Exploring FPGA Dynamic Reconfiguration for Sound Source Localization and Tracking Algorithms

By

Christian Serge Ibala

MS

A thesis submitted for the Degree of Doctor of Philosophy (PhD)

Department of Electronic and Computer Engineering
University of Limerick
Limerick
Ireland

Supervisors: Prof. Khalil Arshak, Dr. Ciaran MacNamee

Submitted to the University Of Limerick
December 2013
DECLARATION

I hereby declare that this work is entirely of my own doing and that it has not been submitted as an exercise for a degree at any other University.

_____________________________________

Serge Christian Ibala
ACKNOWLEDGEMENTS

Firstly, I would like to thank Jesus for giving me health, strength and courage when the task and obstacles appeared daunting and insurmountable. He has been with me the whole journey and I also would like to thank Jesus for allowing me to trust in him.

I wish to say a very special thank you to my supervisors: Professor Khalil Arshak and Doctor Ciaran MacNamee, for their guidance and advice.

I wish to thank Professor Carlos Valderrama, for assisting me in developing, fine-tuning and publishing portions of my PhD. His lab was the perfect place for me to do research. It was a pleasure working with Professor Carlos Valderrama.

I wish to thank Dr Sosthene Nicaise Ibala for his technical assistance to interface the FPGA results to National Instrument LabView software for a better way to visualize the results.

I wish to thank my wife Loreen Faye Ibala, for standing by me and believing in me.

Finally and not the least I would like to thank all my family {Solange Patricia, Regine Gracia, Mireille Claudia, Patrick Yvon and Franck Olivier} for all their support and love. I could not have done it without them. They have helped me forget my difficulties and brought a lot of laughter into my life during this time.
ABSTRACT

The ever-increasing complexity of new electronic systems designs, combined with the need for very high performance and low power consumption have driven technology development, methodologies and design flows, including the widespread use of reconfigurable FPGA technologies in systems design. The in-system reconfigurability of FPGAs offers the possibility of reconfiguring the device for different functionality without halting the application. Adding this functionality requires special design flows.

This thesis first explores the use of different FPGA reconfigurable design flows, using a sample design, which includes a sensor and wireless transmitter along with the FPGA. Two reconfigurable flows were investigated, Dynamic Partial Reconfiguration (DPR) and Multi-Boot (MB). These flows were also used in FPGA implementations of algorithms for the localization and tracking of sound sources. It is shown that the less-studied Multi-Boot flow has some unexpected advantages over the better-known DPR technique.

Applications based on sound source localization and tracking require specific algorithms, which can adapt themselves to disturbing noise or the mobility of the target. This work proposes a novel hybrid algorithm with the ability to adapt its search spectrum according to the target movement. This property allows a good balance between required power processing and real-time response, especially when implemented on a FPGA. Moreover, the algorithm fits quite well on a Dynamic Partial Reconfiguration (DPR) or Multi-Boot (MB) strategy. The approach combines the Generalized Cross Correlation (GCC) and the Delay and Sum Beamforming (DSB) algorithms in such a manner that reductions of more than 80% in DSB computation can be obtained while preserving precision and improving real-time capabilities. The novel algorithm is also applied to a number of application topics such as speaker recognition and multiple sound source tracking that use different localization and beamforming techniques. The performance of the author’s hybrid algorithm in these applications is studied and it is shown to retain its advantages in these cases.
GLOSSARY OF TERMS

ASIC          (Application Specific to Integrated Circuits)
CIC  (Cascade Integrator Comb)
CMOS          (Complementary Metal-Oxide Semiconductor)
CMT  (Clock Management Tile)
CPU  (Central Processor Unit)
DOA  (Direction of arrival)
DFT   (Discrete Fourier Transform)
DSB   (Delay Sum Beamforming)
DSP   (Digital Signal Processing)
EDK          (Embedded development Kit)
FOV   (Field of View)
FSK                         (Frequency shift Keying)
FSM          (Finite State Machine)
HDL         (Hardware Description Language)
FFT   (Fast Fourier Transform)
FIR   (Finite Impulse Response)
FPGA       (Field Programmable Gate Array)
GCC (Generalized Cross Correlation)
GPP          (General Purpose Processor)
GPS          (Global Positioning System)
ICAP        (Integrated Configuration Access Port)
ICON        (Integrated Controller)
IFFT  (Inverse Fast Fourier Transform)
ILA   (Integrated Logic Analyzer)
ISE           (Integrated Software Environment)
LABVIEW  (Laboratory Virtual Instrument Engineering Workbench)
LUT   (Look up Table)
MEMS  (Miniature Electro-Mechanical Systems)
MUSIC  (Multiple Signal Classification)
MVDR  (Minimum Variance Distortionless Response)
NOC  (Network on Chip)
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOSS</td>
<td>(Number of Small Squares)</td>
</tr>
<tr>
<td>PHAT</td>
<td>(Phase Transform)</td>
</tr>
<tr>
<td>SCOT</td>
<td>(Smoothed Coherence Transform Weighting Function)</td>
</tr>
<tr>
<td>SNR</td>
<td>(Signal-to-Noise Ratio)</td>
</tr>
<tr>
<td>SNRG</td>
<td>(Signal-to-Noise Ratio Gain)</td>
</tr>
<tr>
<td>SPI</td>
<td>(Serial Peripheral Interface)</td>
</tr>
<tr>
<td>SRP</td>
<td>(Steered Response Power)</td>
</tr>
<tr>
<td>SS</td>
<td>(Small Square)</td>
</tr>
<tr>
<td>TDOA</td>
<td>(Time Difference of Arrival)</td>
</tr>
<tr>
<td>USB</td>
<td>(Universal Serial Bus)</td>
</tr>
<tr>
<td>VAD</td>
<td>(Voice Activity Detector)</td>
</tr>
<tr>
<td>VIO</td>
<td>(Virtual Input Output)</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

DECLARATION .................................................................................................................................. I
ACKNOWLEDGEMENTS .................................................................................................................. II
ABSTRACT .................................................................................................................................... III
GLOSSARY OF TERMS ................................................................................................................ IV
LIST OF FIGURES ........................................................................................................................ X
LIST OF TABLES .......................................................................................................................... XIII

CHAPTER 1 INTRODUCTION ...................................................................................................... 1
1.1. Motivation for this Research .............................................................................................. 1
1.2. Contribution of the Thesis ................................................................................................. 2
1.3. First Case Study .................................................................................................................. 4
1.4. Second Case Study .............................................................................................................. 6
1.5. Scientific Output of this Work ........................................................................................... 7
1.6. Outline of the Thesis .......................................................................................................... 9
1.7. Summary ............................................................................................................................ 10

CHAPTER 2 Reconfigurable Flows to Increase Architectures Flexibility .............................. 13
2. Declaration and Presentation ............................................................................................... 13
2.1. Purpose of this Chapter ..................................................................................................... 13
2.2. Definitions and Terminology ............................................................................................ 13
2.3. Dynamic Reconfiguration State of Arts .......................................................................... 15
2.3.1. Introduction ................................................................................................................ 15
2.4. Reconfigurable Design Flow and Strategies .................................................................. 17
2.5. Architectures Conversion and Choice ............................................................................. 21
2.5.1. Flow Choice MB vs PR ............................................................................................ 22
2.5.2. Converting a PR Architecture to MB ....................................................................... 24
2.6. Task scheduling and Power consumption ........................................................................ 26
2.7. FPGA prototype of a medical ASIC capsule for reconfiguration (First case study). 30
2.8. MB and PR Software Verification Approach ............................................................ 36
2.9. Design Optimization Options .................................................................................... 39
2.10. Summary .................................................................................................................... 42
2.11. Future Work and Application to Sound Source Computation................................... 43

CHAPTER 3 Sound Localization and Algorithm Specification ........................................ 45
3. Declaration and Presentation ..................................................................................... 45
3.1. Purpose of this Chapter .............................................................................................. 45
3.2. Introduction and Definitions ...................................................................................... 45
3.3. Wave Propagation and BeamForming Background Theory ...................................... 46
3.4. Microphone Geometric Configuration and State of the Art ..................................... 51
   3.4.1. Microphone Principle and Description .............................................................. 54
   3.4.2. System Specification and Signals Generation .................................................... 56
3.5. Summary .................................................................................................................... 58

CHAPTER 4 Combining Sound Source Tracking Algorithms Based on Microphone Arrays to Improve Real-Time Localization ................................................. 59
4. Declaration and Presentation ..................................................................................... 59
4.1. Purpose of this Chapter .............................................................................................. 59
4.2. Introduction ............................................................................................................... 60
4.3. Microphone arrays and beamforming Algorithms .................................................... 61
4.4. The Generalized Cross Correlation Algorithm (GCC) .............................................. 63
   4.4.1. Microphone sub-array Triangulation ................................................................. 64
   4.4.2. Summary ............................................................................................................ 67
   4.4.3. GCC and β-PHAT ............................................................................................. 67
   4.4.4. Proposed GCC Architecture and its associated Throughput ......................... 70
   4.4.5. Summary ............................................................................................................ 72
4.5. Delay and Sum Beamforming (DSB) ....................................................................... 72
   4.5.1. DSB Architecture throughput and number of operations ............................... 76
4.5.2. Summary ............................................................................................................ 78
4.6. DSB Acceleration Techniques.................................................................................. 78
  4.6.1. GCC1-DSB ......................................................................................................... 78
  4.6.2. GCC2-DSB ......................................................................................................... 80
  4.6.3. GCC-DSB ........................................................................................................... 80
  4.6.4. GCC-DSB Computation Considerations ............................................................ 85
4.7. Results and GCC-DSB Architecture Proposal .......................................................... 87
  4.7.1. GCC localization error and reduced NOSS approximation ............................... 87
  4.7.2. Optimized GCC-DSB Architecture ................................................................ 88
4.8. Novel Interface Drivers to Combine Real-time Localization and Tracking
  Algorithms............................................................................................................................ 89
  4.8.1. GCC-DSB Driver ............................................................................................... 89
  4.8.2. DSB-DSB Driver ................................................................................................ 92
4.9. Summary .................................................................................................................... 97

CHAPTER 5 Expanding the Developed Novel Hybrid Algorithm to Multiple Sound
Source Localization and Beamforming Exploration.......................................................... 99
  5. Declaration and Presentation ..................................................................................... 99
  5.1. Purpose of this Chapter.......................................................................................... 99
  5.2. Introduction .......................................................................................................... 100
  5.3. Novel Hybrid Algorithm Applied to Multi-Source Localization ............................ 101
     5.3.1. Evaluation of the MUSIC Algorithm for Multiple Sound Sources Location .. 109
  5.4. DSB AND MVDR BEAMFORMING ................................................................. 113
     5.4.1. MVDR Weight Computation Using the GCC or DSB Computation Results .. 115
     5.4.2. MVDR and DSB Computation Selection and Wiener Filter ......................... 121
  5.5. Combining the GCC-DSB and MVDR Architectures ............................................. 125
  5.6. Pre-computation of Parameters for GCC-DSB and MVDR .................................... 128
     5.6.1. Distance Pre-Computations ........................................................................... 130
     5.6.2. Angular inclination Pre-Computations ............................................................ 131
     5.6.3. Weight .......................................................................................................... 132
     5.6.4. Sample shift ................................................................................................... 133
     5.6.5. Inter-Action between Hardware and Software for Pre-Computation .......... 137
  5.7. Summary and further work..................................................................................... 138
CHAPTER 6 Low Latency Hybrid Speaker Localization Algorithm Implemented in an FPGA Architecture .............................................................................................................. 141

6. Declaration and Presentation .............................................................................................................. .......................................................... 141
6.1. Purpose of this Chapter .................................................................................................................. 141
6.2. Introduction ..................................................................................................................................... 141
6.3. System Full Architecture .............................................................................................................. 142
6.4. Acquisition Chain Processing Element (PE) .................................................................................. 146
   6.4.1. Filters and Framing modules .................................................................................................... 147
   6.4.2. VAD and FFT Modules ........................................................................................................... 149
6.5. Computation Chain Processing Element (PE) ............................................................................... 162
   6.5.1. Frequency Domain Computation ............................................................................................. 162
   6.5.2. Time Domain Computation after the IFFT ............................................................................... 168
6.6. System Partitioning for a Reconfigurable Architecture ................................................................. 172
6.7. Brief Presentation of the System Verification .................................................................................. 176
6.8. Summary and Further Work .......................................................................................................... 178

CHAPTER 7 Conclusions and Perspectives .................................................................................. 181

7. Declaration and Presentation .............................................................................................................. 181
7.1. Purpose of this Chapter .................................................................................................................. 181
7.2. Conclusion ..................................................................................................................................... 181
7.3. Perspective ................................................................................................................................... 183
LIST OF FIGURES

Figure 1.1: Cell Phone Microscopes ................................................................. 1
Figure 1.2: System Monitoring the Intestinal Pressure and Temperature .......... 5
Figure 1.3: System Monitoring Capsule .......................................................... 5
Figure 1.4; Sound Source Localization Principle ............................................. 6

Figure 2.1: Partial Reconfiguration Design Partition ........................................ 15
Figure 2.2: Normal and Multi-BOOT Flow ...................................................... 17
Figure 2.3: HDL State Machine for MB Control [2.17] .................................... 18
Figure 2.4: Detailed Partial Reconfiguration Flow Steps [2.18] ......................... 19
Figure 2.5: Reconfiguration Task in a Running System [2.19] ......................... 20
Figure 2.6: Multi-Boot MB (left) and Partial Reconfiguration PR (right) Basic Implementation Architecture ................................................................. 21
Figure 2.7: Partial Reconfiguration Minimum Resource to Control Partial Bitstream Reading ................................................................. 22
Figure 2.8: FPGA PR Internal System Partition ............................................. 24
Figure 2.9: MB Equivalent Architecture of Figure 2.8 .................................... 25
Figure 2.10: Tasks Dependency Graph ........................................................... 26
Figure 2.11: Partial Reconfiguration Configuration Latency [2.18] .................. 27
Figure 2.12: (a) Pipeline or Latency Optimization and (b) Low Power Execution Mode ---- 28
Figure 2.13: MB Equivalent of Figure 2.10 PR ............................................ 30
Figure 2.14: FPGA Prototyping ASIC Flow [2.27] ........................................ 31
Figure 2.15: Building Block of the Transmitter Part of the Capsule ................. 32
Figure 2.16: Building Block of the Receiver Part of the Capsule .................... 33
Figure 2.17: State Machine Controlling the Pressure and Temperature Reading ............................... 34
Figure 2.18: States Machine Controlling the Temperature Reading using MB Flow ............................... 35
Figure 2.19: State Machine Controlling the Pressure Reading using MB Flow ............................... 35
Figure 2.20: Transmitter FPGA Compressor Units FSM ............................. 37
Figure 2.21: Mixed Simulation in the Design Cycle ........................................ 38
Figure 2.22: Systems Monitoring of the FPGA Die Temperature ................... 39
Figure 2.23: Parallel Processor Speed Multipliers Based upon Amdhal’s Law [2.47] ............................... 40
Figure 2.24: Most Flexible Architectures ...................................................... 42
Figure 2.25: Different Approaches to FPGA Partial Reconfiguration [2.22] ....... 44

Figure 3.1: Signal Received by a Linear Aperture [3.2] .................................. 48
Figure 3.2: Planar Wave Fronts Arrival in Far-Field Mode [3.2] ...................... 48
Figure 3.3: Arrival of Wave Fronts at a Microphone in a Near-Field Approximation [3.2] ........................................... 50
Figure 3.4: Nested Linear Microphone Arrays Configuration [3.7] .................. 52
Figure 3.5: Directivity Pattern for Varying Effective Array Length (f=1 kHz, N=5) [3.2] ....... 53
Figure 3.6: Microphones Different Directional Responses [3.1] ..................... 54
Figure 3.7: Two-ADMP421 Microphone Connected to a Single DATA Wire [3.10] ............................... 55
Figure 3.8: 2D 3x3 FOV with 150cm x 150cm Size and a 50cm x 50cm Resolution .......... 57

Figure 4.1: Signals Arriving on Microphones (blue). Delayed Signals (red) ...................... 62
Figure 4.2: Cross Correlation Correspondences with the Number of Microphones .......... 64
Figure 4.3: Estimation of the Sound Source Position using GCC Sub-Array [4.9] .......... 65
Figure 4.4: Estimation of the Incident Angle [4.9] .......................................................... 66
Figure 4.5: Estimation of the Sound Source Distance [4.9] .............................................. 66
Figure 4.6: Computation Load of the GCC with SCOT or PHAT Weight ......................... 69
Figure 4.7: Localization and Tracking of Sound Source with the GCC [4.9] ...................... 70
Figure 4.8: Full FOV Representation for DSB Approach .................................................. 73
Figure 4.9: Delay-Sum Beamformer Block Diagram in Time Domain [4.10] ...................... 75
Figure 4.10: Delay-Sum Beamformer Block Diagram in Frequency Domain [4.10] .......... 76
Figure 4.11: GCC1-DSB Algorithms FOV Partition .......................................................... 79
Figure 4.12: GCC2-DSB Algorithms FOV Partition .......................................................... 80
Figure 4.13: Reducing the Search Region for the GCC-DSB Algorithm ......................... 82
Figure 4.14: Search Region Contraction of the FOV Internal Description ....................... 85
Figure 4.15: FPGA Memory Structure ............................................................................. 86
Figure 4.16: Reduced DSB NOSS Computation Depending on E and the FOV Size ........ 88
Figure 4.17: Combining GCC-DSB Block Diagram ......................................................... 89
Figure 4.18: GCC-DSB and DSB-DSB Interface Drivers Block Diagram ....................... 92
Figure 4.19: GCC-DSB Distance Pre-Computation ......................................................... 93
Figure 4.20: Different Sound Source Tracking Shapes ..................................................... 95
Figure 4.21: Different Sound Source Tracking Shapes ..................................................... 95
Figure 4.22: DSB Computation Load Compare to GCC-DSB and DSB-DSB ................... 96

Figure 5.1: 16x16 Small Square FOV with 4 Microphones and Two Speakers ............... 101
Figure 5.2: GCC Based Energy Computation in every Direction of the FOV ................. 103
Figure 5.3: FFT and IFFT Load Depending on the Number of Microphones and Configurations ................................................................................................................. 105
Figure 5.4: Detection of two Sound Sources in the FOV with N = 8 and β = 0.8 [5.26] ... 106
Figure 5.5: Estimation Error of the DOA with two Sound Sources [5.26] ....................... 107
Figure 5.6: Comparison of Secondary Speaker Error using SRP GCC and Donohue Temporal and Frequency Mode [5.26] ........................................................................... 108
Figure 5.7: Flow Chart of the Computation of the Sound Source DOA using MUSIC [5.14] ................................................................................................................................................ 110
Figure 5.8: Speakers DOA Detection using MUSIC Algorithm ....................................... 113
Figure 5.9: Structure of a Delay-and-Sum Beamformer [5.19] ....................................... 114
Figure 5.10: Structure of a Frequency-Domain Broadband Beamformer by Narrowband Decomposition [5.19] ................................................................................................. 115
Figure 5.11: DSB Directivity [5.20] .................................................................................. 120
Figure 5.12: MVDR Directivity [5.20] ............................................................................. 120
Figure 5.13: MVDR Algorithms Block Diagram ............................................................. 126
Figure 5.14: Combined Flow of the GCC-DSB-MVDR .................................................... 127
Figure 5.15: Compilation Order Dataflow ....................................................................... 128
Figure 5.16: Nine Points FOV Example ........................................................................... 130
Figure 5.17: DSB Distance Pre-Computation ................................................................. 131
Figure 5.18: DSB Weight Pre-Computation ................................................................... 133
Figure 5.19: DSB Sample Shift Pre-Computation ......................................................... 134
Figure 5.20: DSB SNRG Pre-Computation ..................................................................... 135
LIST OF TABLES

Table 2.1: Minimum Resources for Multi-Boot and Partial Reconfiguration ......................... 23
Table 2.2: Flow and Cost Comparison ............................................................................................ 23
Table 2.3: Control state machine resource Figure 2.17 and 2.19 ............................................... 36
Table 2.4: Code Coverage Report File ............................................................................................. 37

Table 3.1: Sound Speeds in Different Material .............................................................................. 47

Table 4.1: Frames Size and Acquisition Time .............................................................................. 61
Table 4.2: Computation Load of the GCC ...................................................................................... 71
Table 4.3: GCC Architecture Throughput Depending of the Weight ........................................... 72
Table 4.4: Computation Load of the DSB ...................................................................................... 77
Table 4.5: DSB Throughput for N=4, Ns=512 Clock = 200 MHz .................................................. 78
Table 4.6: Computation LOAD of the GCC-DSB ......................................................................... 96

Table 5.1: Points Reference to the sound source location of Figure 5.6 axis ........................ 109
Table 5.2: MVDR and DSB BEAMFORMING SERIAL THROUGHPUT for N = 8 and NS = 512 ................................................................. 122
Table 5.3: Distance Computation of all the FOV Small Square to each Microphone .......... 130
Table 5.4: Weight Computation of each FOV Small Square ......................................................... 132
Table 5.5: Shift Computation of each FOV Small Square ............................................................ 133
Table 5.6: Weight and Shift Computation using ML505 BOARD ............................................... 138

Table 6.1: Bit Growth per State in the Mean Computation .......................................................... 151
Table 6.2: Cordic Algorithm Computation Equation [6.13] ......................................................... 159
Table 6.3: 64-POINT FFT ALGORITHM COMPLEXITY [6.15] ............................................... 161
Table 6.4: Combined GCC-DSB IP Cores Resources ................................................................. 164
Table 6.5: DSB Computation Throughput with N=4, Ns=512 ................................................... 171
Table 6.6: DSB Throughput using the Hybrid Algorithm with a Hardware Contribution N=4 ................................................................................................................................................ 172
CHAPTER 1 INTRODUCTION

1.1. Motivation for this Research

Electronic systems development is becoming more complex, fast and power consuming [1.1]. For example, Industry has developed new mobile phones, which have multiple devices in one; a camera, a radio, internet access and a GPS (Global Positioning Systems) etc...

Recently Berkeley University, California has used a cell phone as a medical microscope that provides an affordable and reliable method to diagnose diseases by phone for those patients who live in remote areas. They can use the cell phone to diagnose skin diseases or blood diseases such as malaria see Figure 1.1 [1.2].

![Cell Phone Microscopes](image)

Figure 1.1: Cell Phone Microscopes

The complexity of a design which embeds all the concepts described above and the miniaturization of modern electronic systems, makes it impossible to get the design right the first time. Therefore all designers need to have processes available that allow them to correct any errors in their code and reprogram their hardware in a matter of seconds. These constraints justify the use of Field Programmable Gate Arrays (FPGA) for this work as FPGA flow accelerates design cycle time.
1.2. Contribution of the Thesis

The main problems addressed in this thesis are outlined below:

The inherent reprogrammability of FPGA devices opens up a number new design possibilities. In particular, the ability to support reprogramming of FPGA while in mission mode, allows systems with smaller size and cost to be developed. Enabling this approach requires modifications to the traditional design used for FPGA development, and one of the objectives of this thesis is to find the most efficient design flow for a given application. An application that uses FPGA technology to prototype an Application Specific Integrated Circuit (ASIC) was selected and two possible reprogrammability approaches along with their associated design flows were investigated.

A related opportunity involves utilizing the inherent support for parallel processing available in FPGAs, to solve a set of problems with the computational requirements that are challenging for conventional sequential processors. One such problem is the ability to locate and track a sound source in real-time. While a number of sound source localization algorithms have been developed, these high computational requirements rendering them unsuitable for real-time localization. This thesis applies FPGA technology to this problem, by investigating the parallel implementation of a number of sound source localization algorithms and evaluating their capacity to locate a sound source in real-time. It is found that only by combining two existing algorithms in a novel way can this problem be solved.

It is further found that the reprogrammability design flow identified in the first part of the thesis can be applied to the sound source localization application, opening the possibility of a smaller and lower cost design than would otherwise be required.

In the light of these problems, the main contributions of this thesis are:

- At the algorithm level, a novel hybrid algorithm was created for localization and tracking of sound sources in real-time that reduces the computation burden of the well-studied Delay and Sum Beamforming (DSB) Algorithm by more than 80% in a serial approach and even more when implemented on parallel architectures. This new algorithm's high level of abstraction allows it to be applied on any sound source localization algorithm that uses a defined Field Of View (FOV) as a region in which
to localize the sound source. The novel algorithm developed in this research is then expanded to the localization of multiple sound sources in a confined area. This methodology assesses the limitation of this novel algorithm in term of its results in different situations, such as the case of multiple sound sources. Initially this approach did not score so well when applied to a scene with multiple speakers. Therefore a new microphone geometric configuration was proposed to attenuate the effect of interferences between speakers, the effect of which, whether destructive or constructive was such that no speaker could be distinguished from the other (see Section 5.7). The creation of this new algorithm enables this work to exploit better the capability of parallel and reconfigurable architectures in terms of power processing, consumption, and resources utilization. Finally, this work will open the way to voice recognition systems to be able to track one speaker among multiple speakers.

- At the design level, to explore novel reconfigurable architectures design techniques, a new design approach combining both, Multi-boot (MB) and Dynamic Partial Reconfiguration (DPR) is proposed. The proposed approach was validated by the redesign of an Application Specific to Integrated Circuits (ASIC) medical capsule to read the intestinal state. In that work the transition from one dynamic reconfigurable technique to another (DPR to MB) was explored.

- At the architecture level, this work investigates how to increase resources reusability, data agility\(^1\) and proposes a flexible and scalable architecture to assess that capacity on systems dealing with multiple tasks or mathematical computations. This work also evaluates the flexibility of the algorithm to be parallelized or serialized and evaluates if the resource utilization and throughput can be improved. These outcomes are linked to data handling (sequentially or fully pipelined) and task partitioning. Partition and data handling also have a big impact when migrating to DPR or MB from a normal flow.

These ideas were developed using a number of major projects or case studies. The first case study investigated reconfiguration techniques using Xilinx FPGAs and mostly involved design and architectural considerations, while the second case study developed new sound localization algorithms, which were then applied to a number of topical sound source

\(^1\) Data agility: the agility is the ability of data to move fast and easily.
problems. In addition, FPGA implementations of the new algorithms were designed and the architectural insights from the earlier case study applied to this application area as well.

The remainder of Chapter 1 is divided as follow: Section 1.3: presents the first case study and the second case study is outlined in Section 1.4. Section 1.5 outlines the scientific output of this work. Section 1.6 presents the outline of the thesis. Section 1.7 summarizes this chapter and briefly describes the other chapters.

