and the magnitude are represented separately. The first bit x_{N-1} (i.e., the MSB) represents the sign and the remaining N-1 bits the magnitude. The representation becomes:

$$X = \begin{cases} \sum_{n=0}^{N-2} x_n 2^n & x_{N-1} \geq 0 \\ -\sum_{n=0}^{N-2} x_n 2^n & x_{N-1} < 0 \end{cases}$$

The range of SM numbers is $[-(2^{N-1}-1), 2^{N-1}-1]$. The advantages of this representation is a simplified prevention of overflows, but the disadvantage is that addition must be split depending on which operand is larger.

3.3.6 Timing requirements

Once the audio data is sampled and quantized it is necessary to stablish a reliable and fast communication protocol between the ADC and a processing system. In our application, the

ADC should indicate to the receiver that every 20,8 ms a new sample of 16-bits will be sent. For those purposes, the ADC provides control signals and timing requirements which allow the different systems to acknowledge digital data transmission from the ADC.

Figure 3.7 boils down the general timing requirements during a conversion cycle of the ADC. First, the system is initialized by sending a *RESET* high pulse. Then, a low pulse of *CONVST* must be applied to start the conversion. Note that since the data can be read from two output lines, D_{outA} and D_{outB}, two conversion lines are available, $CONVST_A$ and $CONVST_B$. In our design, we are just reading data from one line, D_{outA}, so we can apply the same signal to $CONVST_A$ and $CONVST_B$. After a *CONVST* low pulse has been sent, the ADC will assert the *BUSY* signal, which falling edge will mark the end of the conversion process. The time *BUSY* remains high is called the conversion time *(t_{conv})*, while the cycle time ($t_{cycle} = T_s$) is the time between rising edges of two consecutive *CONVST* low pulses. When a conversion is finished, the sampled data is available in the ADC output and can be can read through the D_{out} lines in parallel or serial mode when CS_n is low. Serial implementation was chosen in our design. As it can be seen in Figure 3.7, it is possible to read the previous samples during a new conversion (i.e., while *BUSY* is high), which is useful to adjust the f_s and reduce throughput if necessary. To sum up, after *BUSY* signal is deasserted and while CS_n is low the acquisition platform can read the data through D_{outA} in serial mode.

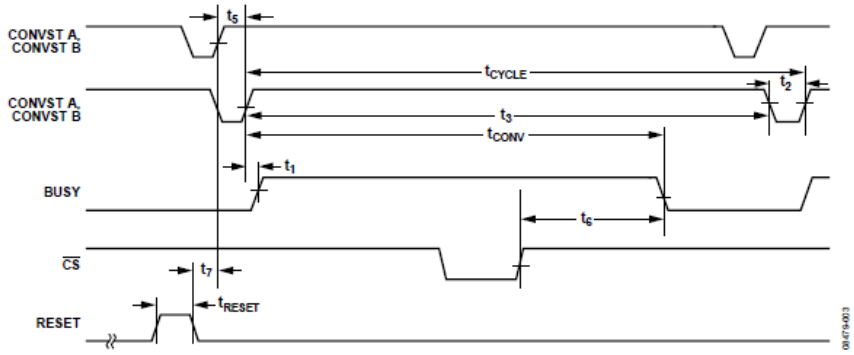


Figure 3.7 – AD7606 conversion time diagram [4]

While CS_n is low, the ADC sends a new bit of information every **falling edge** of *SCLK*. Figure 3.8 displays the process, where the transmission starts from the MSB (the sign) of the first ADC input channel *V1*. Note that our design uses just D_{outA}, which will transmit the eight channels from *V1* to *V8*. From this information, the minimum low pulse width of CS_n can be calculated as follows:

$$t_{read} = \frac{channels * resolution}{f_{sclk}} = \frac{8 * 16}{f_{sclk}}, \tag{3.2}$$

where f_{sclk} is *SCLK* frequency, $channels$ the number of analog input channels for one ADC and $resolution$ the number of bits of an audio sample. F_{sclk} determines the reading speed and is provided by the transmission system. However, timing requirements regarding the

AD7606 Datasheet [4] must be fulfilled. For instance, [4] requests t_{reset} high pulse width from Figure 3.7 to last more than 50 ns in order to be a valid signal. Regarding f_{sclk}, [4] displays a maximum serial reading frequency of 15 MHz when V_{drive} = 3,3V. According with Equation 3.2, if $f_{sclkMAX}$ = 15 MHz the minimum t_{read} necessary for reading one conversion will be equal to 9μs. This value represents the minimum width of a CS_n low pulse.

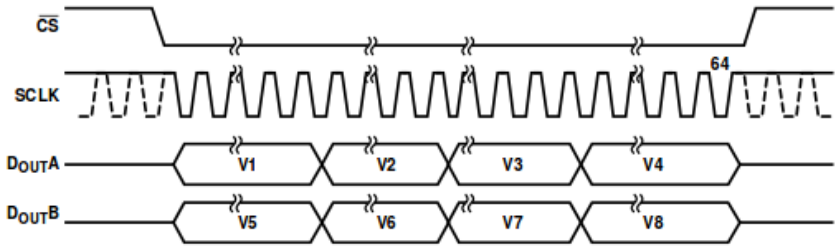


Figure 3.8 – AD7606 conversion tiime diagram [4]

3.4 Summary

In this chapter, we presented an overview of the Pyramic array, its different components and the design considerations behind this new technology. The first steps of the DSP chain presented in Figure 3.1 have been introduced. Particularly, we explained the characteristics of the microphone transducers and the ADC converters implemented in the array. Nexct chapter will present the FPGA based processing system chosen for this platform.

4 FPGA Technologies: Terasic DE1-SoC Board

4.1 Introduction

FPGAs design still a rapidly evolving field, each year main developers such as Altera and Xilinx are building faster and more powerful chips containing reconfigurable hardware. Those achievements motivate the use of FPGA for Digital Signal Processing (DSP) applications and to substitute traditional processors in DSP systems as the one presented in Figure 3.1. As a consequence, in order to understand why FPGAs are preferred over more traditional solutions, this chapter will introduce FPGAs technologies and will present the device upon this project was built on: the DE1-SoC Board.

4.2 FPGA Technology

4.2.1 FPGA Benchmark

Field Programmable Gate Arrays (FPGAs) are a member of a class of devices called field-programmable logic (FPL). FPLS are defined as programmable devices containing repeated fields of arrays of small logic blocks and elements. FPGAs, similarly to ASIC (Application Specific Integrated Circuit), allow application specific design. However, building a programmable gate array solution allows full control over the design implementation without the lag and need for any physical IC fabrication facility. Several advantages offered by FPGAs over ASIC technologies are [17]:

- Die size and weight reduction.
- Parallelization leads to higher throughput.
- Better security against unauthorized copies.
- Reduced device and inventory cost.
- Reduced board test costs.

On the other hand, in the field of DSP programmable digital signal processors (PDSPs)

have enjoyed tremendous success during the last decades. Their architecture is based on a reduced instruction set computer (RISC) paradigm that allows performing fast multiply-and-accumulate (MAC) operations, which are the basic block of signal processing algorithms. By using a multistate pipeline architecture, PDSPs can achieve MAC rates limited only by the speed of the array multiplier. However, compared with FPGAs, which allow hardware parallelization, FPGAs can perform more efficient for high-bandwidth signal processing applications such as multimedia information like audio [17].

Summarizing, FPGAs can still be comparable to ASICs in terms of performance while they do not require additional semiconductor manufacturing steps. The fact FPGAs allow reprogrammable hardware designs offers a lot of flexibility to the implementation. On the other hand, the main drawback regarding FPGAs design is that power consumption usually is higher than for traditional systems, especially at higher frequencies. In the end, it is expected FPGAs will dominate most front-end (sensor) applications like FIR filters or FFTs, while PDSP will dominate applications that require complex algorithms. This will lead to complex mixed implementations where the designer needs to bear specific knowledge about software and hardware design in order to build powerful real-time embedded systems.

4.2.2 Intelectual Property (IP) Cores design

Although FPGAs are known for their capability to support rapid prototyping, the rapidly increasing complexity of FPGAs are forcing a methodology shift towards the use of intellectual property (IP) macrocells or mega-core cells. Those cells provide the designer with specific functions, such as FIFOs, UARTs, oscillators, SPI or FIR filters. Therefore, the designer only needs to specify certain attributes of the system in order to match the design requirements and the synthesizer will generate a hardware description code or schematic for the resulting solution. Therefore, the design cycle is reduced while providing a good utilization of the device resources. The main type of IP cores can be divided into three categories:

1. **Soft Core**: It is a behavioral description of a component that needs to be synthesized with FPGA vendor tools such as *Quartus Prime Programmer*. Modifications in the hardware description language (HDL) file provided in VHDL or Verilog can be made. Altera Cyclone V family devices implement a Nios II softcore processor, which is a processor implemented with Logic Elements (i.e., Look Up Tables and D-type Flip Flops) in an FPGA.
2. **Parameterized Core**: It is a structural description of a component. While the parameters of the design can be changed before synthesis, the HDL is usually not available. Altera provides most of the cores in this format, which allows certain flexibility but are not compatible with other FPGA vendors. For instance, the FIR filters designed in Section 5.3.4 are based on those type of IP blocks.
3. **Hard Core**: A fixed netlist core is a physical description of the IP system. When real-time constraints are required, the cores are optimized for a specific device family (such as

Cyclone V). Those cores have fixed parameters but allow integration and simulation in larger projects. Altera's Cyclone V provides a hardcore processor (HPS) along with an FPGA in the same die, merging simultaneous software and hardware design, which will be discussed in Chapter 6.

In the end, when building an FPGA design it is necessary to choose between design flexibility (soft core) or fast results and reliability of data (hardcore). While soft cores are more flexible and parameterized, the debug time can be longer. While hard cores are verified in silicon, can reduce development, test, and debug time but no VHDL code is available to look at. Finally, parameterized cores are a compromise between flexibility and reliability of the generated code.

4.2.3 FPGA Design

When building an FPGA system based project, first the designer needs to organize the objectives of the project as we did in Chapter 1. Later on, it is necessary to understand the timing requirements of the system as we introduced in Section 3.3.6. After that, the design is completed by drawing block diagrams as the ones that will be introduced in Section 5.2.3. Once the design is completed we can start building and simulating the sytem.

For programming FPGAs two main HDL languages are available: **VHDL** and Verilog. Nowadays, there exist new tools which perform automatic translation from C to HDL. However, through this project, VHDL has been used in Altera-supplied software. However, *Quartus Programmer* generates some IP cores and files in Verilog format. *Quartus Programmer* software is a fully integrated system with VHDL and Verilog editor, synthesizer, simulator, and bitstream generator. For simulation, Modelsim has been used, which also supports Verilog and VHDL test files. On the other hand, Saleae Logic Analyzer for time testing and timing verification has been employed. For more details about the required software refer to Appendix B.

4.3 Terasic DE1-SoC Board

Terasic DE1-SoC board [22](Figure 4.1), from the Altera Cyclone V family, has been implemented as the processing system for the Pyramic array project. An overview of the system is made in the following sections.

4.3.1 Specifications

The main features of the DE1-SoC board implemented for the design of the PyraMic system are listed below. For a complete overview of all the available modules refer to [23].

FPGA Device

- Cyclone V SoC **5CSEMA5F31C6** Device

- Dual-core ARM CORTEX-A9 (HPS)
- 85K Programmable Logic Elements
- 4'450 Kbits embedded memory
- 2 Hard Memory Controllers

Memory Device

- **1 GB** (2x256Mx16) DDR SDRAM on HPS.
- **MICRO SD** Card Socket on HPS.

Communication

- 10 100 1000 Ethernet.
- USB to UART (micro USB type B connector).

Connectors

- One 40-pin Expansion Header (GPIO0) to acquire the data from the AD7606 device.

Audio

- 16/24/32-bit, line-out CODEC.

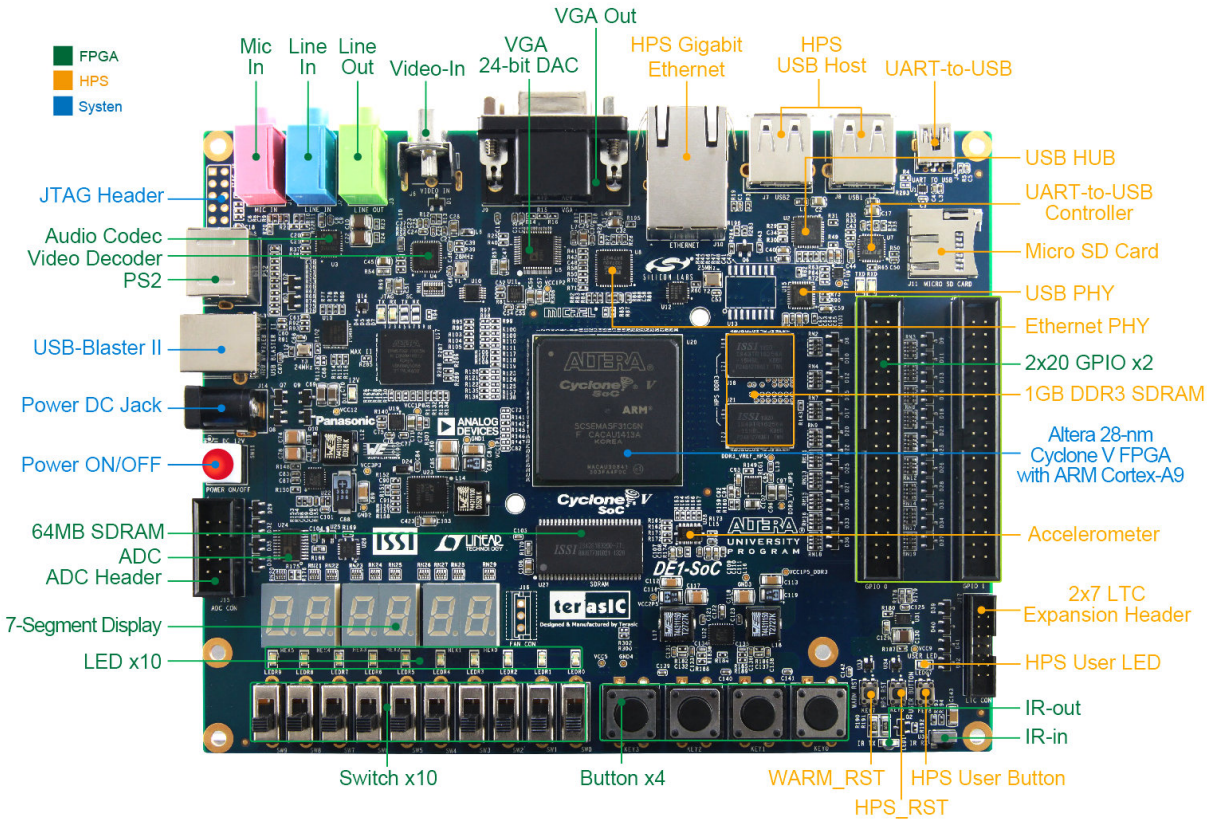


Figure 4.1 – Terasic DE1-SoC Board [22].

Switches, Buttons and Indicators

- 4 User Keys (FPGAx4).
- 10 User switches (FPGA x10).
- 11 User LEDs (FPGA x10;HPS x1).
- 2 HPS Reset Buttons.
- 2 Six 7-segment displays.

Power

- 12V DC input.

4.3.2 Cyclone V Overview

The Cyclone V device consists of a hard processor system (HPS) and an FPGA portion in a single-die system on chip (SoC). The HPS contains a microprocessor unit (MPU) subsystem with dual ARM Cortex-A9 MPCore processors, flash memory controllers, SDRAM L3 interconnect, on-chip memories, support peripherals, interface peripherals, debug capabilities, and phase-locked loops (PLLs). The dual-processor HPS supports symmetric (SMP) and asymmetric (AMP) multiprocessing.

The FPGA portion of the device contains the FPGA fabric, a control block (CB), phase-locked loops (PLLs), and depending on the device variant, high-speed serial interface (HSSI) transceivers, hard PCI Express (PCIe) controllers, and hard memory controllers (Figure 4.2).

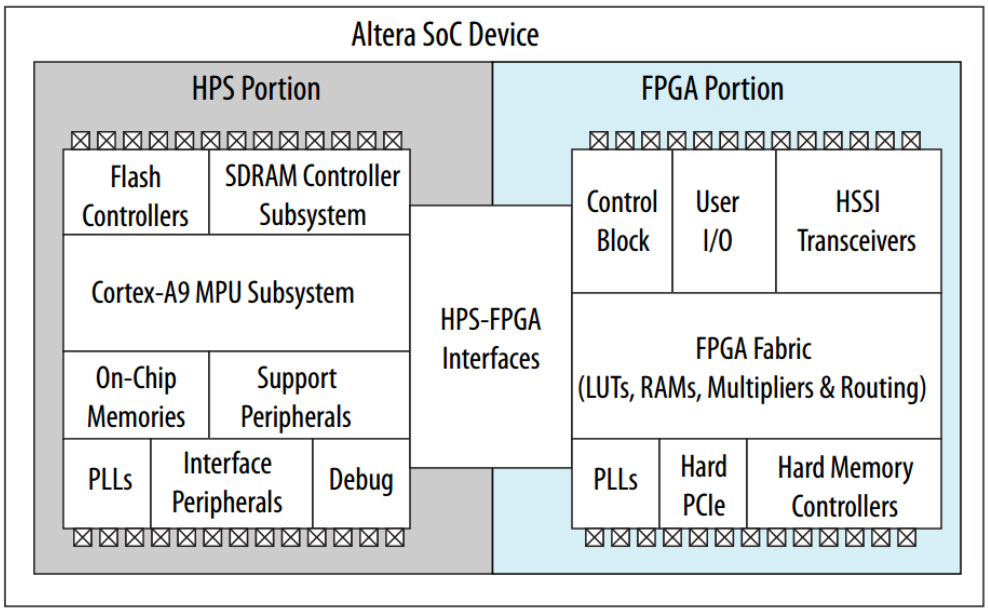


Figure 4.2 – Altera SoC Device Block Diagram [22]

The FPGA portion of the device starts when the it is released from reset state (e.g., power up). The HPS contains exclusively hard logic. The Cyclone V SoC can be used in 3 different

configurations:

- FPGA-only
- HPS-only
- **FPGA and HPS**

On the other side, the FPGA has to be configured either through

- **the HPS**
- JTAG configuration through an externally supported device such as *Quartus Primer* programmer.

The MPU subsystem can boot from

- **flash devices connected to the HPS pins**
- from memory available on the FPGA portion of the device after the FPGA is configured by an external source

4.4 Pyramic array overall design

Pyramic array interacts with DE1-SoC board using both the FPGA and HPS part. An overall view of the system architecture is displayed in Figure 4.3. Note that the system clock is 50 MHz and the reset is active low. Additionally, the FPGA part features a Nios II softcore processor [6] which is not used for this project. Chapter 5 focuses exclusively on the design of custom components and IP cores implemented in the FPGA part, while Chapter 6 explains in more detail the HPS unit, how it is configured for the project and how it interacts with the FPGA part.

Before explaining the custom components designed for the FPGA part of the microphones array, it is necessary to understand how to interconnect different components in the FPGA part of Altera devices. For those purposes, the following section will introduce the Avalon bus interface.

4.5 FPGA interconnections: Avalon Interface

The Avalon bus provided by Altera is an internal interface which allows connecting components inside the FPGA design [2]. Besides that, the Avalon Interface performs arbitration between different slaves and masters units, i.e., it determines which component can access the data and address buses each cycle. The Avalon interface standardizes interfaces for high-speed data streaming, reading and writing registers and memory, and controlling off-chip devices and custom components. These standard interfaces are designed into the components available in *Qsys* program, available within the *Quartus* programming tools. The main interfaces are briefly described below:

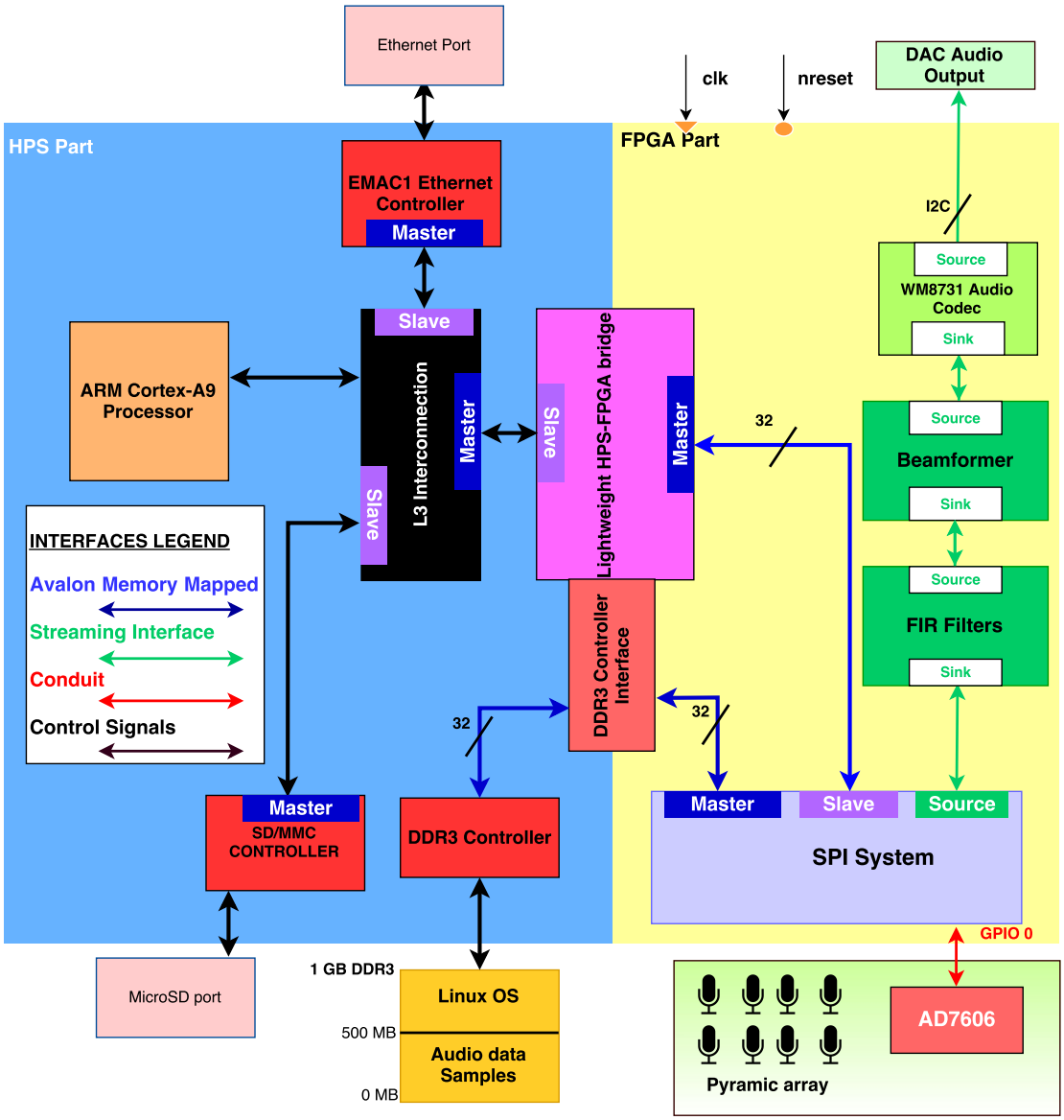


Figure 4.3 – Pyramic array embedded system overview.