1.3. First Case Study

The first case study presented in this thesis arose from the requirement to prototype an electronic ASIC device using an FPGA. A number of reconfigurable flows were investigated to find the most efficient FPGA implementation. These flows were also applied to the sound source localization work in the second case studies although these were much more complex designs.

The purpose of the first design is to monitor the pressure and temperature of the intestinal system as shown in Figure 1.2 below. The capsule depicted in Figure 1.3 is swallowed by a patient to monitor the pressure and temperature of his intestine and the pH if necessary. The capsule may take 12-36 hours to pass from the mouth to anus and may exceed this by up to 7 days for a patient suffering from constipation. Therefore, the main challenge in this design was to design and develop an efficient power management technique to reduce the overall power consumption of the capsule system, leading to an extended battery lifetime. All details about the functionality of that capsule are in [1.3].

The research work was to prototype the electronic ASIC part of the design shown in Figure 1.3 using an FPGA. The main task was to design a control system to monitor the receiving and transfer of data from the intestine to the capsule and from the capsule to the wireless system. A sequential state machine was necessary as pressure and temperature in the intestine are read at different times and for power consumption reasons. This work is briefly discussed in Chapter 2.
Figure 1.2: System Monitoring the Intestinal Pressure and Temperature

Figure 1.3: System Monitoring Capsule
1.4. Second Case Study

One of the main achievements of this research is a novel algorithm that tracks a sound source or sources in real-time. The sound source localization is performed with an improved delay and sum beamforming (DSB) computation methodology. The proposed hybrid algorithm developed in this research first computes the Generalized Cross Correlation (GCC) to create a reduced search spectrum for the DSB algorithm. This methodology reduces by more than 80% the DSB localization computation burden. Moreover, for high frequencies sound sources beamforming, the DSB will be preferred to the Minimum Variance Distortion-less Response (MVDR) for logic and power consumption reduction.

The aim of this work is to localize a sound source as depicted in Figure 1.4. The example shows one sound source located between the microphones 3 and 4 at a distance of 7 meters above a linear 8-microphone array at an angle of 81.87 degrees from the center zero. The scale at the far right of Figure 1.4 is the Steer Power Response (SRP), which will be explained in Chapter 4.

The implementation of a beamforming algorithm supporting many microphones over a wide band of frequencies presents a significant computational challenge in terms of required real-time processing power. Algorithms such as the GCC (Generalized Cross Correlation) and DSB (Delay and Sum Beamforming) adapt better to distributed processing. With the
appropriate communication strategy and optimal resources utilization, the computation volume of the DSB algorithm is reduced without affecting the localization precision. The DSB algorithm applications can be applied to the Smartphone, Digital video cameras, Bluetooth headsets, Video phones, Teleconferencing systems, breast cancer detection [1.4] and other applications. As further work, the author will propose fuzzy logic rules based classification for speaker recognition to track a specific speaker among multiple others[1.5].

1.5. Scientific Output of this Work

A series of papers have been published some of which have been translated and republished in German. Below is the list of papers.


1.6. Outline of the Thesis

This paragraph presents the outline of the remainder of this PhD thesis.

Chapter 2: This chapter will present two dynamic reconfiguration approaches called dynamic partial reconfiguration (DPR) and Multi-Boot (MB) flow. Both have been chosen as they increase hardware flexibility. The medical capsule case study discussed above will be used as an example to present the DPR and MB flows and discuss their advantages and drawbacks. Another contribution of this work is to reduce the area/power overhead of FPGAs by applying Multi-Boot instead of DPR. Previous work so far has only investigated partial reconfiguration design flow. Therefore, this work will also present a migration approach from partial reconfiguration to Multi-Boot.

Chapter 3: This chapter will present the following topics: The background theory on signal propagation, including (mathematical background and fundamental theory) and characteristic, different applications that benefit from sound source localization or share the same algorithms; sensors used to capture the sound source; their configurations, their features. Finally, this chapter will present different kinds of noise that can affect sound source localization.

Chapter 4: The purpose of this chapter is to present a novel hybrid algorithm created during this research work to accelerate the computation of the Delay and Sum Beamforming (DSB) algorithm by reducing its search spectrum without affecting its performance. The DSB throughput can be improved using the hybrid algorithm which confines the DSB search spectrum to the sound source direction of arrival (DOA). The DOA is first found by computing the GCC (Generalize Cross Correlation) Algorithm. Then the hybrid algorithm drivers convert the GCC results to a reduced FOV (Field Of View) used by the DSB. The results improve the DSB throughput by more than 80 % and make it a good candidate for real-time application. Besides the algorithmic contribution aforementioned, this work proposes to merge the DSB and GCC architecture for a possible hardware implementation of the hybrid algorithm at a later stage.
Chapter 5: The main purpose of this chapter is to expand on the novel hybrid algorithm developed in this work to localize multiple sound sources simultaneously. The approach developed will be compared to an existing algorithm called MUSIC (MUltiple SIgnal Classification). This chapter will present a pre-computable parameters approach to reduce the DSB computation load using 9 points (Field Of View) as an example. Both the DSB (Delay Sum Beamforming) and MVDR (minimum-variance distortionless response) coupled to a Wiener filter will be presented along with their context of use. Finally a common architecture combining the novel hybrid algorithm and the beamforming (using adaptive and time invariant algorithm) will be presented before concluding this chapter and outlining in further work some ideas to reduce interference between speakers are proposed.

Chapter 6: This chapter will present the overall system and modules to implement the novel hybrid Algorithm and Beamforming presented in Chapter 4 and Chapter 5 to achieve a real-time implementation. This chapter will also elaborate a strategy to increase module re-usability, control bit growth and dataflow. Finally, this chapter will present the partition of the novel algorithm developed in this work to support the dynamic reconfiguration flow described in Chapter 2.

Chapter 7: This chapter presents the conclusions drawn from the work and outlines further research perspectives that could expand on the results of this thesis.

1.7. Summary

Detecting sound source direction of arrival (DOA) with the GCC prior to DSB computation reduces the DSB algorithm throughput. The architecture and flow proposed to implement this novel computation methodology is similar to the approach elaborated in the first case study explains in Chapter 2. From Chapter 3 to Chapter 6 this work presents its novel algorithm and architecture to compute the DSB in real-time. Finally, Chapter 7 expands this work to multiple source localization and tracking with voice recognition and Chapter 8 concludes this thesis and presents further work.
Note: The applications covered in this thesis include cases where area and resource optimization in the FPGAs are critical. As a result very high level programming languages, such as System C, System generator or LabView are not used as their area optimization is poor. Verilog was the preferred hardware description language used in this work.

While not essential to their implementation, it is shown that both their case study of Chapter 2 and the later sound source localization implementations can benefit from the reconfiguration flow. The sound source localization designs have tighter time constraints due to the higher application execution speed.
CHAPTER 2 Reconfigurable Flows to Increase Architectures Flexibility

2. DECLARATION AND PRESENTATION

I hereby declare that this chapter is entirely of my own doing and that it has not been submitted as an exercise for a degree at any other University. The documentation used to support this work is on Xilinx Website, in various papers that were published during this work and in the case study.

2.1. Purpose of this Chapter

This chapter will present the two FPGA dynamic reconfiguration approaches that are currently used, called dynamic partial reconfiguration (DPR) flow and Multi-Boot (MB) flow. These flows are used in applications where increased hardware flexibility is important. The medical capsule case study will be used as an example to present the DPR and MB flows and to discuss their advantages and drawbacks. The main contribution of this work is to reduce the area required for the FPGA logic as well as its power consumption by applying multi-boot instead of DPR. Previous work has only investigated partial reconfiguration design flow. Therefore, this work will also present a migration approach from partial reconfiguration to Multi-Boot.

2.2. Definitions and Terminology

Ideally every designer would like to have a device that would be able to adapt to the application on the fly. Such a hardware device is called a reconfigurable hardware or reconfigurable device or reconfigurable processing unit (RPU) in an analogy to the Central Processing unit (CPU) [2.1]. Configuration or reconfiguration is the process of changing the structure of a reconfigurable device at start-up-time or at run-time [2.1]. MB and DPR have
opened the way to many techniques that improve and facilitate logic reusability and power consumption control. The most common related to DPR are:

- Area Blanking
- Data Changing
- Scalability
- Partial bitstream relocation
- Persistence

MB does not have specific terminology, but the DPR vocabulary can be understood by referring to Figure 2.1.

- **Partial Reconfiguration Region** (PRR): The area of the device defined as the region to reconfigure dynamically. A single netlist instance is assigned to the PRR at any given time. The number of netlists that can populate that region is unlimited.
- **Partial Reconfiguration Module** (PRM): represents the netlists that will populate the PRR. Typically, more than one PRM is defined for each PRR. The ports pins and sizes should be identical for each PRM and its associated PRR.
- **Static Logic or Base Region**: The remaining design logic that is not defined in any PRM.
- **Partial Bitstream**: The partial bitstream file represents a PRM variant configuration.
- **Merge Bitstream**: A merge bitstream file containing a single configuration of the design including a single active PRM for each PRR. The merge bitstream is used as the initialization bit file for the device.
2.3. Dynamic Reconfiguration State of Arts

2.3.1. Introduction

In recent years, hardware reconfiguration architectures received increasing attention due to their flexibility and short design time. Their distinctive main feature is the capability of changing the system structure in a field, a process known as reconfiguration. Reconfiguration enables fast prototyping, incremental architecture refinement and post-implementation error correction [2.2]. Field-programmable Gate Arrays (FPGAs) emerged in the ‘80’s as reconfigurable logic devices. Twenty years ago, in addition to their reconfigurable main feature, FPGA vendors started combining general-purpose processors and FPGA fabric on a single chip. Current FPGA technology allows the integration of several Generic Purpose Processors (GPPs), on-chip memory, custom accelerators, peripherals and other complex blocks [2.3][2.4][2.5]. Reconfigurable platforms based on FPGAs have quickly become the main option to address the flexibility of software with the performance of hardware. FPGA-based systems are faster than a pure software approach in terms of computational power, less static than traditional ASIC-based solutions and with better time-to-market [2.6].

Modern FPGAs include enough resources to build a complete SOC (System-on-Chip). However, many real-life applications are too large to fit in the logic available on a single chip [2.6]. This is a rapidly changing area and the electronic industry is changing the boundaries of what is possible, as shown by the introduction of Zynq FPGA series [2.38] to take an example.
In addition, the high cost per unit is also a barrier for an expansion of utilization of FPGAs in embedded systems [2.7]. The area/power overhead of FPGAs can be reduced by using reconfigurable techniques such as partial dynamic reconfiguration (PDR) and Multi-Boot [2.8].

Dynamically reconfigurable architectures are those architectures based on FPGAs that allow run-time partial reconfiguration, the partial modification of the FPGA without stopping the execution of the remainder of the FPGA that is being modified at the cost of reconfigurable regions and latency overhead [2.1][2.9]. Compare to the partial reconfiguration, in the multi-booting approach the data in the FPGA are lost during reconfiguration [2.10].

Dynamic reconfiguration has been an area of intensive research during the last two decades. There is now a good understanding on the reconfiguration mechanisms, and alternative reconfigurable architecture are being explored [2.11][2.12][2.13]. However, reconfiguration overhead is still a bottleneck for most applications, especially when considering the clock speed gap between standard microprocessors and FPGAs. Dealing with Dynamic Reconfigurable Architectures (DRA) requires taking into account several issues: reconfiguration management, persistence management, IP integration for reconfigurable areas, reconfiguration process activation, and task migrations. Managing dynamic reconfiguration hardware implies having to deal with a new degree of freedom for systems designers. To the traditional Hardware/Software partition problem, the static and dynamically reconfigurable hardware partition problem is added with the objective to obtain a near optimal scheduling of tasks in heterogeneous architecture [2.14]. The use of this kind of architectures allows the incorporation of new features into the already deployed design and for the components to be updated at run-time [2.15].

Using MB and PR flow with HDL (Hardware Description Language) good coding practice can improve FPGA resources utilization when using simple rules [2.16] such as:

- Use of active high control signals
- Map logic in unused BRAMs.
- Reduce shift register resets
- Instantiate only when necessary
- Reduce inputs per logic

The remainder of Chapter 2 is divided as follow: Section: 2.4 will present the reconfigurable design flow and strategy. Section 2.5 will discuss PR and MB architectures and criteria for
choosing PR or MB flow and Architectures conversion from PR to MB. Section 2.6 will discuss task scheduling, power consumption and hardware partition for both flows. Section 2.7 presents the HDL partition to apply PR or MB flow using the first case study. Section 2.8 will discuss the verification at the software level of the system presented in section 2.7. Section 2.9 presents the design optimization choice. Section 2.10 summarizes this chapter and Section 2.11: presents possible further works.

2.4. Reconfigurable Design Flow and Strategies

Modern design approaches are based on system level specification and refinement [2.13][2.14]. They suggest a design space exploration step as a preliminary estimate of performance requirements before architecture selection [2.15][2.16]. The main parameters to consider, in addition to reconfiguration time for reconfigurable systems, are cost, design time, area, required throughput and power consumption.

Figure 2.2 shows the normal flow used to produce a bitstream required to program a FPGA. This is a straightforward flow. With a similar flow, MB uses partitioning to split the full system onto as many bitstreams as required to fit onto the target FPGA. In that case MB produces a hierarchical scheduler where each State Machine (SM) is in charge of the next partition reconfiguration. Figure 2.2 also shows the detailed steps of the normal and MB flows as they differ only by a State Machine (SM) defined as piece of HDL code used to control the MB reconfiguration depicted by Figure 2.3. The values required at every state from {Dummy word to type 1 no op} are defined in [2.17].

Figure 2.2: Normal and Multi-BOOT Flow
Figure 2.3: HDL State Machine for MB Control [2.17]

Referring to Figure 2.4 the Partial Reconfiguration (PR) flow splits the system between static and reconfigurable bitstreams. The static part includes the reconfiguration sequence and a set of reconfigurable regions (RRs) firstly occupied by an empty netlist bitstream or blank box (BnkB). The reconfigurable modules (RM) generated during the partitioning of the full system will later occupy the reconfigurable regions. That flow is somewhat more complex compared to MB. In the PR flow, RMs are selected according to their usability and availability. The design of a RR must be compliant to the set of RMs required for size and
interface, so this is an issue for scalability. Moreover IP or dedicated blocks are forbidden as RM candidates. They remain in the static region even when they are not always required. In addition the scheduler can only be implemented onto a microcontroller adding area overhead.

The area overhead is not only explained by the used of microcontroller, but also by the implementation flow imposed by the PR. Figure 2.4 also shows that in the PR flows, modules synthesized for each PRR and static regions need to be done separately. That approach prevents the optimization across boundaries thus increasing logic utilization, however its advantage is that it can prevent reverse engineering. In the PR flow, every PRM is floorplanned and can only occupy 80% of PRR as some of PRR resources are used by static routing which reduces the FPGA implemented region. PR flow creates multiple bitstream that may require a certain memory size depending on the application. On the other hand, the design flexibility increases with the number of RMs for a specific reconfigurable region. Figure 2.5 is fully explained in [2.18]

![Figure 2.4: Detailed Partial Reconfiguration Flow Steps](image)

Figure 2.4: Detailed Partial Reconfiguration Flow Steps [2.18]
Figure 2.5 presents the PR reconfiguration process. It shows that the region-containing task B reconfigures itself while task A and the rest of the FPGA is running. It shows that a certain time between the termination of task B and the beginning of task C calculated as $t_3 - t_2$ should be carefully scheduled to avoid loss of data. Figure 2.5 also shows that PR only takes place after FPGA full configuration and that only one region can be reconfigured at a given time [2.19]. Therefore assuming that task A and task B had the same amount of logic resources reconfiguring task A right after task B would take $2^*(t_3-t_2)$ time, as parallel configuration is not possible. In case of successive reconfiguration of all the PRR, the MB equivalent will require approximately the same amount of time although the time required in the MB flow is less linked to the size of the logic changed but rather on the amount of logic in the FPGA.

![Figure 2.5: Reconfiguration Task in a Running System [2.19]](image-url)
2.5. Architectures Conversion and Choice

In this section, partial reconfiguration (PR), multi-boot (MB) architectures, and their required components are described. In both cases, an external memory is necessary to store different configuration bitstreams. PR architectures uses a microprocessor (µP) while MB can be implemented only with a simple finite state machine (FSM) (see Figure 2.3). Figure 2.6 shows the basic components used for MB and PR flow.

Figure 2.6: Multi-Boot MB (left) and Partial Reconfiguration PR (right) Basic Implementation Architecture

In Figure 2.6 (left) the FPGA contains the current user application executing on a reserved reconfigurable region (gray). Through the configuration interface, ICAP (integrated Configuration Access Port) the FPGA downloads the bitstream (BSx) stored on the external memory. The control unit decides which application/bitstream is downloaded from memory at a given time.

The PR implementation on the right hand side of Figure 2.6 contains in its static region an ICAP and two additional blocks, a micro-processor (µP) the minimum architecture of which is shown in Figure 2.7 and a ChipScope module (optional). The external memory (SysACE memory) stores the bitstream of the reconfigurable modules (RMx) that are placed on the PRR. Generally a processor (a PowerPC hard core or Microblaze soft-core) is used to control the bitstream download process [2.20]. In this work, the FPGA uses a SysACE peripheral (System Advanced Configuration Environment) to access the off-chip memory [2.21] and uses ChipScope for hardware debugging in real-time.
2.5.1. Flow Choice MB vs PR

There are several reasons that might lead a designer to convert a PR design to MB or to consider only MB. The most important are:

- Design has a tight timing issue. Applying a PR flow will potentially cause a design to fail timing as the clock degradation is estimated to be around 10%.
- All PRMs allocated to the same PRR must have the same interface.
- The PRR packing density should not exceed 80% per slice. Packing above that limit might cause routing issues.
- The overall implementation time increases. Longer design run-times are common in most PR cases compared to MB, as all the above additional requirements are factored into the overall solutions.
- Certain IP are not permitted into PRRs for the time being as most of them have integrated clock module, I/O Buffer, ICAP, DCM etc.
- When most of the design resources are not supported in the reconfigurable region [2.22].

The minimum resource utilization required for MB or PR implementation is an important decision factor. To verify the minimum resources of both PR and MB a simple design that includes only two LEDs was designed using a PR and MB flow. As shown in Table 2.1, only a small percentage of logic is required to implement MB compared to PR. The main resources include global buffers (BUFG), Slice LUTs (SLUT), I/O (IOBS) and (BRAM36). That difference is mainly due to the need for a microprocessor to control PR. In case of MB, a
simple fully synchronized FSM is used see Figure 2.3. Therefore, MB can be implemented in a smaller and cheaper FPGA see Table 2.2 below.

Table 2.1: Minimum Resources for Multi-Boot and Partial Reconfiguration

<table>
<thead>
<tr>
<th>Device Usage</th>
<th>Multi-Boot</th>
<th>Partial Reconfiguration</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFGs</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>BSCANs</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>RAM36</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>SLICE Register</td>
<td>231</td>
<td>4210</td>
</tr>
<tr>
<td>SLICE LUTS</td>
<td>187</td>
<td>2921</td>
</tr>
<tr>
<td>ICAP</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SLICE LUTS FLIP FLOP</td>
<td>32</td>
<td>5159</td>
</tr>
<tr>
<td>IOBs</td>
<td>7</td>
<td>36</td>
</tr>
</tbody>
</table>

Cost has always been the main criterion that decides choice of technology. Table 2.2 shows that five Spartan-3A700 boards provide more logic resources at lower cost than 1 Virtex-5 (ML505). Moreover, the MB flow can use a free version of ISE whereas the PR flow requires the full version of ISE/EDK and PlanAhead tools which are quite costly.

Thus whenever resources, utilization, cost or timing are an issue, MB will be a better choice compared to PR. The main concern when converting PR to MB is the change of hardware description language (HDL) to target a smaller board.

Table 2.2: Flow and Cost Comparison

<table>
<thead>
<tr>
<th>Logic Resources and Cost</th>
<th>ML 505 (DPR)</th>
<th>SPARTAN3A (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Number</td>
<td>7.200</td>
<td>5.888 * 5 = 29440</td>
</tr>
<tr>
<td>I/O</td>
<td>480</td>
<td>372 * 5 = 1860</td>
</tr>
<tr>
<td>DSP</td>
<td>48</td>
<td>N/A</td>
</tr>
<tr>
<td>Multiplier</td>
<td>0</td>
<td>20 * 5 = 40</td>
</tr>
<tr>
<td>DCM</td>
<td>12</td>
<td>8 * 5 = 40</td>
</tr>
<tr>
<td>Price USD</td>
<td>1195</td>
<td>200*5 = 1000</td>
</tr>
<tr>
<td>Tools</td>
<td>ISE/EDK/PLANHEAD</td>
<td>ISE</td>
</tr>
<tr>
<td>FLOW</td>
<td>DPR</td>
<td>MB</td>
</tr>
</tbody>
</table>
2.5.2. Converting a PR Architecture to MB

To convert PR architecture to MB an evaluation of the number of PRR, the interaction between them and the static region is required. To illustrate that approach the first case study is presented.

Medical industries rely on electronics to monitor the state of human organ function. Those systems are continuously sensing several parameters, such as intestinal pressure, pH, and temperature. In Figure 2.8 sensors can collect temperature (Temp), pressure (PRES) or pH data. Data collected by the 3 sensors are processed by their respective static region and passed to their PRR for further treatment such as the computation of the average or variance etc. Figure 2.8 shows a single FPGA partition for the implementation of the medical device described above that would use a PR flow instead of an ASIC.

![Figure 2.8: FPGA PR Internal System Partition](image)

The MB equivalent architecture of Figure 2.8 is distributed over several MB FPGAs as represented by Figure 2.9. In MB architecture, each PR region and associated RMs corresponds to a MB FPGA with its external memory containing the bitstream of each RMs. The static regions (STx) are located on an additional MB FPGA (FPGA₀) interconnected to the others by synchronization and data interface. The details of the implementation of these interfaces including their complexity (Data, address, I/O, clock path etc) are documented on the Dini Group Website [2.23]. Considering the number of PRRs available \((N_{PRR})\) independently of knowing the specific allocation of tasks, the number of FPGAs required to build an MB equivalent implementation \((N_{MB})\) are modeled by equation (2.1):

\[N_{MB} = \frac{N_{PRR}}{N_{MB}}\]
\[N_{MB} = 1 + N_{PRR}\] (2.1)

Where \(N_{MB}\) is the number of FPGAs required to build an equivalent implementation of the design using MB and \(N_{PRR}\) is the number of Partial Reconfiguration Regions in the original design.

Equation (2.1) shows that \((N_{MB})\) can be quite big if PR is widely used. However, a simpler configuration controller (MB control) and cheaper set of FPGAs can be used for MB work compared to PR (See Table 2.2 as a reference). Figure 2.9 shows that all the static tasks of Figure 2.8 PR flow are allocated on FPGA0, nevertheless, static tasks that only need to be active around the same time as their PRM can be allocated into their respective MB FPGAs. This approach should mainly be used for primitives forbidden in the PRRs. In that case equation (2.1) is modeled by equation (2.2) provided \(N_{PRR} > 1\).

\[N_{MB} = N_{PRR}\] (2.2)

Depending of the design processing optimization whether latency or power consumption mode and the data exchange scheduling, the numbers of MB FPGA equivalent to PR vary as in equation (2.3).

\[2 \leq N_{MB} \leq 1 + N_{PRR}\] (2.3)
2.6. Task scheduling and Power consumption

Along with design partition, scheduling is one of the most important pre-design strategies for any system design. Scheduling is a design decision of choosing between two tasks ready at the same time as to which one needs to go first in order to improve the design performance. The design performance is a compromise between “Speed, Logic, resources used, power consumption, and design complexity”. Scheduling is strongly influenced by the algorithm that is being implemented whether a mathematical operation such as Fast Fourier Transform (FFT) or conditional operation such as a (Communication Protocol). Most of the computation done in FPGA can be presented in two particular formats: Sequential execution and ideal pipelined execution. All the other computations are a combination of these two modes. Most applications have a dependency graph similar to Figure 2.10.

![Figure 2.10: Tasks Dependency Graph](image)

Each function or task corresponds to a node and the execution sequence (including incoming data) or data flow follows the arrow direction. An outgoing arrow can be mutually exclusive, if necessary, the pre-processor node makes that decision. In some cases a task can always be activated while other alternative tasks, as it happens with task X (coloured in yellow), activated alternatively by task B while task V is always executed depending on the design functioning mode. Alternative tasks also mean that their functionality can be substituted on the fly. The coloured tasks (Z, Y, C and X) are not always required, when not required they are replaced by a blank bitstream (BnkB). These functions will be executed by a reconfigurable module and occupy PR regions.

A node must start executing as soon as an incoming edge provides the required activation and as the data becomes available. For instance, node D must wait for the
activation/data coming from nodes V, X or C. However, in our example, it can be possible that there are only one or two paths available at a time, as indicated by the alternative path (dashed arrow) passing through Z, X or Y. To preserve a similar processing throughput the different paths must have the same execution delay. That condition can easily be satisfied if the system is located in the static region but the necessity to reduce power consumption using PR adds the reconfiguration time in each path to the throughput computation. The different paths are A-Z-B-V, A-Z-B-X and A-Y-C. If \( t_x \) represents the time to complete the execution of task X and \( d_x \), the task reconfiguration delay when the task is located on a reconfigurable region, the following condition must be satisfied.

\[
t_{A-D} = \begin{cases} 
 t_A + (t_z + d_z) + t_B + t_y \\
 t_A + (t_x + d_x) \\
 t_A + (t_y + d_y) + (t_C + d_c) 
\end{cases} 
\]  

(2.4)

In equation 2.4, \( t_{A-D} \) is the throughput of the system presented in Figure 2.10. The longest reconfigurable module should be located in the branch Y, C as it is in the shortest path. The reconfiguration delay \( d \) mainly depends on the configuration clock speed \( C_{clk} \), bus width \( BW \) and the bitstream size \( (BS) \) [2.24], this concept is independent of the FPGA family and can also be applied to the newer Virtex 7 series FPGA [2.39] (see equation (2.5)).

\[
d = \frac{BS}{(C_{clk} \times BW)} 
\]  

(2.5)

![Figure 2.11: Partial Reconfiguration Configuration Latency [2.18]](image-url)
Figure 2.11 shows the configuration latency between two FPGA devices Virtex-2Pro and Virtex-4. In both cases the bigger the device is the longer the configuration takes. It also shows that the technology is becoming faster. Virtex-4 configures itself almost 10 times faster than Virtex-2 Pro [2.25]. Even with newer Virtex families such as Virtex-5 and Virtex-6 the partial reconfiguration speed is similar to Virtex-4 as the ICAP speed has remained the same [2.40]. In most cases, the assumption can be made that the reconfiguration delay is small compared to the completion of each task. However it must be considered, not only to correctly compute schedule deadlines, but also because it contributes to the total energy consumption.