- **Avalon Streaming Interface (Avalon-ST)**: An interfaces which supports unidirectional flow of data between source and sink units.
- **Avalon Memory Mapped Interface (Avalon-MM)**: An address-based read/write interface typical of master-slave connections.
- **Avalon Conduit Interface**: An interface type that acoomodates individual signals or groups of signals that do not fit into any of the other Avalon types. They can be connected inside a Qsys system or they can be connect to other modules in the design or to FPGA pins.
- **Avalon Clock Interface**: An interface which drives or receives clock. For our system a clock of 50 MHz has been used.
- **Avalon Reset Interface**: An interface that provides reset connectiviy. For our system a active low reset signal has been used.

4.5.1 Avalon Memory-Mapped Interfaces

Avalon Memory-Mapped (Avalon-MM) interfaces can be used to implement read and write interfaces for master and slave components. For instance, a Direct Memory Acces (DMA) component can access memory by implementing Memory-Mapped transfers. The main signals of the interface are described in Table 4.1. The widths of the signals are the ones chosen for the Pyramic implementation.

Design example: Burst write transfer

A burst executes multiple transfers as a unit, rather than treating every word independently. Burst may increase throughput for slave ports that achieve greater efficiency when handling multiple words at a time, such as DDR3 memories. The net effect of bursting is to lock the arbitration for the duration of the burst, defined by the Burstcount output signal.

When a *burstcount* of <*n*> is presented at the beginning of the burst, the slave must accept<n> successive units of *writedata* to complete the burst. Arbitration between the master-slave pair is locked until the burst is completed. The main advantage of this lock is that it guarantees that no other master can execute transactions on the slave until the write burst is completed.

Figure 4.4 shows a *write* burst transfer where *burstcount* = 4, i.e., the bus is locked for 4 cycles to send four consecutive *writedata* signals. The numbers in the timing diagram, mark the following transitions:

1. The master asserts *address*, *burstcount*, *write* and drives the first unit of *writedata*, indicating the first writing transfer cycle.

Table 4.1 – Avalon MM signals

Signal role	Width (bits)	Direction	Description
address	32	Master to Slave	Indicates the memory address where the data is write/read to/from.
byteenable	4	Master to Slave	Each bit corresponds to a byte in writedata and readdata. For a 32-bit slave (4-bits): - If 1111 writes full 32 bits - If 0011 writes lower 2 bytes - If 1100 writes upper 2 bytes
read	1	Master to Slave	Asserted to indicate a read transfer.
readdata	32	Slave to Master	The *readdata* driven from the slave to the master in response to a read transfer.
write	1	Master to Slave	Asserted to indicate a write transfer.
writedata	32	Master to Slave	The *writedata* driven from the master to the slave in response to a write transfer.
waitrequest	1	Slave to Master	Asserted by the slave when it is unable to responde to a read or write request. Forces the master to wait until the interconnect is ready to proceed with the tranfer. A master must take not assumption about the assertion of waitrequest. Even at the start of the transfer, the master initiates the transfer needs to wait unitl waitrequest is deasserted.
burstcount	4	Master to Slave	Used by bursting masters to indicate the number of transfers in each burst. For instance, a 4-bit burstcount signal can support a maximum burst count of 8.

2. The slave immediately asserts textitwaitrequest, indicating it is not ready to proceed with the transfer. The Master has to wait before starting the transfer.
3. When *waitrequest* is low the slave captures *addr1*, *burstcount*, and the first unit of *writedata*. On subsequent cycles of the transfer, **address** and **burstcount** are ignored.
4. The slave captures the second unit of data at the rising edge of *clk*.
5. The burst is paused while *write* deasserted.
6. The slave captures the third unit of data at the rising edge of *clk*.
7. The slave asserts *waitrequest*. Consequently, all outputs are held constant through another clock cycle.
8. The slave captures the last unit of data on this rising edge of *clk*. The slave write burst ends.

The *beginbursttransfer* signal is asserted for the first clock cycle of a burst and is deasserted on the next clock cycle. However, this signal has not been implemented in our FPGA design.

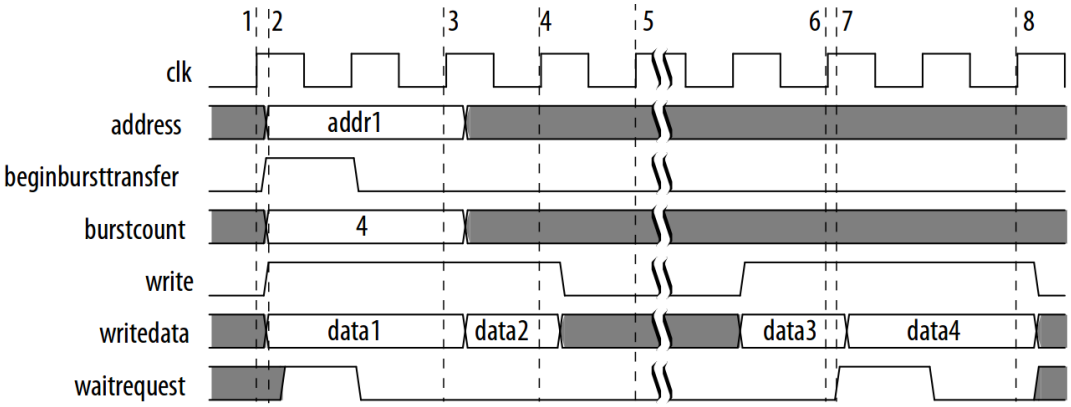


Figure 4.4 – Slave write burst of length 4 [2].

4.5.2 Avalon Streaming Interfaces

Avalon Streaming (Avalon-ST) interface drives unidirectional data with high bandwidth and low latency. For DSP applications, this interface allows to directly stream data between different modules in an FPGA design with a lot of flexibility. The interface supports from a single stream of data without knowledge of channels or packet boundaries till more complex protocol capable of burst and packet transfer with packets interleaved across multiple channels. Figure 4.5.2 represents each signal direction, while the function of each signal is described below:

- *startofpacket*: Marks the active cycle containing the start of the packet. It is only interpreted when valid is asserted.

- *endofpacket*: Marks the active cycle containing the end of the packet. It is only interpreted when valid is asserted.

- *ready*: On interfaces supporting backpressure, the sink asserts *ready* to mark the cycles where transfers may take place, i.e, if *ready* is asserted on cycle <n>, next cycle is considered a ready cycle. [1].

- *valid*: This signal qualifies data on any cycle where data is being transferred from the source to the sink. On each valid cycle the *data*

- *data*: Carries the bulk of the information being transferred from the source to the sink. The data consists of one or more symbols being transferred on every clock cycle. The *dataBitsPerSymbol* parameter defines how the data signal is divided into symbols.

- *channels*: It is driven by the source to indicate the channel to which the data belongs. In each active cycle, all of the data transferred belongs to the same channel, while the source may change to a different channel on successive active cycles. Additionally, the *maxChannel* parameter indicates the maximum channel number. If the number of channels an interface supports changes dynamically, *maxChannel* is the maximum number the interface can support.

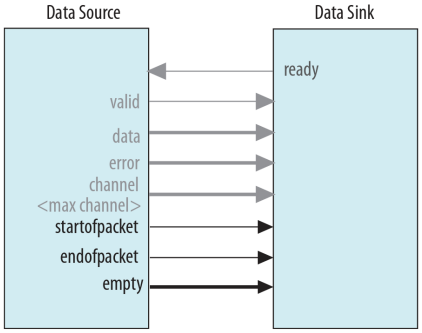


Figure 4.5 shows a 32-bit example where *dataBitsPerSymbol*=8. Data transfers occurs on cycles 1,2,4,5, and 6, when both *ready* and *valid* are asserted. Since during cycle 1, *startofpacket* is asserted the first 4 bytes of packet are transferred. During cycle 6, *endofpacket* is asserted and *valid* is high as well, which indicates that this cycle is the end of the packet data transfer. *empty* has a value of 3, which indicates that at the end of the packet 3 of 4 symbols are empty, however this signal is not implemented in our design, neither *error* signal.

4.5.3 Avalon Conduit Interfaces

Avalon Conduit interfaces group an arbitrary collection of signals, which direction can be seen as input, output or bidirectional by the FPGA device. Conduits can be used to interconnect different modules inside a Qsys system or to drive off-chip device signals, such as memory controllers or data and control signals for devices such as the Pyramic array. An important

[1] Inded, in a more general case if ready is asserted on cycle <n>, <n + *readyLatency*> should be considered as ready cycle. However, in our Pyramic implementation *readyLatency* = 0 by default

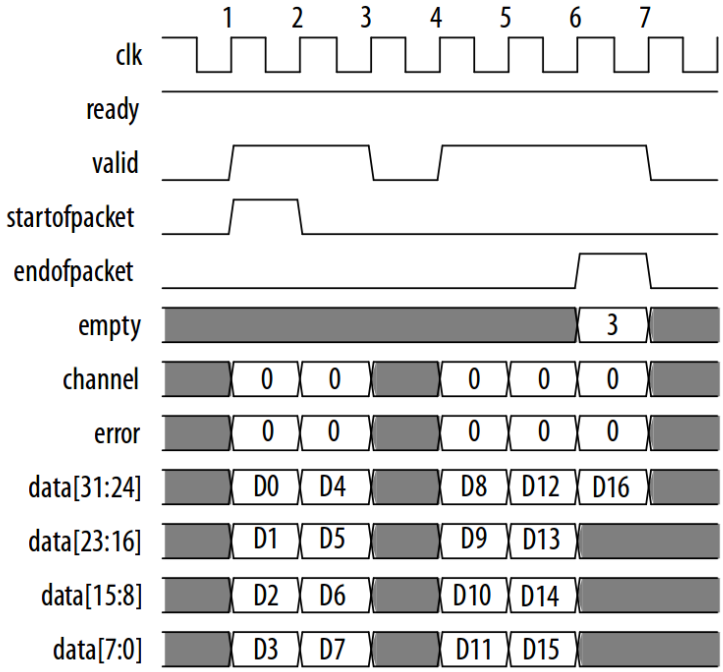


Figure 4.5 – 32 bits single channel Avalon Streaming Interface with packet transfer.

constraint of Conduit signals is that when connecting them the roles and widths must match and the directions must be opposite.

4.6 Summary

This chapter introduced FPGA technologies and compared its performance with ASIC and PDSP implementations. The IP core design philosophy is presented, while some FPGA design methodologies are described. Next, Terasic DE1-SoC from Altera Cyclone V family is presented, along with its main modules and a complete overview of the Pyramic array architecture implemented in the DE1-SoC board. Finally, the most important Avalon interfaces are explained in order to understand better the FPGA designs which will be explained in the following chapter.

5 FPGA Design

5.1 Introduction

In order to communicate with the Pyramic array, a parallel Master SPI interface has been designed in the FPGA part of the DE1-SoC board. The interface conveys the output samples from the AD7606 to an FPGA platform. A custom programmable interface (PI) in VHDL is designed for those purposes. Additionally, real-time FIR filters designed for delay-and-sum beamformer algorithms are presented as well. The output of the beamformer is converted send to an audio line out DAC located in the DE1-SoC board.

Through this chapter we will present a top-bottom approach, i.e., from an overall architecture of the system to a detailed explanation of each block design and performance.

5.2 SPI Communication Design

5.2.1 SPI review

A Serial Peripheral Interface (SPI) is a synchronous communication standard developed by Motorola, mainly used for short distances communications such as in embedded systems. It is a full-duplex interface, in which a Master unit starts a transmission/reception of information to/from a selected slaves. The main signals which compose a basic SPI interface are listed below, while Figure 5.1 represents a generic block diagram of an SPI system.

- *SCLK* Serial clock provided by the Master unit. Usually the data in transmitted/received in the falling edge of SCLK to/from the selected Slave.
- *MOSI*, Master out Slave in, data sent by the Master unit to the selected slave.
- *MISO*, Master in Slave out, data sent to the Master unit by the selected slave.
- *SS*, Selected Slave, indicates which slave have access to the Master. There is one by slave and usually, it is active low.

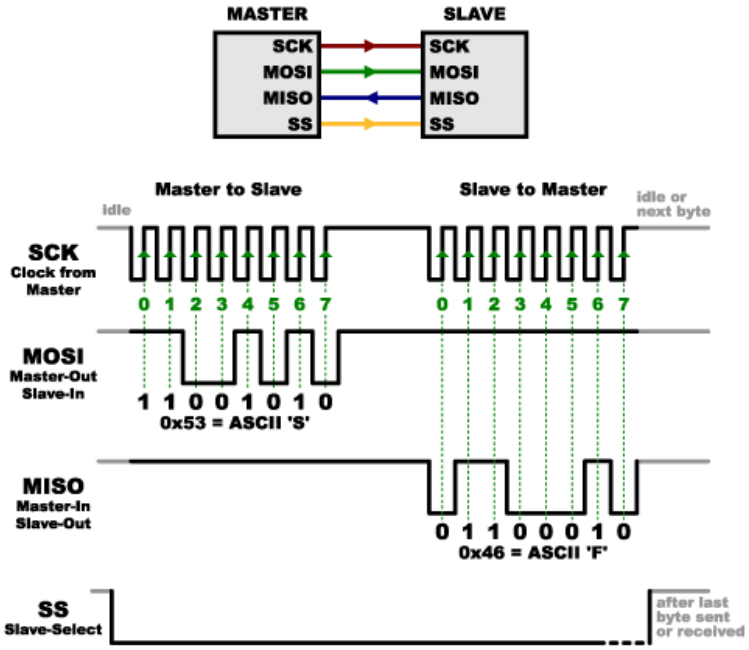


Figure 5.1 – Generic SPI communication block diagram.

5.2.2 SPI communication design

The Pyramic array communicates with the DE1-SoC device from Altera following an SPI communication protocol. The ADC sends the sampled data to the FPGA device, which receives the data and sends it to memory and to other FPGA modules for further processing. Those are the main design considerations of the developed *SPI_System* module:

- The communication should be single-master multi-slave, where the FPGA is the Master and the ADC modules connected in daisy-chain by pairs are the Slaves.
- The data transmission should be unidirectional from Slave to Master, i.e., only the *MISO* signal is transmitted.
- The microphones data should be available as Avalon Streaming Interface and Avalon MM.
- The memory distribution will be as in Figure 5.2. The data will be written in 32-bits memory addresses, each of them containing two microphone signals of 16-bits each.
- The system should allow a burst transfer to memory with a *burstcount* of 4, meaning that each burst transfer will convey eight channels from the Pyramic array.
- CS_n corresponds to *Slave-Select* signal, *SCLK* to the *SCLK* from the AD7606, while *MISO* will correspond to the serial output D_{outA}.
- All the ADC slave modules should send the information at the same time. Therefore, several *MISO* lines will be implemented to perform serial transfers in parallel.
- The data transmission should be circular when a *Start* signal is high as indicated in Figure 5.3. Moreover, two flag signals, *Buffer1* and *Buffer2*, will mark the beginning, half

and end of each circular period.

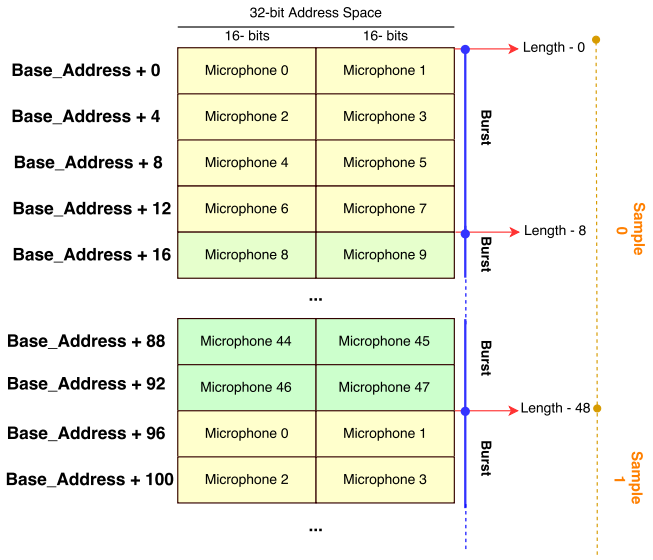


Figure 5.2 – Pyramic samples memory distribution

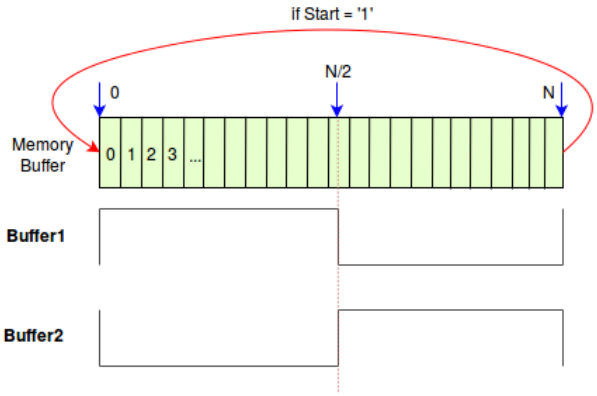


Figure 5.3 – Circular buffering of samples with Buffer1 and Buffer2 flags

5.2.3 SPI System block diagram

SPI_System is the FPGA HDL module developed through this project which interfaces a parallel SPI master communication. The module receives conduit signals from the Pyramic ADCs and sends the audio data through a DMA unit to memory. Moreover, the data is sent to a streaming interface. All the process is controlled by a slave module. Figure 5.4 displays the full block diagram of the *SPI_System*, the main components are described below:

- *SPI_Controller*: The SPI controller module is in charge of interacting directly with the AD7606 unit through conduit signals. It implements the SPI communication in order to receive the microphones data from the ADC in parallel from all the microphone arrays. Moreover, the data is stored in registers which can be read in real-time by a Direct Memory Access (DMA) unit and it is controlled by a *SPI_Slave* slave module.
- *SPI_DMA*: This module is a Direct Memory Access (DMA), which is able to write directly in the different memories of the DE1-SoC board. It receives the data from the *SPI_Controller* module and sends it to an Avalon MM SDRAM Controller located in the HPS part. It is controlled by the *SPI_Slave* module.
- *SPI_Slave*: This Programmable Interface (PI) can be accessed by the main program to control the SPI communication. It is possible to change its register values to send instructions or receive information from the system. Its main functionalities are to set the length of the acquisition, the base address and start and stop the acquisition.
- *SPI_Streaming*: It receives the data from the DMA unit as Avalon ST interface. This way, the audio data can be sent to other modules for further processing in the FPGA. In our implementation the data is driven to an FIR filters bank.