The total energy consumption can be computed as the addition of the task’s execution power consumption $P_p$ during the time $t_p$, and the power consumed during reconfiguration $P_r$ [2.26] with the delay $t_r$:

$$E_T[X] = P_p \cdot t_p + P_r \cdot \frac{BS}{(C_{clk} \cdot BW)}$$

Equation 2.6 shows that reducing the reconfigurable bitstream size and the processing time will reduce the system energy consumption. Assuming the same execution time for all the tasks, the longest path of Figure 2.10 is modeled by equation (2.7) which presents the system throughput in sequential mode.

$$t_{A-D} = 5t_A + d_z + d_x$$

In the pipeline mode the system throughput is modeled by equation (2.8)

$$t_{A-D} = t_A$$

The total power consumption is also dependent on the computation mode see Figure 2.12.
Figure 2.12 (a) shows that in pipeline mode if both branch \{(A, Z) and (A, Y)\} had to be used then the PR flow could not be implemented. But when they have an alternate use then node Z can be populated by a blank bitstream (BnkB) while Y node is active and vice versa. Pipeline optimization mode is mainly used for high-speed application such as Internet connection, Military radar imaging, etc… Figure 2.12 (b) represents the sequential approach that can be used for less computation intensive applications such as the medical device discussed below. This sequential approach best suits the MB and PR flow. In both modes, the power consumption will be computed with the assumption that each task has the same power consumption $P_A$.

In pipeline mode the power consumption varies from $8 \times P_A \leq P_T \leq 25 \times P_A$ but if all the nodes of Figure 2.10 are running at the same time or just a specific branch is running in the sequential mode it varies from $P_A < P_T \leq 8 \times P_A$. Thus to reduce power consumption a sequential mode should be used whereas the pipeline mode is used to improve throughput.

The PR graph in Figure 2.10 could work using MB after the task redistribution presented in Figure 2.13. The first proposition in Figure 2.13 (a) shows that all the reconfigurable task that happen around the same time are placed in the same FPGA. Therefore the static tasks \{A, B, V, D\} are in FPGA$_0$ and the reconfigurable tasks are divided between FPGA$_1$ and FPGA$_2$ respectively for the tasks \{Y, Z\} and \{X, C\}. In this approach \{X, C\} task needs synchronizing before reconfiguration.

The case study shown in Figure 2.13 (b) presents the advantage of the MB over the PR; it shows that MB does not need a static or reconfigurable part to function. FPGA$_1$ can reconfigure the task A along with task \{Z and Y\} and keep it identical after reconfiguration. The same truth applies to tasks \{B, V, D\} along with the reconfigurable task \{X, C\} in FPGA$_2$. Figure 2.13 keeps approximately the same throughput as Figure 2.10 with a reduced power consumption as the computation process is shared over multiple FPGA.
2.7. FPGA prototype of a medical ASIC capsule for reconfiguration (First case study)

In today’s practice more than 41% of ASIC developers prototype their ASIC designs using FPGAs [2.27]. Utilizing FPGAs to support ASIC prototyping is a viable way of addressing the time to market constraint. By incorporating the prototyping as part of the verification methodology, a designer may find a functional bug that was not detected during simulation but could only be uncovered during full system verification. This ability to find problems before marketing the design could save a company millions of dollars in reengineering and tooling costs. Moreover the migration from a FPGA to ASIC is easily achieved with the availability of structured ASICs from various suppliers, the migration to production once the FPGA has been verified is greatly shortened, so time to market concerns can be alleviated and a lower cost product can be obtained. Figure 2.14 shows the prototype flow from an FPGA to an ASIC and its explanation is given in [2.27].
More details on the full design of the medical ASIC capsule are given in [2.28]. This section expands on the ASIC case to present a concrete example of MB and PR. The main blocks of the system are the transmitter block in Figure 2.15 and the receiver block in Figure 2.16. These blocks show that the FPGA is mainly partitioned into the following modules:

A serial peripheral interface (SPI), run length encoding (RLE), a compressor and high data link control (HDLC) compressor units. The operation of the system units and the flow of data through the system are controlled by a main finite-state machine (FSM) controller using a dual block ram to store the data.

At the receiver end, Figure 2.16 shows the organization of the system. Data recovery is needed to extract the clock from received bit stream. The HDLC de-framer and RLE de-compressor blocks are designed to reconstruct the original data bytes sent by the transmitter. More details on this work are given in [2.28].
Figure 2.15: Building Block of the Transmitter Part of the Capsule
The Finite state machine (FSM) in Figure 2.17 is one of the many state machines of the system. It is a simplified version of the RLE compressor unit in Figure 2.15. The state machine in Figure 2.17 was simplified to monitor only pressure and temperature for visibility and understanding purpose. The pressure monitoring was given more importance than temperature. The temperature is only read once after a significant change in a pressure reading or after every 127 readings of the pressure if no change has occurred.

From the above description, it is obvious that the reading of the temperature sensor is independent from the pressure sensor reading. Therefore two separate state machines can be extracted from Figure 2.17. Figure 2.18 represents the temperature state machine and Figure 2.19 represents the pressure state machine. Both tasks can share the same PRR or have separate PRRs if the design is expected to grow. Figure 2.18 and Figure 2.19 show the simplification of the state machine of Figure 2.17 when applying a MB strategy on a normal design flow whereas Table 2.3 show the reduction in the logic resource utilization.
The state machine diagram of Figure 2.18 shows that the reading of temperature has only three states, although every state requires many logic resources. It is important to remember that there is a requirement to have seven clock cycles between two consecutive reconfigurations. However, the number of FPGA reconfigurations is not limited.
The number of states shown in Figure 2.19 is drastically reduced when moving from the normal to the PR or MB flows, as shown by the number of states in Figure 2.18 and Figure 2.19.

The 24 states in Figure 2.17 reduced to 3 states for temperature reading and 12 states for pressure reading, which represents a reduction of more than 35%. This reduction is mainly because the states responsible for handling the transition between pressure and temperature reading or their scheduling are no longer necessary. It also proves that many resources are allocated to data control, scheduling, coding and decoding of instructions. From Table 2.3, it is obvious that MB reduces the logic resources of Figure 2.17 compared to Figure 2.19.
Using PR flow in this case would have an inverse effect as the minimum resources necessary to implement the PR requires more resources than the implementation of the state machine of Figure 2.17 (see Table 2.1). Table 2.3 also shows that Figure 2.17 implementation has almost twice as much logic compared to Figure 2.19 using MB flow, although the implementation of MB requires the additional state machine shown in Figure 2.3 to control the re-boot process of Figure 2.19 using ICAP.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Before Multi-Boot Figure 2.17</th>
<th>After Multi-Boot Figure 2.19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Slice</td>
<td>234</td>
<td>125</td>
</tr>
<tr>
<td>Number of Slice Flip Flop</td>
<td>282</td>
<td>166</td>
</tr>
<tr>
<td>Number of 4 inputs LUT’s</td>
<td>378</td>
<td>187</td>
</tr>
<tr>
<td>Number of RAMS Used</td>
<td>122</td>
<td>56</td>
</tr>
</tbody>
</table>

### 2.8. MB and PR Software Verification Approach

The MB and PR basic verification is similar to the normal flow and is performed at the software level with a simple simulator such as ModelSim from Mentor Graphic that helps to compute code coverage to verify the system. ModelSim code coverage can provide graphical and report file feedback on which statements, branches, conditions, and expressions in the source code that were executed. It also measures bits of logic that have been toggled during the execution. Therefore it can be considered as a trace tool and probe at the same time that provides a history of the software execution. As code execution is almost invisible without an accurate trace tool, it is common for entire blocks or modules of code to go unexecuted during test routines or redundant user case suites. Coverage metrics showing which functions are not executed are useful for writing new, additional tests or identifying unused “dead” code. In applications where code size is critical, removing dead code reduces both waste and the risk of errors in the targeted designs. In many cases code coverage can also be used to analyze errant behavior and the unexpected execution of specific branches or paths. The Finite State Machine (FSM) can be extracted with code coverage as in Figure 2.22 for the compressor unit.
Figure 2.20: Transmitter FPGA Compressor Units FSM

Figure 2.20 FSM shows the number of states involved in the design and the interaction between those states. A test bench was written to examine the behavior of the HDL design. Table 2.4 presents the report file that needs to be examined to understand the test bench coverage. More details on the system verification using code-coverage is described in [2.29].

<table>
<thead>
<tr>
<th>File: BlockRamControl.v</th>
<th>Enabled Coverage</th>
<th>Active</th>
<th>Hits</th>
<th>% Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stmts</td>
<td>86</td>
<td>83</td>
<td>96.5</td>
</tr>
<tr>
<td></td>
<td>Branches</td>
<td>52</td>
<td>50</td>
<td>96.2</td>
</tr>
<tr>
<td></td>
<td>Conditions</td>
<td>6</td>
<td>6</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>States</td>
<td>9</td>
<td>9</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>Transitions</td>
<td>26</td>
<td>12</td>
<td>46.2</td>
</tr>
</tbody>
</table>

However the ModelSim tool itself cannot simulate the design interaction between two separate FPGA that communicate via a Radio Frequency (RF) model transmitter and
receiver as shown in Figure 2.15 and Figure 2.16. Therefore a high level language such as System Generator using the flow presented in Figure 2.21 is the preferred approach.

Figure 2.21: Mixed Simulation in the Design Cycle

At the Hardware level both PR and MB need a real-time debugging tool such as Chipscope Pro to verify the reconfiguration sequence. This approach is simplified for MB as the only software necessary to implement that approach is ISE whereas in PR flow ISE, EDK and PlanAhead tool must co-exist with Chipscope Pro. More details on hardware simulation have been published by the author in [2.44].

In term of power consumption, for new generation FPGA the power consumption can be computed at run-time using Chipscope-Pro that can provide die temperature, voltage and many more parameter details of the system. Figure 2.22 shows the die temperature, which is an important parameter to guarantee the timing between FPGA components. More details on the concept have been published by the author in [2.30].
2.9. Design Optimization Options

Increased throughput and power consumption reduction by reducing algorithm logic switching is the main aim of this work. A parallel architecture immensely improves a system throughput at the cost of logic resources and power consumption. Although transistor densities have grown exponentially from continued innovations in process technology, the power needed to switch each transistor has not decreased as expected due to slow reductions in the threshold voltage (Vth) [2.45]. In Digital CMOS (Complementary Metal-Oxide Semiconductor) circuits, the power consumption is composed of static and dynamic power. Dynamic power is approximated as in equation (2.9) [2.46] and it represents the dominating part of the total power consumption. Therefore the primary determinant of performance today is often power efficiency, not necessarily insufficient area.

\[
P_{\text{dyn}} = \frac{1}{2} V_{dd}^2 f_c C_L \alpha
\]

where \( V_{dd} \) is the supply voltage, \( f_c \) clock frequency, \( C_L \) is the load capacitance \( \alpha \) is the switching activity.

A low power architecture is always desirable and it is achieved by reducing the switching activities of the nets in the design as well as the utilization of the FPGA resources [2.46]. Increasing resources reusability and the use of enable registers signals will reduce respectively the resources utilization and net switching activities. Amdahl’s law expresses the relationship between the expected speedup of the parallelized implementations of an algorithm relative to the serial algorithm under the assumption that the algorithm size remains
the same when parallelized. For example if for a given problem size a parallelized implementation of an algorithm can run 12% of the operations arbitrarily quicker (while the remaining 88% of the operations are serial) then Amdahl’s law states that the maximum speed up of the parallelized version is: \( 1 / (1 - 0.12) = 1.136 \) times as fast as the non-parallelized implementation. Equation (2.10) is the Amdahl’s law mathematical model.

\[
\text{SysAcc} = \frac{1}{(1 - P) + \frac{P}{N}} \quad \text{(2.10)}
\]

Where SysAcc is the system global acceleration, P is the percentage of parallelism and N is the number of processors. The system acceleration reaches its maximum value when N >> P as shown in equation (2.11).

\[
\lim_{n \to +\infty} \frac{1}{(1 - p) + \frac{p}{n}} = \frac{1}{1 - p} \quad \text{(2.11)}
\]

![Parallel Processor Speed Multipliers Based upon Amdahl’s Law](image)

Figure 2.23: Parallel Processor Speed Multipliers Based upon Amdhal’s Law [2.47]

Figures 2.23 shows that when the parallelism p is 75%, the Speed multipliers approach 4 as the number of processors n gets larger; it also shows that at a certain stage adding more processor will not speed up the system. If p is 95% and n is 100, the speed multiplier is 16.8 and with 100 processors and 99% parallelism, the Amdahl speed multiplier is 50 [2.47]. When the stage at which adding more processor does not speed up the system, a serial
architecture for the remaining resources will contribute to logic resource reduction depending of the connection between resources. This will be demonstrated later in Chapter 6.

The connection between modules whether it be point-to-point, point-to-bus or Network On Chip (NOC) is the main factor to improve logic reusability. The NOC followed by the point to bus represents the highest flexibility connection. A good compromise between these three types of connection is possible and computing static parameters of the algorithm with software before compile time will also reduce the logic resources.

The DPR and MB flow procedure mentioned above can also reduce power consumption and logic resources at a given time in the hardware. Both flow partition their logic resources mainly in 3 groups:

- **First: Mutually exclusive modules**: These modules are interchangeable at run-time by their mutually exclusive counterparts or replaced by empty bistreams (EB) if not in used. The EB is mainly used to reduce logic resources therefore power consumption.

- **Second: Logics and primitives**: These are primitives or logic such as the Integral Configuration Access Port (ICAP) that are present in the design at all times but are required only periodically for hardware configuration. These primitives will be gate clocked and the controlling logic disables them when the FPGA is not in a reconfigurable mode in order to reduce power consumption.

- **Third: Control modules**: They handle task scheduling; synchronize data transfer between clock domains, when not in use these modules will run with slow clock for power consumption reasons.
2.10. Summary

In the light this work, it is seen that it is simpler to implement a reconfigurable design using the MB flow than the PR flow, and that this is also a lower cost solution. So a PR should only be used when a MB cannot satisfy the design constraint. Beside the testcase, used in this chapter is so slow that the reconfiguration time can be ignored during the system scheduling which represents an extra factor for the use of MB instead of PR. Nevertheless, incorporating both flows in the same design, on separate boards can increase tremendously the FPGA flexibility. That approach will give a reconfigurable capacity to the static region.

Looking at Figure 2.9 above, all FPGA 1, FPGA 2, FPGA 3 can be used as Partial Reconfiguration Region equivalent to PRR1 up to PRR3 of the Figure 2.8 and the region static 1, static 2 and static 3 in Figure 2.8 as the equivalent of FPGA 0 in Figure 2.9. In Figure 2.9 scenarios, the system flexibility is increased, as there is no longer an implementation limit depending on the primitive type, but also the static region corresponding to the respective PRR can be changed in a MB flow if necessary. Consequently, smaller FPGAs can be used in Figure 2.9 compared to Figure 2.8. The contribution of the proposed approach is to reduce development time, cost and power consumption by distributing the PR tasks over multiple smaller and less expensive MB FPGAs. Nevertheless, hybrid architectures that combine MB and PR presented in Figure 2.24 could have a reconfigurable static module placed in MB FPGA and the remaining logic in the PR FPGA. This configuration can resolve the issue of reconfigurable IP modules that remained on the static region due to the PR flow limitation.
2.11. Future Work and Application to Sound Source Computation

Reducing the resource utilization or power consumption is no longer the only research subject related to PR or MB. Subjects that are more complex have surfaced with the purpose of giving hardware a new level of flexibility. Concepts such as persistence, partial bitstream relocation, preventing reverse engineering, bitstream compression or bitstream encryption are other aspects that can be investigated.

There are also some competing approaches to PR and MB that are being investigated by the electronic industry.

- The growing number of software applications written using multiple threads can benefit from multi-core parallelism [2.31][2.32][2.41].

- Another software technique similar to multi-booting, thread warping, dynamically remaping critical code regions processor instructions to FPGA circuits using runtime synthesis [2.33]. The technique improves performance, speeds up individual threads which allows more threads to execute concurrently [2.34]. Binary-level partitioning or techniques using dataflow provide competitive results in terms of performance and energy [2.42].

MB and PR also require a scheduling and tasks allocation process where mutual exclusive tasks can be identified and allocated to PR regions and tasks reordered according to the reconfiguration time overhead. Both graph scheduling and resources allocation, can be automated by certain design tools [2.35][2.36][2.27][2.43].

Figure 2.25 shows two distinct approaches when using partial reconfiguration. The approach number (1) with internal microprocessor is mainly used when the overall system is solely composed of FPGAs. In this case an entire microprocessor needs to be dedicated to the reconfiguration purpose even if no other task requires the microprocessor's contribution in the entire system. At the present time only large scale processors such as Microblaze or PowerPC are available for this purpose, although this problem could be alleviated by using smaller processors or dedicated reconfiguration IP. The approach number (2) is used in hybrid system composed of (ASIC and FPGA) where the reconfiguration task can be assigned to the ASIC and therefore reduce the FPGA resources utilization.
This chapter has presented an effective but heretofore little used technique for FPGA reconfiguration, called Multiboot (MB). In the following chapters, which develop computationally intensive sound source localization techniques, it is shown that the multiboot and partial dynamic reconfiguration techniques can also be applied to this set of problems when FPGAs are used to implement the algorithms. In most cases, the two flows are applied to these more complex applications and conclusions drawn about their suitability.
CHAPTER 3 Sound Localization and Algorithm Specification

3. DECLARATION AND PRESENTATION

Most of the references used in this background chapter comes from, “Sound Capture and Processing” by Ian Tashev, Ian Cowan “Tutorial on Microphone arrays” and other reference papers.

3.1. Purpose of this Chapter

This chapter will present the following topics: The background knowledge on signal propagation and characteristic, different applications that benefit from sound source localization or share same algorithms, sensors used to capture sound source, their configurations, their features and then will briefly present tools necessary to perform the simulation model for localization and tracking algorithms. Finally, although it is not the main topic of this research study, this chapter will present different kinds of noise that can affect sound source localization.

3.2. Introduction and Definitions

The localization of sound sources using sensors arrays is a well-studied problem. The main sensors used for the localization purposes are antennas (active sensors) or microphones (passive sensors). The problem under investigation in this work did not require sending signals back to the source, so microphone arrays were used for these studies.

Microphones are sensors for capturing small changes in air pressure called sound and convert them into an electrical signal. The need to convert the sound to an electrical signal arose with the first telephone devices [3.1]. The microphone arrays used in this work consists of multiple microphones placed at different spatial locations built upon knowledge of sound
propagation principles. Multiple microphone inputs can be manipulated to enhance or attenuate signals emanating from particular direction [3.2]. Certain microphones are more sensitive to a signal coming from one direction than the other, which is known as directivity pattern.

Although the most common applications of this technology includes intelligent environments and teleconferencing, this technology can also be applied to two other applications that are less known; breast cancer detection and hearings aids [3.3] [3.4][3.5]. The focus of this work is to use a microphone arrays to receive acoustic signals, or more specifically, speech signals to detect their direction of arrival (DOA). While the use of sensor arrays for speech processing is a relatively new area of research, the fundamental theory is common to all sensor arrays, being based on the theory of wave propagation.

Sound source localization involves heavy computational loads, so the author concluded that the parallel processing capabilities supported by FPGA implementations make this technology a better candidate for implementing these algorithms than a general purpose processor or even a dedicated Digital Signal Processor (DSP). Previous results reported by other researchers, who have investigated similar applications, support this conclusion [3.14].

The remainder of Chapter 3 is divided as follow: Section 3.3 will present sound source propagation theory and beamforming theories. Section 3.4 presents the microphones’ geometric configuration and the type of noise considered. Section 3.5 present the system specification and briefly introduce signals generation. Section 3.6 summarizes this chapter.

3.3. Wave Propagation and BeamForming Background Theory

Sound waves propagate through fluids as longitudinal waves. The molecules in the fluid move back and forth in the direction of propagation, producing regions of compression and expansion. By using Newton’s equations of motion to consider an infinitesimal volume of the fluid, the equation governing the wave’s propagation can be developed. A generalised wave equation for acoustic waves is quite complex as it depends upon properties of the fluid. However, assuming an ideal fluid with zero viscosity; the wave equation can be derived [3.2].
\[ \nabla^2 X(t, r) - \frac{1}{c^2} \frac{\partial^2}{\partial t^2} X(t, r) = 0 \quad (3.1) \]

Where \( X(t, r) \) is a function representing the sound pressure at a point in time and space \( r [x, y, z] \). \( \nabla^2 \) is the Laplacian operator, \( C \) is the sound propagation speed depending upon the pressure and the density of the fluid, and is approximately 340 m/s in air. The wave equation (3.1) is known as the governing equation for a wide range of propagating waves, including electromagnetic waves. The solution to the differential wave equation can be derived using the method of separation of variables. The solution is well known and for a monochromatic plane wave is given \([3.2]\) as:

\[ x(t, r) = A \cdot e^{j(\omega t - k \cdot r)} \quad (3.2) \]

Where \( A \) is the wave amplitude, \( \omega = 2\pi f \) is the frequency in radians per second and the wave number vector \( k \) indicates the speed and the direction of wave propagation and is given by

\[ K = \frac{2\pi}{\lambda} [\cos \theta \sin \phi \quad \sin \theta \sin \phi \cos \theta] \]

with the wavelength \( \lambda \) related to \( C \) by the sample relation

\[ \lambda = \frac{C}{\omega} \]}

The speed of sound depends on the medium in which it moves. Table 3.1 presents different speeds depending on the material. This chapter will focus on sound propagation in the air [3.1].

<table>
<thead>
<tr>
<th>MATERIAL</th>
<th>AIR</th>
<th>WATER</th>
<th>ALUMINIUM</th>
<th>STEEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sound Speed (m/s)</td>
<td>343</td>
<td>1480</td>
<td>6400</td>
<td>5050</td>
</tr>
</tbody>
</table>

The propagation of a planar wave is illustrated in Figure 3.1. This is for the simple case of planar waves being received by a linear aperture. The aperture referring to a spatial region that receives propagating waves. The aperture response as a function of frequency and direction of arrival is known as the aperture directivity pattern or beam pattern.
Figure 3.1: Signal Received by a Linear Aperture [3.2]

The response of a receiving aperture is inherently directional in nature, because the amount of signal seen by the aperture varies with the direction of arrival (see Figure 3.1). This discussion will later show in Chapter 4 that the best localization is achieved when the signal seen by the sensor aperture is a maximum [3.2]. As seen from the diagram the maximum occurs when the sensor is facing the signal.

Figure 3.2: Planar Wave Fronts Arrival in Far-Field Mode [3.2]

The planar waveforms seen in Figure 3.2 appear when the sound source is placed in a far field compared to the microphone aperture. Conventional analysis and synthesis techniques for acoustic arrays are based on the assumption that the distance between the arrays is small compared to the distance of the sound source to the array. That restriction guarantees that only plane-waves need to be considered. Under this far-field assumption, the
main beam of the array is steered by delaying the individual sensor signals by an amount of sample equal to the propagation delay of a plane wave prior to summing [3.2]. The main condition that needs to be respected to be in far-field is governed by the equation 3.3 below
\[ |r| > \frac{2(Nd)^2}{\lambda} \]  
(3.3)

Where \( r \) represents the distance of the sound source to the reference microphone in the microphone arrays (see Figure 3.2), \( N \) is the number of microphones in the array (see Figure 3.2), \( d \) is the distance between two consecutives microphone (see Figure 3.2) and \( \lambda \) is the wavelength and must be computed with the highest signal frequency.

Figure 3.2 shows a planar arrival of the sound wave on a linear microphone arrays. It also shows that the actual distance traveled by the wave between adjacent sensors is modeled as in equation (3.4).
\[ d' = d \cos \phi \]  
(3.4)

Deriving from equation (3.4), the extra distance travel by the sound wave from the microphone \( n = 0 \) to the microphone \( N \) can be expressed by equation (3.5).
\[ d' = N.d \cos \phi \]  
(3.5)

The directivity pattern in a far-field mode is given by equation (3.6) below:
\[ D(f, \phi) = \sum_{n=1}^{N} w_n(f).e^{-\frac{j2\pi(N-1)d\cos \phi}{\lambda}} \]  
(3.6)

Where \( w_n \) represents the microphones weight which will be defined later.

Although the far-field assumption is the most used when working with microphone arrays, there are common applications where the plane-wave assumption can be an error. An example is the deployment of speech-bandwidth microphone arrays in small room. In such cases, the sound source is located in the arrays near field and wave front curvature is detected within the aperture of the array. Figure 3.3 shows a spherical waveform at the microphone level [3.2].

49
In the Figure 3.3 configuration, the distance traveled by waves between two consecutives microphones is modeled by equation 3.7.

$$d' = d_1(r, \phi) - d_0(r, \phi)$$  (3.7)

The distance from the microphone 0 to the microphone n is expressed as in equation (3.8):

$$d' = d_n(r, \phi) - d_0(r, \phi)$$  (3.8)

Where $d_n(r, \phi)$ is defined as in the equation (3.9):

$$d_n(r, \phi) = \sqrt{r^2 + 2r(x_n - x_0)\cos\phi + (x_n - x_0)^2}$$  (3.9)

If all the microphones are equidistant then equation (3.9) becomes equation (3.10):

$$d_n(r, \phi) = \sqrt{r^2 + 2r(n.d)\cos\phi + (n.d)^2}$$  (3.10)

The directivity in the near-field approach is then defined as in equation (3.11).

$$D(f, \phi) = \frac{\sum_{n=1}^{N} d_1(r, \phi) w_n(f) e^{j\frac{2\pi}{\lambda}(d_n(r, \phi) - d_0(r, \phi))}}{d_n(r, \phi)}$$  (3.11)

In the near field approach an attenuation factor needs to be taken into account and it is defined as:

$$\alpha_n = \frac{d_1(r, \phi)}{d_0(r, \phi)}$$
From the directivity pattern, the sensor array is capable of enhancing a signal arriving from a certain direction with respect to signal arriving from all the other directions. This enhancement is based purely on the direction of arrival, and is independent of the characteristics of the desired and undesired signals.