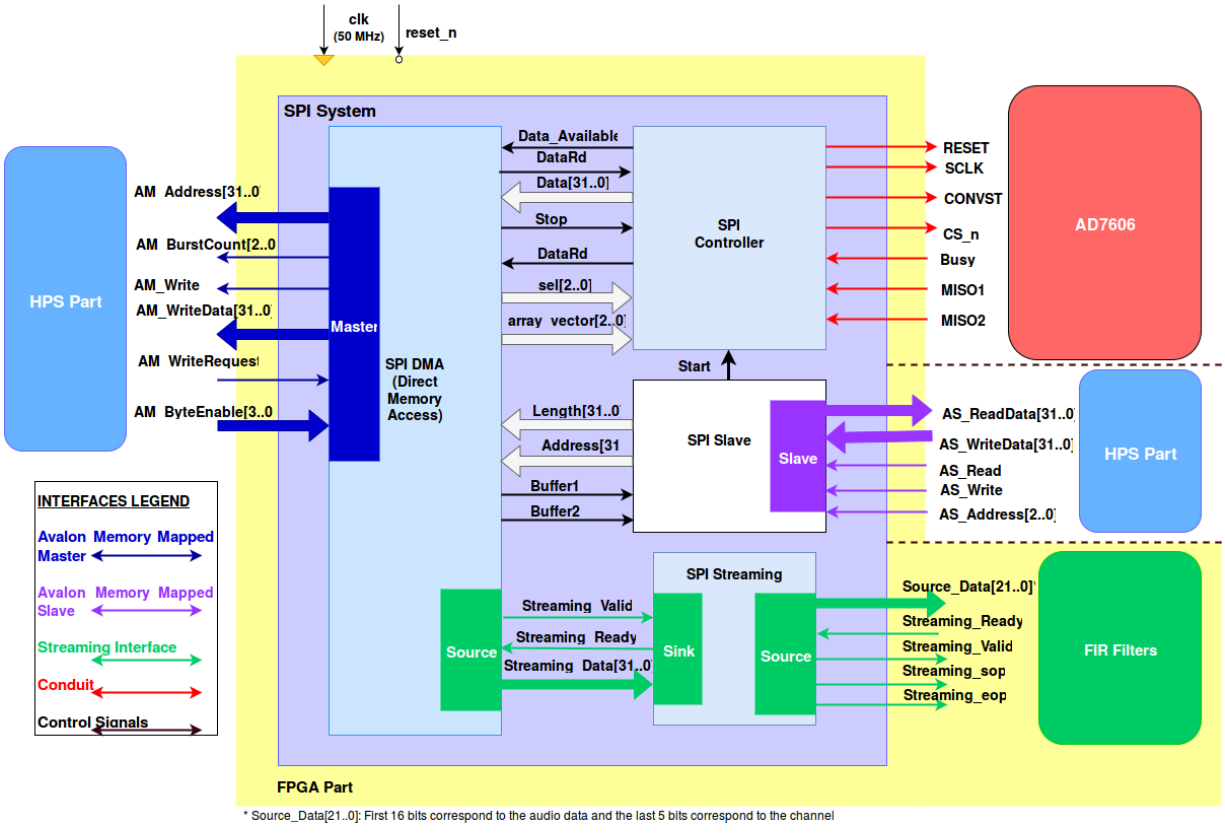


Figure 5.4 – SPI System Block Diagram.

SPI Controller module

The SPI Controller module communicates directly the FPGA with the ADCs of the Pyramic array. Figure 5.5 details the inner architecture of the system. The signals *SCLK*, *RESET* and CS_n are sent to the three ADCs according with the timing requirements of Figure 3.7 and [4]. *SCLK* frequency is 12,5 MHz, which is generated by dividing the system clock *clk* of 50 MHz by four. A *D* type Flip-Flop synchronizes the *BUSY OR* input signal with the system clock in order avoid metastability.

The FPGA is the Master, which receives data from six slaves (the ADCs) in parallel, connected by pairs with daisy-chain connection and each of them sending a different *MISO* signal. Figure 5.6 displays the waveform outputs recorded with the *Saleae Logic Analyzer*. Although just *MISO_00* signal is displayed for simplicity purposes, note that *MISO_01*,*MISO_10*, *MISO_11*, *MISO_20* and *MISO_21* are recorded in parallel and present the same behavior as *MISO_00*. There are as many MISO signals as the number of arrays we are using, by default we use six arrays, each of which bears eight microphone channels. When CS_n is low, every ***SCLK* falling edge** a bit is sampled for each microphone, starting from the MSB. An internal counter counts 16 falling edges of SCLK. Then, another counter increments the channel number and the **Serial to Parallel** register stores the sample from the current channel of the six arrays. The next sample of 16 bits from the next channel is recorded in the same way till the eight microphone channels of each board are stored.

Once in the register, the data can be read in parallel and stored in an array of 48 FIFOs called **FIFO_MIC** module. FIFO (First In First Out) modules are very recurrent in data buffering for streaming applications and interfaces design. In this project Altera SCFIFO (Single Clock FIFO) IP Core [9] has been used. The main signals of the FIFO are as follows:

- *clock*: Connected to the system frequency of 50 MHz. It sets the writing and reading speed.
- *data*: 16-bits input data in the writing side.
- *q*: 16-bits output data in the reading side.
- *rdreq*: When asserted a new output value each clock cycle is read.
- *wrreq*: When asserted a new output value each clock cycle is written.
- *usedw*: Number of words (samples of 16-bits) written in the FIFO. Maximum capacity is of three words per FIFO.

When the **Parallel to Serial** register samples all the 48 microphone channels it sends a *wrreq* signal to the 48 FIFOs module and it loads a sample in each FIFO. When there are two or more words written in every FIFO (i.e., $usedw \geq 2 \; \forall$ FIFO), *Data Available* will send a positive pulse. Then, the DMA unit will send a *wrreq* signal through *DataRd* signal. The samples will be stored in an output register and sent through a multiplexer to the DMA unit in pairs through *Data[31..0]* signal. The *array_vector[2..0]* and *sel[2..0]* signals control the array and channel number to be transmitted.

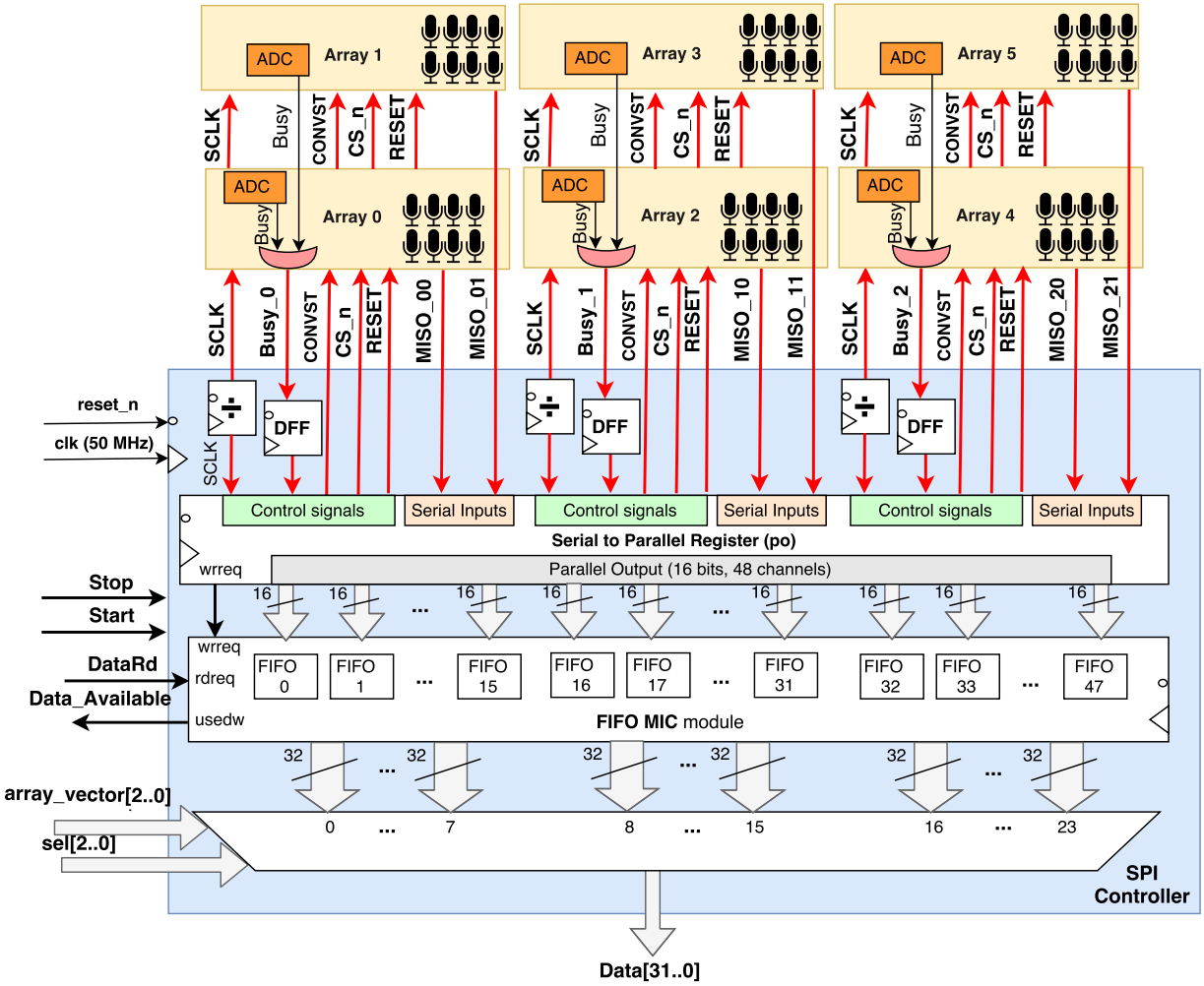


Figure 5.5 – Block diagram of the SPI Controller module

Regarding the timing of the system, *SCLK* bears a frequency of 12,5 MHz. Therefore, according to Equation 3.2, the time needed to complete an acquisition of one sample will be equal to 10μs, which matches with the results in Figure 5.6. If f_s is equal to 48 kHz (21 μs), since it is possible to read during conversion (i.e., while BUSY is high) we can perform real-time processing of the audio samples during the 11 μs available between samples. Once the data has been acquired, some clock cycles will be spent sending the data to memory through a DMA unit.

SPI DMA module

A Direct Memory Access (DMA) module is a Master unit that can transfer data directly to memory controllers. It is a programmable interface that must be programmed by the processor before it is operational. When the access bus is busy, the DMA performs arbitration and waits until the bus is available to send the next sample. Moreover, the DMA can perform burst

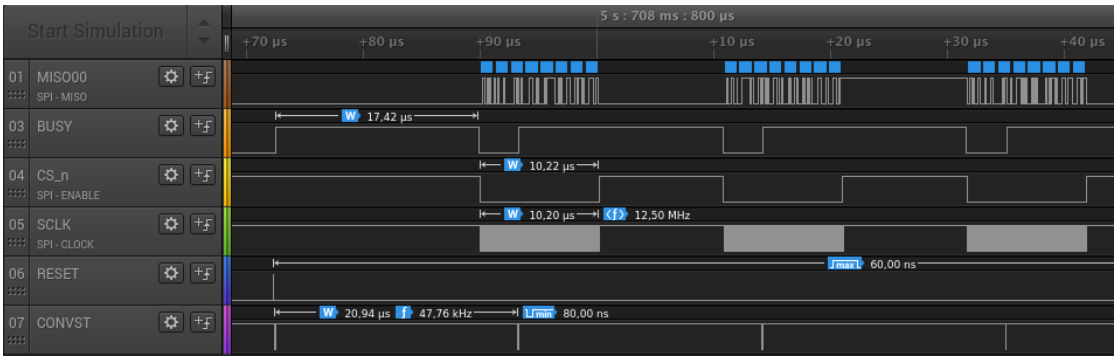


Figure 5.6 – SPI communication waveforms from Saleae Logic Analyzer.

transfers as the one explained in Section 4.5.1, reducing the transfer latency.

The 32-bits *Length* and *Address* signals are given by the slave module and indicate the length of the acquisition and the base address where the data will be written in the physical memory. Note that *length* it is not the number of samples per channel, although it represents the total number of samples of all the channels of the array. The 16-bits microphone signals are stored in pairs in the 32-bits address space as indicated in Figure 5.2.

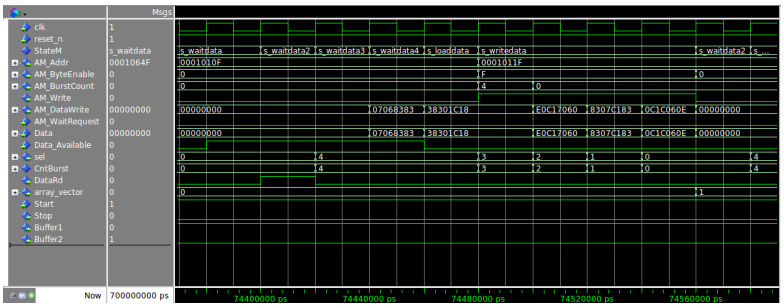


Figure 5.7 – SPI DMA module timing simulation in Modelsim.

The DMA unit will have access to the SDRAM memory controllers to write the audio data into a 1 GB DDR3 memory included in the DE1-SoC device. In our design, only **500 MB are available to store the audio data**. Therefore, if we record the 48 microphone channels at f_s = 48 kHz using all the available physical memory the maximum acquisition is equal to:

$$t_{acq} = \frac{500MB}{resolution * channels * f_s} = \frac{500MB}{16bits * 48channels * 48kHz} = 108.5s \qquad (5.1)$$

In order to overcome this limit, the DMA can write the data in a circular buffer. A circular buffer starts writing new data in the base address when the buffer overflows. *Address* indicates the base address where to start writing the data each period, while *Length* indicates the period duration (i.e., the size of the buffer). To exit from circular buffer mode the Start needs to be is low. In that case, the system transfers the data till the current period finishes. Additionally,

the DMA makes available *Buffer1* and *Buffer2* signals to the SPI slave to announce when the system reaches the middle and the end of each acquisition period. The timing diagram of those signals is presented in Figure 5.3.

DMA State machine

Figure 5.8 displays the main states of the DMA unit developed inside the *SPI_System* block. The state machine is synchronous to the **rising edge** of *clk*. The main functions of each state are described bellow:

- *s_idle*: It resets all the signals of the module to their default values. It waits till the *Start* signal is asserted by the Slave module.
- *s_waitdata*: It waits for the FIFOs of the *SPI_Controller* module to accumulate enough data to write in the memory. This happens when *Data_Available* is high. At that moment the DMA unit sends a read request to the FIFOs of the *SPI_Controller* module through *DataRd* signal.
- *s_waitdata2*: It is an intermediate state which deasserts *DataRd* signal. The system goes through *s_waitdata3* and *s_waitdata4* to wait three cycles for the *SPI_Controller* FIFOs sending all the parallel data to the output registers.
- *s_loaddata*: It loads the configuration of the Avalon MM Master interface as in number 2 in Figure 4.4. The parameters loaded for a write transfer are:
  - *AM_Write* = '1'
  - *AM_WriteAddress* = Address
  - *AM_BurstCount* = "100"
  - *AM_ByteEnable* = "1111"

  When *AM_WriteRequest* = '0' the system switches to the next state and the data transfer begins.
- *s_writedata*: It is the state where the samples are writen to memory. Each transfer takes places only when *AM_WriteRequest* = '0', i.e., when the bus is not busy. This state asserts the *sel* and *array_vector* signals to select the data from the output multiplexer of the *SPI_Controller* module. The microphone signals are selected as indicated in Table 5.1. For each value of *array_vector* a burst transfer is performed of four cycles, for *sel* from 1 to 4. The value of *array_vector* is incremented and if the new value is lower the total number of arrays (usually 6) the system returns to *s_waitdata2* to begin a new burst transfer. Otherwise the system goes to *s_waitdata* to wait for the ADC to send more samples. The length and address registers are changed after each transfer. If the circular buffer mode is not enabled (i.e., Start = '0'), when length signal is null it indicates the end of the acquisition and the state machine returns to *s_idle* state. This state asserts the *Streaming_Valid* and *Streaming_Data* signals depending on the value of the *Streaming_Ready* value.

Table 5.1 – Microphones transfer multiplexer selection.

array_vector	sel	Selected Mic
0 (MISO_00)	1	Microphone 0 Microphone 1
	2	Microphone 2 Microphone 3
	3	Microphone 4 Microphone 5
	4	Microphone 6 Microphone 7
1 (MISO_01)	1	Microphone 8 Microphone 9
	2	Microphone 10 Microphone 11
	3	Microphone 12 Microphone 13
	4	Microphone 14 Microphone 15
2 (MISO_11)	1 ... 4	Microphone 16 ... Microphone 23
3 (MISO_12)	1 ... 4	Microphone 24 ... Microphone 31
4 (MISO_21)	1 ... 4	Microphone 32 ... Microphone 39
5 (MISO_22)	1 ... 4	Microphone 40 ... Microphone 47

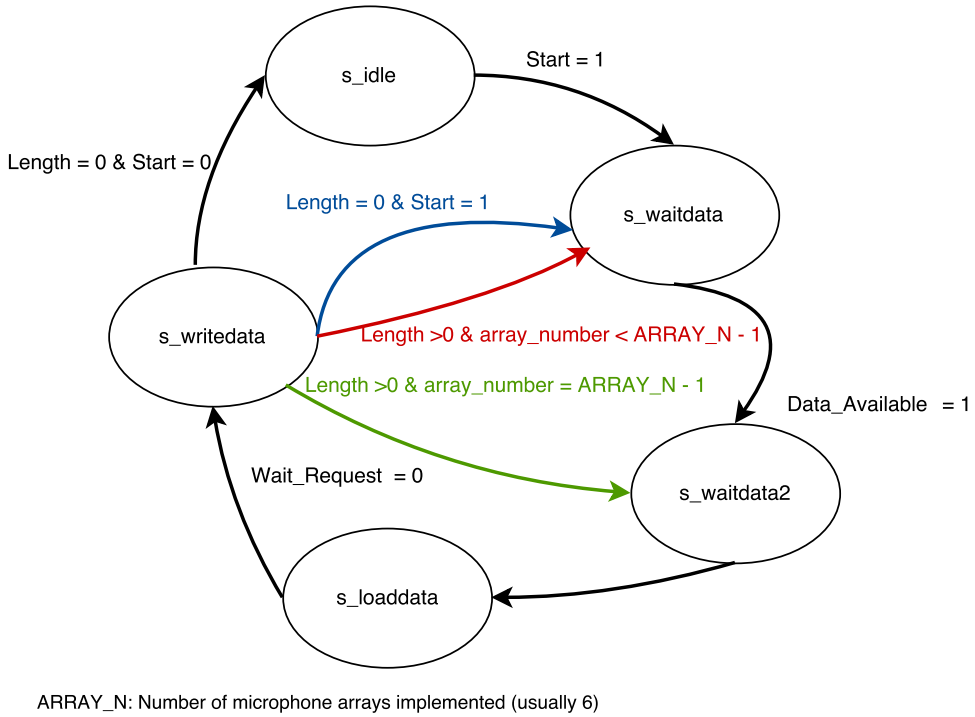


Figure 5.8 – SPI DMA unit state machine.

SPI Slave module

The *SPI_Slave* registers values control the *SPI_System* parameters. The registers can be accesed by bidirectional Avalon MM Slave interface. Table 5.2 summarizes the designed register table. For instace, if we want the acquisition to start writing in the memory address *0x0000FFFF* we ought to assert *AS_ Write* signal, load the desired value in *AS_WriteData* and select the correct *AS_Addr* value according with Table 5.2 ('000' in this example). The SPI slave can be accessed by a processor to set the base address, the acquisition period length, to start/stop the acquisition and to monitor the state of the acquisition through *Buffer1* and *Buffer2*. Figure 5.9 is a Modelsim simulation with multiple readings and writings in different address offset positions. In practice, the processor needs to increment the address offset by four instead of by one. This is due to the fact that the standard data packet size is 8 bits, and the bus width is 32 bits.

SPI Streaming module

The *SPI_Streaming* module sinks the audio samples from the *SPI_DMA* module and sends the streaming data to the parallel FIR filters bank through an Avalon Streaming interface. Figure 5.10 represents the timing diagram of the system. The system internal architecture is composed of one FIFO which receives the data from the DMA unit and sends the streaming data to the FIR filters. The *FIFO_Streaming* is a DCFIFO (Dual Clock FIFO) [9] which input

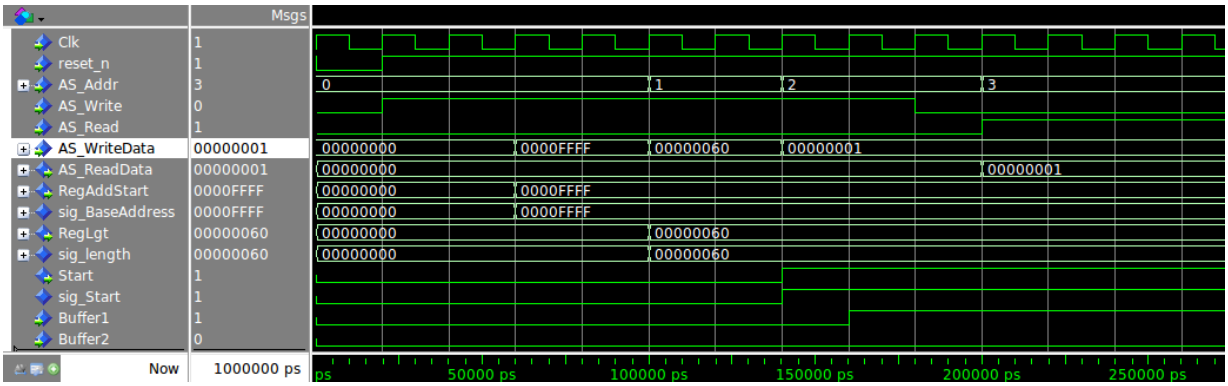


Figure 5.9 – SPI Slave module timing simulation in Modelsim

Table 5.2 – SPI Slave Table

Address	Name	Direction	Function
000	sig_BaseAddress	Write/Read	Initial base address for writing in memory
001	sig_length	Write/Read	Acquisition period length
010	sig_Start	Write/Read	When '1' the acquisition starts. When '0' the acquisition stops at the end of the acquisition period
011	Buffer1	Read	Active high flag indicating the first half of the acquisition period
100	Buffer2	Read	Active high flag indicating the second half of the acquisition period

and output data sizes are different, therefore it can use different clocks for reading and writing. The signals are connected as following:

- *data*: The input signal connected to *Streaming_Data* from the DMA unit. It is a 32-bit signal.
- *rdclk*: The clock which determines the reading speed, it is connected to the system clock of 50 MHz.
- *rdreq*: Read request signal, it is asserted each time the DMA unit sends a sample from all the microphone channels.
- *wrclk*: The clock which determines the writing speed, it is connected to the system clock of 50 MHz.
- *wrreq*: Write request signal, connected to the *Streaming_Valid* signal coming from the DMA.
- *q*: The output signal of 21 bits width. The first 16 bits of this signal represent the data input for the FIR filters, while the remaining 5 bits are the channel number of the sample (from 0 to 47). This particular configuration is built to match the FIR filter input protocol [3], which requires to send the *channel* and *data* information in the same signal.
- *wrfull*: It indicates if the FIFO is full on the writing side. It is connected to the *Streaming_Ready*

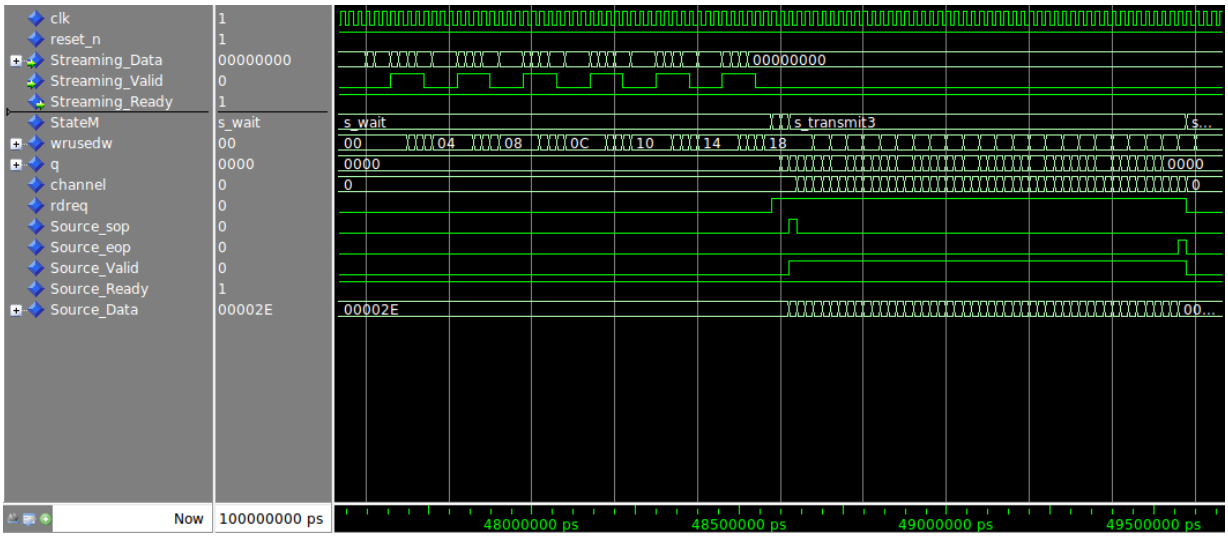


Figure 5.10 – SPI Streaming timing simulation in Modelsim

signal to indicate to the DMA unit that if the FIFO is full the system is not ready to receive new information.