In general, the complex weightings \( w_n(f) \) is expressed in terms of its magnitude and phase components as:

\[
w_n(f) = a_n(f)e^{j\varphi_n(f)}
\]  

(3.12)

Where \( a_n(f) \) and \( \varphi_n(f) \) are real, frequency dependent amplitude and phase weights respectively. By modifying the amplitude \( a_n(f) \), the shape of the directivity pattern can be modified. Similarly, by modifying the phase weights, \( \varphi_n \), the angular location of the response’s main lobe is controlled. Beamforming techniques are algorithms for determining the complex sensor weights \( w_n(f) \) in order to implement a desired shaping and steering of the array directivity pattern. In this way the response of the array is controlled to enhance specific signals, provided the direction of the signal source is known with some accuracy. This condition is often met in many speech and speaker recognition applications [3.1].

This background theory given above is necessary to understand beamforming concepts and approximation made in the following chapters. More details on beamforming are in [3.1], [3.2].

### 3.4. Microphone Geometric Configuration and State of the Art

Microphone array geometry is mainly dependent on the sound source direction of arrival (DOA) and the location of the microphone arrays in the observation space. The distance between the microphones and their number are fixed. A system with 4 microphones and a distance of 4cm is presented in the following discussion.

This work will focus on the linear microphone arrays shown in Figure 3.4. Although different distance between microphones might improve beamforming depending on the signal frequency, a constant distance between microphones is kept to simplify the presentation of the hybrid algorithm developed to accelerate sound source localization.
Figure 3.4 represents the state of the art to improve localization as well as beamforming in a linear microphone arrays configuration. When building a system with the microphone as shown Figure 3.4, spatial aliasing must be taken into account. Spatial aliasing is one of the cause of secondary lobe also called (Grating Lobes) in the literature [3.7]. To reduce spatial aliasing, the distance between microphones must be constrained as in equation (3.13).

\[ d < \frac{\lambda_{\text{min}}}{2} = \frac{C}{2 \cdot f_{\text{max}}} \]

(3.13)

Where \( d \) is the distance between two consecutives microphones, \( C \) is the speed of sound defined in Table 3.1, \( f_{\text{max}} \) is the highest frequency of the signal and \( \lambda_{\text{min}} \) the wavelength.

Equation (3.13) is known as the spatial sampling theorem, and must be adhered to in order to prevent the occurrence of signal aliasing in the directivity pattern of a sensor array. If \( d \) is fixed to hold the spatial aliasing constraint for a maximum frequency, then there is a significant loss of resolution at low frequencies (Figure 3.4) presents a possible solution. Nested microphone arrays are not used in this research as they only provide a small improvement in the accuracy of the sound source localization but do not improve sound source localization throughput.

Figure 3.5 shows that the distance between microphones is a factor that needs to be taken seriously as it greatly impacts the directivity, especially for multiple speakers tracking in a confined area.
Directionality describes the pattern in which the microphone’s sensitivity changes when the sound source changes positions in space. Figure 3.5 shows how directivity varies depending on the distance between microphones. However few microphones are built in with a certain fixed directivity see Figure 3.6. The Bidirectional Response microphones are those used in the Laptop and speakerphones. The Hyper-Cardioids response microphones are good for theatrical scenes, capturing the sounds of the stage and suppressing sound coming from the audience [3.1] e.g (clapping or coughing). More types of directivity examples and their potential applications can be seen in [3.8]. This approach is a plus when the application is well known and when designers can guarantee that desired signals fall in the main lobe and the undesired does not.

However, the drawback with using the directional microphones illustrated in Figure 3.6, is that if the desired signal does not fall in the main lobe, it must be physically steered in the direction of the desired source. This is not practical in many situations, with one of the main reasons being the increased cost and risk of failure due to additional mechanical or electromechanical components. In additions, such a system could add more noise to measurements or limit the ability to track moving sources [3.8].

Figure 3.5: Directivity Pattern for Varying Effective Array Length (f=1 kHz, N=5) [3.2]
In this research an Omni-directional microphone was preferred meaning that it has the same sensitive in every direction. However a microphone with a 180° degree sensitivity is sufficient for the purposes of this investigation.

3.4.1. Microphone Principle and Description

Almost every microphone first converts the changes in air pressure to mechanical movements, due to a flexible membrane that bends under forces caused by the difference in air pressure on the two sides. These mechanical movements are then converted to an electrical signal [3.1]. The ways different microphones do this conversion categorizes their classification into various groups such as carbon, piezoelectric, electrodynamics, condenser and Electro-mechanical system (MEMS) microphones. In this work, digital MEMS microphones are used. They are manufactured from a silicon crystal with the same technology used for manufacturing integrated circuits. Conversion of the movements of the diaphragm to an electrical signal is based on varying the capacitance between the diagram and
the base, they are condenser microphones. The advantages of MEMS microphones are their small size (3x3x1 mm) and in addition, the accompanying electronic circuitry (preamplifiers, frequency correction, even part of the ADC – so called “digital MEMS”) can be integrated into the microphone chip using the same manufacturing process [3.1].

Analog Devices ADMP421 MEMS Microphones are high performance acoustic transducers featuring an extended wideband frequency response. The ADMP421 has many features that make it very suitable for design and easy to interface (See Figure 3.7). The most important feature comes from the MEMs output data pins, which can time share the same wire with another microphone that reduces the number of wires by half. Moreover when the clock is turned off, or the clock frequency falls below 1 kHz, the microphones enter into sleep mode. In this mode, the microphone data output is in a high impedance state. The current consumption in sleep mode is less than 50 µA [3.9]. More details on the MEMS microphones can be found in [3.9], [3.10].

![Figure 3.7: Two-ADMP421 Microphone Connected to a Single DATA Wire [3.10]](image)

Another very important challenge when working on sound source localization is the ambient noise and the noise generated by the MEMs microphone when converting the wave front to an electric signal. Different beamforming techniques are appropriate for different noise conditions. There are three main categories of noise fields to be defined for microphone array applications. These categories are characterized by the degree of correlation between noise signals at different spatial locations [3.2]. A commonly used measure of the correlation is the coherence function, which is defined as in equation (3.14).
\[ \Gamma_{ij}(f) = \frac{\Phi_{ij}(f)}{\sqrt{\Phi_{ii}(f) \cdot \Phi_{jj}(f)}} \] (3.14)

\( \Phi_{ij} \) is the cross-spectral density between signals \((i,j)\). In a coherent noise field \( \Gamma_{ij}(f) = 1 \) and in incoherent noise field \( \Gamma_{ij}(f) = 0 \). In a diffuse noise field, this is a noise of equal energy that is propagated in all directions simultaneously. Sensors in a diffuse noise field will receive noise signals that are lowly correlated, but have approximately the same energy. Many practical noise environments can be characterized by a diffuse noise field, such as office or car noise [3.2]. The coherence between the noise at any two points in a diffuse noise field is a function of the distance between the sensors, and can be modeled as in equation (3.15):

\[ \Gamma_{ij}(f) = \text{sinc} \left( \frac{2\pi f \cdot d_{ij}}{C} \right) \] (3.15)

where \( d_{ij} \) is the distance between sensors \((i,j)\).

3.4.2. System Specification and Signals Generation

To produce this work the following system was built. The distance between the microphones and their number is fixed. A system with 4 microphones and a distance of 4cm is presented in the following discussion.

Both signal frequencies of 44.1 or 48 kHz can be used but the sample resolution can be limited to 16 bits instead of 24 bits for resources utilization and power consumption reduction. In general the assumption can be made that the human speech is a signal with frequency content mostly concentrated between 100 and 8000 Hz. But it can be further reduced to 300-3400 Hz, so called “telephone quality” because this is the bandwidth used in communications [3.1].

This system is limited in terms of speed by the acquisition of data by the MEMs microphones, which is limited to 2.4 MHz and by the computation chain speed, which depends on the hardware chosen as a free running clock or system clock (100 MHz for Virtex-5 or 200 MHz for Virtex-6). Inside the FPGA the system clock can be increased using a Clock Management Tile (CMT) primitive to accelerate the computation of selected modules.
The choice of the field of view (FOV) area and its resolution is related to the application. A FOV is the region in the space where the sound source is likely to be found and the resolution is the measurement of the smallest distinguishable region. The number of small squares (NOSS) is related to the FOV area and its resolution and they are linked by equation (3.16).

\[ \text{NOSS} = \frac{\text{FOV}}{\text{Resolution}} = \frac{L \times H}{\Delta x \times \Delta y} \]  

\[ \text{(3.16)} \]

Figure 3.8: 2D 3x3 FOV with 150cm x 150cm Size and a 50cm x 50cm Resolution

In Figure 3.8 L and H are the length and the height of the FOV and \( \Delta x \) and \( \Delta y \) are the length and height of a small square (SS) of the FOV. This concept will be explained further in Chapter 4. A two-dimension system is sufficient to demonstrate the novelty of the algorithm developed in this work.
3.5. Summary

The purpose of this chapter was to familiarize the reader with the vocabulary that will be used in the remainder of this work and to present general theory about sound source localization. Based on the analysis of the background theory presented in this chapter, the following design decisions were made for the research project:

- A MEMS Microphone was chosen because its output is in digital format, which simplifies connection to any FPGA.
- The far field approximation will mainly be used,
- A two-dimensional system is sufficient for this research.
- A Linear microphone configuration with a fixed distance between microphones was preferred
- The experimental set up allowed the effects of different FOV areas and resolutions to be explored. These will be discussed in Chapter 4 and Chapter 5 to assess their impact on a real-time application

As further work, different microphone position configuration can be investigated to assess their contribution to the accuracy of sound source localization. Other applications such as cancer tumor detection or hearing aid machines can profit from the algorithm that will be used in Chapter 4 for sound source localization [3.3][3.5].
4. DECLARATION AND PRESENTATION

This Chapter is the fruit of two published papers entitled: “Combining sound source tracking algorithms based on microphone arrays to improve real-time localization” [4.16] and “Novel interface drivers to combine real-time localization and tracking algorithms”[4.14]. These two papers are listed in Section 1.5 of Chapter 1. The MATLAB code used throughout this chapter is provided in Appendix [A].

4.1. Purpose of this Chapter

The purpose of this chapter is to present a novel hybrid algorithm created during this research work to accelerate the computation of the Delay and Sum Beamforming (DSB) algorithm by reducing its search spectrum without affecting its performance. The DSB throughput can be improved using the hybrid algorithm that confines the DSB search spectrum to the sound source direction of arrival (DOA). The DOA is first found by computing the GCC (Generalized Cross Correlation) Algorithm. Then the hybrid algorithm drivers convert the GCC results to a reduced FOV (Field Of View) used by the DSB. The results improve the DSB throughput by more than 80% and make it a good candidate for real-time application. In addition to the algorithmic contribution, this work proposes a means of merging the DSB and GCC architectures for a possible hardware implementation of the hybrid algorithm at a later stage.
4.2. Introduction

The auditory system of living creatures provides a vast amount of information about the world, such as the localization of sound sources. For us humans, it means to be able to focus our attention on events and changes surrounding us, such as a cordless phone ringing, a vehicle honking, a person who is talking to us, etc. Hearing complements other senses such as vision by being omni-directional. Hearing is capable of working in the dark and is not limited by a physical structure such as a wall [4.1].

Sound source localization (SSL) involves finding a set of coordinates of the sound source relative to a point in space. Many applications ranging from teleconferencing to artificial perception use the localization and spatial filtering of one or more acoustic sources. Sound propagation itself is a three-dimensional process, so capturing it in one single point with one microphone is not sufficient. It is even more insufficient if ambient noise, reverberation and multiple sound sources are involved. Using several closely positioned microphones creates the ability to listen to the sound coming from one direction, while suppressing noises and interference sounds coming from other directions. Beamforming is a signal processing technique used to both locate the sound source and concentrate the sensor beam on that particular direction.

This work will assess the capacity of the proposed novel hybrid algorithm to achieve sound source localization in real-time. In the most optimistic scenario, real-time in this work is defined as the capacity to locate the sound source in the frame j before the acquisition of the frame j+1 has been completed. In the most pessimistic scenario, it is the capacity to use a buffering system to allow each audio frame to be processed without regard to the processing latency for individual frames.

Table 4.1 shows audio frames length based on the sampling frequency and the computation time required to achieve real-time using the most optimistic definition. During a short period between 5 to 100 ms, the speech sound is stationary. However, over a longer period such as 250 ms or more the signal characteristic changes to reflect the different variations of speech [4.2].
Table 4.1: Frames Size and Acquisition Time

<table>
<thead>
<tr>
<th>Frames size $N_s$</th>
<th>Time (ms) Fs = 44.1 KHz</th>
<th>Time (ms) Fs = 48 KHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>5.805</td>
<td>5.34</td>
</tr>
<tr>
<td>512</td>
<td>11.61</td>
<td>10.67</td>
</tr>
<tr>
<td>1024</td>
<td>23.22</td>
<td>21.34</td>
</tr>
<tr>
<td>2048</td>
<td>46.44</td>
<td>42.67</td>
</tr>
<tr>
<td>4096</td>
<td>92.88</td>
<td>85.34</td>
</tr>
</tbody>
</table>

In most cases, it is preferable to use the smallest frame size 256 for the following reasons:

- Faster sound source localization (higher throughput)
- Less resources storing elements necessary for computation
- Possibility of using a radix-4 approach for FFT computation

The remainder of this chapter is organized as follows: Section 4.3 will present microphone arrays, localization and tracking algorithms. Sections 4.4 will describe the GCC algorithm. Section 4.5 presents the DSB algorithm. Section 4.6 will present the contribution of this work, which is the creation of a hybrid algorithm that combines the GCC with the DSB. Section 4.7 will discuss this work's contributing results and propose combined architecture for the GCC-DSB algorithm. Section 4.8 presents novel drivers to allow the GCC to communicate to the DSB. Section 4.10 summarizes the chapter.

4.3. Microphone arrays and beamforming Algorithms

Microphone arrays, consisting of a set of microphone sensors, spatially arranged in specific patterns, have been studied for more than three decades. One of the most important functionalities of microphone arrays is to extract the speech of interest from its observations corrupted by noise, reverberation, and competing sources. This is done by aiming the beam towards the desired sound source. As a result, signals from this so-called “look-direction” are reinforced while signals from all other directions are attenuated [4.3]. The basic idea behind beamforming is to sum up the contribution of each microphone after appropriate filtering and look for directions that maximize that sum [4.4].
Figure 4.1 give an illustration of the beamforming concept. The blue signals represent the signal arriving at the microphones and the red signals are the blue signals shifted to match the reference blue signal. In this case the signals arrive first at Microphone 3, so the red signals are shifted to match with it.

A forward or backward shifting can be applied to the signals. In this work, a forward shifting was preferred since it reduces the number of toggle bits of the counting register. After the signal shifting, the signals are added constructively. Equation (4.1) represents the number of sample shifts from the other microphone signals compared to the reference Microphone 3. As the sound source is far enough away from the microphone arrays, the difference between the signal received by the $n^{th}$ microphone $x_n$ and the reference point of the array is a pure delay [4.5] expressed as in equation (4.1):

$$
\tau_n = \frac{f_s \cdot d_n \cos \phi}{C}
$$

(4.1)

Where $\tau_n$ represents the delay per number of samples, $f_s$ is the sample frequency, $d_n$ is the extra distance travelled by the wave sound to reach the $n^{th}$ sensor compared to the reference sensor $\phi$ is the wave front incident angle.
No multipath contribution is taken into account in the equation (4.1). The assumption is made that the sound wave propagation is identical to the sound source to each microphone. The only sources of noise are the microphone themselves and stationary noises. No diffuse noise field model will be presented in this chapter [4.6].

There are two major groups of microphone arrays processing algorithms: time-invariant and adaptive [4.7]. The first group is optimal under the assumption of isotropic ambient noise. These algorithms are fast and it is simple to get a real-time working implementation using them. Acoustic adaptive algorithms are able to adapt automatically their response to different weightings or time-delays to minimize the output noise. Adaptive algorithms are most suitable when there are punctual noise sources in low reverberant conditions. However, they require more CPU power and are complex to implement. Both approaches assume identical sound source capture and are affected by mismatch caused mainly by the manufacturing tolerances of the microphones used. This Chapter focuses on two time invariant algorithms, the GCC and DSB for sound source localization. Some adaptive beamformings are similar to time invariant algorithm, when noise and sensors are spatially uncorrelated [4.8]. They are discussed in Chapter 5.

4.4. The Generalized Cross Correlation Algorithm (GCC)

The GCC computation burden is linked to the number of cross-correlations it needs to compute. The number of cross-correlation is modeled as in equation (4.2)

$$C_N^P = \frac{N!}{P!(N - P)!}$$  \hspace{1cm} (4.2)

Where $P$ is always equal to 2 as the cross-correlation is always computed between two microphones. $N$ represents the number of microphone in the array = [4, 8, 16, 32, 64] etc.

The number of cross-correlation and IFFT (Inverse Fast Fourier Transform) modeled as in equation (4.2) varies according to the number of microphones as shown in Figure 4.2. The central processing unit (CPU) power increases with the number of microphones. For 64 microphones (NB MICRO), the number of cross correlations (CROSS-COR-NB) is so immense that it is almost impossible to have a real-time application without a very flexible architecture.
4.4.1. Microphone sub-array Triangulation

The GCC Algorithm only provides an angular localization, but the sound source distance to the microphone arrays is unknown to the user. To find the sound source location using the GCC two microphone sub-arrays are required as shown in Figure 4.3.(a) and Figure 4.3.(b). The first sub-microphone arrays is composed of microphones \{1, 2, 3\} whereas the second is composed of microphones \{4, 5, 6\}. Using both sub-microphone arrays to localize the same sound source, they return respectively the angles $\theta_1, \theta_2$ [4.9]. The crossing of the two lines drawn with an inclination $\theta_1$, and $\theta_2$ is the sound source location which is defined as point \((x_s, y_s)\), as seen in Figure 4.3.
The angle $\theta$ represents the direction of arrival (DOA) of the sound source with reference to the center of the array. The lack of accuracy is one of the major problems with the microphones sub-arrays approach. Figure 4.3 (b) shows that slight errors in the computation of $\theta_1$ or $\theta_2$ angle will lead to a wrong detection of the sound source location [4.9]. Using the estimation of angle $\theta_1$, $\theta_2$ as in Figure 4.3, the coordinates of the point $(x_s, y_s)$ [4.9] can be computed as in equation (4.3) and (4.4) or as in (4.5) and (4.6).

\begin{align*}
    r &= \sqrt{x^2 + y^2} \quad (4.3) \\
    \theta &= \arctan \frac{y}{x} \quad (4.4) \\
    \text{point } (x_s, y_s) &= \left( \frac{D \sin(\theta_1 + \theta_2)}{2 \sin(\theta_2 - \theta_1)}, D \frac{\sin \theta_1 \sin \theta_2}{\sin(\theta_2 - \theta_1)} \right) \quad (4.5) \\
    \text{point } (r, \theta) &= \left( \sqrt{x^2 + y^2}, \arctan \frac{y}{x} \right) \quad (4.6)
\end{align*}

$D$ is the distance between the centers of the two microphones sub-arrays. The distance from the center of the microphone arrays to the sound source is denoted by $r$.

Figure 4.4 and 4.5 were drawn, by using the Figure 4.3 configuration with eight microphones at a distance of 5 cm from each other. A sound source is placed at a distance of 2 meters from the microphone arrays and at different angles varying from $0^\circ$ to $180^\circ$ with a step of $10^\circ$. 

![Figure 4.3: Estimation of the Sound Source Position using GCC Sub-Array [4.9]](image)
Figure 4.4: Estimation of the Incident Angle [4.9]

Figure 4.4 shows that the estimated sound source incident angle has an error which varies from 1° to 20° and that the closer the sound source is to 90° the less the error.

Figure 4.5: Estimation of the Sound Source Distance [4.9]

Figure 4.5 shows that the estimation error is even worse when trying to estimate the distance of the sound source to the microphone arrays reference point using the sub-
microphone arrays technique. The estimation error can vary from 1 to 7 meter from the exact
distance of the sound source. In this case also the error is smaller when the DOA angle of the
sound source is close to 90°.

4.4.2. Summary

It has been shown that the GCC is not very accurate, but can be used as a first
approximation of the position of the sound source. The number of operations due to the
cross-correlation can also be reduced by taking one every three cross-correlations for instance.
The next section will describe how to find the sound source direction of arrival (DOA) angle \( \phi \) for a single linear microphone arrays.

4.4.3. GCC and \( \beta \)-PHAT

The concept used in this section is the time delay of arrival (TDOA). The main
approach is to find the time shift \( \tau \) between a signal acquired by two different microphones
\( f(t) \) and \( g(t) \) that gives the maximum Cross-Correlation (CC) \( R(\tau) \) between them. The cross
correlation is defined as in equation (4.7).

\[
R(\tau) = \int_{-\infty}^{\infty} f^*(t) g(t + \tau) dt
\]  

(4.7)

\( f^* \) represents the complex conjugate of function \( f \). In the discrete time domain, equation (4.7) becomes equation (4.8).

\[
R(k) = \sum_{n} f^*(n) g(n + k)
\]  

(4.8)

In many case, for computation load reasons, it is better to work in the frequency domain.
Equation (4.8) then becomes equation (4.9).

\[
R(\omega) = \int_{-\infty}^{\infty} F^*(\omega) \cdot G(\omega) e^{j\omega \tau} d\omega
\]  

(4.9)

\( F(\omega) \) and \( G(\omega) \) are respectively the Fourier transform of functions \( f(t) \) and \( g(t) \). In
practice, the cross correlation function is computed on frames of a finite length \( N_f \). Equation
(4.9) then becomes equation (4.10).

\[
R(k) = \sum_{n=0}^{N_f-k} f^*(n) g(n + k)
\]  

(4.10)
Using the inverse Fourier transform of the function $F^*$ and $G$ the computation of equation (4.10) becomes equation (4.11).

$$R(k) = IFFT[F^*(\omega) \cdot G(\omega)]$$  \hspace{1cm} (4.11)

The index $k$ that gives the highest cross correlation $R$ from signal $F^*(w)$ and $G(w)$ will be the object of interest, it is given by equation (4.12).

$$\tau_{fg} = \arg\max_k R(k)$$  \hspace{1cm} (4.12)

The cross-correlation function as defined in equation (4.7) is very sensitive to noise or to a slight change to the signals’ amplitude, therefore a corrective or normalization factor is necessary and defined as in equation (4.13).

$$\varphi_{PHAT}(\omega) = \frac{1}{|F^*(\omega).G(\omega)|^\beta}$$  \hspace{1cm} (4.13)

$\beta$ is a constant value that can vary from 0 to 1. If $\beta = 0$ then equation (4.15) is a simple cross-correlation such as equation (4.9), if $\beta = 1$ then the influence of the magnitude is totally removed from equation (4.15). In $\varphi_{PHAT}^{(w)}$, PHAT is the acronym for phase transformation.

Another normalization or coefficient factor called the smooth coherence transformation (SCOT) with an approximately similar result as the PHAT is defined in equation (4.14).

$$\varphi_{SCOT}(\omega) = \frac{1}{\sqrt{(F^*(\omega)F(\omega)).(G(\omega)G^*(\omega))}}$$  \hspace{1cm} (4.14)

The computation of the GCC could be faster when using the SCOT factor with equation (4.15) rather than the PHAT normalization factor.

$$R(\tau) = \int_{-\infty}^{\infty} \varphi(\omega)F^*(\omega) \cdot G(\omega)e^{j\omega \tau} d\omega$$  \hspace{1cm} (4.15)

In the discrete time domain equation (4.15) becomes equation (4.16).

$$R(k) = IFFT\left(\frac{FFT(f^*(t)).FFT(g(t))}{\left|FFT(f^*(t)).FFT(g(t))\right|^\beta}\right)$$  \hspace{1cm} (4.16)
Using SCOT or PHAT to compute one bin of equation (4.16) does not influence its numerator, so the result is common for both factors as shown in equation (4.17).

\[ F^*(\omega).G(\omega) = (a + ib)^*(c - id) = (ac - bd) + i(bc - ad) \]  \hspace{1cm} (4.17)

The Denominator of one bin computed with the PHAT and SCOT weight are described respectively with equation (4.18) and equation (4.19).

\[
\left| F^*(\omega).G(\omega) \right| = \sqrt{(a + ib)^*(c - id)} = \sqrt{(ac - bd)^2 + (bc - ad)^2} 
\]  \hspace{1cm} (4.18)

\[
\left| F^*(\omega).F(\omega) \right| \left| G(\omega)G^*(\omega) \right| = \sqrt{(a^2 + b^2)(c^2 + d^2)} 
\]  \hspace{1cm} (4.19)

Figure 4.6 shows the difference in the numbers of multiplications and additions when computing the GCC (equation (4.16)) using SCOT or PHAT weight. The difference in computation load depends on the frame size. Therefore, the importance of the weight choice can be seen over a long computation period. For example for a 1.161 second recording at a sample frequency of 44.1kHz, 100 frames of \( N_s = 512 \) length are processed. In this scenario, the GCC computation using PHAT weight, requires 51,200 more multiplications and additions compared to the computation using the SCOT weight.

![Figure 4.6: Computation Load of the GCC with SCOT or PHAT Weight](image)

The maximum delay computed in equation (4.12) using (4.16) symbolized in the time domain by \( \tau_{ij} \) is then used to compute the DOA angle \( \phi \) between two microphones using equation (4.20).
\[ \phi_{ij} = \arccos \left( \frac{c \cdot \tau_{ij}}{d_{ij} \cdot f_s} \right) \]  

(4.20)

To increase the robustness of the estimation, the average is computed between every microphone pair as in equation (4.21).

\[ \phi = \frac{1}{C_N^2} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \phi_{ij} \]  

(4.21)

### 4.4.4. Proposed GCC Architecture and its associated Throughput

Figure 4.7 shows the block diagram of the sound source localization (SSL) using the GCC technique. The block (1) in Figure 4.7 is the audio system acquisition chain, the block (2) is the GCC computation chain and the block (3) is the computation of the sound source DOA. Figure 4.7 shows an interpolation module on every branch. This approach is very useful to reduce the sampling frequency and by so doing reduces the hardware storage required [4.10]. The interpolation approach is not required in this work as most of the FPGA have sufficient Block-RAM to store data.

![Figure 4.7: Localization and Tracking of Sound Source with the GCC [4.9]](image-url)
Table 4.2 shows the number of operations and Block-RAM read accesses to compute the GCC using PHAT weight with a frames length $N_s = 512$ samples. This computation does not include the IFFT computation. The term complex in Table 4.2 refers to the numbers of multiplications between two complex numbers.

Table 4.2: Computation Load of the GCC

<table>
<thead>
<tr>
<th>GCC (PHAT)</th>
<th>$N_s = 512$ and $\beta = 1$ Group $(2+3)$ Fig 4.7</th>
<th>Number of Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLK-read</td>
<td>$2 \times (C^p_n \times N_s)$</td>
<td>6144</td>
</tr>
<tr>
<td>MULT</td>
<td>$2 \times (C^p_n \times N_s) + ((C^p_n \times N_s) \times \text{complex}) + 1$</td>
<td>18433</td>
</tr>
<tr>
<td>ADD</td>
<td>$3 \times (C^p_n \times N_s) + (2 \times N_s) + (C^p_n - 1)$</td>
<td>10245</td>
</tr>
<tr>
<td>DIV</td>
<td>$2 \times (C^p_n \times N_s) + 1$</td>
<td>6145</td>
</tr>
<tr>
<td>SQRT</td>
<td>$(C^p_n \times N_s)$</td>
<td>3072</td>
</tr>
</tbody>
</table>

Table 4.3 presents the GCC throughput using SCOT and PHAT weights. It shows that the impact of the PHAT weight compared to the SCOT weight is negligible in terms of throughput, so the weight choice can be based on the resource utilization and quality of sound source detection. In Table 4.3, the parallelism section represents the duplication of the GCC module whether with a pipelined or sequential approach. The pipelined approach only applies to the computation of the square root and division for the time being. Table 4.3 computations are made assuming that most operations are performed in one clock cycle except for the division and square root (8 clock cycles). The number of microphones for this experiment is $N=4$ and the number of clock cycles by frames (NCCF) is given by equation (4.22).

Based on the results in Table 4.2 and with the number of microphones for this experiment set to $N=4$, Table 4.3 can be computed using the number of clock cycles by frames (NCCF) in equation (4.22) divided by the clock frequency.

$$NCCF = \text{Mult} + BLK_{\text{read}} + \text{Add} + 8(\text{Div} + \text{Sqrt}) \quad (4.22)$$
Table 4.3: GCC Architecture Throughput Depending of the Weight

<table>
<thead>
<tr>
<th>GCC Throughput CLK (200 Mhz)</th>
<th>( \phi_{SCOT} )</th>
<th>( \phi_{PHAT} )</th>
<th>( \beta = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>540 µs</td>
<td>543 µs</td>
<td></td>
</tr>
<tr>
<td>Parallel Sequential</td>
<td>270 µs</td>
<td>272 µs</td>
<td></td>
</tr>
<tr>
<td>Parallel Pipeline</td>
<td>109 µs</td>
<td>110 µs</td>
<td></td>
</tr>
</tbody>
</table>

4.4.5. Summary

GCC is a very fast algorithm as shown by comparison of the throughput times shown in Table 4.3 to the constraint set in Table 4.1. Figure 4.2 shows that for a system that has more than eight microphones, either a huge parallel architecture is required or an alternative approach to compute the GCC must be found to account for the algorithm asymmetry. The purpose of such an approach would be to reduce the number of cross-correlations and IFFT calculations. Another drawback of the GCC is its lack of accuracy in providing the exact DOA of the sound source and its location as shown by Figure 4.4 and Figure 4.5. Therefore, a slight error should be taken into consideration when detecting the sound source DOA angle.

4.5. Delay and Sum Beamforming (DSB)

The DSB is a localization and beamforming algorithm that localizes the sound source through hypothesis testing and pick as a sound source location the point in space which produces the highest Steered Response Power (SRP) [4.11]. Therefore, a region called a field of view (FOV) must be defined giving all the positions in that region where the sound source can possibly be found. Figure 4.8 shows an example of a FOV with a certain number of small squares (NOSS) which are resolution dependent. The NOSS represents the number of points in the FOV where the sound source can be located. The resolution is the measurement of the smallest distinguishable region, which is a possible location of the sound source. Thus, The DSB computation load is linked to the NOSS defined in Chapter 3 equation (3.16). In Figure 4.8, the NOSS equal 256. L and H are the length and the height of the FOV and \( \Delta x \) and \( \Delta y \) are the length and height of a small square (SS) of the FOV. The red point in the FOV represents a possible location of the sound source.
Figure 4.8: Full FOV Representation for DSB Approach

$h^+_n$ and $h^-_n$ are lines that are drawn above and below the line representing the direction of arrival of the sound source computed by the GCC to account for the GCC slight inaccuracy presented in Figure 4.4 and Figure 4.5. The cone presented by \{0, X, Z\} will then be the new region where DSB algorithm would be applied to locate the sound source. More details about $h^+_n$ and $h^-_n$ are given later in this Chapter.

Lets assume that N is the number of microphones in a microphone arrays. The signal received at a microphone m where m = 1,..,N is modeled as in equation (4.23):

$$x_m[t] = s(t) + n_m(t) \tag{4.23}$$

$n_m(t)$ is the additive noise and s(t) the sound source signal. Even if the reverberation is ignored, the signal will arrive at each microphone at different times. The DSB selects the location in the space that maximizes the sum of the delayed received signals [4.11]. Mathematically the output of the delay and sum (DSB) at time t is modeled as defined in the equation (4.24).
\[ y[t] = \frac{N_s}{m=1} \sum w_{im} x_m (t - \tau_{im}) \]  

(4.24)

\( w_{im} \) is the weight of the i-th small square SS\( i \) relative to the microphone m. \( \tau_{im} \) is the sound travel time between the point \( i \) to the microphone m. \( w_{im} \) is modeled as in equation (4.25).

\[ w_{im} = \frac{1/d_{im}}{\sum_{n=1}^{N} 1/d_{im}} \]  

(4.25)

\( d_{i,m} \) is the distance between the point \( i \) of the FOV to a microphone m. \( d_{i,m} \) is modeled as in equation (4.26).

\[ d_{im} = \sqrt{(x_i - x_{mic})^2 + (y_i - y_{mic})^2 + (z_i - z_{mic})^2} \]  

(4.26)

The pairs \( (x_i, y_i) \) and \( (x_{mic}, y_{mic}) \) are respectively the coordinates of the point \( i \) and the microphone m. The term \( (z_i - z_{mic}) \) is ignored for simplification reasons, which will imply that the sound source is located at the same height as the microphone arrays. The term \( \tau_{im} \) is defined as in equation (4.27) below.

\[ \tau_{im} = \frac{d_{im}}{c} \]  

(4.27)

The weight \( W_{N,NOSS} \) and the distance \( D_{N,NOSS} \) are matrices of N microphones by the number of NOSS as shown in equation (4.28)

\[
\begin{pmatrix}
  w_{1,1} & \ldots & w_{1,NOSS} \\
  w_{N,1} & \ldots & w_{N,NOSS}
\end{pmatrix}
\quad
\begin{pmatrix}
  d_{1,1} & \ldots & d_{1,NOSS} \\
  d_{N,1} & \ldots & d_{N,NOSS}
\end{pmatrix}
\]  

(4.28)

The SRP is defined in the time domain by equation (4.29) proposed by Donohue [4.12].

\[ SRP_i = \sum_{n=1}^{N_s} \left[ \sqrt{\left( \sum_{m=1}^{N} w_{im} x_m (n - \tau_{im}) \right)^2 - \sum_{m=1}^{N} w_{im}^2 x_m^2 (n - \tau_{im})^2} \right] \]  

(4.29)

The point with the highest SRP is the sound source location (see equation (4.30)).
Multiple other variants of DSB in the time domain such as the interpolation and shifted-sideband beamforming and in the frequency domain such as DFT (Discrete Fourier Transform) and phase shift beamforming are described in [4.10]. There are not described more in this research as the main novelty which is to improve the DSB throughput by reducing the FOV size remains unchanged. Figure 4.9 and Figure 4.10 represents respectively the architecture of delay and sum beamforming in the time domain modeled by equation (4.24) and the Frequency domain beamforming modeled as in equation (4.31).

$$y[t] = \sum_{m=1}^{Ns} w_{im} X_{m}(f) e^{-j2\pi f\tau_{im}}$$  \hspace{1cm} (4.31)
4.5.1. DSB Architecture throughput and number of operations

Figure 4.9 and Figure 4.10 provide a general overview on the computation steps of the DSB but they do not include the signal acquisition chain and so do not constitute a complete system. Figure 4.11, however, is the complete block diagram of a sound source localization (SSL) system using DSB. Figure 4.9 and Figure 4.10 appear in Figure 4.11 as part 2 and part 3 respectively. Figure 4.11 however is the complete block diagram of sound source localization (SSL) using DSB. The part 1 of the block diagram is the acquisition chain, which is composed of a sigma-delta demodulation, implemented with a Cascade Integrator Comb (CIC), a Voice Activity Detector (VAD) and a Framer. All these modules are detailed further in Chapter 6. DSB (Figure 4.11) and GCC (Figure 4.7) acquisition chains are similar and therefore will be merged in the hybrid algorithm architecture introduced in Section 4.7. The part 2 of Figure 4.11 is composed of (IFFT, $\beta$-PHAT). It is partly similar to Figure 4.7 part 2 and can better share their resources if the SCOT weight approach is used for the GCC computation. The $\beta$-PHAT in DSB algorithm in Figure 4.11 is modeled as in equation 4.32 below.

$$W(f) = \frac{S(f)}{|S(f)|^\beta} \tag{4.32}$$
$S(f)$ is the signal spectrum, $W(f)$ is the modified spectrum after the $\beta$-PHAT with $\beta$ varying between 0 to 1. Figure 4.11 part number 3 is composed of modules $\{D, W, \tau\}$ representing the delay and sum computation which are defined in equations 4.25, 4.26 and 4.27. Finally, the SRP returns the SSL point as defined in equation (4.30).

![Figure 4.11: DSB Block Diagram](image)

Table 4.4 represents the number of operations and Block-RAM read accesses necessary to compute the DSB in an FPGA implementation.

<table>
<thead>
<tr>
<th>Function</th>
<th>Localization</th>
<th>$N_{oas} = 256$</th>
<th>$N_{oas} = 32$</th>
<th>$N_{oas} = 25$</th>
<th>$N_{oas} = 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Bk_{read}$</td>
<td>$[(N_s \cdot N_{data} + (N_s \cdot N_{weights}) \cdot N_{oas} + N_{oas} + [(2N_s \cdot N)<em>{\beta - PHAT}] \cdot N</em>{oas}]$</td>
<td>1575168</td>
<td>196896</td>
<td>153825</td>
<td>73836</td>
</tr>
<tr>
<td>$MULT$</td>
<td>$[(N_s \cdot N_{weights} + (N_s \cdot N_{DPSB} + (N_s \cdot N_{DPSB}) + (1_N \cdot N_{DPSB})) \cdot N_{oas} + [(2N_s \cdot N_{oas} + [(2N_s \cdot N)<em>{\beta - PHAT}] \cdot N</em>{oas}$</td>
<td>2228480</td>
<td>278560</td>
<td>217625</td>
<td>104460</td>
</tr>
<tr>
<td>$ADD$</td>
<td>$[(N_s \cdot (N - 1))<em>{DPSB} + (N_s \cdot (N - 1))</em>{DPSB} + (N_s \cdot (N - 1))<em>{DPSB}] \cdot N</em>{oas} + [(N_s \cdot N)<em>{\beta - PHAT}] \cdot N</em>{oas}$</td>
<td>1572864</td>
<td>196608</td>
<td>133600</td>
<td>73728</td>
</tr>
<tr>
<td>$DIV$</td>
<td>$2N_s \cdot N_s \cdot N_{oas}$</td>
<td>1048576</td>
<td>131072</td>
<td>102400</td>
<td>49152</td>
</tr>
<tr>
<td>$SQRT$</td>
<td>$(N_s \cdot N) \cdot N_{oas}$</td>
<td>524288</td>
<td>65536</td>
<td>51200</td>
<td>24576</td>
</tr>
</tbody>
</table>

Table 4.5 shows that the DSB cannot be computed sequentially in real-time for any NOSS value. As defined above the computation must be completed in less than 11.61 ms for a frame size of $N_s = 512$ (see Table 4.1). The 200 MHz clock represents the quartz speed for
the Xilinx Virtex-6 FPGA family. The sequential computation is preferred as it is more suitable to reconfigurable flows and for power consumption reduction.

<table>
<thead>
<tr>
<th>DSB Throughput</th>
<th>NOSS = 128</th>
<th>NOSS = 256</th>
<th>NOSS = 512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>45 ms</td>
<td>90 ms</td>
<td>180 ms</td>
</tr>
<tr>
<td>Pipeline</td>
<td>17.38 ms</td>
<td>34.75 ms</td>
<td>75.5 ms</td>
</tr>
<tr>
<td>Parallel Sequential</td>
<td>22.5 ms</td>
<td>45 ms</td>
<td>90 ms</td>
</tr>
<tr>
<td>Parallel Pipeline</td>
<td>8.69 ms</td>
<td>17.37 ms</td>
<td>37.5 ms</td>
</tr>
</tbody>
</table>

4.5.2. Summary

Although DSB is known to be accurate, its computation loads are colossal, therefore cannot be computed in real-time without using multiple parallel resources with fast running logic.

4.6. DSB Acceleration Techniques

The above analysis has shown that DSB computation is very tedious, complex and long. It has also shown that the GCC is a fast algorithm compared to the DSB but is less accurate. The author’s main contribution in this field is to create a novel hybrid algorithm that combines the speed of the GCC and the accuracy of the DSB. Therefore novel drivers need to be created to enable the DSB to interpret the result of the GCC. The main concept is to reduce the Search Region of the FOV to localize faster the sound source using the DSB. This novel hybrid algorithm uses first the GCC to pre-compute the sound source direction of arrival (DOA) angle $\phi$ varying from $0^\circ$ to $180^\circ$. Three different approaches developed during this work have been compared for their ease of computation and their aptitude to reduce the FOV NOSS. They are referred to as GCC1-DSB, GCC2-DSB and GCC-DSB.

4.6.1. GCC1-DSB

In this approach, the FOV is divided in 4 regions as denoted in Figure 4.12 as {C1, C2, C3, C4}. Once the GCC is computed, only regions crossed by the GCC angle $\phi$ are computed using the DSB. Figure 4.12 shows 4 different scenarios, in two cases, $\phi < 45^\circ$ and $\phi > 135^\circ$.
which are Figure 4.12(a) and Figure 4.12(d), respectively. The DSB search region is reduced by 75 % as only the region C3 or C2 needs to be searched using the DSB. Figure 4.12 (b) and Figure 4.12 (c) correspond to $45^\circ < \phi < 90^\circ$ and $90^\circ < \phi < 135^\circ$ respectively, and they reduce the FOV spectrum by 50% as only regions {C3,C4} and {C2, C1} need to be searched. Looking at Table 4.5 with the results of this approach a NOSS of 512 will be reduced to 256 with 50% reduction or 128 with 75% but it is still impossible to compute a 512 NOSS in a real-time in a sequential mode. Although 50% or 75% computation reduction are good results and the GCC1-DSB is easy to compute, some improvement could still be made to meet real-time constraint. This is the motivation for proposing GCC2-DSB.

Figure 4.12: GCC1-DSB Algorithms FOV Partition
4.6.2. GCC2-DSB

The principle of GCC2-DSB is similar to the GCC1-DSB with the advantage that it can find the region of interest inside the four regions \{C1, C2, C3, C4\} as shown in Figure 4.13. The region of interest is delimited by the rectangle \{L/2, \Delta h\} in the direction of angle \phi as shown by Figure 4.13. \Delta h could be computed as in equation 4.33.

\[
\Delta h = \frac{L}{2} \cdot \tan \phi
\]  

(4.33)

![Figure 4.13: GCC2-DSB Algorithms FOV Partition](image)

This approach improves the GCC1-DSB but both the GCC1-DSB and GCC2-DSB still suffer from the same deficiency. The GCC1-DSB and GCC2-DSB do not handle efficiently an angle of arrival of \(\phi = 90^\circ\). If \(\phi = 90^\circ\) all the four regions need to be computed. Another approach called GCC-DSB which solves the issue of \(\phi = 90^\circ\) will be presented to accelerate the DSB computation.

4.6.3. GCC-DSB

In this approach after the computation of the angle \(\phi\), a region is delimited around \(\phi\), to take into account the approximation error made when computing \(\phi\). That region is delimited by \(\phi + \varepsilon\) and \(\phi - \varepsilon\), where \(\varepsilon\) is user defined or can follow the results found in Figure 4.4. Thus, two lines with angles \(\phi - \varepsilon\) (\(h^-\)) and \(\phi + \varepsilon\) (\(h^+\)) are drawn from the FOV origin up to.
their intersection with the FOV border see Figure 4.8. The area inside the region delimited by the points O, X, Y, Z of Figure 4.14 becomes the region of interest. The sound source is represented by the red square inside the cone. The DSB is then computed only within that region. These regions of interest are computed in different manners depending on $\phi$. Figure 4.14 (a), (b), (c), (d) is an example of 4 different scenarios of SSL.
Figure 4.14: Reducing the Search Region for the GCC-DSB Algorithm
Both upper corners of Figure 4.14 (a), (b), (c), (d) referenced by angle $\delta_1$ and $\delta_2$ are key points for that computation. Those angles are computed as defined in equation 4.34:

$$
\delta_1 = \arctan\left(\frac{H}{L/2}\right) \quad \text{and} \quad \delta_2 = 180 - \arctan\left(\frac{H}{L/2}\right)
$$

(4.34)

The regions where the DSB needs to be computed changes according to the angle $\phi$. They are shown below:

- **If $(\phi - \epsilon \leq 0)$** then the region where we need to compute the DSB is defined by the coordinates:

  $$
  \{0,0\}; \left\{\left(\frac{L}{2}\right), 0\right\}; \left\{\left(\frac{L}{2}\right), \left(\frac{L}{2} \cdot \tan(\phi + \epsilon)\right)\right\}; \\
  h_n^+ = n\Delta x \tan(\phi + \epsilon) \\
  h_n^- = 0
  $$

  $$
  n \in [0, L/2]
  $$

  (4.35)

- **If $(\phi - \epsilon > 0$ and $\phi + \epsilon \leq \delta_1$)** then the region is:

  $$
  \{0,0\}; \left\{\left(\frac{L}{2}\right), \left(\frac{L}{2} \cdot \tan(\phi - \epsilon)\right)\right\}; \left\{\left(\frac{L}{2}\right), \left(\frac{L}{2} \cdot \tan(\phi + \epsilon)\right)\right\}; \\
  h_n^+ = n\Delta x \tan(\phi + \epsilon) \\
  h_n^- = n\Delta x \tan(\phi - \epsilon)
  $$

  $$
  n \in [0, L/2]
  $$

  (4.36)

- **If $(\phi - \epsilon < \delta_1$ and $\phi + \epsilon > \delta_1$)** then the region is:

  $$
  \{0,0\}; \left\{\left(\frac{L}{2}\right), \left(\frac{L}{2} \cdot \tan(\phi - \epsilon)\right)\right\}; \left\{\left(\frac{L}{2}\right), H\right\}; \left\{H \cdot \cotan(\phi + \epsilon), H\right\}
  $$

  $$
  h_n^+ = n\Delta x \tan(\phi + \epsilon) \\
  h_n^- = H \\
  h_n^+ = H \cdot \cotan(\phi + \epsilon) \\
  h_n^- = n\Delta x \tan(\phi - \epsilon)
  $$

  $$
  0 < n < H \cdot \cotan(\phi + \epsilon) \\
  H \cdot \cotan(\phi + \epsilon) < n < L/2
  $$

  (4.37)

- **If $(\phi - \epsilon > \delta_1$ and $\phi + \epsilon < 90^\circ$)** then the region is:

  $$
  \{0,0\}; \left\{H \cdot \cotan(\phi - \epsilon), H\right\}; \left\{H \cdot \cotan(\phi + \epsilon), H\right\}
  $$

  $$
  h_n^+ = n\Delta x \tan(\phi + \epsilon) \\
  h_n^- = n\Delta x \tan(\phi - \epsilon) \\
  h_n^+ = H \\
  h_n^- = H \cdot \cotan(\phi + \epsilon)
  $$

  $$
  0 < n < H \cdot \cotan(\phi + \epsilon) \\
  0 < n < H \cdot \cotan(\phi - \epsilon)
  $$

  (4.38)

- **If $(\delta_1 < \phi - \epsilon < 90^\circ$ and $90^\circ < \phi + \epsilon < \delta_2$)** then the region is:

  $$
  \{0,0\}; \left\{H \cdot \cotan(\phi - \epsilon), H\right\}; \left\{-H \cdot \cotan(\phi + \epsilon), H\right\}
  $$

83
\[ h^+_n = -n\Delta x \tan((180 - (\phi + \varepsilon))) \quad \text{for } -H \cdot \cotan(\phi + \varepsilon) < n < 0 \] \[ h^-_n = n\Delta x \tan(\phi - \varepsilon) \quad \text{for } 0 < n < H \cdot \cotan(\phi - \varepsilon) \] (4.39)

- If \((\phi - \varepsilon < \delta_2 \text{ and } \phi + \varepsilon > \delta_2)\) then the region is:
  \[ \{0,0\}; \{-H \cdot \cotan(\phi - \varepsilon), H\}; \left\{\left(-\frac{L}{2}\right), \left(-\frac{L}{2}\right) \cdot \tan(\phi + \varepsilon)\right\} \]
  \begin{align*}
  h^+_n &= -n\Delta x \tan(\phi + \varepsilon) \quad \text{for } -L/2 < n < 0 \\
  h^-_n &= H \quad \text{for } -L/2 < n < -H \cdot \cotan(\phi - \varepsilon) \\
  h^-_n &= -n\Delta x \tan(\phi - \varepsilon) \quad \text{for } -L/2 < n < -H \cdot \cotan(\phi - \varepsilon) \\
\end{align*} (4.40)

- If \((\phi - \varepsilon > \delta_2 \text{ and } \phi + \varepsilon < 180^\circ)\) then the region is:
  \[ \{0,0\}; \left\{\left(-\frac{L}{2}\right), \left(-\frac{L}{2}\right) \cdot \tan(\phi - \varepsilon)\right\}; \left\{\left(-\frac{L}{2}\right), \left(-\frac{L}{2}\right) \cdot \tan(\phi + \varepsilon)\right\} \]
  \begin{align*}
  h^+_n &= -n\Delta x \tan(\phi + \varepsilon) \quad \text{for } -L/2 < n < 0 \\
  h^-_n &= -n\Delta x \tan(\phi - \varepsilon) \quad \text{for } -L/2 < n < 0 \\
\end{align*} (4.41)

- If \(\phi - \varepsilon < 180^\circ \text{ and } (\phi + \varepsilon > 180^\circ)\) then the region is:
  \[ \{0,0\}; \left\{\left(-\frac{L}{2}\right), \left(-\frac{L}{2}\right) \cdot \tan(\phi - \varepsilon)\right\}; \left\{\left(-\frac{L}{2}\right), 0\right\} \]
  \begin{align*}
  h^+_n &= 0 \\
  h^-_n &= -n\Delta x \tan(\phi - \varepsilon) \quad \text{for } -L/2 < n < 0 \\
\end{align*} (4.42)

These regions were computed with the assumption that \(\varepsilon = 10^\circ\). The GCC-DSB achieves from 80\% up to 90\% reduction of the search spectrum of the DSB depending on the DOA of the sound source. Referring to Table 4.5 the throughput for a FOV with a NOSS = 256 would be 10\% of 90 ms so it could then be computed in real-time. However for a NOSS = 512, it would be necessary to use another hardware speed-up technique in conjunction with the hybrid algorithm to achieve the desired performance. This is discussed further in Chapters 6 and 7 (see Parallel sequential Mode).
4.6.4. GCC-DSB Computation Considerations

The main disadvantage of the GCC-DSB is its computation complexity which includes a succession of division, cotangent and tangent approximations using numbers with fractional part. Figure 4.15 shows that for every $\Delta x$ the number identifying every small square between the $h^+_i$ and $h^-_i$, $h^+_2$ and $h^-_2$ up to $h^+_n$ and $h^-_n$ will be stored in the memory and their SRP computed to find the SSL.

![Figure 4.15: Search Region Contraction of the FOV Internal Description](image)

The issue with the GCC-DSB is that while the region to be searched to find the SSL is contiguous in space it is not contiguous in terms of memory organisation. In this sense, it is unlike GCC1-DSB and GCC2-DSB in which the memory organisation is contiguous. For
those algorithms only the starting and ending addresses were needed to find the remaining points.

Figure 4.16 (a) shows the structure of the FOV memory that stores all the parameters of each small square where the sound source can potentially be after the GCC-DSB computation. Figure 4.16 (b) shows that in the GCC-DSB approach although the first index of the desired small square is located at the address \{0.1\} presented as \(\Delta y_1\), the next addresses in the FPGA can be located at the address \{9,10,11\} presented as \(\Delta y_2\). The GCC-DSB memory structure adds an extra lattency to the computation of the DSB. However, the extra latency does not really impact the computation of the DSB as all the points of the reduced FOV do not need to be found to start the DSB computation, the first point of interest will trigger the DSB computation. The search of the GCC-DSB reduced FOV can thus be done in parallel with the DSB computation. Thus it has been shown that intelligent design of the memory organization can overcome the potential disadvantage of the GCC_DSB algorithm.

Figure 4.16: FPGA Memory Structure
4.7. Results and GCC-DSB Architecture Proposal

4.7.1. GCC localization error and reduced NOSS approximation

The investigation carried out on the GCC-DSB shows that this approach reduced the DSB computation load in the worst scenario by 80% which make it better than the GCC1-DSB and GCC2-DSB therefore this work will focus on that approach. The GCC localization precision factor $\epsilon$ can be linked to the angle of arrival of the sound source $\phi$ as shown by the estimation presented in Figure 4.4. The error approximation $\epsilon$ decreases when the sound source is in the vicinity of 90° degree and increases elsewhere as shown in Figure 4.4. However for simplification purpose the GCC error $\epsilon$ can be fixed and linked to the highest error in Figure 4.4 although this is detrimental to angles close to 90° degrees.