- *wrusedw*: It counts the number of samples written in the FIFO. When more than 24 words have been written in the FIFO (i.e., 1 word = 32 bits = 2 samples) the FIFO can start streaming data packages from the DMA unit to the FIR filters.

Really, the system has two sources, one for the right channel and another one for the left channel. Both channels signals are identical at the output of the *SPIStreaming* module. This allows performing stereo processing of the microphone audio signals.

Next section details how the output data from the *SPI_Streaming* system interacts with the FIR filters bank in order to perform real-time delay-and-sum beamforming.

5.3 FPGA implementation of Delay-and-Sum beamformer

In this section, an FIR filter design process for delay-and-add beamforming applications will be briefly reviewed, followed by an explanation of the filters implementation in an FPGA based platform.

5.3.1 Digital filters

Digital filters are typically used to modify or alter the attributes of a signal in the time or frequency domain. The most common digital filter is the linear time-invariant (LTI) filter. An LTI interacts with its input signal through a process called linear convolution, denoted by $y = h*x$, where h is the filter's impulse response, x is the input signal, and y is the convolved output.

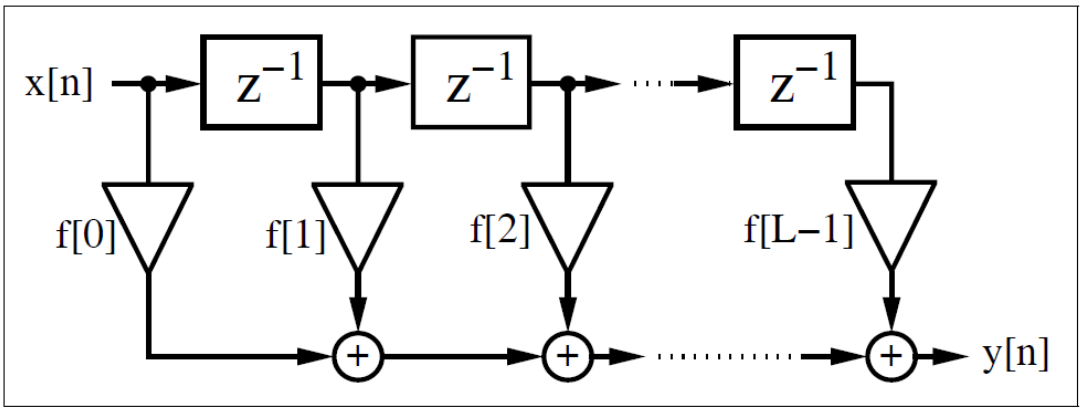


Figure 5.11 – Direct form FIR filter [17, p. 166].

The linear convolution process is formally defined by:

$$y[n] = x[n] * y[n] = \sum_k x[k]h[n-k] = \sum_k h[k]x[n-k]. \tag{5.2}$$

LTI digital filters are generally classified as being *finite impulse response* (i.e., FIR), or *infinite impulse response* (i.e., IIR). As the name implies, FIR filter consists of a finite number of sample values, reducing the above convolution sum to a finite sum per output sample instant. An IIR filter, however, requires that an infinite sum be performed [17].

Digital filters are rapidly replacing analog filters, which were implemented using RLC components and operational amplifiers. Analog filters were mathematically modeled using ordinary differential equations of Laplace transforms. They were analyzed in the time or s (also known as Laplace) domain. Analog prototypes are now only used in IIR design, while FIRs are typically designed using direct computer tools and algorithms.

5.3.2 FIR theory

An FIR with constant coefficients is an LTI (Linear Time Invariant) digital filter. LTI systems are linear, which implies that the relationship between the input and the output is linear. The second property, time invariant, means that whether a given input is applied to the system now or t seconds from now, the output will be identical except for a t seconds shift. Therefore, an FIR filter of order or length L can be defined by a transfer function $h[n]$, which output is given by a *finite* version of the convolution sum between an input time-series $x[n]$ and $h[n]$ (5.2):

$$y[n] = x[n] * h[n] = \sum_{k=0}^{L-1} f[k]x[n-k], \tag{5.3}$$

where *f[k]* are the filter's L coefficients. They also correspond to the FIR's impulse response. For LTI systems it is sometimes more convenient to express 5.2 in the z-domain with

$$Y(z) = F(z)X(z), \tag{5.4}$$

where *F(z)* is the FIR's *transfer function* defined in the z-domain by

$$F(z) = \sum_{k=0}^{L-1} f[k]z^{-k}, \tag{5.5}$$

In practice, the Lth-order LTI FIR filter can be graphically intepreted in Figure 5.11. It can be seen to consist of a collection of a "tapped delay line", adders and multipliers. One of the operands presented to each multiplier is an FIR coefficient, often referred to as a "tap weight". The *roots* of *F(z)* define the zeros of the filter.

5.3.3 Filter-and-Sum Beamformer

Consider an array of M microphones and a far-field sound source at direction ϕ. Each of the microphones captures the same signal slightly delayed owing to the different times the sound needs to reach the corresponding microphone. The sound is captured from a far-field source according to Equation 2.3. The most intuitive approach to estimate the source signal is to delay each of the microphone signals in a way that the sound from the source signal are in phase, and to sum them. Then, we will take the source signal (the same in all channels after the delay) to have an amplitue M times bigger than in each of the channels. This is usually compensated by dividiing the sum by the number of channels [21].

$$Y(f) = \frac{1}{M}\sum_{m=0}^{M-1} e^{j2\pi f}\frac{\|p_m\|\cos((p_m)-\phi_0)}{v}X_m(f), \tag{5.6}$$

where ϕ_0 is the listening direction. Under the assumption of omnidirectional microphones ($U_m(f) \equiv 1$) and perfect conditioning circuit and ADC system ($A_m(f) \equiv 1$) for the band of interest, combining equations 2.2, 2.4 and 5.6 we obtain

$$Y(f) = S_0(f) + \frac{1}{M}\sum_{m=0}^{M-1} e^{j2\pi f}\frac{\|p_m\|\cos((p_m)-\phi_0)}{v}N_m(f). \tag{5.7}$$

If we assume the captured noise is not correlated accross the microphones and is modeled as a zero-mean Gaussian process $N_m(f) = N(0,\lambda(f))$, then the output signal becomes

$$Y(f) = S_0(f) * N(0,\frac{\lambda(f)}{M}). \tag{5.8}$$

Since the multiplicand after *S(f)* value will be smaller or equal to 1, in most cases the magnitude of a sound source coming from a direction different from the listening direction will be supressed to a certain degree.

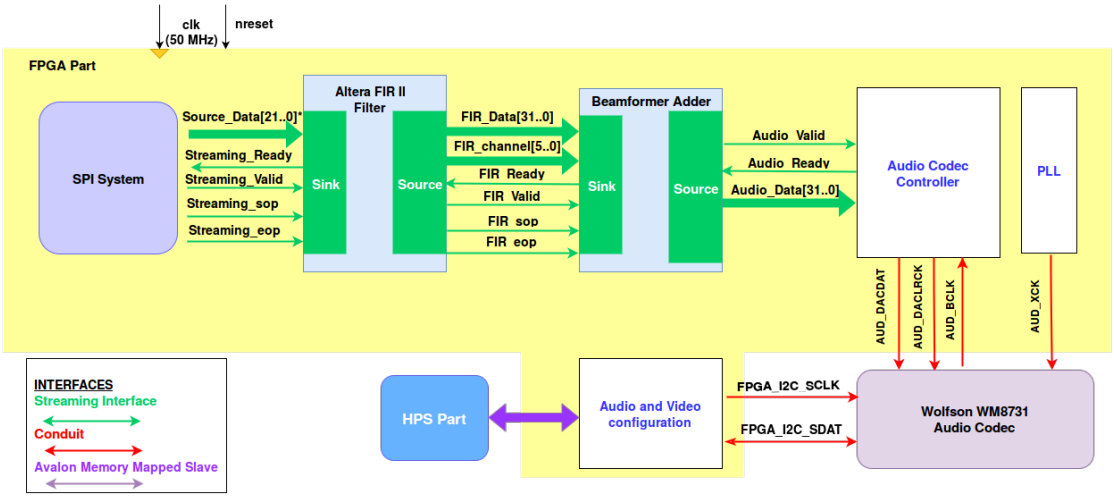


Figure 5.12 – Parallel delay-and-sum beamformer block diagram.

Therefore, Equation 5.7 can be easily generalized in matrix form to

$$Y(f) = W(f)X(f). \tag{5.9}$$

The spectrum of each input audio frame **X** is an *MxK* complex matrix, where *M* is the number of microphones and *K* is the frame size, i.e., the number of frequency bins. *X(f)* is a *Mx1* complex vector. The frame indices are omitted for simplicity. The equations above are known as a "filter-and-sum beamformer" because the weights matrix *W(m)* acts as a filter for each channel. That is, first we filter each of the 48 channels separately and then we sum them. The filter-and-sum beamformer has more degrees of freedom and allows to perform flexible microphone arrays processing [21].

5.3.4 FIR filters design in FPGA

In Section 5.3.3 a general delay-and-sum beamformer using FIR filters was presented. This technique has been designed for the Pyramic array system, for which a real-time 48 parallel FIR filter cascade has been designed in an FPGA platform. The generated beam is output through the Audio Codec of the DE1-SoC. We will explain the design in the following section.

Overall Design

Figure 5.12 summarizes the overall architecture of the implemented custom delay-and-sum beamformer FIR filters bank. The FIR filters have been built using IP cores offered by Altera, which can generate FIR filters [3]. The data recorded through the SPI communication serves as input to the 48 filters bank. The filters bank filters the data, which is added in the next stage, a *Beamformer_Adder* custom component. The resulting audio beam is sent to the Audio Codec, which sends the data to the audio out line of the DE1-SoC board.

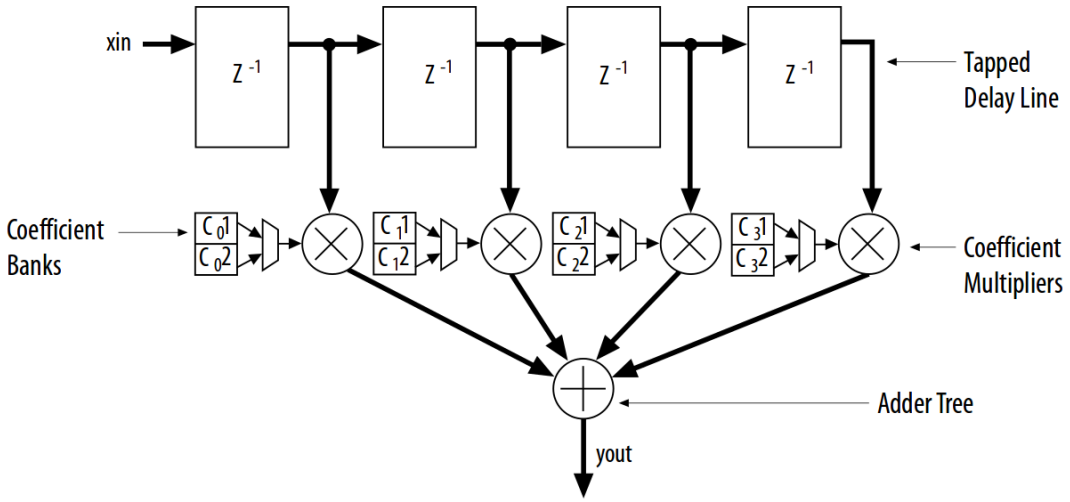


Figure 5.13 – Basic FIR Filter with Weighted Tapped Delay Line [3].

IP Core FIR Filter design

The Altera FIR II IP core provides a fully-integrated finite impulse response (FIR) filter function optimized for use with Altera FPGA devices. The FIR II IP core has an interactive parameter editor that allows creating easily custom FIR filters. For instance, the filter type can be defined as single rate, decimation, interpolation, and fractional rate. The core is easy to integrate using Avalon Streaming (Avalon-ST) interfaces. Moreover, it supports run-time coefficient reloading capability and multiple coefficient banks. Figure 5.13 shows the internal architecture of the filter, where the number of tapped delay lines represent the filter order L and the coefficient multipliers allow multiple coefficient banks multiplexing.

The main parameters applied in our implementation are as follows:

- **Filter Specifications**:
  - Fractional Rate
  - Interpolation factor: 2
  - Decimation Factor: 3
  - Maximum number of channels: 48
  - Input clock: 50 MHz
  - Clock Slack: 0
  - Input sample rate: 48 kHz
  - No coefficient reload
  - No back pressure support
- **Coefficient Settings**:
  - Symmetry: Non Symmetry
  - L-th Band Filter: All taps
  - Coefficients Scaling: Auto

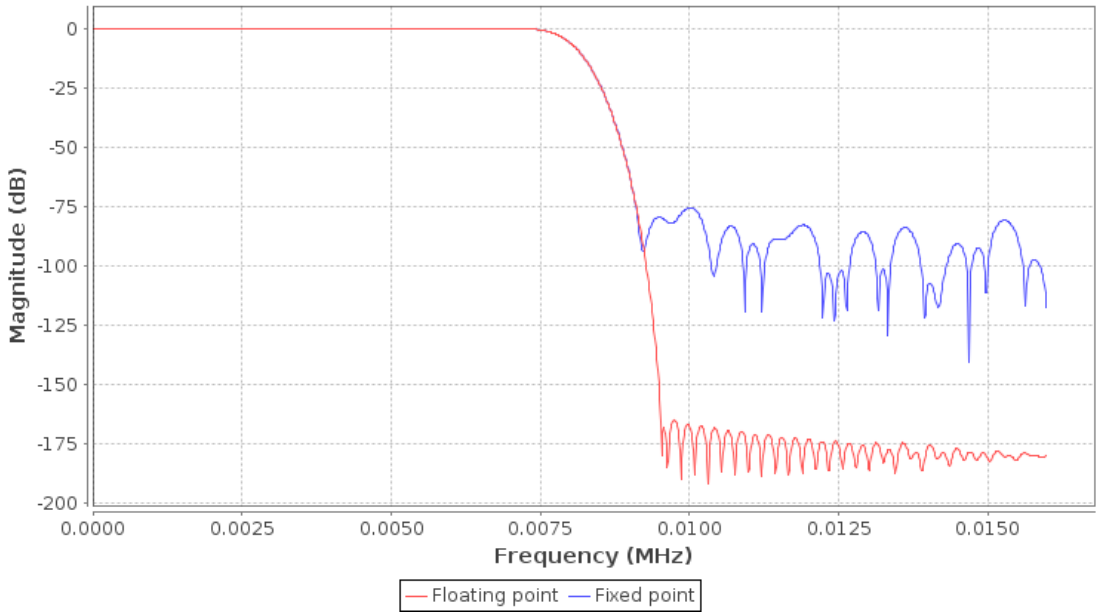


Figure 5.14 – FIR filter frequency response from Altera FIR IP Core [3]

  - Coefficients Data Type: Signed Fractional Binary
  - Coefficients Width: 15 bits
  - Coefficients Fractional Width: 15 bits
- **Input/Output Options**:
  - Input Type: Signed binary
  - Input Width: 16 bits
  - Output Type: Signed binary
  - Truncate 7 LSB to remove from the output
  - Output Width: 32 bits
- Implementation Options: Default parameters.
- No reconfigurable carrier mode.

The filter coefficients can be loaded from a *.txt* file. We tested the filter banks with the coefficients array of Appendix D, which frequency response is shown in Figure 5.14. The cutoff frequency is 8 kHz, suitable for speech processing applications. Note that since the filter performs Fractional Rate the f_s is different at the input and output of the module:

$$f_{sout} = \frac{f_{sin} * Interpolation}{Decimation} = \frac{48kHz * 2}{3} = 32kHz \tag{5.10}$$

5.3.5 System Timing

One of the reasons why the Altera FIR IP core was chosen is that it allows Time-Division Multiplexing (TDM) to optimize hardware utilization. The TDM factor is the ratio of the clock

rate to the sample rate. For example, implementing a filter with a TDM factor of 2 can halve the required hardware.

To achieve TDM, the IP core requires a serializer and deserializer before and after the reused hardware to control the timing. The ratio of system clock frequency to sample rate determines the amount of resource saving except for a small amount of additional logic for the serializer and deserializer.

$$TDM = \frac{f_{clk}}{f_s} = \frac{50MHz}{48kHz} = 1041 \tag{5.11}$$

With this TDM is possible to have 48 FIR filters in parallel just by using one physical FIR filter, which reduces a lot the hardware resources of the system. The input data is streamed just using one wire to the FIR II IP core. Since there are 1041 clock cycles between samples, the IP core can process all the 48 channels by using just one FIR filter.

The data is sent to the *Beamformer_Adder* module as Streaming ST interface. Note that the bits per packet (*dataBitsperSymbol*) in the FIR IP core is equal to the data width, while usually in most of the IP cores the bits per packet is equal to 8. For instance, the *dataBitsPerSymbol* of Figure 4.5 is equal to 8, thus we need four wires to send a 32 bits data. If we would like to connect those wires to the input of the FIR core, since we are using a single wire we should change the *dataBitsPerSymbol* parameter to 32 bits and send all the data through the same wire. If *channel* signal is available it should be concatenated to the *data* signal and its length should be added to the *dataBitsPerSymbol* parameter. This way all the information (the channel number and the data) can be sent through the same wire.

Beamformer Adder

The *Beamformer_Adder* module receives filtered data from the FIR filters bank, adds it and sends the output to an Audio line out in the DE1-SoC board.

It receives the filtered *FIR_data* from the FIR filter and the *FIR_channel* number in two different wires. The FIR filter asserts *FIR_Valid* to indicate the data is ready to be sent. When *FIR_sop* is high the first channel is sent, while when *FIR_eop* is high indicates the last channel is transmitted. During this time, the module saves the 48 channels data in the *FIR_Output* register. When the last channel is sent (i.e., *FIR_eop* = '1'), the module adds all the filtered channel values and divides this addition by *M*, which is the number of microphones. The *FIR_Valid* indicates when the register is available to receive more data. The process is described in Figure 5.15.

Audio Codec

The output an the *Beamformer_Adder* module can be listened through the Audio Codec DAC output line. The DE1-SoC device possesses a Wolfson-WM8731 audio CODEC which can

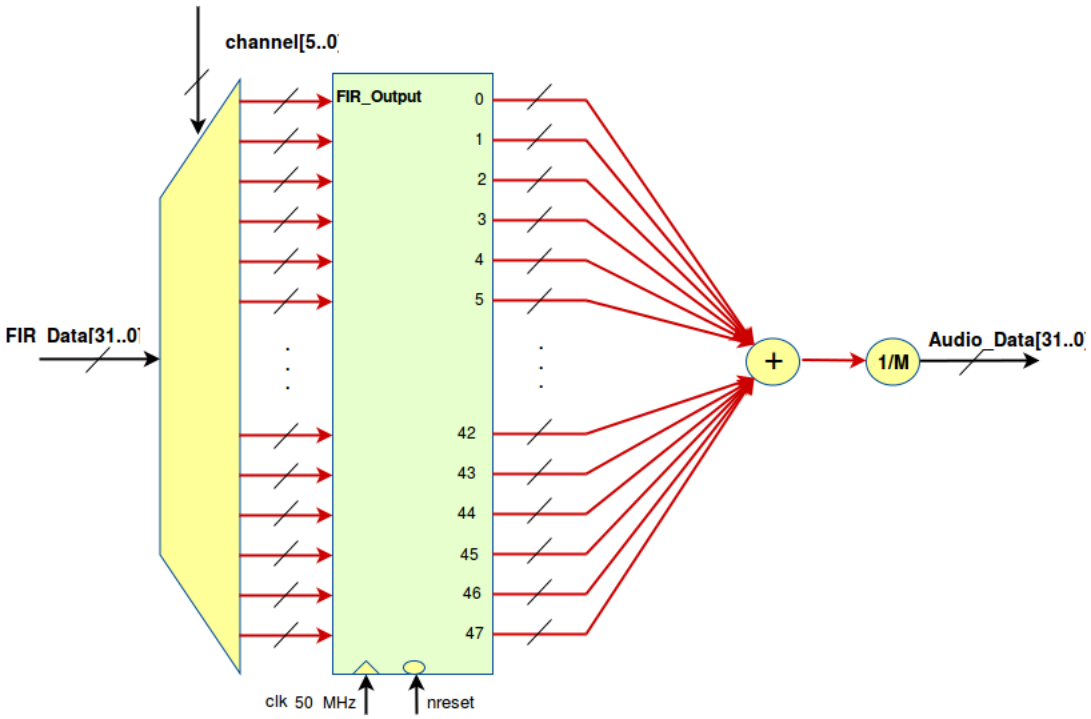


Figure 5.15 – Beamformer adding the output of the FIR filters module

be controlled thanks to the *Audio and Video Config* and *audio_controller* IP cores [1]. The connection diagram between those modules and the design is displayed in Figure 5.12. The *Audio and Video Config* module controls the Wolfson codec through an I2C interface. At the same time, the *Audio and Video Config* module is controlled as an Avalon MM slave by a processor Master unit (the HPS in our system). The *audio_controller* sends the data to the output, input or microphone input lines through the WM8731 audio codec, as displayed in Figure 5.16. The parameters chosen for this project are the followings:

- Left justified data
- Line out
- 32 bits data width
- 32 kHz output frequency
- Streaming interface

The input to the *audio_controller* is a simple Avalon ST interface where the data is sent when the audio codec is ready to accept data (i.e., *Streaming_Ready* high as well). To indicate the reliability of the transmitted data *Streaming_Data*, the *Streaming_Valid* signal is asserted every cycle a new sample is sent.

A PLL module has been added to the design in order to generate *AUD_XCK* signal, which value is 12,288 MHz to match the specifications of defined in [26]. Moreover, [26] provides a more detailed information about the different signals to interface the audio ADCs and DACs

Table 5.3 – Hardware resources

Logic Utilization (in ALMs)	2,195/32,070 (7 %)
Total registers	3831
Total pins	201 / 457 (44%)
Total block memory bits	511,700 / 4,065,280 (13%)
Total PLLs	1/6 (17%)
Total DLLs	1/4 (25%)

with the FPGA.

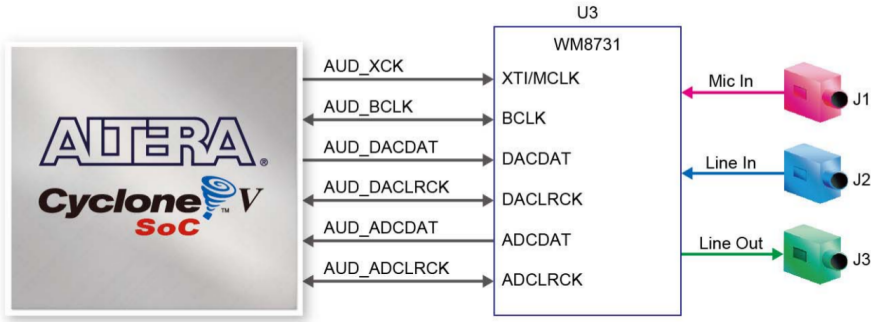


Figure 5.16 – Connections between the FPGA and audio CODEC [23].

5.4 Conclusions

Through this Chapter we described the SPI module and the FIR filters bank developed for delay-and-add beamforming applications. In the end, the SPI communication has been tested and verified its correct performance, while the FIR filters have some problems to perform real-time processing yet. Even if the described design is correct and the audio signal from the beamformer can be listened at the Audio Codec line out, there is some saturation in the value of the data. We believe the problem is with the coefficients values and widths, which parameters can be change in order to address this problem. Moreover, it would be interesting to use the coefficient reloading option of the FIR IP core to be able to change the look-up direction in real-time.

However, the VHDL designs work properly while the hardware resources spent on processing such big amount of data are not very big. Table 5.3 summarizes the FPGA resources for the Pyramic array FPGA project after compiling the SPI communication interface along with the 48 FIR filters bank with 143 coefficients per filter.

Regarding timing, the time it takes to write through the SPI communication in memory a sample from all the microphone channels is equal to 0,64 μs. With a sampling frequency of 48 kHz there is a free time of 21.09 μs where the FPGA can perform further processing.

6 HPS design and implementation

6.1 Introduction

Even if a programmable DSP system offers many advantages, usually it is convenient to communicate the FPGA with a processing unit embedded as a Hard Processor System (HPS). As explained in Section 4.2.2, HPS are not synthesized on FPGA fabric, rather they are real processors. The Cyclone V family allows mixed designs where a single chip contains both an HPS and an FPGA device. The HPS consists of an ARM-A9 MPCore, which is a general-purpose *application processor*. The HPS-FPGA combination delivers the flexibility of programmable logic with the power and cost savings of hard IP cores, while the board size is reduced since we reduce the cost of adding a discretely embedded processor. In addition, this type of embedded systems allows differentiating the end product between the hardware and software design. Therefore, the system development supports virtually any interface standard.

Through this chapter, we will introduce the main characteristics of the Cyclone V HPS. Based on this architecture, we will explain the implementation followed for the Pyramic array design, which involves the communication between the FPGA and the HPS unit, data storage in a DDR3 memory and the installation of a Linux OS in order to control the system through a WebServer. This allows storing the data in an SD card plugged into the DE-1 and to send it through the network to a local computer.

6.2 Cyclone V Hard Processor System [18]

The DE1-SoC HPS contains a microprocessor unit (MPU) subsystem with dual ARM Cortex-A9 MPCore processors, flash memory controllers, SDRAM L3 interconnect, on-chip memories, support peripherals, interface peripherals, debug capabilities, and phase-locked loops (PLLs). The dual-processor HPS supports symmetric (SMP) and asymmetric (AMP) multiprocessing (Figure 4.1).

The HPS and FPGA portions of the device are distinctly different, while the HPS can boot

from

- the FPGA fabric,
- **an external flash** or
- JTAG

the FPGA must be configured either through

- **the HPS**, or
- an externally supported device such as the *Quartus Prime* programmer.

The HPS and FPGA parts of the device each have their own pins. However, pins are not freely shared between the HPS and FPGA fabric. The ***FPGA I/O pins*** are configured by an ***FPGA configuration image*** through the HPS or any external source supported by the device. Moreover, the HPS and FPGA portions have distinct power supplies and power on independently. Meaning that the HPS can be powered without turning on the FPGA portion of the device. Nonetheless, to switch on the FPGA portion the HPS must be already powered on at the same time as the FPGA.

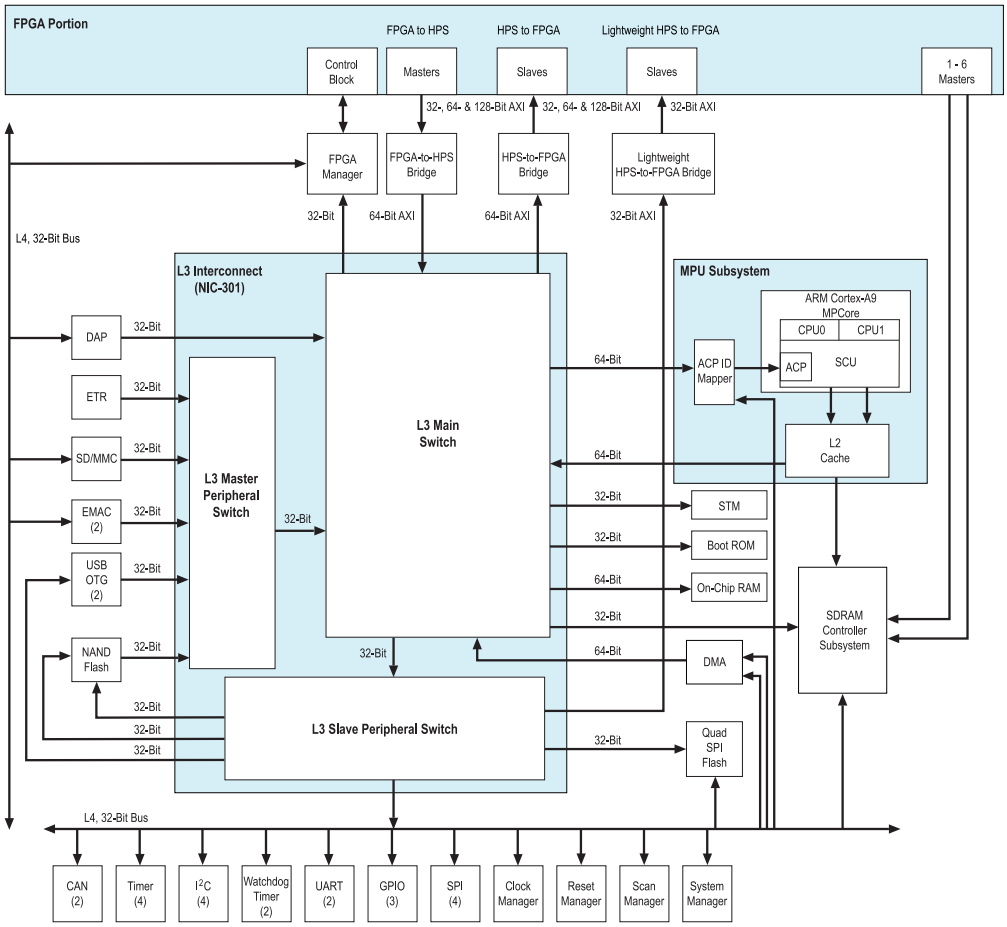


Figure 6.1 – HPS block diagram [7].

6.2.1 HPS features

Figure 6.1 shows the architecture of the HPS system, which consists of a dual-core 925 MHz ARM Cortex-A9 MPCore processor, a rich set of peripherals, and a share multiport SDRAM memory controller [7]. The following modules were implemented in the design of the Pyramic array:

- **Masters**:
  - MPU subsystem featuring dual ARM Cortex-A9 MPCore processors.
  - One Ethernet media access controller (EMAC1):
    - **EMAC1 pin**: HPS I/O Set 0 pin
    - **EMAC1 mode**: RGMII
  - ARM CoreSight debug components
  - One SD/MMC controller:
    - **SDIO pin**: HPS I/O Set 0
    - **SDIO mode**: 4-bit Data
- **Slaves**:
  - 64 KB on-chip RAM
  - 64 kB on-chip boot ROM
  - UART0 Controller:
    - **UART0 pin**: HPS I/O Set 0
    - **UART0 mode**: No flow control
  - Reset manager
  - Clock manager
  - Scan manager
  - System manager
  - FPGA manager

The Ethernet MAC, USB OTG, NAND flash controller, and SD/MMC controller modules have an integrated DMA controller. An integrated DMA controller module provides up to eight channels of high-bandwidth data transfers. Note that peripherals that communicate off-chip are multiplexed with other peripherals at the HPS pin level, which allows choosing which peripherals to interface with other devices on the PCB.

The MPU subsystem has an interrupt controller, one-general-purpose timer and one watchdog timer per processor and one memory management unit (MMU) per processor. Besides that, the HPS masters the L3 interconnect and the SDRAM controller subsystem.

6.2.2 HPS-FPGA Bridges

The FPGA can exchange information with the HPS part thanks to the HPS–FPGA bridges, which support the Advanced Microcontroller Bus Architecture (AMBA®) Advanced eXtensible Interface (AXI™) specifications, consist of the following bridges:

- **FPGA-to-HPS AXI bridge**-a high-performance bus supporting 32, 64, and 128-bit data widths that allow the FPGA fabric to issue transactions to slaves in the HPS.
- **HPS-to-FPGA AXI bridge**-a high-performance bus supporting 32, 64, and 128-bit data widths that allow the HPS to issue transactions to slaves in the FPGA fabric.
- **Lightweight HPS-to-FPGA AXI bridge**- a lower latency 32 bit width bus that allows the HPS to issue transactions to slaves in the FPGA fabric. This bridge is primarily used for control and status register (CSR) accesses to peripherals in the FPGA fabric.

The HPS–FPGA AXI bridges allow masters in the FPGA fabric to communicate with slaves in the HPS logic, and vice versa.

6.2.3 HPS Address Map

HPS Address spaces

The HPS address map specifies the address of slaves, such as memory and peripherals, as viewed by the HPS masters. The HPS is divided into 3 nonoverlapping address spaces of 4 GB size each: one for the MPU subystem, another for the L3 interconnect and for the SDRAM controller subsystem [5].

The window regions provide access to other address spaces. The thin black arrows in Figure 6.2 indicate which address space is accessed by a window region (arrows point to accessed address space). For example, accesses to the ACP window in the L3 address space map to a 1 GB region of the MPU address space.

The SDRAM window in the MPU address space can grow and shrink at the top and bottom (short, blue vertical arrows) at the expense of the FPGA slaves and boot regions. The blue bidirectional arrow indicates that the ACP window can be mapped to any 1 GB region in the MPU address space, on gigabyte-aligned boundaries.

HPS Peripheral Region Address Map

Each peripheral slave interface has a dedicated address range in the peripheral region. Table 6.1 lists the base address and address range size for some slaves interfaces in the HPS part. The *SPI_System* and *audio_and_video_config* modules implemented in Chapter 5 are seen as slave modules connected to the lightweight bus from the HPS side. Note that if a peripheral is connected to a bus, its address is obtained by adding its offset in the bus to the bus' address, which in this case should be the lightweight FPGA slaves bus base address. For a complete table showing all the HPS peripheral region address map refer to [5].

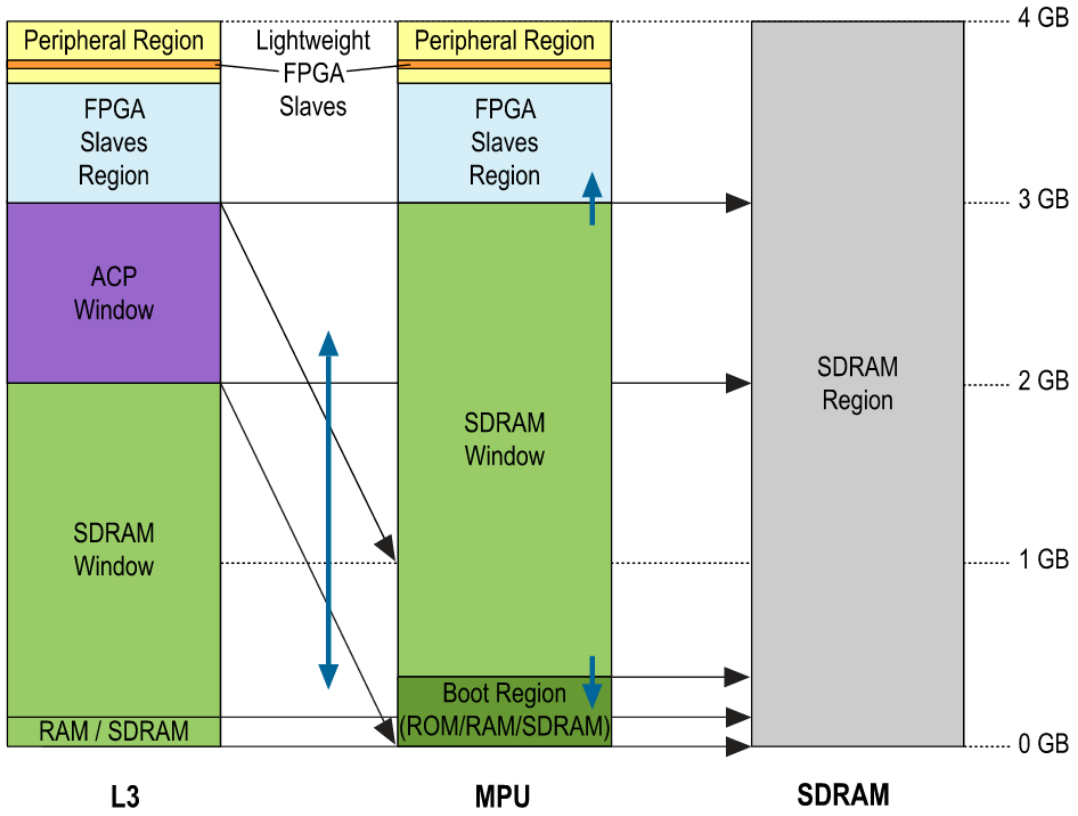


Figure 6.2 – HPS address spaces relationship [5].

6.3 SDRAM Controller

The SDRAM controller subsystem supports DDR2, DDR3, or LPDDR2 devices up to 4 Gb in density operating at up to 400 MHz (800 Mbps data rate).

The HPS SDRAM controller subsystem contains a multiport SDRAM controller and DDR PHY that are shared between the FPGA fabric (through the FPGA-to-HPS SDRAM interface), the level 2 (L2) cache, and the level 3 (L3) system interconnect. The FPGA-to-HPS SDRAM interface supports AMBA AXI and *Avalon® Memory-Mapped (Avalon-MM)* interface standards and provides up to six individual ports for access by masters implemented in the FPGA fabric.

HPS and FPGA fabric masters have access to the SDRAM controller subsystem. The DE1-SoC SDRAM controller is connected to 1 GB DDR3 memory. In our design the SPI Master unit can access the SDRAM controller through an Avalon MM communication protocol.

6.4 HPS Booting and FPGA Configuration

It is important to understand how the HPS boots and how the FPGA can be configured through the HPS [8]. Figure 6.3 illustrates the typical bootflow of an HPS ARM processor system.

Table 6.1 – HPS peripheral region address map

Slave identifier	Slave title	Base address	Memory span
SPI	SPI_System_0	Lightweight_Base + 0x0000_0000	32 bits
Audio Module	audio_and_video_config_0	Lightweight_Base + 0x0000_0001F	16 bits
SDMMC	SD MMC0	0XFF70_4000	4kB
EMAC1	EMAC1	0xFF70_2000	8 kB
Lightweight FPGA slaves	FPGA slaves accessed with lightweight HPS-to-FPGA bridge	0xFFD0_5000	2 MB

First the boot process starts when the processor is released from reset, e.g., on power up. At this point, CPU1 is helding reset and CPU0 jumps to the reset exception address (usually 0x0) where it starts executing code out of the Boot ROM mapped to this address. The Boot ROM bears the first code that runs the device, which is hard coded by Altera. Boot ROMs main purpose is to set up the device so it can fetch the next stage of boot code into the On-Chip RAM and pass control over to it. Then, the preloader performs additional HPS initialization for the HPS to look like a conventional ARM processor. The preloader brings up the SDRAM controller, basic I/O pins and loads the next boot stage from Flash to SDRAM. Moreover, it fetches the next piece of code that corresponds to the user Boot Loader, which is an *u-boot* script in our case. As the user Boot Loader runs, the appropriate Operating System will run in the ARM processor, which in our configuration it consists on an *Ubuntu Core 14.04.4*. Finally, the OS application is launched. The FPGA is configured through the HPS at the Boot Loader stage.

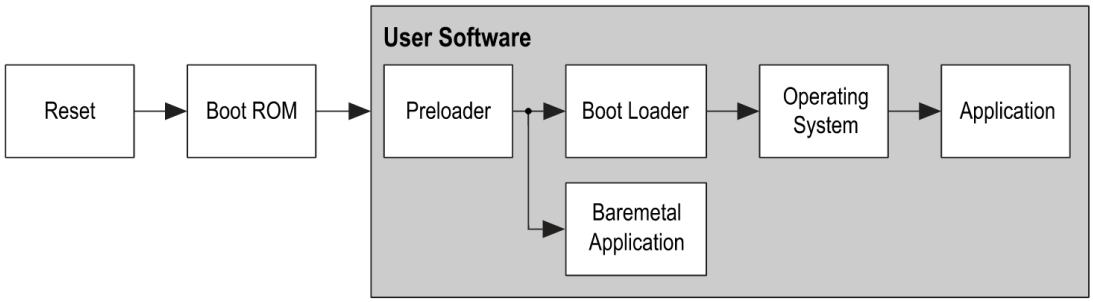


Figure 6.3 – HPS Boot flow [8] .

Note in Figure 6.3 that what it comes after the preloader depends on the application the user wants to run. The reset, boot ROM, and preloader stages are always present in the HPS boot flow. However, the HPS can execute two types of applications:

- Bare-metal applications (without operations system)
- Applications on top of an **Operating System** (linux)

The processor can boot from the following sources:

- NAND flash memory through the NAND flash controller
- **SD/MMC flash memory** through the SD/MMC flash controller
- SPI and QSPI flash memory through the QSPI flash controller
- FPGA fabric on-chip memory

The choice of the boot source is done by modifying the *BOOTSEL* and *CLKSEL* before the device is powered up. Therefore, the Cyclone V family normally uses a **PHYSICAL DIP SWITCH** to configure the *BOOTSEL* and *CLKSEL*. However, it is important to note that **the DE1-SoC can only boot from SD/MMC flash memory**, since the *BOOTSEL* and *CLKSEL* values are hard-wired on the board. Even if the HPS contains all the necessary controllers, the board does not have a physical DIP switch to modify the *BOOTSEL* and *CLKSEL* values. The location of the DIP switch is present underneath the board.