The size of the reduced NOSS can be computed at run-time with the knowledge of the FOV dimension. Another way to approximate the reduced NOSS at run time, looking at Figure 4.14.d is to compute the triangle area formed by the points (0, X, Y) and divide it by the resolution as shown in equation (4.43).

$$\text{NOSS}_{\text{GCC-DSB}} = \frac{B \cdot H}{2 \cdot \Delta x \cdot \Delta y}$$  \hspace{1cm} (4.43)

Where H is the FOV height, B being the base of the triangle and equal to the distance between points (X,Y), $\Delta x$ and $\Delta y$ are the FOV resolution. Unfortunately equation (4.43) does not take into account the small squares of the FOV that are crossed by the lines drawn with angles $\phi - \epsilon (h^-_n)$ and $\phi + \epsilon (h^+_n)$. 
Another approach is to compute the reduce NOSS at compile time for every angle going from $0°$ to $180°$ degree for a given localization error $\varepsilon$. Figure 4.17 shows that when $\phi$ is around $65°$ degree with a chosen error $\varepsilon = 5°$, the highest possible NOSS that needs to be computed for the DSB is around 45 points. For a 256 points FOV, 45 points represent more than 80% reductions for the DSB computations. Moreover to remove from the hardware system the complexity of computing equation (4.34) to equation (4.42) all the points in Figure 4.17 can be stored in a Block Ram at compile time for each angle. Figure 4.17 also shows that the precision of the GCC greatly impacts the reduction of the DSB computation region.

### 4.7.2. Optimized GCC-DSB Architecture

The combined architecture proposed in this work uses a common acquisition chain for both the GCC and the DSB algorithm as shown in Figure 4.18. This approach reduces the number of CIC (Cascade Integrator-Comb) and FFT modules for the acquisition chain by 50%. A few other modules are similar such as the IFFT or the $\beta$-PHAT, and they will be explored in Chapter 6 to find different approaches to merge them and reduce the resource utilization.
4.8. Novel Interface Drivers to Combine Real-time Localization and Tracking Algorithms

Although research on algorithms for sound source localization and tracking using microphone arrays have been carried out for years, providing an interface to switch from one algorithm to the other is rather new [4.14]. This work provides some drivers that will allow the Delay Sum Beamforming (DSB) algorithm to understand and interpret the Generalized Cross Correlation (GCC) results to make the DSB computation faster. This work also shows that if there is knowledge of the sound source speed, a DSB-DSB algorithm could slightly improve the GCC-DSB throughput.

4.8.1. GCC-DSB Driver

Algorithm 4.1 shows a part of the driver used to convert the GCC angle to a DSB FOV. Depending on the FOV shape, angle $(\phi - \epsilon) > 90^\circ$ could be treated as being symmetric around the $90^\circ$ y-axis. Algorithm 4.1 below was coded with the main consideration being that the center of the FOV is in middle of the microphone arrays (see Figure 4.8). Thus, two
correction factors $X_{offset}$ and $Y_{offset}$ were required, with $X_{offset} = (1/2)(L/\Delta x)$ and $Y_{offset} = 2\times X_{offset}$. Results computed by this algorithm are stored in the FPGA’s BlockRAM. Ceiling and floor approximation are used because all the FOV points are integers. Division by $\Delta x$ and $\Delta y$ shows that a FOV with a unitary resolution or a power of 2 is a good choice to avoid costly operations. In the Algorithm, the constant $k$ defined as in equation (4.44) comes from equation 4.36 written as $h_n^+ = n\Delta x \tan(\phi + \epsilon) = n.k1$ and $h_n^- = n\Delta x \tan(\phi - \epsilon) = n.k2$

$$k = \frac{k1 - k2}{\Delta Y} = \frac{\Delta X. (\tan(\phi + \epsilon) - \tan(\phi - \epsilon))}{\Delta Y}. \quad (4.44)$$

The else clause of the Algorithm 4.1 is taken care of by the symmetry of 90° degree y-axis as mentioned above, so it has not been shown here although it was implemented in the application. This work has been published by the author in [4.14].
if \( \varphi < 90 \)
\[ \text{cnt}_\text{wr}_\text{blk} = 0; \]
\[
\text{for}(i = 0; i = i + \Delta x; i \leq \frac{L}{2\Delta x} - 1)
\]
\[ h_n^+ = (i + 1) \cdot k_1; \]
\[ \text{if} (h_n^+ > H) \]
\[ \max_j = \frac{H - \text{cei}(h_n^-)}{\Delta y}; \]
\[ \text{else} \]
\[ \max_j = (i + 1) \cdot k; \]
\[
\text{for}(j = 0; j = j + \Delta y; j < \text{cei}(\max_j))
\]
\[ \text{ptx} = i \Delta x; \]
\[ \text{pty} = j \Delta y + ik_2; \]
\[ nb = \text{ptx} + x_{\text{offset}} + \text{floor}(\frac{pty}{\Delta y}) \cdot y_{\text{offset}}; \]
\[ \text{cnt}_\text{wr}_\text{blk} = \text{cnt}_\text{wr}_\text{blk} + 1; \]
\[ \]
\[ \text{else} \]
\[
\text{for}(i = 0; i = i + \Delta x; i \leq \frac{L}{2\Delta x} - 1)
\]
\[ h_n^+ = (i + 1) \cdot k_1; \]
\[ \text{if} (h_n^+ > H) \]
\[ \max_j = \frac{H - \text{cei}(h_n^-)}{\Delta y}; \]
\[ \text{else} \]
\[ \max_j = (i + 1) \cdot k; \]
\[
\text{for}(j = 0; j = j + \Delta y; j < \text{cei}(\max_j))
\]
\[ \text{ptx} = i \Delta x; \]
\[ \text{pty} = j \Delta y + ik_2; \]
\[ nb = x_{\text{offset}} - \text{ptx} + \text{floor}(\frac{pty}{\Delta y}) \cdot y_{\text{offset}}; \]
\[ \text{cnt}_\text{wr}_\text{blk} = \text{cnt}_\text{wr}_\text{blk} + 1; \]

Algorithm 4.1 GCC Drivers

Figure 4.19 is the equivalent of Figure 4.18 with the drivers’ interface added to the block diagram. Figure 4.19 also shows that there is a possible parallelism between the \{GCC, GCC-Driver\} and the \{β-PHAT, IFFT\} computation. In Figure 4.19 Driver GCC corresponds to Algorithm 4.1 while the Driver DSB corresponds to Algorithm 4.2 below.
4.8.2. DSB-DSB Driver

The aim of the Driver-DSB in Figure 4.19 is to use the first DSB localization point found with the Hybrid algorithm approach to define a reduced region around it and track the sound source inside it (see colored region inside Figure 4.20). If the motion speed of the sound source is known in advance, it is then possible to couple in the same work the GCC-DSB and the DSB-DSB. The region covered by the sound source is then defined as in equation 4.45.

\[ X_{shift} = Y_{shift} = V \cdot T_{trpt} \] (4.45)

\( V \) is the speed of the sound source (which can be a human walking as he talks). \( T_{trpt} \) is the throughput, the time between two consecutive frames computation. \( X_{shift} \), \( Y_{shift} \) represents the distance covered by the sound source on y or x axis.
Figure 4.20: Different Motion of the Sound Source

In Figure 4.20 example, the number of points to compute to localize the sound source is defined as in equation 4.46. In this case $X_{\text{shift}} = Y_{\text{shift}} = 2$.

$$DSB_{\text{NOSS}} = \left(2 \cdot X_{\text{shift}} + 1\right) \ast \left(2 \cdot Y_{\text{shift}} + 1\right)$$  \hspace{1cm} (4.46)

The indices of each of the points in the colored restricted area surrounding the sound source depicted in Figure 4.20 as blue, black, green or orange are computed as in Algorithm 4.2.

VL-start represents the smallest index of the colored square and I-max and J-max represent a possible move of the sound source on the X and Y-axis. $y_{\text{up}}, y_{\text{down}}, y_{\text{left}}, y_{\text{right}}$ represent respectively the maximum distance the speaker can travel if he moves up, down or left, right in the restrictive colored FOV region shown in Figure 4.20. The rest of the parameters are similar to Algorithm 4.1. This work has also been published by the author in [4.14].
Algorithm 4.2 DSB-Driver computations

Algorithm 4.2 computes the new $DSB_{NOSS}$ using the colored square region in Figure 4.20. The new $DSB_{NOSS}$ (see Figure 4.20) is computed for every frame. From position (1 in pink) to position (2 in green), and from position (2 in green) to position (3 in blue) the sound source remains in the same square. The move from position (3 in blue) to position (4 in black) should never happen, as the speaker goes out of the region he was expected to be in, if this does happen the whole localization process needs to be recomputed. To detect if the sound source is still in the region where it is being tracked equation 4.47 could be used. This is suggested for investigation in future work.

$$VL_{start} = V L_{pt} - [X_{ml}(1 + 2 \cdot X_{offset}];$$

$$i_{max} = Y_{mup} + Y_{mdw};$$

$$j_{max} = X_{ml} + X_{mr};$$

$$\text{for}(i = 0, i \leq i_{max}, i++)$$

$$\text{for}(j = 0, j \leq j_{max}, j++)$$

$$F[j + i(i_{max} + 1)] = VL_{start} + (j + i \cdot 2 \cdot X_{offset})$$

$$\text{end}$$

$$\text{end}$$

Although multiple tracking shapes could have been used as shown by Figure 4.21, they should be selected with a pre-knowledge of the tracked object direction. For example, for a person walking on a straight street, a rectangular shape will be better, but for a teacher walking in a class-room a square tracking shape is more appropriate. When no details are given about the area in which the person will move, a circle area is better as the center is equidistant from all the borders. However, the least expensive region in term of computation (hardware logic utilization) are the rectangular and the square areas as they fit the FOV small square correctly.

$$FOV_{max} \geq 2^* \sum_{i=1}^{DSB_{noSS}} \frac{x_i}{n_{x_{fov}} n_{y_{fov}}} (4.47)$$
Table 4.6 shows the number of operations necessary for a GCC-DSB NOSS of 32 after a GCC computation returning a direction of arrival $\phi = 53.5^\circ$. The proposition of this work reduces the Block-RAM read access by 87%, the number of multiplication by 86.5%, the number of addition by 87%, the number of division by 87% and the number of Square root by 87%. Thus the approach developed in this work accelerates the computation of the traditional DSB by 87%.
Table 4.6: Computation LOAD of the GCC-DSB

<table>
<thead>
<tr>
<th>Localization</th>
<th>(N_{\text{ops}} = 256) (DSB)</th>
<th>(N_{\text{ops}} = 32) (DSB+GCC)</th>
<th>(N_{\text{ops}} = 25) (DSB+GCC)</th>
<th>(N_{\text{ops}} = 12) (DSB+GCC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLK_read</td>
<td>1575168</td>
<td>203940</td>
<td>159959</td>
<td>79980</td>
</tr>
<tr>
<td>MULT</td>
<td>2228480</td>
<td>296993</td>
<td>236058</td>
<td>122893</td>
</tr>
<tr>
<td>ADD</td>
<td>1572864</td>
<td>206853</td>
<td>163845</td>
<td>83973</td>
</tr>
<tr>
<td>DIV</td>
<td>1048576</td>
<td>137217</td>
<td>108545</td>
<td>55297</td>
</tr>
<tr>
<td>SQRT</td>
<td>524288</td>
<td>68008</td>
<td>54272</td>
<td>27648</td>
</tr>
</tbody>
</table>

Figure 4.22 shows that the highest number of computations in the DSB computation is multiplication and that access to Block-RAM is frequent. Figure 4.22 could be used as an indication of the FPGA to select for a possible implementation. Figure 4.22 also shows that not much is gained between the GCC-DSB and DSB-DSB in terms of throughput. But in term of logic resources the DSB-DSB is preferable.
4.9. Summary

This work has shown that the proposed hybrid algorithm using the GCC-DSB reduces the computation load of the DSB by more than 87 % and if the speaker can be tracked using the DSB-DSB approach then the computation reduction is 88% which can be seen by comparing the result of Table 4.6 NOSS 32 to Table 4.4 NOSS 32. The same result can be seen in Figure 4.22. The DSB-DSB algorithm is only computed after the first localization of the sound source therefore DSB-DSB complement the Novel Hybrid Algorithm proposed in this research. The DSB-DSB main advantage is that it uses fewer logic resources because the GCC logic is no longer necessary.
CHAPTER 5 Expanding the Developed Novel Hybrid Algorithm to Multiple Sound Source Localization and Beamforming Exploration

5. DECLARATION AND PRESENTATION

This chapter expands on the Hybrid algorithm developed in Chapter 4 and applies it to multiple sound source localization and further discusses beamforming techniques for microphone focussing. This work has been published in a paper entitled “Faster Sound Source Localization and Tracking with Limited Computation Load” [5.28], which is listed in the scientific output of this thesis presented in Chapter 1. The MATLAB and the Verilog code used throughout this chapter is provided in Appendix.

5.1. Purpose of this Chapter

The main purpose of this Chapter is to expand on the novel hybrid algorithm developed in this work to localize multiple sound sources simultaneously. The approach developed in this work will be compared to an existing algorithm called MUSIC (MUltiple SIgnal Classification). This Chapter will present a pre-computable parameters approach to reduce the DSB computation load using 9 points (Field Of View). Both the DSB (Delay Sum Beamforming) and MVDR (Minimum-Variance Distortionless Response) coupled to a Wiener Filter will be presented with their context of use. Finally a common architecture combining the novel hybrid algorithm and the beamforming (using adaptive and time invariant algorithms) will be presented before concluding the Chapter. Some ideas on how to reduce interference between speakers will also be proposed as further work at the end of this Chapter.
5.2. Introduction

Rapid advancement in adaptive beamforming applications such as (sonar and radar) algorithms has greatly increased the computation and communication demands on beamforming arrays, particularly for applications that require autonomous and real-time computations. Parallel processing for adaptive beamformers can significantly reduce execution time, cost and increase scalability and dependability [5.1]. Parallelism is well defined and Amdahl's law is referenced in [5.2] and multiple papers have defined power consumption control [5.3], [5.4], [5.5]. In this work the beamforming task is performed by the DSB (Delay Sum Beamforming) or MVDR (minimum-variance distortionless response) algorithms depending on the signal frequency. In addition, MVDR algorithm can be coupled to filters such as the Wiener Filter to improve its directivity. Localization and tracking of sound sources is done by the (GCC and DSB) as explained in Chapter 4. Nevertheless, a variant of the GCC algorithm based on the computation of energy in every direction of the FOV is used to account for the limitation of the Chapter 4 approach. Although the DSB localization is accurate, it does not achieve a maximization of the Signal-to-Noise Ratio (SNR) especially in low frequencies [5.6]; the DSB SNR can be improved by combining it to Dolph-Chebychev technique to attenuate the height of the side lobes [5.7].

The remainder of this Chapter will be divided as follows: Section 5.3 expands on work done in Chapter 4 by applying the hybrid algorithm to the localization of multiple sound sources and presents another localization algorithm called MUSIC (MUltiple SIgnal Classification) and compare it to the algorithm developed by this author. Section 5.4 presents the DSB and MVDR beamforming, their condition of utilization in this work and possible improvement of the MVDR algorithm using a Wiener Filter. Section 5.5 Presents the GCC, DSB and MVDR architecture, their computation load and combined architecture. Section 5.6 presents the parameters that can be computed at compile time to alleviate the DSB computation load (A 9 points FOV example is presented). Section 5.7 concludes this chapter, and as a further work proposes a solution to speaker interferences to improve localization.
5.3. Novel Hybrid Algorithm Applied to Multi-Source Localization

This section presents the limitation of the novel hybrid algorithm presented in Chapter 4 to track multiple sound sources and how it has been adapted to add that functionality. Figure 5.1 shows an example of a 16x16 FOV (Field Of View), where two sound sources are emitting simultaneously. This discussion will show how the novel hybrid algorithm was adapted to localize both sound sources and achieve real-time performance.

![Figure 5.1: 16x16 Small Square FOV with 4 Microphones and Two Speakers](image)

In real-time applications execution speed is an important concept, the algorithm that requires the least logic resources and achieves the highest throughput for a given task is preferred. The computation is achieved in real-time if a frame of 512 or 1024 samples is processed respectively at 11.6 or 23.16 ms for a sampling frequency of 44.1 KHz. Before describing localization algorithm a brief signals model and notation [5.8] is presented.

$(\cdot)^H$ denotes Hermitian transpose, $(\cdot)^T$ denotes the transposition and $(\cdot)^*$ denotes the complex conjugate.

Let $\{S_i(t)\}_{i=1}^L$ be the temporal waveforms of the sources, where $L$ is the number of sources. The assumption is made that the sources are independent and stationary over several adjacent $N_s$ samples intervals. This is expressed mathematically as: $E[S_i(n), S_j(n-l)] = 0$. The signal at the microphone $i$ is modeled as in equation (5.1).
Where $a_{pi}$ is the attenuation coefficients from the sound source $p$ to the microphone $i$, $t$ is the time index, $\tau_{pi}$ is the travel time from the sound source $p$ to the microphone $i$ and $n_i(t)$ is the noise sensed at the microphone $i$. Let us assume that $s_p(t)$ and $s_q(t)$ are the signals that come from the two sound sources presented in Figure 5.1 in blue and red. Then the signals received at the microphone $(i,j)$, $x_i(t)$ and $x_j(t)$ respectively, are written as in equation (5.2) and equation (5.3) if noise is ignored.

$$x_i(t) = a_{pi}s_p(t-\tau_{pi}) + b_{qi}s_q(t-\tau_{qi})$$  \hspace{1cm} (5.2)

$$x_j(t) = a_{pj}s_p(t-\tau_{pj}) + b_{qj}s_q(t-\tau_{qj})$$  \hspace{1cm} (5.3)

In a far field approximation the attenuation coefficient approximates to 1. To be precise the attenuation factor is defined as in equation (5.4), where $d_r$ is the distance between the source and the reference point, and $d_n$ is the distance between the source and nth sensor.

$$a_n = \frac{d_r}{d_n}$$  \hspace{1cm} (5.4)

The Cross-Correlation between the signals $x_i(t)$ and $x_j(t)$ at the microphone defined in equation (5.2) and (5.3) respectively is defined in the frequency domain by equation (5.5).

$$FFT\ (x_i(t)).FFT\ (x_j(t))^* = a_{pi}.a_{pj}.S_p(w)^2.e^{-jw(\tau_{pi}-\tau_{pj})} +$$

$$b_{qi}.b_{qj}.S_q(w)^2.e^{-jw(\tau_{qi}-\tau_{qj})} +$$

$$+ S_p.S_q(a_{pi}.b_{qj}.e^{-jw(\tau_{pi}-\tau_{qj})} + a_{pj}.b_{qi}.e^{-jw(\tau_{qi}-\tau_{pj})})$$  \hspace{1cm} (5.5)

The different angles of arrival ($\phi_1$ and $\phi_2$) of the sound source in Figure 5.1 are difficult to compute using the approach presented in Chapter 4 equation (4.20) [5.9]. For multiple sound source localization, this work used an approach called one step time delay direction of arrival (1TDOA) [5.10]. This is based on the estimation of the signal power coming from the direction of the sound source. This approach detects the direction of arrival of the sound source by computing the signal energy in every direction of the field of view (FOV). The highest energy represents the direction of arrival of the sound source. The FOV is then

102
discretized in V points selected in a way to provide sufficient spatial resolution $\varepsilon$ and the same distance to the center of the FOV by using points on a circle drawn from the center of the microphone arrays. The circle inside the FOV is then discretized in V points selected from $[0^\circ..180^\circ]$. The First point could be $\phi = 0^\circ$ and then the point S and so on up to $180^\circ$ as shown in Figure 5.2.

The sound source location is mathematically modeled as in equation (5.6).

$$S(V) = \arg_v \max(E(V))$$  \hspace{1cm} (5.6)

$E(V)$ is the estimated power coming from a single direction V from $[0..180]$ and is modeled as in equation (5.7).

$$E(V) = \left| \sum_{m=1}^{N} \frac{F_s}{2} \int X_m(f) \exp(-j2\pi \tau_m) df \right|^2$$  \hspace{1cm} (5.7)

Here $\tau_m$ is the estimated travel time from position s to microphone m. Expanding the terms of equation (5.7) leads to equation (5.8).

$$E(V) = \sum_{m=1}^{N} \left| \frac{F_s}{2} \int X_m(f) df \right|^2 + 2 \sum_{r=1}^{N} \sum_{r \neq s} \left| \frac{F_s}{2} \int X_r(f) X_s^*(f) \exp(j2\pi (\tau_r - \tau_s)) df \right|$$  \hspace{1cm} (5.8)
The first term is position independent and a constant for the current frame, while the second is a cross-correlation with a delay of \( (\tau_r - \tau_s) \). As explained in Chapter 4 it is important to add to the cross-correlation a weighting factor. Therefore the next logical step after removing the position independent part is to add the weightings \((W_{rs})\). Equation (5.8) is then modeled as in equation (5.9) below.

\[
E(V) = \sum_{r=1}^{N} \sum_{r \neq s} \int_{-F_s/2}^{F_s/2} W_{rs}(f) X_r(f) X_s^*(f) \exp(j2\pi f(\tau_r - \tau_s)) df
\]

Looking at equation (5.9) it represents the inverse Fourier transformation of the factor \(X_r X_s^*\) at one point that is the GCC function for specific delay \( (\tau_r - \tau_s) \). \(W_{rs}(f)\) is the weight of the GCC as described in Chapter 4 equation (4.13) as in equation (5.10).

\[
\phi_{PHAT}(\omega) = \frac{1}{\left| X_r^*(\omega) X_s(\omega) \right|^\beta}
\]

In the literature \(\beta = 0.8\) represents the best value to overcome reverberation [5.11]. The power 0.8 in equation (5.10) is very challenging to compute in hardware. The best approximate value in terms of computation is \(\beta = 0.75\) [5.12]. However a logarithmic computation approach can be useful. For instance if \(Y = \sqrt[q]{X^q}\), \(X\) can be rewritten using Napierian logarithm \(Y = e^p\) the same approach can be applied with binary base as modeled in equation (5.11). Where \(q = 3\), \(p = 4\) and \(b\) being the logarithmic base. The reason a binary was used is that in the FPGA, this computation becomes a right or a left shift.

\[
Y = b^p \ln_b X
\]

Although the main contribution of this research is to improve the DSB throughput reducing the number of computations will control power consumption. Figure 5.3 shows that creating sub-array microphone drastically reduces the number of IFFT computation when using the GCC approach. Figure 5.3 results can be reproduced using equation (4.2) of Chapter 4.
The purpose of Figure 5.4, Figure 5.5 and Figure 5.6 is to present the limitation of the novel hybrid algorithm presented in Chapter 4.

Figure 5.4 presents the limitation of the novel algorithm to detect multiple sound sources at close proximity of the microphone arrays. It shows the localization of two sound sources at different distances from the microphones. The closer the sources are to the microphones the more difficult it is to find their direction of arrival (DOA). When the sound sources are located at a distance from 5 cm to 25 cm of the microphone arrays, it is impossible to detect them as shown by data1 to data5 curve. The localization accuracy becomes reliable beyond 50 cm as shown by data6 and data7. The distance of the sound sources to the microphone arrays is not the only parameter that plays an important role when multiple speakers’ detection is involved.

Figure 5.4 also shows that the second sound source has a reduce height peak which might cause a detection issue. Another parameter is the angular distance necessary between both sources to avoid any masking of one source by the other. An angular distance of (25-35) degree is necessary meaning that the number of sources in the FOV should be limited to 4 for a FOV of 180° degree. More limitation are presented in Figure 5.5 and Figure 5.6. This work has been published by the author in [5.26].
The localization of two speakers simultaneously creates a dominant speaker, called primary source, which can mask totally or partially the secondary speaker due to the interference. This interference increases the error margin of the speaker’s localization.

Figure 5.5 shows the novel hybrid algorithm estimation error of the secondary and primary source location in green and in black respectively. This figure summarizes the result of a test, the purpose of which is to detect the highest possible error attached to the primary and the secondary speakers in the scenario where both speakers can be distinguished. The significance of this experiment is that it helps to determine the size of the cone around the direction of arrival (DOA) of the sound source as shown in Figure 5.1.

A random signal generator is used to produce signals whose phase varying from 175° to 95° degree with 5° degree steps for the primary source and the secondary source varying from 5° to 85° with the same steps. From those random signals 306 frames are chosen where both the primary speaker and the secondary speaker are detected. Figure 5.5 is a plot of individual frame numbers against the error angle, with frames ordered according to the size of the error associated with each one. Thus it may be seen that 282 frames have a localization
error ranging from $[1^\circ…3^\circ]$ and the remaining 24 frames have a localization error ranging from $[4^\circ…7^\circ]$ for the primary source in black.

For the secondary speaker only 83 frames out the 306 have an acceptable error ranging from $[5^\circ…9^\circ]$ degrees, while the remaining 223 frames or 73% of the frames have a localization error varying from $[14^\circ..18^\circ]$ degrees. The high detection error for the secondary speaker shows that the novel hybrid algorithm is better suited for systems detecting only one speaker. This work has been published by the author in [5.26].

Figure 5.5: Estimation Error of the DOA with two Sound Sources [5.26]

Figure 5.5 is drawn without taking into account the interference between the primary sound source and the secondary, meaning that the error is computed only on frames where both sound sources are detected. In real life, there are scenarios where the primary sources can completely mask the secondary speaker or cancel the entire signal due to the interferences between them. Figure 5.6 results represent the localization estimation error of the secondary speaker taking into account the interferences between speakers using three different approaches. If due to the interferences the secondary speaker is totally masked by the primary speaker the error assigned to that frame is $180^\circ$ degree as the speaker could have been located anywhere in the FOV whose range is from $0^\circ$ to $180^\circ$ degree. The algorithms used are the SRP GCC which scores poorly as the GCC is not well suited to multi-target localization [5.13], the Donohue temporal approach, which carries a small error because a fractional part is ignored during the data shift as integer value are required in that approach, and the frequency domain algorithm, which outperforms the two previously quoted algorithms. Table 5.1 below represents the connection between the number in the horizontal axis of Figure 5.6 and the angular position of the primary and secondary source. It is seen that Point
Number 1 represents the greatest separation between the two sources, with Point Number 17 representing the smallest separation. The speakers’ positions were varied as described in the experiment of Figure 5.5.