Figure 6.4 shows the scheme under which the HPS first boots from one of its non-FPGA fabric boot sources, then software running on the HPS configures the FPGA fabric through the FPGA manager. The software on the HPS obtains the FPGA configuration image from any of its flash memory devices or communication interfaces. In our application, it is loaded from the SD/MMC memory controller, which reads an image saved in an SD card. The software provided by users and the boot ROM are not involved in configuring the FPGA fabric.

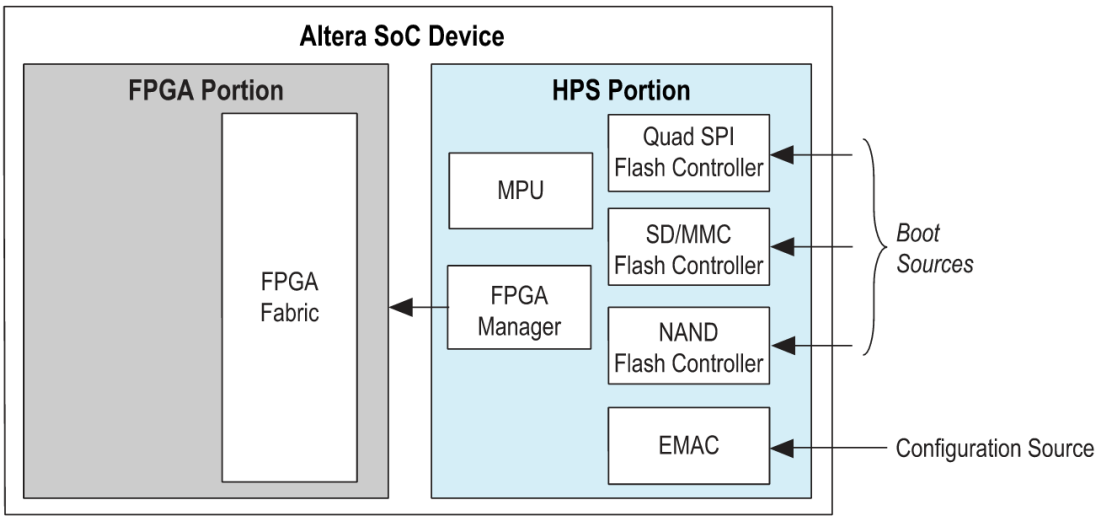


Figure 6.4 – HPS Boots and Performs FPGA Configuration

Preloader

Everything in the user software box can be customized. The preloader is the most important of the boot stages, it is actually in charge of the following actions:

- Initialize the SDRAM device
- Configure the HPS I/O through the scan manager
- Configure the pin multiplexing through the system manager
- Configure the HPS clocks through the clock manager
- Initialize the flash controller (NAND, SD/MMC, QSPI) that contains the next stage boot software
- Load the next boot software into the SDRAM controller and pass control to it

The preloader does not release CPU1 from reset as the subsequent stages of the boot process are responsible for it.

6.5 Pyramic Hybrid System

Exclusively using the FPGA part of Cyclone V is easy, the design process followed in Chapter 5 is identical in any other Altera FPGA. Design an HDL module in *Quartus Prime* editor, simulate it and then program the FPGA through *JTAG configuration*. However, Pyramic implements a more powerful design by combining the FPGA portion with the HPS processor available in Cyclone V devices, creating a hybrid system. In our design, the Pyramic array is an embedded hybrid system which HPS part is configured as follows:

- The HPS is able to use the **Ethernet port** on the board. Moreover, it is possible to share internet with the DE1-SoC board. The device has a **static IP address**. *Apache http webserver* has been downloaded to write *php* applications.
- The HPS is able to use the **microSD** card port on the board to which the system will be able to write anything. The inserted microSD size is 32 GB and it has two partitions: *a2* (24 GB) and *fat32* (512 MB).
- The HPS will program automatically the FPGA part reading a binary *.rbf* file saved in a *fat32* partition in the microSD card.
- The HPS is able to use the **UART** controller. The board can be controlled through *minicom* connection.
- The **Lightweight FPGA-to-HPS bridge** with Avalon MM interfaces is used, allowing the **HPS** to control slaves modules from the FPGA such as the *SPI_System* and *audio_and_video_config* modules.
- The HPS system will run on a Linux based OS.
- The **FPGA-to-HPS SDRAM controller** interfaces the FPGA with a 1GB DDR3 memory. The transaction is an **Avalon-MM Bidirectional** interface of 32-bits.
- 500 MB of the DDR3 memory are allocated for the OS Kernel. The rest 500 MB are used to store audio samples from the Pyramic array.

6.5.1 HPS and FPGA Design

Bare-metal Application

Bare-metal software enjoys the advantage of having no OS overhead. The code can access directly HPS peripherals by using their **physical memory-mapped**, as no virtual memory system is being used.

Application Over an Operation System (Linux)

One of the main advantages of running code on a Linux OS is that the kernel releases CPU1 from reset upon boot, thus all processors are available. Moreover, the kernel initializes almost all HPS peripherals available, which is possible since the linux kernel has access to a big amount of device drivers. Furthermore, a multithread code is much easier to write. Finally, the Linux kernel can run more programming languages other than C, such as Python, which is available for ARM processors.

However, when running an embedded system on top of an operating system the programmer needs to cope with **virtual memory** system placed by the OS. Hence, the program cannot access directly teh HPS peripherals through their physical memory-mapped addresses. Instead, one first needs to map the physical addresses of interest into the running program's virtual address space. Only then it will be possible to access a peripheral's registers. Ideally, the programmer should write a device driver for each specific component that is designed to have a clean interface between user code, and device accesses.

Summarizing, bare-metal and Linux applications can do the same things. In general, programming on top of Linux provides higher level syntax compared to bare-metal code, as its advantages outweigh its drawbacks we implemented linux OS for Pyramic array. Following Chapter 13 of [18], we installed an *Ubuntu Core 14.04.4* OS in the DE1-SoC which boots the HPS ARM system and interfaces the Pyramic array.

6.6 Project Structure

The development process of a hybrid system containing an FPGA and HPS design involves more files than an FPGA-only design. We followed the folder structure shown in Figure 6.6.2 to organize the Pyramic array project.

6.6.1 Hardware design files

"hw" contains all hardware-related files developed through the FPGA design of Chapter 5. Its structure is:

- "modelsim" contains all the simulation and testbench files implemented in *Modelsim*.

For instance, those files were used to generate Figure 5.9.

- "quartus" stores The "Pyramic_array" project developed in *Quartus Prime* project. Besides that, it contains all the compilation files. Moreover, the Pins assignments *.xsl* file for the DE1-SoC connections is stored in this folder.
- "hdl" contains the custom VHDL components developed through this project. The *DE1_SoC_top_level* entity with all the system interconnections is stored in this folder as well.

6.6.2 Software design files

All the software-related files written for initializing and programming the HPS processor are saved in "sw". In our design, we are just using the HPS processor. However, if Nios2 based applications were developed the "nios" folder can be used to store the programming files. The main subfolders of the "hps" directory are described below:

- "preloader" stores the preloader which boots the HPS as explained in Section 6.4.
- "u-boot" stores the U-boot preloader capable of loading the Linux kernel into the system.
- "linux" It stores the latest Linux sources. The "rootfs" contains all the configuration of the Linux kernel running on the system. The username and password can be configured in this file.
- "application" contains the software project developed in *Eclipse DS-5* platform to program the ARM processor. To control the Pyramic array system, the "Mic_Array_HPS" project has been created. Figure 6.6 shows the flow chart of the developed application. The red squares are mandatory steps in any software application for this design. The main steps are described below:
    1. Load the 'duration' and 'audio_folder' arguments to the main function. Otherwise, default values are used.
    2. Open physical memory devices from the 1GB DDR3 memory.
    3. Physical to virtual memory mapping for the OS Kernel.
    4. SPI System control. Data can be written and read from software to the *SPI_System* module to control its slave registers.
    5. Read data from DDR3 memory to a buffer.
    6. Save the microphones data in *.wav* format after the full acquisition is completed.
    7. *scp* microphones data.
    8. Virtual to physical memory mapping for the OS Kernel.
    9. Close physical memory devices.

The "sdcard" directory contains all the final targets needed to create a valid SD card from which the DE1-SoC can boot. The SD Card is composed of a *fat32* and *a2* of 512 MB and 24 GB respectively. The *fat32* contains all the necessary FPGA and system images to configure the HPS system when powering on the board. The *a2* contains the Linux root directory folders.

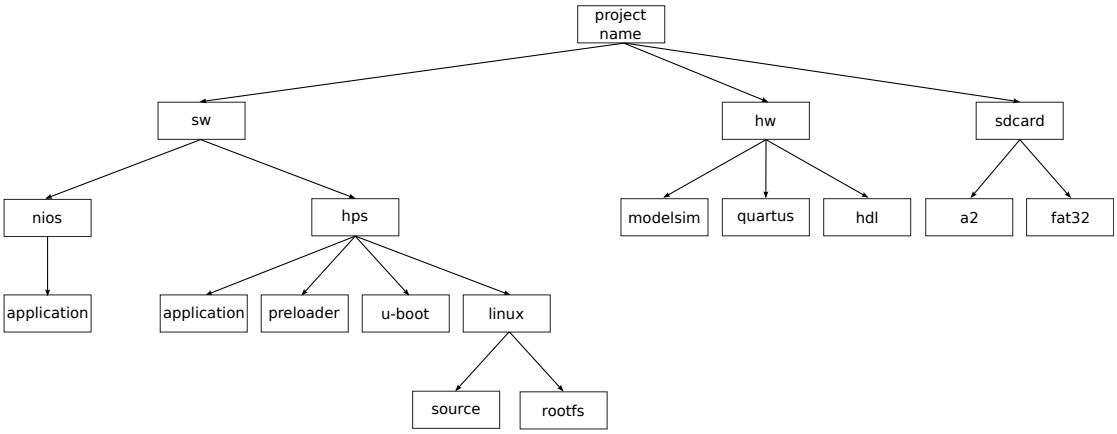


Figure 6.5 – Pyramic project structure

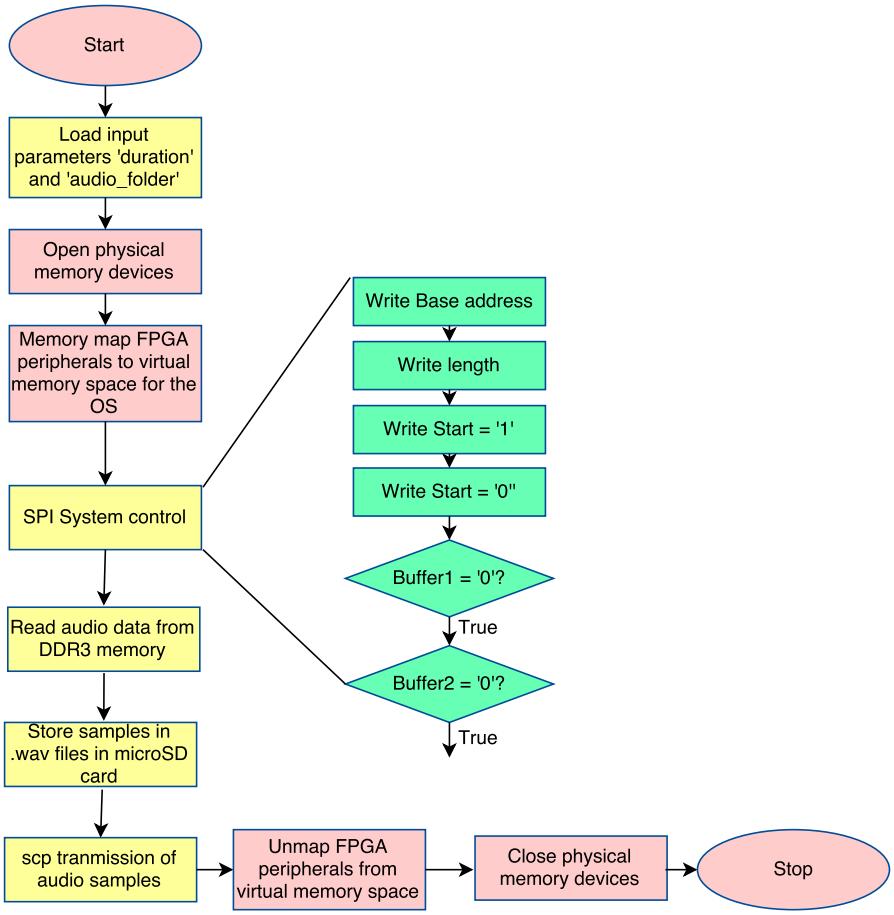


Figure 6.6 – *Mic_Array_HPS* application flow chart

6.7 Conclusions

Through this Chapter we explained the main features of the HPS part of the Cyclone V family devices. Combining HPS and FPGA system allows designing more reliable embedded system architectures. The Pyramic array is controlled by the HPS hardcore module.

The HPS allows fast communication between the FPGA and HPS sides, as well as a fast data transaction through SDRAM controllers. Moreover, the HPS can be programmed as a normal microcontroller and it can obtain internet thanks to sharing the connection with the host computer. Next Chapter will explain some results obtained from implementing the Pyramic array.

7 Results and algorithms implementation

7.1 Experiment conditions

The Pyramic array was tested in the Laboratory of Audiovisual Communications (LCAV) at EPFL, in the *INR019* Audio room. The test consisted of several recordings where audio files are played from different speakers distributed as in Figure 7.1. The Pyramic array was positioned in the position (0,0). The microphones are:

- *MISO_00*: from Microphone 0 to Microphone 7. Connected to J4 in Figure A.7.
- *MISO_01*: from Microphone 8 to Microphone 15. Connected to J4 in Figure A.7.
- *MISO_10*: from Microphone 16 to Microphone 23. Connected to J3 in Figure A.7.
- *MISO_11*: from Microphone 24 to Microphone 31. Connected to J3 in Figure A.7.
- *MISO_20*: from Microphone 32 to Microphone 39. Connected to J2 in Figure A.7.
- *MISO_21*: from Microphone 40 to Microphone 47. Connected to J2 in Figure A.7.

First, a frequency sweep generated by a Python script was played from the speakers. The excitation was an exponential sweep of 6 seconds with a 0,1 seconds margin from 100 Hz to 7000 kHz. The rest of the recordings were speech signals of 5 seconds duration each. Moreover, noise was recorded at the end. Appendix B explains where the audio recordings can be found in the DVD attached to this project.

The room conditions were 25.4 C and 57.4% humidity. The direction of arrival algorithms were implemented in Python 2.7.12. The positions of the microphones is presented in Appendix D.

7.2 Microphone Signals

Figure 7.3 displays the microphone outputs from the speech signal coming out from the fourth speaker. Only the signals from the 16 first microphones are displayed, i.e, the first two microphone arrays. The data is successfully recorded by each microphone and we can appreciate the delay and attenuation between microphones due to the different positions of

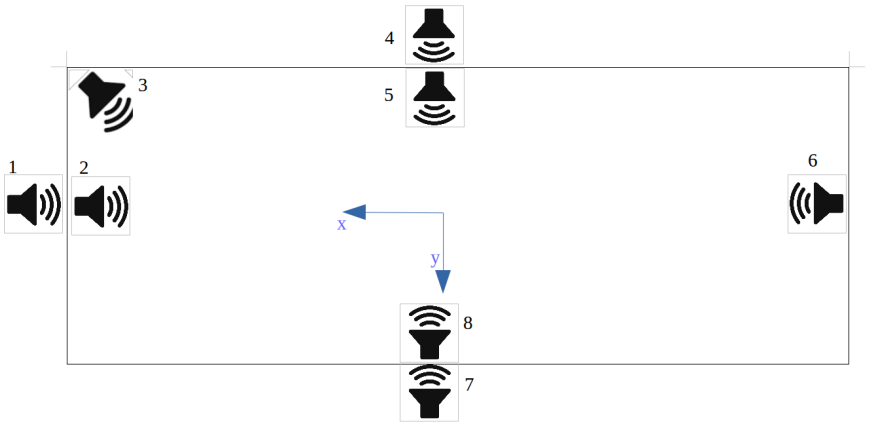


Figure 7.1 – Speakers set up and order

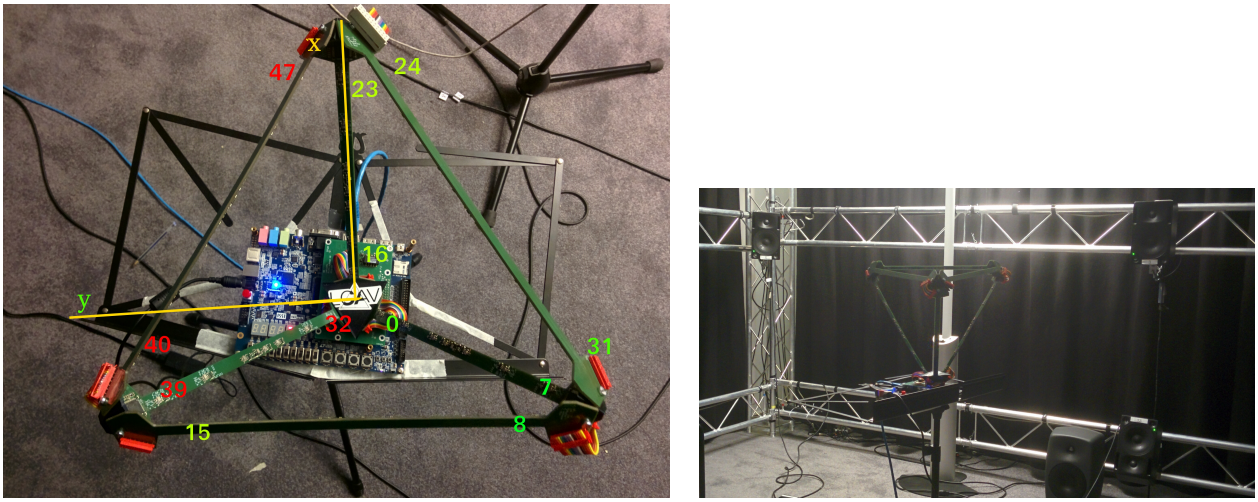


Figure 7.2 – Pyramic set up

the microphone transducers within the array.

Moreover, Figure 7.4 shows the spectrogram of a frequency sweep coming from Speaker 2 to *Microphone 0*. The microphone shows a homogeneous frequency response along all the frequency range.

7.3 SRP algorithm results

Finally, offlin direction of arrival algorithms developed at LCAV by Eric Bezzam and Robin Scheibler have been tested using the Pyramic data. A beamforming algorithm based on the Steered-Response-Power (SRP) method explained in Section 6.2 of [21] was chosen. For the results presented in this report, the algorithm was tested with a single speech signal coming

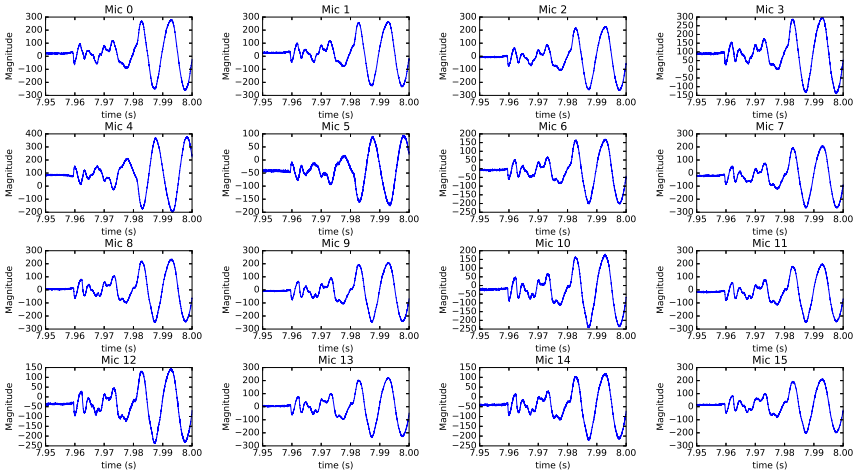


Figure 7.3 – Time waveforms of 50 ms for the first 16 microphones of the Pyramic array

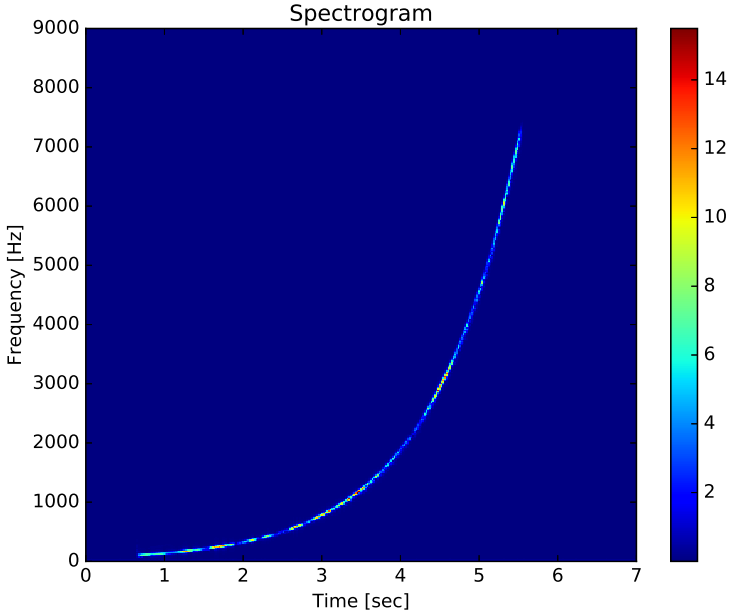
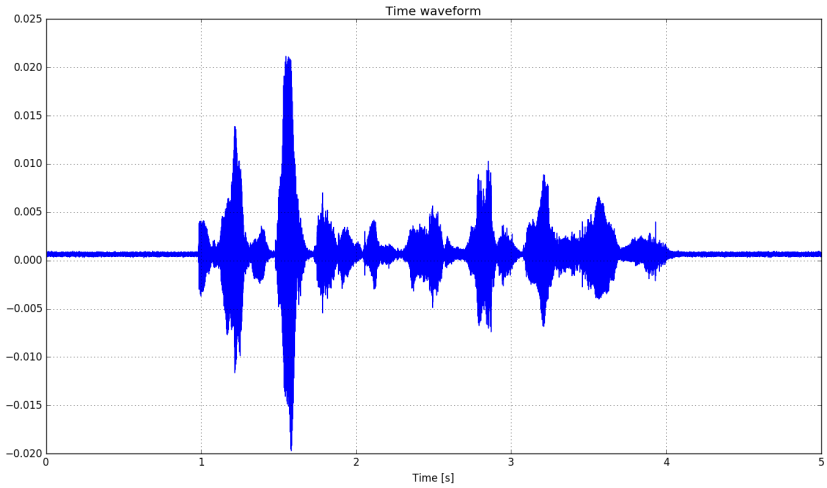


Figure 7.4 – Frequency sweep captured by *Microphone 0*.