Figure 5.6: Comparison of Secondary Speaker Error using SRP GCC and Donohue Temporal and Frequency Mode [5.26]
Table 5.1: Points Reference to the sound source location of Figure 5.6 axis

<table>
<thead>
<tr>
<th>Points Number</th>
<th>Primary Source degree</th>
<th>Secondary Source degree</th>
<th>GCC Secondary Source Error (degree)</th>
<th>Donohue Secondary Source Error (Time)</th>
<th>Donohue Secondary Source Error (Frequency)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>175°</td>
<td>5°</td>
<td>103.8665°</td>
<td>104.2119°</td>
<td>102.3839°</td>
</tr>
<tr>
<td>2</td>
<td>170°</td>
<td>10°</td>
<td>102.3617°</td>
<td>102.2801°</td>
<td>102.0491°</td>
</tr>
<tr>
<td>3</td>
<td>165°</td>
<td>15°</td>
<td>101.8806°</td>
<td>101.6956°</td>
<td>102.6389°</td>
</tr>
<tr>
<td>4</td>
<td>160°</td>
<td>20°</td>
<td>102.8371°</td>
<td>102.4669°</td>
<td>101.9318°</td>
</tr>
<tr>
<td>5</td>
<td>155°</td>
<td>25°</td>
<td>104.6337°</td>
<td>103.0454°</td>
<td>101.7799°</td>
</tr>
<tr>
<td>6</td>
<td>150°</td>
<td>30°</td>
<td>107.1017°</td>
<td>103.4151°</td>
<td>102.6177°</td>
</tr>
<tr>
<td>7</td>
<td>145°</td>
<td>35°</td>
<td>107.8491°</td>
<td>106.0698°</td>
<td>103.5670°</td>
</tr>
<tr>
<td>8</td>
<td>140°</td>
<td>40°</td>
<td>110.0628°</td>
<td>108.9622°</td>
<td>106.2645°</td>
</tr>
<tr>
<td>9</td>
<td>135°</td>
<td>45°</td>
<td>112.0654°</td>
<td>110.2221°</td>
<td>104.1124°</td>
</tr>
<tr>
<td>10</td>
<td>130°</td>
<td>50°</td>
<td>115.5741°</td>
<td>110.8254°</td>
<td>99.5118°</td>
</tr>
<tr>
<td>11</td>
<td>125°</td>
<td>55°</td>
<td>116.4505°</td>
<td>114.2612°</td>
<td>95.0248°</td>
</tr>
<tr>
<td>12</td>
<td>120°</td>
<td>60°</td>
<td>118.7310°</td>
<td>114.9188°</td>
<td>95.7442°</td>
</tr>
<tr>
<td>13</td>
<td>115°</td>
<td>65°</td>
<td>120.0530°</td>
<td>116.2233°</td>
<td>102.8262°</td>
</tr>
<tr>
<td>14</td>
<td>110°</td>
<td>70°</td>
<td>122.7594°</td>
<td>118.9943°</td>
<td>111.3025°</td>
</tr>
<tr>
<td>15</td>
<td>105°</td>
<td>75°</td>
<td>130.1288°</td>
<td>123.8018°</td>
<td>119.2127°</td>
</tr>
<tr>
<td>16</td>
<td>100°</td>
<td>80°</td>
<td>154.3841°</td>
<td>150.8377°</td>
<td>146.6654°</td>
</tr>
<tr>
<td>17</td>
<td>95°</td>
<td>85°</td>
<td>180.0000°</td>
<td>180.0000°</td>
<td>180.0000°</td>
</tr>
</tbody>
</table>

Figure 5.5 and Figure 5.6 shows that the novel hybrid algorithm developed in Chapter 4 is not suited for multiple speakers’ detection therefore another algorithm, called MUSIC (MUltiple SIgnal Classification), known to be able to handle multiple sources will be evaluated to see if it can improve the localization of multiple speakers.

5.3.1. Evaluation of the MUSIC Algorithm for Multiple Sound Sources Location
The MUSIC (MUltiple SIgnal Classification) algorithm is a well-known high-resolution method for source localization. However, there are two issues regarding the MUSIC algorithm, which constrain its application for sound localization in practice. One is the heavy computational cost, while the other is the need for previous knowledge about the actual number of sources present in the input signal [5.14]. When the number of sound sources is known, MUSIC has better performance than all the others algorithms previously presented with an approximate 7-degree angular resolution [5.15]. Figure 5.7 presents the computation flow chart of the MUSIC algorithm to detect the direction of arrival of the sound sources.

![Flow Chart of the Computation of the Sound Source DOA using MUSIC](image)

**Figure 5.7: Flow Chart of the Computation of the Sound Source DOA using MUSIC [5.14]**

Sound sources captured via a filter and the FFT are described in Chapter 6. After the FFT the Correlation Matrix of the vector signals $X(k) = [X_1(k), ..., X_N(k)]^T$ of each microphones, with Ns samples is carried out using equation (5.12).

$$R_{xx} = E \left\{ x(t) x^H(t) \right\} = \frac{1}{N_s} \sum_{k=1}^{N_s} X_k X_k^H$$  \hspace{1cm} (5.12)
Where \( E\{\} \) denotes the statistical expectation and \( X^H \) denotes the Hermitian (Complex-conjugate) transpose of \( X \). The correlation matrix is then defined as in equation (5.13)

\[
R = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{N} R_{ij}
\]

(5.13)

The steering position vector is defined as \( a(\delta_i) \) with \( \delta_i(\phi) \) being a function of the DOA \( \phi \). The angle \( \phi \) is the scanning angle over the region of interest for the N microphones. \( \delta_i(\phi) \) is defined as equation (5.14) with \( i \in [1..N] \). The equation (5.14) has been explained in Chapter 3.

\[
\delta_i(\phi) = \frac{-2\pi f(d_1 - d_0) \cos \phi}{C}
\]

(5.14)

The steering position vector is then defined as in equation (5.15)

\[
A = [a(\delta_1), a(\delta_2), ..., a(\delta_N)]
\]

(5.15)

The steering vector matrix depends on the position of the reference microphone and the number of speakers. It is defined as in equation (5.16).

\[
A = \begin{bmatrix}
1 & 1 & \ldots & 1 \\
e^{j\delta_1} & e^{j\delta_2} & \ldots & e^{j\delta_L} \\
\vdots & \vdots & \ddots & \vdots \\
e^{j(N-1)\delta_1} & e^{j(N-1)\delta_2} & \ldots & e^{j(N-1)\delta_L}
\end{bmatrix}
\]

(5.16)

The Eigen decomposition of \( R_{xx} \) is computed using equation (5.17).

\[
\det [R_{xx} - \lambda_i I_N] = 0
\]

(5.17)

Where \( I_N \) is the identity matrix and \( \{ \lambda_1, \ldots, \lambda_N \} \) are the eigenvalues of \( R_{xx} \). This means that \( N\)-\( L \) of the eigenvalues of \( R_{xx} \) are equal to the noise variance \( \sigma^2_N \) that is mathematically modeled in equation [5.15] (5.18):

\[
\lambda_{L+1} = \lambda_{L+2} = \ldots = \lambda_N = \lambda_{\min} = \sigma^2_N
\]

(5.18)

In practice, however when the autocorrelation matrix \( R_{xx} \) is estimated from the finite data sample set of the received signals, all the eigenvectors corresponding to the noise space will not be exactly identical [5.15] [5.16]. Therefore techniques such as AIC (Akaike’s
Information Criterion) and MDL (Minimum Description Length Criterion) are used to determine the number of sound sources and the number eigenvalues related to the noise. In the MDL-based approach, the number of signals \( l \) is obtained as the value of \( l \in \{0, 1, \ldots, N-l\} \), which minimizes the following criterion in equation (5.19):

\[
L(l) = -\log \left( \prod_{n=l+1}^{N} \lambda_n^{1/(N-l)} \right) + \frac{1}{2} \log (2N-l) \log N
\]  

Similar to MDL, a technique called the Akaike Information Theoretic Criterion (AIC) was proposed for estimation of the number of signals or model order. The number of signals \( l \) is determined as the argument that minimizes the criterion in equation (5.20).

\[
L(l) = -\log \left( \prod_{n=l+1}^{N} \lambda_n^{1/(N-l)} \right) + \frac{1}{2} (N-1) \log N
\]  

The criteria presented in equations (5.19) and (5.20) are very complex to compute in real-time. To get around this difficulty an alternative approach is proposed as a further work by the author later in this Chapter. The eigenvector associated with a particular eigenvalue \( \lambda_i \), denoted as \( q_i \), satisfies equation (5.21).

\[
R_{xx} - \lambda_i I_N | q_i = 0 \quad i = l+1, l+2, \ldots, N
\]  

The eigenvectors associated with the \( N-l \) smallest eigenvalues are orthogonal to the \( l \) steering vectors that make up the vector \( A \) as shown in equation (5.22).

\[
\{a(\delta), \ldots, a(\delta_i), \ldots, a(\delta_l)\} \perp \{q_{l+1}, \ldots, q_N\}
\]  

Therefore the narrow band MUSIC response is computed as in equation (5.23). More details on the computation of the MUSIC algorithm are provided in \[5.14\][5.15].

\[
P_{\text{MUSIC}}(\phi) = \frac{a^H a}{a^H Q Q^H a(\phi)}
\]  

The DOA of sound sources using the MUSIC Algorithm is then computed on all the signal spectrum and it is modeled as in equation (5.24).

\[
P_{\text{WB MUSIC}}(\phi) = \frac{1}{N_s} \sum_{k=1}^{N_s} P_{\text{MUSIC}}(\phi)
\]
Figure 5.8 compared to this work in Figure 5.4 shows that the MUSIC algorithm is more directive, meaning that there are less interferences between sound sources. This signifies that more sound sources can be localized in the FOV compared to this author's results. However, the MUSIC Algorithm computation is more complex and has a higher throughput. An estimation of the MUSIC throughput has been published during this research by the author in [5.27].

![Figure 5.8: Speakers DOA Detection using MUSIC Algorithm](image)

### 5.4. DSB AND MVDR BEAMFORMING

The process of computing the output signal as a linear combination of the signals from the microphones of the array, equivalent to the output of highly directional microphones, is called beamforming. The name comes from the beam-like shape of the directivity pattern [5.17]. Let’s suppose we have an array consisting of N microphones and denote the microphone outputs as $x_n(k)$ (n= 1,2,…,N), beamforming is achieved based on manipulating the signals $x_n(k)$. Many algorithms have been developed over the three decades [5.18]. The simplest one is delay-and-sum beamformer (DSB) as shown in Figure 5.9, which was originally investigated in the underwater acoustics and radar antenna areas. The basic idea is
to delay (or advance) each microphone output by a proper amount of time $Z^{-\tau_N}$ with $\tau_N$ defined as equation (4.1) in Chapter 4. Therefore the signal components from the desired source are synchronized across all sensors. These delayed (or advanced) signals are then weighted and summed together. The weight can be computed as in equation (4.25) or are given the same value across the field of view (FOV).

![Figure 5.9: Structure of a Delay-and-Sum Beamformer [5.19]](image)

In the literature the weights are denoted by the letter W or G and one or the other will be used in this discussion. When the signals are added up together coherently, the desired signal components are reinforced. In contrast, the others sources and noises are suppressed or even eliminated as they are added together destructively [5.19].

The weighting coefficient $g_n$ can be either fixed or adaptively determined. The advantage of the adaptive beamforming over the non-adaptive beamforming can roughly be interpreted as their weight $g_n$ being better adaptively designed to take into account the signal and noise characteristic. Their weights are designed and placed so that noise and interference can be better rejected.

The DSB beamforming technique was developed for narrowband signals. The directivity pattern of the Delay-and-Sum Beamformer is not the same across a wide frequency band. If the DSB beamformer is used, then the noise and interference signals coming from a different direction than the direction in which the beamformer is looking will not be uniformly attenuated over its entire spectrum [5.19]. One way to overcome this problem is to
use harmonically nested sub-arrays. In this case, every sub-array is designed to operate at a single frequency. However, such solution requires a large array with a great number of microphones. Another way to circumvent this problem is to perform narrowband decomposition and to design narrowband beamformers independently at each frequency, as shown in Figure 5.10.

![Figure 5.10: Structure of a Frequency-Domain Broadband Beamformer by Narrowband Decomposition [5.19]](image)

### 5.4.1. MVDR Weight Computation Using the GCC or DSB Computation Results

A general description of a beamformer is modeled as in equation (5.25)

\[
Y(n) = \sum_{i=0}^{N-1} x_i(n) * w_i(n)
\]  

(5.25)

Where \( N \) is the number of microphones, \( x_i \) the input signal at the sensor \( i \), and \((*)\) denotes the convolution with an arbitrary constant design filter \( w_i \), which includes the steering delays in non-adaptive beamformer. In the case of an adaptive beamformer \( w_i \) varies with the sound source direction of arrival (DOA) angle and frequency. Different optimization approaches can be used. One which is commonly used, allows the sound source coming from the DOA in the absence of noise to stay unchanged. That optimization approach is called MVDR. Equation (5.25) can also be computed in the frequency domain as in equation (5.26).
\[ Y(n) = \sum_{i=0}^{N-1} X_i(f) \cdot W_i(f) \] (5.26)

Additionally, the steering is much easier in the frequency domain, as the fractional delay is only a multiplication with linear phase term [5.20]. The weight \( W_i(f) \) is computed as in equation (5.27) as presented in [5.21].

\[ W_{MVDR}^H = \frac{A^H \Phi_{vv}^{-1}}{A^H \Phi_{vv}^{-1} A} \] (5.27)

Where \( \Phi_{vv} \) is a NxN noise cross spectral density matrix between microphones, \( A \) (also name d in the literature) is the propagation vector of the desired speech signal for a linear sensor array [5.22]. A bin can be thought of as a narrow band pass filter and the center frequency of each FFT bin frequency is equal to the bin number multiplied by the sampling rate divided by the FFT size. The bin amplitude reflects the power around the bin frequency as shown in equation (5.28). The parameter vector, \( A \) defined in equations (5.14) and (5.15) also varies with the frequency bin amplitude, computed in equation (5.28).

\[ f = \frac{k f_s}{N_s} \] (5.28)

The FFT bin number is denoted by \( k \), the sample frequency by \( f_s \) and \( N_s \) is the number of samples in the frame. A coefficient \( \alpha_n \) represents the amplitude attenuation factor between the \( n^{th} \) sensor and the reference sensor computed as in equation (5.29) could also be added to every elements of the propagation vector \( A \) to account for the signal attenuation.

\[ \alpha_n = \frac{d_r}{d_n} \] (5.29)

In this discussion, \( \alpha_n \) is approximated to 1. \( d_r \) is the distance between the sound source and the reference point and \( d_n \) is the distance between the sound source and the \( n^{th} \) sensor. In a far field approximation equation (5.29) is equal to 1 but in the near field the approximation has to be computed. Another very important parameter in the MVDR weight is the microphone arrays gain in the direction of the sound source arrival [5.21]. It is defined as in equation (5.30). Equation (5.30) can be compared to signal-to-noise ratio (SNR) in the sound source direction of arrival.
\[ G_{MVDR} = \frac{|W_{MVDR}^H A|}{W_{MVDR}^H \Phi_{vv} W_{MVDR}} \]  

(5.30)

\[ \Phi_{vv} \text{ represents the noise cross spectral matrix and } \Phi_{xx} \text{ is the noisy signal cross spectral density matrix between microphones. These are defined in equation (5.31).} \]

\[
\Phi_{vv} = \begin{pmatrix}
\phi_{v_0v_0} & \phi_{v_0v_1} & \cdots & \phi_{v_0v_{N-1}} \\
\phi_{v_1v_0} & \phi_{v_1v_1} & \cdots & \phi_{v_1v_{N-1}} \\
\vdots & \vdots & \ddots & \vdots \\
\phi_{v_{N-1}v_0} & \phi_{v_{N-1}v_1} & \cdots & \phi_{v_{N-1}v_{N-1}}
\end{pmatrix} \quad \Phi_{xx} = \begin{pmatrix}
\phi_{x_0x_0} & \phi_{x_0x_1} & \cdots & \phi_{x_0x_{N-1}} \\
\phi_{x_1x_0} & \phi_{x_1x_1} & \cdots & \phi_{x_1x_{N-1}} \\
\vdots & \vdots & \ddots & \vdots \\
\phi_{x_{N-1}x_0} & \phi_{x_{N-1}x_1} & \cdots & \phi_{x_{N-1}x_{N-1}}
\end{pmatrix}
\]  

(5.31)

It always more convenient to work with the spectral coherence matrix \( \Gamma_{xx} \) and \( \Gamma_{vv} \) defined in equation (5.32) rather than the spectral density matrix \( \Phi_{vv} \) and \( \Phi_{xx} \).

\[
\Gamma_{vv} = \begin{pmatrix}
\gamma_{v_0v_0} & \gamma_{v_0v_1} & \cdots & \gamma_{v_0v_{N-1}} \\
\gamma_{v_1v_0} & 1 & \cdots & \gamma_{v_1v_{N-1}} \\
\vdots & \vdots & \ddots & \vdots \\
\gamma_{v_{N-1}v_0} & \gamma_{v_{N-1}v_1} & \cdots & 1
\end{pmatrix} \quad \Gamma_{xx} = \begin{pmatrix}
1 & \gamma_{x_0x_1} & \cdots & \gamma_{x_0x_{N-1}} \\
\gamma_{x_1x_0} & 1 & \cdots & \gamma_{x_1x_{N-1}} \\
\vdots & \vdots & \ddots & \vdots \\
\gamma_{x_{N-1}x_0} & \gamma_{x_{N-1}x_1} & \cdots & 1
\end{pmatrix}
\]  

(5.32)

The complex coherence function plays a key role in the design of good beam-patterns. The complex coherence function is defined as the normalized cross-power spectral density of two sensors (microphones) [5.23]. It is modeled as in equation (5.33).

\[ \gamma_{j_i} = \frac{\phi_{j_i} (f)}{\sqrt{\phi_{j_i} (f) \phi_{j_i} (f)}} \]  

(5.33)

The assumption that the noise is stationary, spatially homogenous with a spectral density power \( \phi_{vv} \) leads to the fact that spectral density power between two microphones \( \phi_{v_i} \) can be expressed with the spectral coherent function \( \gamma_{j_i} \) as: \( \phi_{v_i} = \phi_{vv} \gamma_{v_i} v_j \). In that condition the MVDR defined as in equation (5.27) can be computed as in equation (5.34):

\[ W_{MVDR}^H = \frac{A^H \Gamma_{vv}^{-1}}{A^H \Gamma_{vv}^{-1} A} \]  

(5.34)
In a diffuse or spherically isotropic noise field, noise of equal energy propagates in all directions simultaneously. The sensors of a microphone arrays will receive noise signals that are mainly correlated at low frequencies but have approximately the same energy. Diffuse energy fields can serve as a model for many applications concerning noisy environments, for example cars and offices environments [5.21]. The complex coherence function for such a noise field can be approximated as in equation (5.35).

\[
\gamma_{ij}(f) = \frac{\sin(2\pi f d_{ij} / c)}{2\pi f d_{ij} / c}
\]

\(d_{ij}\) represents the distance between two consecutive microphones. All the remaining parameters used in equation (5.35) were defined previously. Equation (5.35) increases the noise in low frequencies. Therefore the uncorrelated noise variance of the sensors \(\sigma_n^2\) can be included in the coherence function along with the diffuse noise spectral density power \(\phi_{nn}(f)\). Equation (5.36) should then be used instead of equation (5.35).

\[
\gamma_{ij}(f) = \frac{\sin(2\pi f d_{ij} / c)}{2\pi f d_{ij} / c(1 + \sigma_n^2 / \phi_{nn}(f))}
\]

From experimental results, the coefficient \(\sigma_n^2 / \phi_{nn}(f)\) is always approximated to a value that varies in the range of -20dB...-40dB [5.22].

Using the DSB result to compute the MVDR weight \(W\) is more accurate as it takes into account the real distance between the sound source and the reference point and not the sound source DOA. The propagation vector \(A\) is computed slightly differently. Its parameter \(\delta_i(\phi)\) presented in equation (5.14) is modeled slightly differently. Its parameter \(\delta_i(\phi)\) presented in equation (5.14) is modeled as in equation (5.37).

\[
\delta_i(\phi) = \frac{2\pi f (r_i - r)}{c}
\]

Where parameters \(r_i\) and \(r\) determine respectively the distance between the sound source and the microphone \(i\) and the distance between sound source to the central position of the microphone arrays [5.20]. \(r_i\) is modeled as in equation (5.38). In this research, both \(r\) and \(r_i\) are computed during the DSB pre-computation presented later in this Chapter.

\[
r_i = \sqrt{r^2 + 2rd_{im}\cos(\phi) + (d_{im})^2}
\]
\( d_{im} \) is the distance between the microphone \( i \) and the middle of the microphone arrays or the reference point and \( \phi \) the incident angle. Using the DSB approach described above, extra care needs to be taken into account before computing the inverse Fourier transformation. Assuming that equation (5.39) represents the audio signal coefficients after the discrete Fourier transform (DFT).

\[
X(k) = a_k + jb_k = A_k e^{j\alpha_k} = A_k (\cos \alpha_k + j \sin \alpha_k)
\] (5.39)

Where \( A_k \) is the amplitude and \( \alpha_k \) the phase of the signal in bin \( k \). The shifting of the coefficient using DSB angle detection is modeled as in equation (5.40):

\[
X\phi(k) = c_k + jd_k = A_k e^{j(\alpha_k + \phi)}
\] (5.40)

\( \phi \) is the phase shift due to the signal DOA angle on every microphone. Therefore the real and the imaginary part \{ \( c_k, d_k \) \} of the audio signal after the MVDR shift phase is modeled as in equation (5.41) and (5.42) for the real and imaginary part:

Real part

\[
c_k = A_k \cos(\alpha_k - \phi_k) = a_k \cos \phi_k + b_k \sin \phi_k
\] (5.41)

Imaginary part

\[
d_k = A_k \sin(\alpha_k - \phi_k) = b_k \cos \phi_k - a_k \sin \phi_k
\] (5.42)

Although the beamforming could be computed solely with the DSB algorithm, looking at Figure 5.11 and Figure 5.12 it may be seen that the beamforming computed with the MVDR is more directive especially for low frequencies. The red band represents the listening region, meaning that at low frequencies the directivity totally disappears for the DSB (see Figure 5.11). The MVDR (see Figure 5.12) has a better directivity compared to the DSB in Figure 5.11, moreover the MVDR can be coupled to a filter to improve its performance.
Figure 5.11 shows that if another sound source or noise is emitted in low frequency the DSB beamforming will not be able to differentiate from the sound source located at 90° and noise coming from anywhere else. Figure 5.11 DSB and Figure 5.12 MVDR also show that in high frequency around 6000 and 7000 Hz any sound source located at 20 to 40° or 140° to 160° will be heard along with the sound source located at 90°. Finally, Figure 5.11 and Figure 5.12 show that in high frequency DSB and MVDR behave similarly.
The red line between 6000 Hz and 7000 Hz is also known in the literature as a ‘grating lobe’. Grating lobes are usually unwanted as they cause the array to pick up signals from directions other than that of the main lobe without attenuation. Grating lobes occur when the extra distance a signal wave front must travel between array elements is a multiple of the signal’s wavelength. In this situation, the signals received by each element are highly correlated and no signal attenuation occurs [5.23].

### 5.4.2. MVDR and DSB Computation Selection and Wiener Filter

As explained above at high frequencies the DSB performs similarly to the MVDR therefore at high frequencies the DSB is preferred since it reduces the computation load as shown in Table 5.1 and at low frequencies the MVDR is preferred as it is more directive (see Figure 5.12). Therefore the signal frequency band needs to be determined to select which of the DSB or MVDR beamforming to use. To detect the signal frequency equation (5.43) or (5.44) is computed. The frequency band [0.3..1.5] kHz is considered low frequency and [1.6..to higher] kHz high Frequency.

\[
X_{BE}(i) = \frac{\sum_{l \in S_l} |X(l)|^2}{\sum_{k=1}^{K} |X(k)|^2}
\]

(5.43)

\(S_l\) denotes the low frequency band and the denominator is the signal's entire spectrum. Although equation (5.43) is a good approach, the computation of division in hardware is costly. Therefore equation (5.44) is preferred where \(p\) is the number of low frequency bins and \(TH\) is a fixed constant that defines the magnitude of the presence of signal in a certain frequency region compared to the overall signal spectrum. In this research it is set to 0.5.

\[
X_{SR} = \arg \max_p \left( \sum_{l=1}^p |X(l)|^2 \leq TH \cdot \sum_{k=1}^{K} |X(k)|^2 \right)
\]

(5.44)

Table 5.2 compares the DSB computation load to the MVDR. It shows that the MVDR uses a lot more resources for the result presented in Figure 5.12. The main handicap of the MVDR approach is the number of divisions as hardware description languages (HDL) today still have limitations when computing division.
Table 5.2: MVDR and DSB BEAMFORMING SERIAL THROUGHPUT for N = 8 and NS = 512

<table>
<thead>
<tr>
<th>OPERATION</th>
<th>DSB LOAD</th>
<th>BURDEN</th>
<th>MVDR LOAD</th>
<th>BURDEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULT</td>
<td>N*Ns</td>
<td>4 096</td>
<td>(2N+1)*N*Ns</td>
<td>69 632</td>
</tr>
<tr>
<td>ADD</td>
<td>Ns*(N-1)</td>
<td>3 584</td>
<td>Ns*(N-1)*(1+2N)</td>
<td>60 928</td>
</tr>
<tr>
<td>DIV</td>
<td>0</td>
<td>0</td>
<td>Ns*N</td>
<td>4 096</td>
</tr>
<tr>
<td>BLK-READ</td>
<td>N*(1+Ns)</td>
<td>4 104</td>
<td>N*Ns*(N+4)</td>
<td>49 152</td>
</tr>
</tbody>
</table>

The beamforming directivity is more accurate when the amplitude attenuation as well as the phase’s differences is taken into account for its computation. While the phase differences are negligible at low frequencies, the amplitude differences are significant, particularly when the sensors are placed in a linear configuration as this maximizes the difference in the distance from the source to each microphone [5.24].