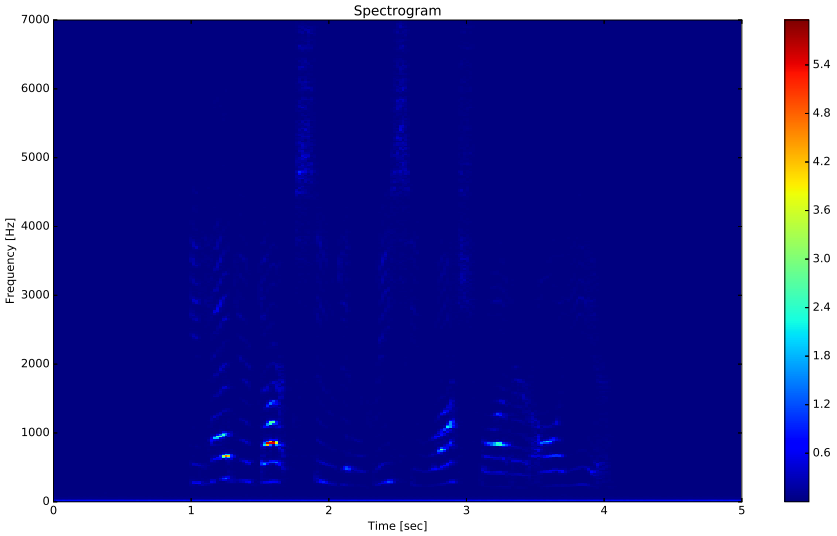
from Speaker 2. Figure 7.5 shows the spectrogram and the time waveform of the speech signal. Moreover, the time waveform allows to identify when the algorithms should proceed, i.e., when there are *voiced segments* in the record. For this test, a 2-D SRP direction of arrival implementation has been made. The *xy* plane is marked by the Pyramic array. The parameters of the algorithms are set as:

- Frequency range: 100 Hz to 7kHz
- Hop size: 1024
- FFT length: 1024
- Number of Snapshots: 25
- Starting time: 1,2 sec
- Far-field model

Finally, the results are displayed in Figures 7.6 and 7.7 in which a magnitude plot and polar coordinates of the direction of arrival response are displayed respectively. We can appreciate how the Pyramic array is able to detect the look-up direction ϕ which in this case corresponds to 0° (since Speaker 2 is located a that angle from the Pyramic array look-up direction). The length of the processed signal was 530 ms, while the processing time for SRP for 48 microphones and the parameters mentioned above is 2,798 seconds.



(a) Time waveform of speech signal captured by *Microphone 0*.



(b) Spectogram of speech signal captured by *Microphone 0*.

Figure 7.5 – In the top panel, the time representation of a speech signal. In the bottom panel, a spectogram representation of the same signal.

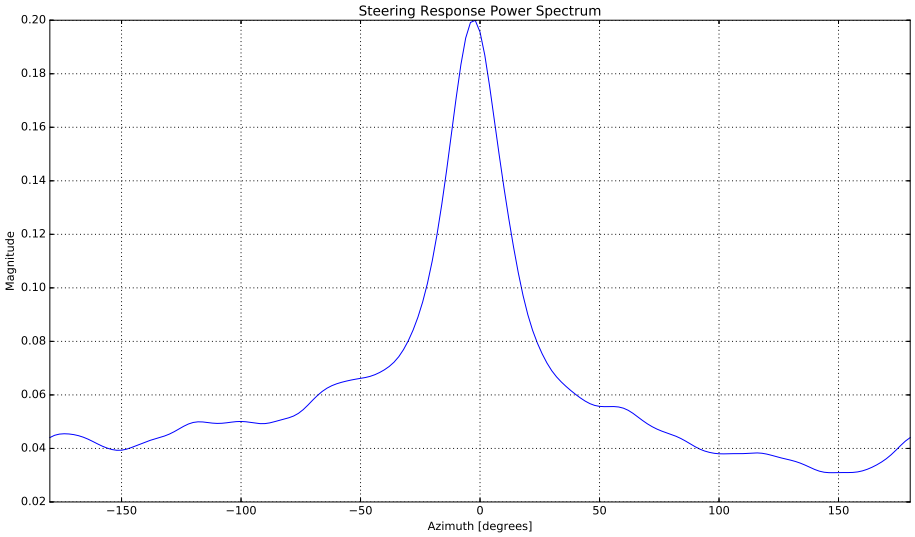


Figure 7.6 – Magnitude plot of SRP algorithms results for single source location.

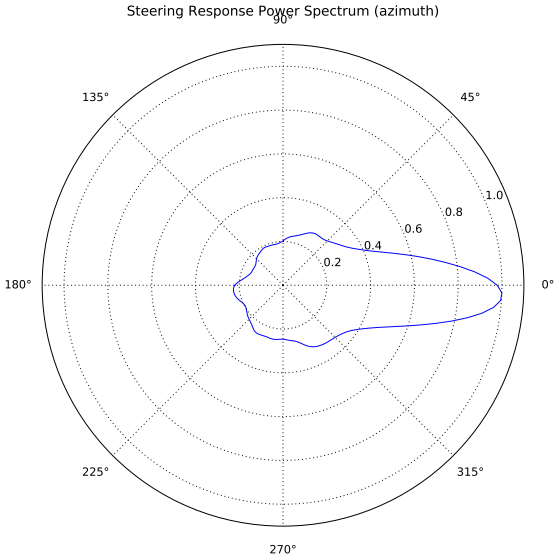


Figure 7.7 – Polar plot result of SRP implementation. The sound source is detected at $\phi = 0°$.

8 Conclusions and further directions for research

8.1 Conclusions

Using a revolutionary technology, the Pyramic microphone array, for capturing multi-channel audio signals, is able to acquire massive amount of acoustic information while implements real-time DSP hardware accelerators. The main goals of the project have been achieved and a functional and reliable system has been implemented.

In order to achieve our goal, we built the Pyramic array, a custom hardware component connected to a DE1-SoC board from Altera Cyclone V family. DE1-SoC device combines a Field Programmable Gate Array (FPGA) and a Hard Processor System (HPS), which allows building reliable embedded systems.

Pyramic delivers a flexible platform for implementing practical real-time DSP algorithms such as beamforming, speech-enhancement, and room shape recognition. The user can change the number of microphones used by the device and test the Pyramic array under different geometries more suitable for the desired application.

Moreover, the design is compatible with other Alteras Cyclone V devices that support HPS and FPGA development. However, depending on the chosen platform, some peripherals might not be available, such as the Audio Codec in the DE0-nano device.

Thanks to the fact that the project directory structure separates the *hw* and *sw* folders, the designer can develop independent hardware or software applications. For instance, in the current state of the design, the DE1-SoC acquires the data from the Pyramic array and saves it in a DDR3 memory, which can be accessed by software applications. This means that the designer can start further advanced software applications in *Eclipse DS5* for the Pyramic array without overlapping the design with the hardware system. On the other hand, for programming the FPGA part of the device the designer should have knowledge of hardware programming.

8.2 Further research

As a continuation of this work, the student should build custom components and hardware accelerators in order to perform real-time processing, which is the core purpose of the Pyramic array. In the software side, the designer can develop advanced software applications where, for instance, Python and C codes interact between them and control the FPGA part of the device.

Moreover, it should be necessary to characterize the microphones response and to analyze the SNR of each microphone to understand better the Pyramic array behavior. For example, one of the main problems encountered when designing the FIR filters was that for high frequencies the filter output overflows. According to 3.5, the microphones might have a non-uniform frequency response, which can affect the way the microphones data should be processed. Understanding better the microphones response will help the student to calibrate the design of the FIR filters. Alternative reason why the FIR filters might not work correctly is due to the coefficients width settings, which might generate overflow of the output data after MAC operations.

Additionally, the Altera FIR II IP Core allows coefficient reloading while the system is running. This can be very practical for real-time applications where the user wants to change the look-up direction of the system.

Another feature that can be improved from the system consists of developing a more complex WebServer interface for the Pyramic array. At the moment, the WebServer simply controls the input parameters of the software application and it is able to start the SPI acquisition.

In the end, the magic of the Pyramic array is that the possible development directions are so broad that the designer bears a lot of freedom to implement its own ideas. Combining software and hardware design gives complete control over the system behavior. The potential of this project for real time implementations is huge. Therefore, we encourage further research on this project to be driven with creativity and enthusiasm.

A Appendix: Schematics

The following circuits and schematics were designed by René Beuchat and Francisco Rojo at LAP, EPFL, during Spring 2015.

A.1 INMP504 conditioning circuit

Figure A.1 contains the schematic of the conditioning circuit for the INMP504 conditioning circuit for *'Microphone 0'* channel. The parameters of the circuit were calculated as follows

$$A = 1 + \frac{R_2}{R_1} = 1 + \frac{4.9k}{100} = 50$$
$$BW = \frac{GBW}{A} = \frac{1.2MHz}{50} = 24kHz$$
$$f_{high} = \frac{1}{2\pi * C_3 * R_{f1}} = \frac{1}{2\pi * 1\mu F * 10K} = 15.9Hz,$$

where f_{low} corresponds to the decouping capacitor cut off frequency, A represents the system gain and BW is the system bandwidth. Moreover, decoupling capacitors C_{22} and C_{23} were chosen according to the documentation of the OPA170 amplifier [25].

A.2 Schematic for the ADC connection diagram

Figure A.2 shows the connection diagram of the AD7606 converter designed according to [4].

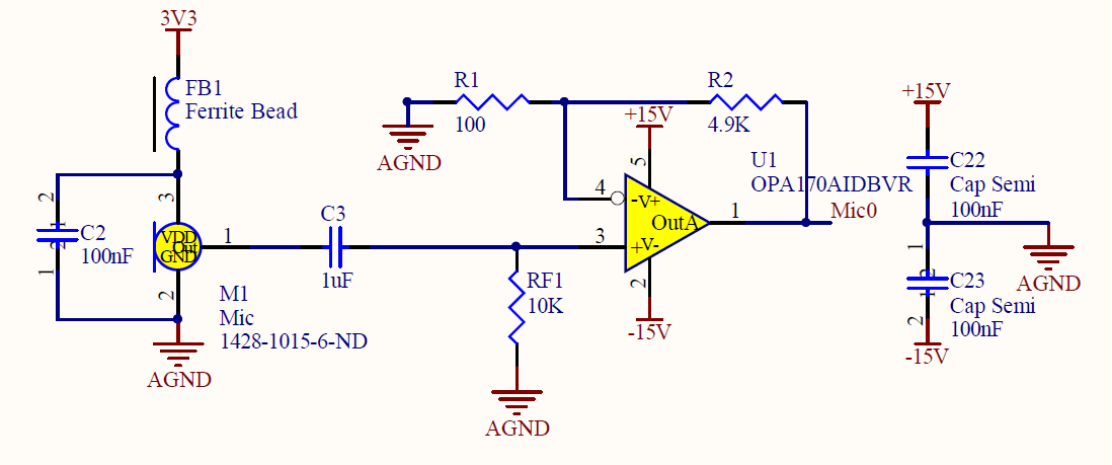


Figure A.1 – Schematic of the preamflifier circuit for one microphone channel

A.3 Microphones power supply and daisy-chain connection

Figure A.3.a displays the circuit diagram of the TPS78033 power converter from 5V to 3,3V, which serves as power supply for the microphones. On the left panel, Figure A.3.b represents the connectors schematic within a single microphone array.This connection layout allows daisy chain wiring. The connectors placed in *J1* and *J2* pins are AMP 1-215079-4 female connectors.

A.4 Pyramic array PCB

Figure A.4 shows a top view from the PCB of a single microphone array board. Components such as microphones and Op Amps were mounted on the boards using SMT (surface mount technology) placement systems at HEPIA, in Geneva. The joint between PCBs is a plastic piece with pyramidal shape designed and 3-D printed at LAP.

A.5 PCB for Power Supply and GPIO connectors

The PCB displayed at Figure A.7 powers the Pyramic array and connects it to the GPIO pins of the DE1-SoC device. The connectors placed in *J2*, *J3* and *J4* pins are AMP 1-215079-4 female connectors, which are compatible with AMP 8-215083 male connectors. The labels of the connectors pins are depicted in Figure A.5. Left panel of Figure A.6 represents the correspondence between the pin labels on the Pyramic array and the *GPIO_0* port of the FPGA. The right panel shows the wiring of the NTA0515mc boost power converter from Murata Power Solution. It converts the 5 V from the DE1-SoC to ±15V for the Op Amps of the Pyramic array. The filter was designed according to the datasheet presented in [20]. The load value of 4,37k was chosen so the current through the boost converter remains lower than 33 mA.

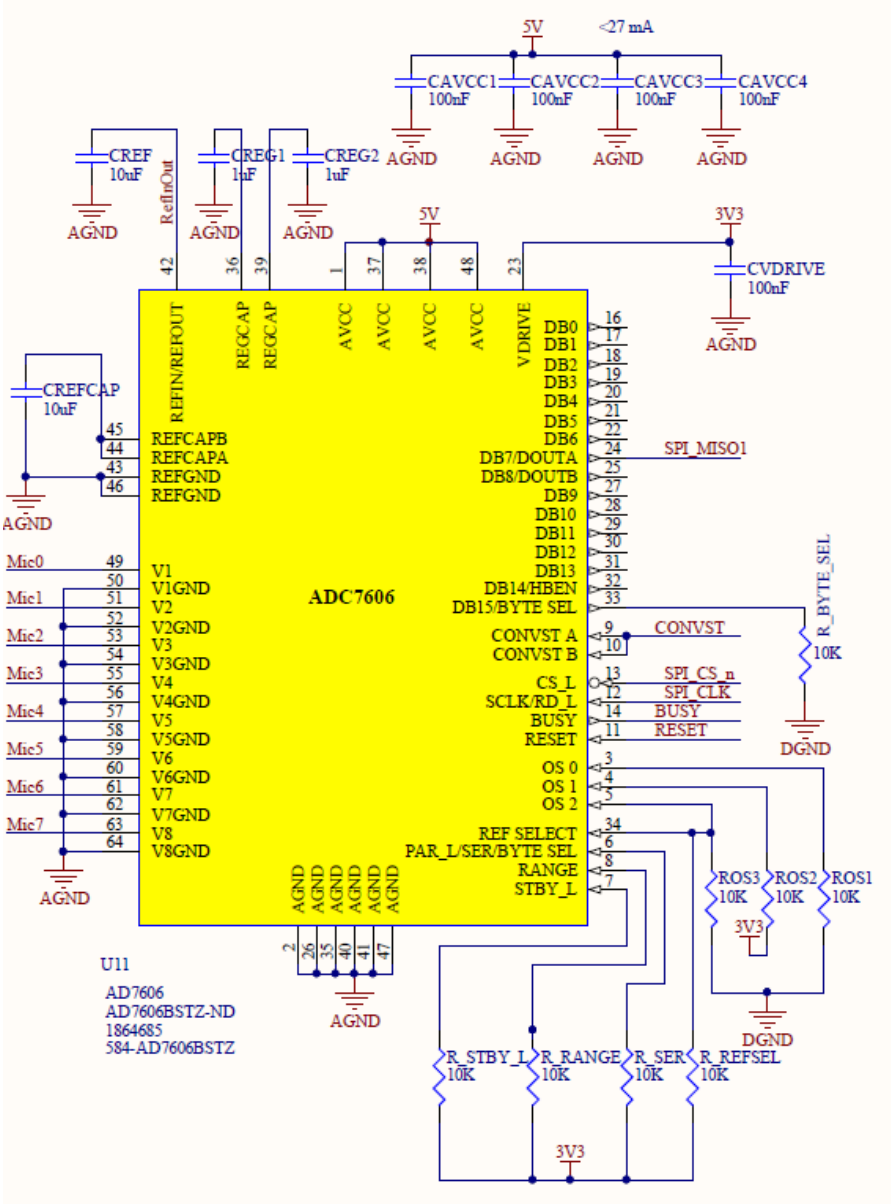
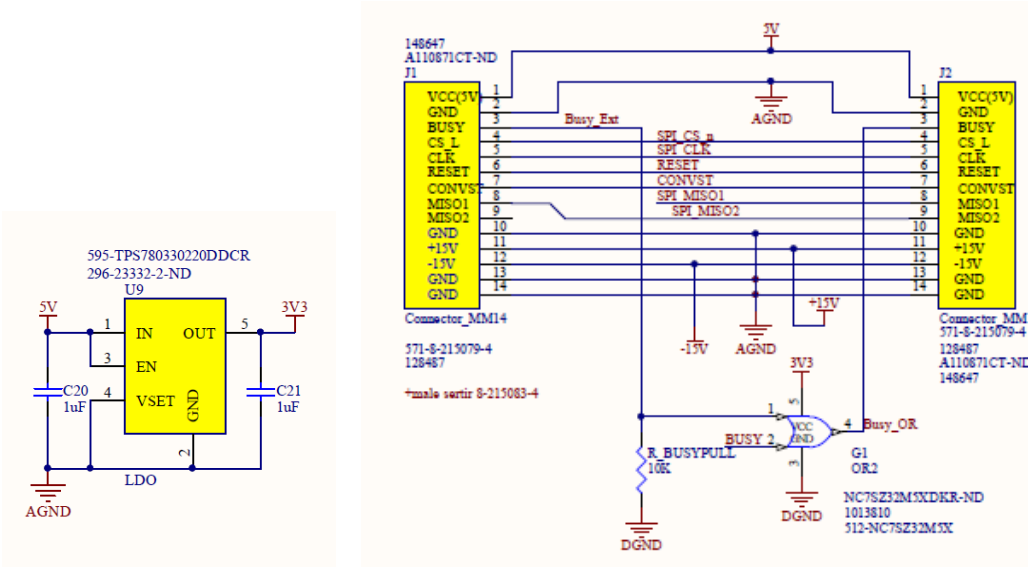


Figure A.2 – Connections diagram of the AD7606



(a) TPS78033 power converter from 5V to 3.3V.

(b) Daisy chain connection diagram between Pyramic PCBs

Figure A.3

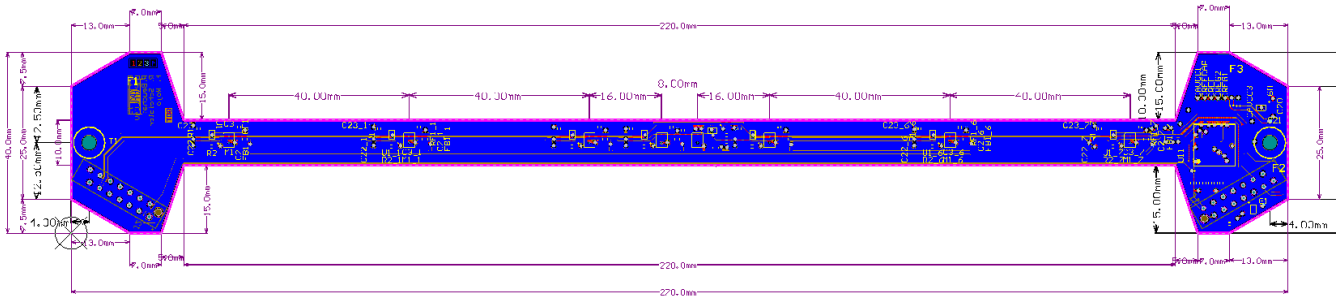


Figure A.4 – Pyramic single array PCB board design

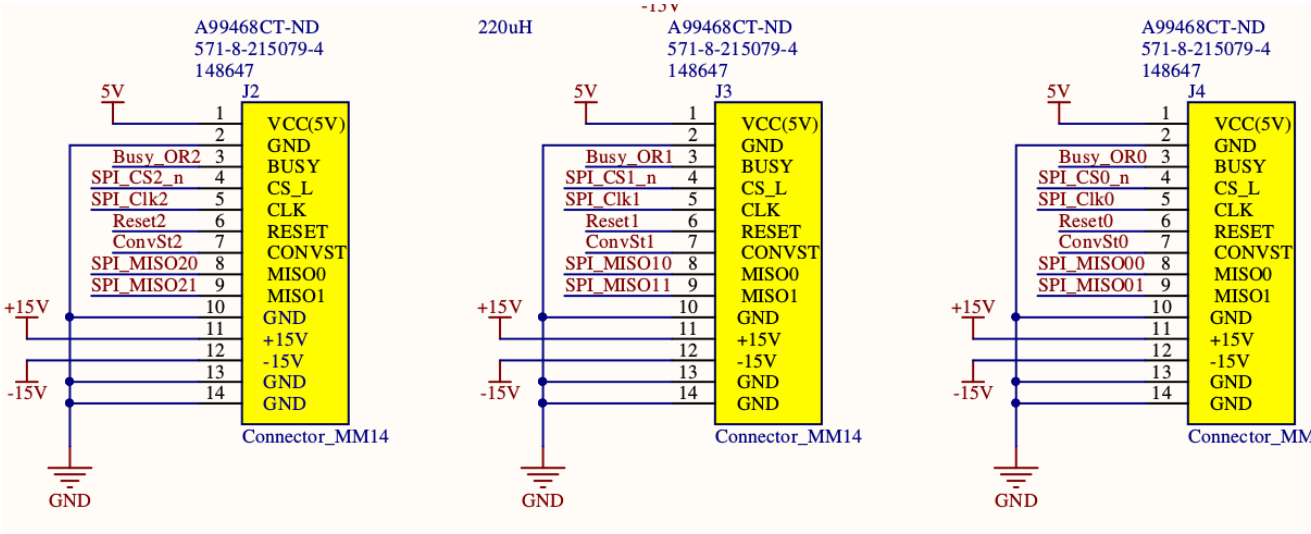
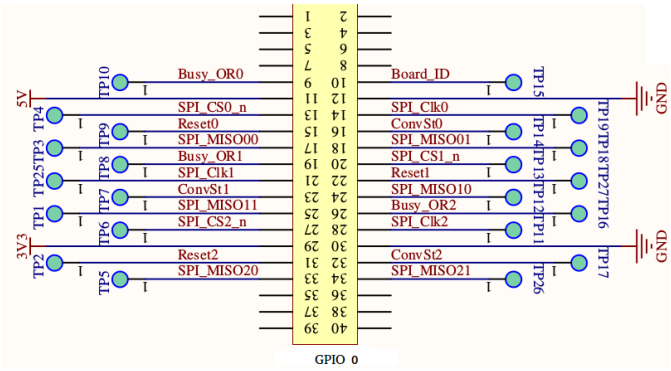


Figure A.5 – Pyramic array input pins labels and connectors



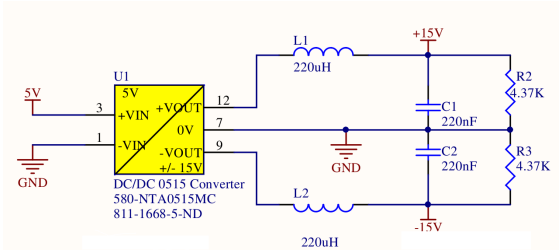


Figure A.6 – In the left panel the GPIO pins labels. In the right pannel the schematic of the Murata Power Converter [20] connections

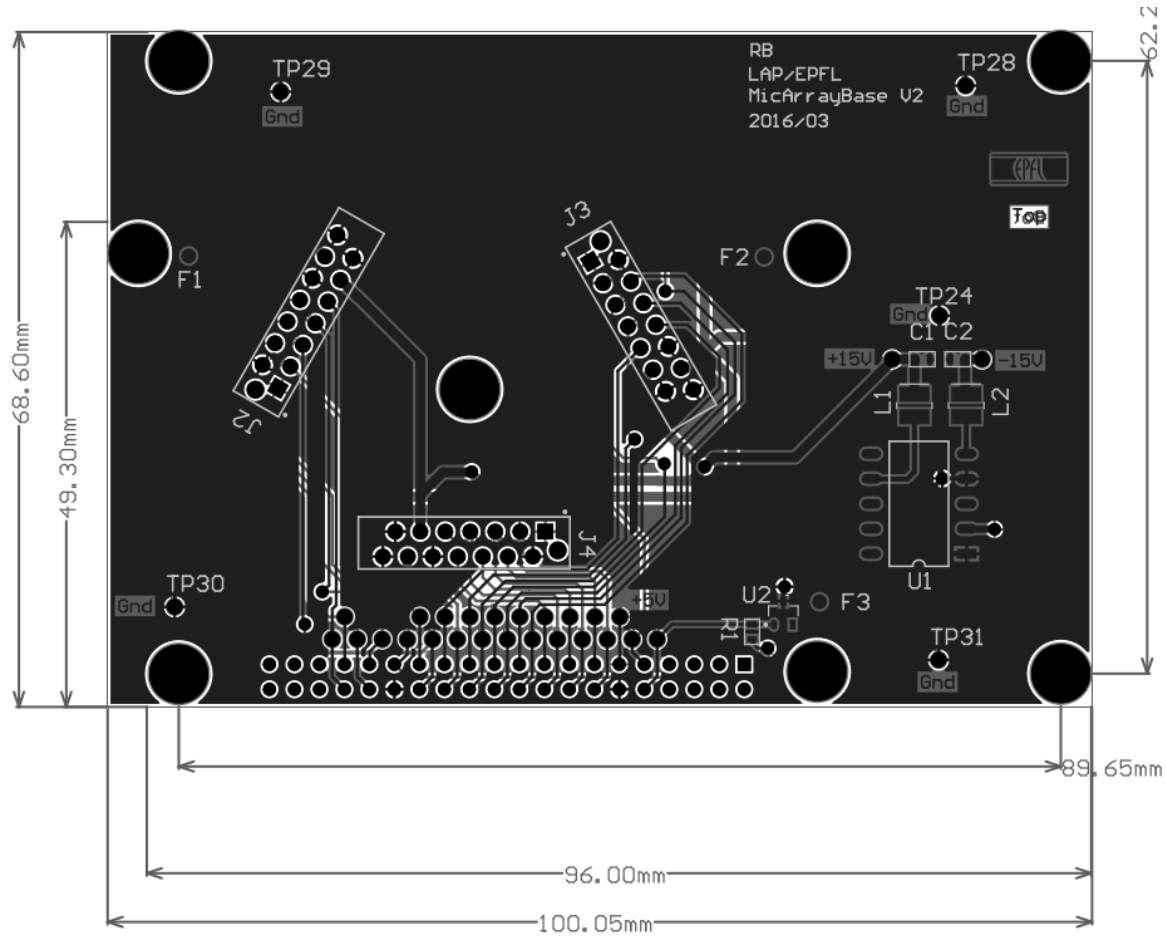


Figure A.7 – Pyramic array base power supply

B Appendix: Development Tools

For hardware development, you can configure the HPS and connect your soft logic in the FPGA fabric to the HPS interfaces using the Qsys system integration tool in the Quartus Prime software. For software development, the ARM-based SoC devices inherit the rich software development ecosystem available for the ARM Cortex-A9 MPCore processor. The software development process for Altera SoCs follows the same steps as those for other SoC devices from other manufacturers.

B.1 Hardware

The project was developed upon Terasic DE1-SoC board, nevertheless it is compatible with other Cyclone V SoC device which gathers the characteristics described in Section 4.3.1.

B.2 Software

The following programs were installed and the operating system used to develop the design was **UBUNTU 16.04.1**:

- *Quartus Prime 16.0 Lite Edition*
- *ModelSim-Altera Starter Edition 10.4d*
- *SoC Embedded Design Suite (SoC EDS): Eclipse for DS-5 v5.23.1*

B.3 Attached multimedia files

The attached DVD is composed of the following sections:

- Audio: It contains the audio recordings for the results of Chapter 7.
- Project: It contains all the design files compressed as described in Section 6.6.
- Logic Analyzer: Salae logic simulation of the SPI System.

- Schematics: It contains *pdf* files of the system schematics.

C Appendix: FIR Filters coefficients

143 coefficients for one FIR filter. In the same file, copy the coefficients as many times as the number of banks you want to implement (usually 48). A comma represents a separation between coefficients, while a space is a separation between filter banks.

```
# banks: 48
# coeffs: 143
-1.92836858E-10, 2.7661817E-9, 1.56636394E-9, -6.68390369E-9, -7.40206242E
  -9, 1.99785879E-8, 2.64397032E-8, -5.82629366E-8, -7.94253102E-8,
  1.56116056E-7, 2.11062408E-7, -3.84096067E-7, -5.10505862E-7, 8.76751151E
  -7, 1.14446625E-6, -1.87627082E-6, -2.40792169E-6, 3.79720804E-6,
  4.79813881E-6, -7.31812953E-6, -9.11797671E-6, 1.35058995E-5, 1.66140745E
  -5, -2.39773828E-5, -2.91543402E-5, 4.11015628E-5, 4.94471105E-5,
  -6.82424538E-5, -8.13015536E-5, 1.10040013E-4, 1.29925719E-4, -1.72723075E
  -4, -2.02255751E-4, 2.64446021E-4, 3.07308265E-4, -3.95640742E-4,
  -4.56549135E-4, 5.79379167E-4, 6.64278091E-4, -8.31751644E-4, -9.48042281E
  -4, 0.00117228603, 0.00132911754, -0.00162446675, -0.00183314038,
  0.00221647127, 0.00249104855, -0.00298233982, -0.00334061758,
  0.00396396701, 0.00442911344, -0.00521462334, -0.00581802684,
  0.00680534563, 0.0075917674, -0.00883688062, -0.0098742217, 0.0114629847,
  0.0128620038, -0.0149388678, -0.0168965785, 0.0197316777, 0.022639064,
  -0.0268084979, -0.031568432, 0.0385531058, 0.0478084522, -0.0628254567,
  -0.088722802, 0.149493423, 0.44977945, 0.44977945, 0.149493423,
  -0.088722802, -0.0628254567, 0.0478084522, 0.0385531058, -0.031568432,
  -0.0268084979, 0.022639064, 0.0197316777, -0.0168965785, -0.0149388678,
  0.0128620038, 0.0114629847, -0.0098742217, -0.00883688062, 0.0075917674,
  0.00680534563, -0.00581802684, -0.00521462334, 0.00442911344,
  0.00396396701, -0.00334061758, -0.00298233982, 0.00249104855,
  0.00221647127, -0.00183314038, -0.00162446675, 0.00132911754,
  0.00117228603, -9.48042281E-4, -8.31751644E-4, 6.64278091E-4, 5.79379167E
  -4, -4.56549135E-4, -3.95640742E-4, 3.07308265E-4, 2.64446021E-4,
  -2.02255751E-4, -1.72723075E-4, 1.29925719E-4, 1.10040013E-4, -8.13015536E
  -5, -6.82424538E-5, 4.94471105E-5, 4.11015628E-5, -2.91543402E-5,
  -2.39773828E-5, 1.66140745E-5, 1.35058995E-5, -9.11797671E-6, -7.31812953E
  -6, 4.79813881E-6, 3.79720804E-6, -2.40792169E-6, -1.87627082E-6,
  1.14446625E-6, 8.76751151E-7, -5.10505862E-7, -3.84096067E-7, 2.11062408E
  -7, 1.56116056E-7, -7.94253102E-8, -5.82629366E-8, 2.64397032E-8,
  1.99785879E-8, -7.40206242E-9, -6.68390369E-9, 1.56636394E-9, 2.7661817E
  -9, -1.92836858E-10, -9.9644808E-10
```

D Appendix: Pyramic Microphones positions

```
# Microphones positions (in cm) for Pyramic array for compilation in Python.
import numpy as np

pos0  = [3.51, 0, 1.13]
pos1  = [5.83, 0, 4.39]
pos2  = [8.15, 0, 7.65]
pos3  = [9.08, 0, 8.95]
pos4  = [9.55, 0, 9.6]
pos5  = [10.48, 0, 10.9]
pos6  = [12.8, 0, 13.5]
pos7  = [15.12, 0, 17.42]

pos8  = [13.7, -2.5, 21.3]
pos9  = [10.23, -4.5, 21.3]
pos10 = [6.77, -6.5, 21.3]
pos11 = [5.38, -7.3, 21.3]
pos12 = [4.69, -7.7, 21.3]
pos13 = [3.3, -8.5, 21.3]
pos14 = [-0.15, -10.5, 21.3]
pos15 = [-3.6, -12.5, 21.3]

pos16 = [-1.75, 3.04, 1.13]
pos17 = [-2.91, 5.05, 4.39]
pos18 = [-4.07, 7.06, 7.65]
pos19 = [-4.54, 7.87, 8.95]
pos20 = [-4.77, 8.27, 9.6]
pos21 = [-5.24, 9.07, 10.9]
pos22 = [-6.4, 11.08, 14.16]
pos23 = [-7.56, 13.1, 17.42]

pos24 = [-3.6, 12.5, 21.3]
pos25 = [-0.5, 10.5, 21.3]
pos26 = [3.3, 8.5, 21.3]
pos27 = [4.69, 7.7, 21.3]
pos28 = [5.38, 7.3, 21.3]
pos29 = [6.77, 6.5, 21.3]
pos30 = [10.23, 4.5, 21.3]
pos31 = [13.7, 2.5, 21.3]

pos32 = [-1.75, -3.04, 1.13]
pos33 = [-2.91, -5.05, 4.39]
pos34 = [-4.07, -7.06, 7.65]
pos35 = [-4.54, -7.87, 8.95]
pos36 = [-4.77, -8.27, 9.6]
```

```
45 pos37 = [-5.24, -9.07, 10.9]
46 pos38 = [-6.4, -11.08, 14.16]
47 pos39 = [-7.56, -13.1, 17.42]
48
49 pos40 = [-9, -10, 21.3]
50 pos41 = [-9, -6, 21.3]
51 pos42 = [-9, -2, 21.3]
52 pos43 = [-9, -0.04, 21.3]
53 pos44 = [-9, 0.04, 21.3]
54 pos45 = [-9, 2, 21.3]
55 pos46 = [-9, 6, 21.3]
56 pos47 = [-9, 10, 21.3]
57
58 # Create Rotation matrix
59 alpha = 120.0*np.pi/180;    # Alpha: Angle to rotate
60 R = np.array([[np.cos(alpha),-np.sin(alpha),0],[np.sin(alpha),np.cos(alpha)
     ,0],[0,0,1]]);
61
62 # construct mic array
63 L = np.array([pos0, pos1, pos2, pos3, pos4, pos5, pos6, pos7, pos8, pos9,
     pos10, pos11, pos12, pos13, pos14, pos15
64       ,pos16, pos17, pos18, pos19, pos20, pos21, pos22, pos23, pos24, pos25,
           pos26, pos27, pos28, pos29, pos30, pos31
65       ,pos32, pos33, pos34, pos35, pos36, pos37, pos38, pos39, pos40, pos41,
           pos42, pos43, pos44, pos45, pos46, pos47],dtype=float)/100
66
67 # Multiply and rotate by alpha (=120  )
68 L = np.dot(L,R)
```

Bibliography

[1] Altera, 101 Innovation Drive San Jose, CA, 95134. *Audio core for Altera DE-Series Boards*, 10 2015.

[2] Altera. *Avalon Interface Specifications*, 2015.

[3] Altera, 101 Innovation Drive San Jose, CA, 95134. *FIR II IP Core User Guide*, 10 2015.

[4] Analog Devices. *8-/6-/4-Channel DAS with 16-Bit, Bipolar Input, Simultaneous Sampling ADC*, 2012.

[5] Altera Corporation. *" Cyclone V Device Handbook, Volume 3: Hard Processor System Technical Referene Manual," 31 July 2014*, 2014.

[6] Altera Corporation. *"Nios II Gen2 Processor Reference Guide"*, 2015.

[7] Altera Corporation. *"Cyclone V Device Overview"*, 2016.

[8] Altera Corporation. *"HPS SoC Boot Guide - Cyclone V SoC Development Kit"*, 2016.

[9] Altera Corporation. *"SCFIFO and DCFIFO IP Cores User Guide"*, May 2016.

[10] Eugene Weinstein, Kenneth Steele, Anant Agarwal, and James Glass. Loud: A 1020-node modular microphone array and beamformer for intelligent computing spaces. MIT Computer Science and Artificial Intelligence Laboratory, 2004.

[11] Fairchild Semiconductor. *NC7SZ32 TinyLogic® UHS Two-Input OR Gate*, 2009.

[12] Florian Parrodin, Janosch Nikolic, Joel Busset and Roland Siegwart. Design and calibration of large microphone arrays for robotic applications. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4596 – 4601, 2012.

[13] InvenSense. *Ultra-Low Noise Microphone with Bottom Port and Analog Output*, 2014.

[14] Ivan Dokmanic, Reza Parhizkar, Andreas Walther, Yue M. Lu, and Martin Vetterli. Raking the cocktail party. *PNAS*, 110(30), 2013.

[15] Ivan Dokmanic, Robin Scheibler, Martin Vetterli. Raking the cocktail party. *IEEE journal of selected topics in signal processing*, 9(5), 2015.

[16] Philipos C. Loizou. *Speech Enhancement*. 3 edition, 2007.

[17] Uwe Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. 3 edition, 2007.

[18] René Beuchat Sahand Kashani-Akhavan. *SoC-FPGA Design Guide*.

[19] M.I. Skolnik. *Introduction to Radar Systems*. McGraw-Hill, New York, 1980.

[20] Murata Power Solutions. *"NTA Series Isolated 1W Dual Output SM DC/DC Converters"*.

[21] Ivan Tashev. *Sound Capture and Processing*. 1 edition, 2009.

[22] Terasic Technologies. *"Terasic- DE Main Boards - Cyclone - DE1-SoC Board," [Online].Available:*.

[23] Altera University Program Terasic Technologies. *DE1-SoC User Manual*, 2014.

[24] Texas Instrument. *TPS780xx 150-mA Low-Dropout Regulator, Ultralow-Power, IQ 500 nA With Pin-Selectable, Dual-Level Output Voltage*, 2007.

[25] Texas Instruments. *OPAx170 36-V, Single-Supply, SOT553, Low-Power Operational Amplifiers Value Line Series*, 2011.

[26] WM8731. *"Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates"*.

JUAN AZCARRETA ORTIZ

Lausanne, Switzerland | From: Vitoria-Gasteiz, Spain | Born: 12/11/1992
jazcarretao@gmail.com | (+34) 669 86 88 46 | Status: Single

WORK EXPERIENCE

09/2016-08/2017 (exp.)	**NTT Communications SL:** Signal Processing Group.	Kyoto, Japan
02/2016-07/2016 (exp.)	**EPFL LCAV and LPA laboratories:** An FPGA based platform for many-channel audio acquisition and algorithms implementation.	Lausanne, Switzerland
02/2015-06/2015	**Broadcom Networks Spain:** Formation in Analog Microelectronics with the aim of increasing knowledge in the design of high performance analog IC.	Barcelona, Spain
02/2014-07/2014	**Basque Center on Cognition, Brain and Language, BCBL**: Model physiological related BOLD signal variations in fMRI experiments.	San Sebastian, Spain
01/2014-06/2014	**Power harvesting from human body at University of Navarra.**	San Sebastian, Spain
07/2013	**Metalúrgicas Alavesas:** PLC programming and control.	Vitoria, Spain
2012	**CEIT**: Undergraduate researcher in the Automations and Electronics department.	San Sebastian, Spain

EDUCATION

09/2015-07/2016 (exp.)	**MS of Science in Electrical Engineering** (UPC.EUROPA program) École Polytechnique Fédérale de Lausanne (EPFL)	Lausanne, Switzerland
09/2014-06/2015	**Master's Degree in Electronics Engineering (MEE)** at Universitat Politécnica de Catalunya (UPC).	Barcelona, Spain
09/2010-07/2014	**BSc in Industrial Electronics Engineering** at University of Navarra, Ranking: **1st** out of 14.	San Sebastian, Spain
Fall 2013	**Electric Engineering** at Cal Poly, Exchange Program.	California**,** USA
Summer Course 2013	**Multimedia Signal and Information Processing**: Aalborg University. Machine learning for speech recognition.	Aalborg, Denmark

ACADEMIC BACKGROUND

02/2016-05/2016	**Introduction to the IoT and Embedded Systems** (UCI, Irvine)	MOOC, Coursera
03/2014-04/2014	**Statistical Analysis of fMRI data** (Johns Hopkins University)	MOOC, Coursera
09/2013-12/2013	**Audio Engineering Club:** Fuzz guitar pedal	California, USA
07/2013	**Anthem construction and design with CST Studio**	San Sebastian, Spain
01/2013-04/2013	**Management and Leadership Skills**	San Sebastian, Spain
02/2013	**Introduction to Digital Sound Design** (Emory University)	MOOC, Coursera
05/2012	**Product Design and Development**	San Sebastian, Spain

AWARDS

2016	**Vulcanus in Japan Programme Scolarship**: 1-year EU students exchange in Japan. **EU-Japan Centre for Industrial Cooperation**
2015	**AGAUR Scolarship 2015:** for mobility programs for outstanding students in Catalonia.
2014	**KutxaBank Award for Academic Excellence**: awarded to the most outstanding graduates in Gipuzkoa.

LANGUAGES

Spanish Mother Tongue | **English** TOEFL 98 | **German** A1
Basque Mother Tongue | **French** B1

PROGRAMMING LANGUAGES

C/C++, Java, Matlab, Python, VHDL

VOLUNTEERING WORK

2010-2014	ASPACE: Spanish Association of cerebral palsy sufferers.	San Sebastian, Spain
06/2013	FSL INDIA: Work Camp in construction and education fields.	Dharamsala, India

ADDITIONAL EXPERIENCE

13/03/2016-18/03/2016	MunEPFL: Russian Federation Delegate at Harvard WorldMun, UNHRC	Rome, Italy
06/11/2015-08/11/2015	MunEPFL: Germany delegate at ManMun, SPECPOL	Manchester, UK
2014	**Founder of AUDIO TECHNOLOGIES TECNUN (ATT).** Ebow Project: Electronic design and construction of an electronic bow.	San Sebastian, Spain
2013	**Class Representative** of Electronics Engineering.	San Sebastian, Spain

Hobbies: Basketball – Music (guitar, accordion, bass) - Reading- Diving- Ski- Kendo- Make it Happen.