In practice, the basic filter beamformer seldom exhibits the level of improvement that the theory promises and further enhancement is desirable. One method to improve the system performance is to add a post-filter to the output of the beamformer. Zelinski proposed a Wiener post-filter using the cross-spectral densities between channels of the microphone arrays. Incorporating a post filter with a beamformer allows the use of knowledge obtained in spatial filtering. In using both spatial and frequency domain enhancement, the use of information about the signal is maximized [5.24]. The Wiener transfer function is modeled as in equation (5.45).

\[
H = \frac{\phi_s}{\phi_s + \phi_n} \tag{5.45}
\]

Using a post filter was suggested by Lefkimmiatis, Bitzer and McCowan [5.21, 5.22, 5.24] to improve the global performance of the beamformer. Simmer and al express the optimal band pass Minimum Mean Squared Error (MMSE) filter solution as a classical Minimum Variance Distortionless Response (MVDR) beamformer followed by a single-channel Wiener filter that is modeled as in equation (5.46).

\[
W_{MMSE}^{H} = \frac{d^H \Phi_{vv}^{-1} \left( \phi_s \right)}{d^H \Phi_{vv}^{-1} d \left( \phi_s + \phi_n \right)} \tag{5.46}
\]

Where \( W_{MMSE}^{H} \) is the optimal filter coefficient vector, \( \phi_{ss} \) and \( \phi_{nn} \) are respectively the (single-channel) target signal and noise (after the MVDR noise filtering) auto-power spectrum vectors, and \( \Phi_{vv} \) is the (multichannel) noise cross-spectral density matrix. Solving equation
(5.46) requires computing all the different parameters in it. The power spectrum of the noise at the output of the beamformer $\phi_{nn}$ is modeled in [5.21] as in equation (5.47).

$$\phi_{nn} = W^H_{MVDR} \Phi_{vv} W_{MVDR} = \left( A^H \cdot \Phi^{-1}_{vv} \cdot A \right)^{-1} \tag{5.47}$$

In the assumption of stationary noise the spectral density matrix $\Phi_{vv}$ can be expressed with the coherence matrix $\Gamma_{vv}$ and the spectral power $\varphi_{vv}$, it is modeled as in equation (5.48).

$$\phi_{nn} = \left( \begin{array}{c} \varphi_{vv} \\ A^H \cdot \Gamma_{vv}^{-1} \cdot A \end{array} \right) \tag{5.48}$$

Signals received at the microphone after the compensation of the propagation phase shift and relative amplitude attenuation are modeled as in equation (5.49).

$$X_j(f) = S(f) + V(f) \tag{5.49}$$

$X_j(f)$ is the noisy signal, $S(f)$ is the sound source signal and $V(f)$ is the noise. Under the assumption that the noise and the signal are uncorrelated meaning that $(\phi_{x,y} = 0, \forall i, j)$. The spectral density can be modeled as in equation (5.50).

$$\begin{align*}
\phi_{x,x} &= \phi_x + \phi_{y,y} \\
\phi_{x,y} &= \phi_x + \phi_{y,y} \\
\phi_{y,x} &= \phi_x + \phi_{y,y}
\end{align*} \tag{5.50}$$

Using the relation $\phi_{y,y} = \varphi_{vv} \varphi_{y,y}$ and making the assumption that noise power spectrum is the same on all sensors which is translated by $(\varphi_{y,y} = \varphi_v, \forall i)$ The equation (5.50) becomes equation (5.51) below.

$$\begin{align*}
\phi_{x,x} &= \phi_x + \varphi_v \\
\phi_{x,y} &= \phi_x + \varphi_v \\
\phi_{y,x} &= \phi_x + \varphi_v \\
\phi_{y,y} &= \phi_y + \varphi_v \\
\phi_{x,y} &= \phi_x + \varphi_v
\end{align*} \tag{5.51}$$

Resolving equation (5.51) gives the result modeled in equation (5.52).

$$\hat{\phi}_{ij} = \max \left\{ \frac{R(\phi_{x,y}) \left| - \frac{1}{2} R(\varphi_{y,y})(\phi_{x,y} + \phi_{y,y}) \right|}{1 - R(\varphi_{y,y})}, 0 \right\} \tag{5.52}$$
\( \hat{\phi}_{ss}^{ij} \) is an estimation of \( \phi_{ss} \) using the microphones i and j. R in equation (5.52) means that only the positive real value of the estimation is considered, the imaginary part is considered to be zero. For N microphones there will be \( C_N^2 \) ways to estimate \( \phi_{ss} \). Taking the average improves the system robustness at the cost of the power required for computation and resource utilization. This approach is modeled as in equation (5.53).

\[
\hat{\phi}_{ss} = \frac{2}{N(N-1)} \sum_{i=0}^{N-2} \sum_{j=j+1}^{N-1} \hat{\phi}_{ss}^{ij}
\]

Using equation (5.51) an estimation of the noise spectral density \( \hat{\phi}_{vv}^{ij} \) is computed and modeled as in equation (5.54).

\[
\hat{\phi}_{vv}^{ij} = \max \left\{ \frac{1}{2} R(\phi_{x_i,x_j} + \phi_{x_j,x_i}) - R[\phi_{x_i,x_j}] \right\} \quad (5.54)
\]

The noise average value is then described as in equation (5.55).

\[
\hat{\phi}_{vv} = \frac{2}{N(N-1)} \sum_{i=0}^{N-2} \sum_{j=j+1}^{N-1} \hat{\phi}_{vv}^{ij}
\]

\( \phi_{nn} \) is finally modeled according to [5.11] as in equation (5.56)

\[
\phi_{nn} = \begin{cases} 
\phi_{vv} & f \leq f_1 \\
\phi_{nn} & f \geq f_1 
\end{cases}
\]

\( f_1 \) is the frequency limit which determines whether the computation of noise spectral density will be computed using equation (5.52) or equation (5.48).

Therefore the Wiener Filter parameters can be computed with the estimation found above and modeled as in equation (5.57)

\[
h(f) = \left( \frac{\hat{\phi}_{ss}(f)}{\hat{\phi}_{ss}(f) + \phi_{nn}(f)} \right)
\]

The computation of Wiener Filter does not require more computation compared to GCC algorithm in Chapter 4. Wiener Filters should mainly be used in conjunction with the MVDR beamforming for low frequency signals.
5.5. Combining the GCC-DSB and MVDR Architectures

The block diagram of the system developed in this work is shown in Figure 5.13. The MVDR beamforming, acquisition chain presented in Figure 5.13 is similar to the GCC-DSB architecture shown in Figure 4.7 in Chapter 4. Therefore their acquisition chains could be merged. Most of the modules in Figure 5.13 are described above. The module denoted H represents the Wiener Filter presented in equation 5.45 with its parameters computed as in equation 5.57.

The unframing module in Figure 5.13 is used to be able to listen to the sound source after it localization. The localization algorithms are also presented in the literature [5.24].

Figure 5.14 is the block diagram of the system developed in this work, combining the GCC, DSB and MVDR algorithms. It shows that the module {GCC, Driver GCC} can be computed in parallel with the {β-PHAT+IFFT} of the DSB and that a parallel computation can also be applied to the modules Driver-DSB and SRP. Although the MVDR could benefit from the result of the GCC and DSB FFT computation, different FFT Windows need to be applied to MVDR depending on the end application. Some of these applications might be Beamforming or Voice recognition. The MVDR branch is dependent on the localization of the sound source to complete its computation as shown by Figure 5.14. The remaining components such as the {Angle Conversion} which convert the DSB localization point to an angle compared to the center of the microphone arrays are necessary for the MVDR computation.
Figure 5.13: MVDR Algorithms Block Diagram
Figure 5.14: Combined Flow of the GCC-DSB-MVDR
Figure 5.15 time diagram represents a simplify implementation order of the main module in Figure 5.14.

1. GCC is computed to detect the sound source DOA, which returns an angle $\phi$.
2. GCC-DSB (Driver GCC) is computed to allow the DSB to interpret the GCC result.
3. DSB is computed to refine the GCC result by finding the exact localization of the sound source distance and angle to the microphone arrays.
4. The Res-CONV is the Driver DSB that converts to the DSB result to a smaller FOV if the speaker speed is known.
5. Finally the MVDR module in Figure 5.14 can be computed.

5.6. Pre-computation of Parameters for GCC-DSB and MVDR

Digital signal processing voice applications require a lot of computation therefore pre-computation of certain parameters at compile-time is crucial. This section will show the impact of the pre-computation of certain parameters on the logic resources utilization and design speed. In this work GCC-DSB algorithms are combined for the sound localization and the MVDR is used for beamforming. All computation happens at run-time for the GCC algorithm. On the other hand, multiple parameters could be computed at compile time for the DSB and the MVDR. In the DSB case the following parameter could be computed at compile time.

The distance in sample unit of point $i$ to every microphone is modeled as in equation (5.58):
\[ Sh_{im} = \frac{r_{im} f_s}{c} \] 

Equation (5.58)

\( r_{im} \) is the distance from the point \( i \) to the microphone \( m \). \( f_s \) is the sample frequency.

\( Sh_{im} \) is a \([N \times NOSS]\) matrix. \( N \) is the number of microphones and \( NOSS \) is the number of small squares in the FOV. The concept of sample shifts presented in Chapter 4 at each microphone allows the sound source signals to be added in a constructive manner. The number of samples in reference to the closest microphone to the sound source is defined in equation (5.59).

\[ Csh_{i,m} = \sum_{i=1}^{NOSS} \sum_{m=1}^{N} (Sh_{i,m} - Min(Sh_{i,m})) \] 

Equation (5.59) shows that the signal arriving to the closest microphone will not be shifted as it represents the shorter distance. Multiple microphones can be located at the shortest distance from certain points of the FOV. For instance in Figure 5.16 microphone A and B are at the same distance from the point 7. Chapter 4 also shows that the weight of every point to each microphone can be pre-computed. A 9 points field of view (FOV) (See Figure 5.17) is used as example to show the DSB computation burden although in the real life case study, FOV contains hundreds or thousands of small squares identified with a number. Each small square stores the following information:

a) The distance from the FOV point to each microphone and to the reference point  
b) The angular inclination to the reference point  
c) Each point’s weight depend on each individual microphone position  
d) Finally, one of the most important parameter is the sound propagation delay.
5.6.1. Distance Pre-Computations

Table 5.3 shows the distance computation from each point of Figure 5.16 to every microphone. The mathematical model of the distance computation shows three main operations, an addition, a power of two and a square root computation. The number of addition and square root computations in Table 5.3 is equal to the number of microphones multiplied by the number of points in the FOV which is \( N \times \text{NOSS} \). The number of power of two operations equal to \( 2 \times N \times \text{NOSS} \). The pre-computation is useful as square root operations use a lot of FPGA resources.

Table 5.3: Distance Computation of all the FOV Small Square to each Microphone

<table>
<thead>
<tr>
<th>Dist</th>
<th>Calcul</th>
<th>Val (m)</th>
<th>Dist</th>
<th>Calcul</th>
<th>Val (m)</th>
<th>Dist</th>
<th>Calcul</th>
<th>Val (m)</th>
<th>Dist</th>
<th>Calcul</th>
<th>Val (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_{1A} )</td>
<td>( 0.25^2 + 1.25^2 )</td>
<td>1.275</td>
<td>( d_{2A} )</td>
<td>( 0.75^2 + 1.25^2 )</td>
<td>1.4577</td>
<td>( d_{3A} )</td>
<td>( 1.25^2 + 1.25^2 )</td>
<td>1.7677</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{1B} )</td>
<td>( 0.25^2 + 1.25^2 )</td>
<td>1.275</td>
<td>( d_{2B} )</td>
<td>( 0.25^2 + 1.25^2 )</td>
<td>1.275</td>
<td>( d_{3B} )</td>
<td>( 0.75^2 + 1.25^2 )</td>
<td>1.4577</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{1C} )</td>
<td>( 0.75^2 + 1.25^2 )</td>
<td>1.4577</td>
<td>( d_{2C} )</td>
<td>( 0.25^2 + 1.25^2 )</td>
<td>1.275</td>
<td>( d_{3C} )</td>
<td>( 0.25^2 + 1.25^2 )</td>
<td>1.275</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{1D} )</td>
<td>( 1.25^2 + 1.25^2 )</td>
<td>1.7677</td>
<td>( d_{2D} )</td>
<td>( 0.75^2 + 1.25^2 )</td>
<td>1.4577</td>
<td>( d_{3D} )</td>
<td>( 0.25^2 + 1.25^2 )</td>
<td>1.275</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7A} )</td>
<td>( 0.25^2 + 0.75^2 )</td>
<td>0.79</td>
<td>( d_{6A} )</td>
<td>( 0.75^2 + 0.75^2 )</td>
<td>1.06</td>
<td>( d_{5A} )</td>
<td>( 1.25^2 + 0.75^2 )</td>
<td>1.4577</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7B} )</td>
<td>( 0.25^2 + 0.75^2 )</td>
<td>0.79</td>
<td>( d_{6B} )</td>
<td>( 0.25^2 + 0.75^2 )</td>
<td>0.79</td>
<td>( d_{5B} )</td>
<td>( 0.75^2 + 0.75^2 )</td>
<td>1.06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7C} )</td>
<td>( 0.75^2 + 0.75^2 )</td>
<td>1.06</td>
<td>( d_{6C} )</td>
<td>( 0.25^2 + 0.75^2 )</td>
<td>0.79</td>
<td>( d_{5C} )</td>
<td>( 0.25^2 + 0.75^2 )</td>
<td>0.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7D} )</td>
<td>( 1.25^2 + 0.75^2 )</td>
<td>1.4577</td>
<td>( d_{6D} )</td>
<td>( 0.75^2 + 0.75^2 )</td>
<td>1.06</td>
<td>( d_{5D} )</td>
<td>( 0.25^2 + 0.75^2 )</td>
<td>0.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7E} )</td>
<td>( 0.25^2 + 0.25^2 )</td>
<td>0.3535</td>
<td>( d_{6E} )</td>
<td>( 0.75^2 + 0.25^2 )</td>
<td>0.79</td>
<td>( d_{5E} )</td>
<td>( 1.25^2 + 0.25^2 )</td>
<td>1.275</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7F} )</td>
<td>( 0.25^2 + 0.25^2 )</td>
<td>0.3535</td>
<td>( d_{6F} )</td>
<td>( 0.25^2 + 0.25^2 )</td>
<td>0.3535</td>
<td>( d_{5F} )</td>
<td>( 0.75^2 + 0.25^2 )</td>
<td>0.79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7G} )</td>
<td>( 0.75^2 + 0.25^2 )</td>
<td>0.79</td>
<td>( d_{6G} )</td>
<td>( 0.25^2 + 0.25^2 )</td>
<td>0.3535</td>
<td>( d_{5G} )</td>
<td>( 0.25^2 + 0.25^2 )</td>
<td>0.3535</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( d_{7H} )</td>
<td>( 1.25^2 + 0.25^2 )</td>
<td>1.275</td>
<td>( d_{6H} )</td>
<td>( 0.75^2 + 0.25^2 )</td>
<td>0.79</td>
<td>( d_{5H} )</td>
<td>( 0.25^2 + 0.25^2 )</td>
<td>0.3535</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Figure 5.17 shows the difference of arrival of the sound source to each microphone with the assumption that the sound source can be located in any of the 9 points of the FOV (see Figure 5.16). It is possible to see that the distances from the points \{1, 4, 7\} is similar to the microphone \{A, B\} and the points \{3, 6, 9\} to the microphones \{C, D\}. The same similarity can be seen from the points \{2, 5, 8\} to the microphone \{B, C\}. Therefore the shape of the FOV can be chosen in such a way that a certain number of distance computations can be deduced by symmetry.

Note that the numbers on the horizontal axis of Figure 5.17 refer to each small square (SS) number seen in Figure 5.16.

![Figure 5.17: DSB Distance Pre-Computation](image)

### 5.6.2. Angular inclination Pre-Computations

The angular inclination of each point to the reference point of the microphone arrays is modeled as in equation (5.60)

$$\phi = \arctan\left(\frac{y}{x}\right)$$  \hspace{1cm} (5.60)

For the points \{2, 5, 8\}, the angle \(\phi\) is 90°. If these parameters are not pre-computed, computation of the arctangent function is required. A calculation of this complexity could require dedicated logic in the form of an IP Core.
5.6.3. Weight

In some DSB applications, a uniform weight is given to all points of the FOV; in this research it is computed as in equation (4.25) in Chapter 4. The same similarity described in the distance computation in Table 5.3 can be seen with Table 5.4. Weights as shown in Table 5.3 are not integer parameters therefore their representation with a fixed points FPGA computation needs to be studied carefully.

<table>
<thead>
<tr>
<th>Weights</th>
<th>value</th>
<th>Weights</th>
<th>value</th>
<th>Weights</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_{1A}$</td>
<td>0.2781</td>
<td>$w_{2A}$</td>
<td>0.2333</td>
<td>$w_{3A}$</td>
<td>0.2006</td>
</tr>
<tr>
<td>$w_{1B}$</td>
<td>0.2781</td>
<td>$w_{2B}$</td>
<td>0.2667</td>
<td>$w_{3B}$</td>
<td>0.2432</td>
</tr>
<tr>
<td>$w_{1C}$</td>
<td>0.2432</td>
<td>$w_{2C}$</td>
<td>0.2667</td>
<td>$w_{3C}$</td>
<td>0.2781</td>
</tr>
<tr>
<td>$w_{1D}$</td>
<td>0.2006</td>
<td>$w_{2D}$</td>
<td>0.2333</td>
<td>$w_{3D}$</td>
<td>0.2781</td>
</tr>
<tr>
<td>$w_{4A}$</td>
<td>0.3042</td>
<td>$w_{5A}$</td>
<td>0.2135</td>
<td>$w_{6A}$</td>
<td>0.1650</td>
</tr>
<tr>
<td>$w_{4B}$</td>
<td>0.3042</td>
<td>$w_{5B}$</td>
<td>0.2865</td>
<td>$w_{6B}$</td>
<td>0.2267</td>
</tr>
<tr>
<td>$w_{4C}$</td>
<td>0.2267</td>
<td>$w_{5C}$</td>
<td>0.2865</td>
<td>$w_{6C}$</td>
<td>0.3042</td>
</tr>
<tr>
<td>$w_{4D}$</td>
<td>0.1650</td>
<td>$w_{5D}$</td>
<td>0.2135</td>
<td>$w_{6D}$</td>
<td>0.3042</td>
</tr>
<tr>
<td>$w_{7A}$</td>
<td>0.3670</td>
<td>$w_{8A}$</td>
<td>0.1545</td>
<td>$w_{9A}$</td>
<td>0.1018</td>
</tr>
<tr>
<td>$w_{7B}$</td>
<td>0.3670</td>
<td>$w_{8B}$</td>
<td>0.3455</td>
<td>$w_{9B}$</td>
<td>0.1641</td>
</tr>
<tr>
<td>$w_{7C}$</td>
<td>0.1641</td>
<td>$w_{8C}$</td>
<td>0.3455</td>
<td>$w_{9C}$</td>
<td>0.3670</td>
</tr>
<tr>
<td>$w_{7D}$</td>
<td>0.1018</td>
<td>$w_{8D}$</td>
<td>0.1545</td>
<td>$w_{9D}$</td>
<td>0.3670</td>
</tr>
</tbody>
</table>

Figure 5.18 shows that more weight importance is given to the points closer to the microphones. The weight magnitude is bigger for points (7, 8, 9) of Figure 5.16 compared to the others.
5.6.4. Sample shift

The sample shift computation in Table 5.5 is similar to the distance computation except that the distance in this case represents the distance in number of samples shifted. It is modeled as in equation (5.58).

Table 5.5: Shift Computation of each FOV Small Square

<table>
<thead>
<tr>
<th>Delay</th>
<th># samples</th>
<th>~ value</th>
<th>Delay</th>
<th># samples</th>
<th>~ value</th>
<th>Delay</th>
<th># samples</th>
<th>~ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_{1A} )</td>
<td>165,37</td>
<td>165</td>
<td>( \tau_{2A} )</td>
<td>188,75</td>
<td>188</td>
<td>( \tau_{3A} )</td>
<td>229,32</td>
<td>229</td>
</tr>
<tr>
<td>( \tau_{1B} )</td>
<td>165,37</td>
<td>165</td>
<td>( \tau_{2B} )</td>
<td>165,37</td>
<td>165</td>
<td>( \tau_{3B} )</td>
<td>188,75</td>
<td>188</td>
</tr>
<tr>
<td>( \tau_{1C} )</td>
<td>188,75</td>
<td>188</td>
<td>( \tau_{2C} )</td>
<td>165,37</td>
<td>165</td>
<td>( \tau_{3C} )</td>
<td>165,37</td>
<td>165</td>
</tr>
<tr>
<td>( \tau_{1D} )</td>
<td>229,32</td>
<td>229</td>
<td>( \tau_{2D} )</td>
<td>188,75</td>
<td>188</td>
<td>( \tau_{3D} )</td>
<td>165,37</td>
<td>165</td>
</tr>
<tr>
<td>( \tau_{4A} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{5A} )</td>
<td>137,59</td>
<td>137</td>
<td>( \tau_{6A} )</td>
<td>188,75</td>
<td>188</td>
</tr>
<tr>
<td>( \tau_{4B} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{5B} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{6B} )</td>
<td>137,59</td>
<td>137</td>
</tr>
<tr>
<td>( \tau_{4C} )</td>
<td>137,59</td>
<td>137</td>
<td>( \tau_{5C} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{6C} )</td>
<td>102,31</td>
<td>102</td>
</tr>
<tr>
<td>( \tau_{4D} )</td>
<td>188,75</td>
<td>188</td>
<td>( \tau_{5D} )</td>
<td>137,59</td>
<td>137</td>
<td>( \tau_{6D} )</td>
<td>102,31</td>
<td>102</td>
</tr>
<tr>
<td>( \tau_{7A} )</td>
<td>45,86</td>
<td>45</td>
<td>( \tau_{8A} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{9A} )</td>
<td>210,79</td>
<td>210</td>
</tr>
<tr>
<td>( \tau_{7B} )</td>
<td>45,86</td>
<td>45</td>
<td>( \tau_{8B} )</td>
<td>45,86</td>
<td>45</td>
<td>( \tau_{9B} )</td>
<td>102,31</td>
<td>102</td>
</tr>
<tr>
<td>( \tau_{7C} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{8C} )</td>
<td>45,86</td>
<td>45</td>
<td>( \tau_{9C} )</td>
<td>45,86</td>
<td>45</td>
</tr>
<tr>
<td>( \tau_{7D} )</td>
<td>210,79</td>
<td>210</td>
<td>( \tau_{8D} )</td>
<td>102,31</td>
<td>102</td>
<td>( \tau_{9D} )</td>
<td>45,86</td>
<td>45</td>
</tr>
</tbody>
</table>

Table 5.5 shows that a fractional error is introduced by taking only the integer part of the number of samples to shift. One way to reduce that error by half is to take the integer
value for the value that has a fractional part less to 0.5 and to round it to the highest integer value when the fractional part is greater to 0.5 while another is to use the frequency approach presented from equation (5.39) to equation (5.42). Figure 5.19 shows the shift of data per microphone for every point of the FOV. It shows that the number of sample shifts vary depending on the distance of the point to the microphone. Figure 5.19 was drawn using data from Table 5.5.

Figure 5.19: DSB Sample Shift Pre-Computation

Figure 5.20 shows the signal-to-noise ratio gain (SNRG) which represents the amplification of the sound source signal depending on its position in the FOV. It is drawn using equation (5.61) and Table 5.3. The SNRG represents the improvement in the SNR at the output of the beamformer compared to the output of a single microphone under the assumption that noise and signal are uncorrelated and stationary [5.27]. The SNRG is defined as in equation (5.61).

\[
SNRG = \frac{\sum_{m=0}^{N\text{OSS}-1} W_{i,m}^2}{\sum_{m=0}^{N\text{OSS}-1} W_{i,m}^2}
\]  

(5.61)
Figure 5.20: DSB SNRG Pre-Computation

Figure 5.20 shows that the signal amplification is optimal when the sound source is at a certain distance from microphone. Looking at the points \{7, 8, 9\} that are the closest to the microphone array, their amplification are the smallest with the middle point having the higher amplification. Figure 5.20 also shows that the points \{1, 2, 3\} which are further away from the microphone have the maximum amplification. An attenuation factor of the sound source could be investigated to better translate the fact that at a certain distance of the microphone the sound source can no longer be detected. Figure 5.20 was drawn using Table 5.3 and equation 5.61.

A complete 256 points FOV is presented in Figure 5.21. It details the principle of the localization of two sound sources. In Figure 5.21, the line with dots represents the sound source direction of arrival angle and the two lines under and above the dots-line delimits the region in which the sound source must be searched. One way to remove the need for drivers shown in Figure 5.14 is to pre-compute the region where the sound source needs to be searched for every sound source (DOA). For the region above microphones (C, D) the points for which the DSB needs to be computed are \{8, 25, 26, 41, 42, 43, 58, 59, 60, etc…\}. For the region above microphones (A, B) the points for which the DSB needs to be computed are \{7, 22, 21, 38, 37, 36, 53, 52, 51, etc…\}.

In the MVDR algorithm, only one parameter can be pre-computed. The inverse coherence matrix $\Gamma_{nn}^{-1}$ can be pre-computed provided the distance between microphones and the frame size remains the same. Pre-computation of parameters for MVDR is more difficult...
as it requires very complex and long operations for matrix inversion. The following steps are necessary to invert the coherence matrix $\Gamma_{vv}$ to compute equations (5.34).

- Check if the matrix has an inverse. Let assume that $V^n$ is a dimensional vector space and $x \in V^n$ if $x = \{x_1, ..., x_n\}$ therefore $\Gamma \in V^{nxn}$, if $\Gamma$ is a $[N \times N]$ matrix. The matrix $\Gamma_{vv}$ can be inverted if there is a matrix $\Gamma_{vv}^{-1}$ so that the equation (5.62) is respected. DET represents the determinant of the matrix $\Gamma_{vv}$.

$$\Gamma_{vv} \cdot \Gamma_{vv}^{-1} = I = \Gamma_{vv}^{-1} \cdot \Gamma_{vv} \quad \text{DET}(\Gamma_{vv}) \neq 0$$ (5.62)

![Figure 5.21: Full FOV Detailed Representations](image)

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>241</td>
<td>242</td>
<td>243</td>
<td>244</td>
<td>245</td>
<td>246</td>
<td>247</td>
<td>248</td>
<td>249</td>
</tr>
<tr>
<td>224</td>
<td>225</td>
<td>226</td>
<td>227</td>
<td>228</td>
<td>229</td>
<td>230</td>
<td>231</td>
<td>232</td>
<td>233</td>
</tr>
<tr>
<td>208</td>
<td>209</td>
<td>210</td>
<td>211</td>
<td>212</td>
<td>213</td>
<td>214</td>
<td>215</td>
<td>216</td>
<td>217</td>
</tr>
<tr>
<td>192</td>
<td>193</td>
<td>194</td>
<td>195</td>
<td>196</td>
<td>197</td>
<td>198</td>
<td>199</td>
<td>200</td>
<td>201</td>
</tr>
<tr>
<td>176</td>
<td>177</td>
<td>178</td>
<td>179</td>
<td>180</td>
<td>181</td>
<td>182</td>
<td>183</td>
<td>184</td>
<td>185</td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>162</td>
<td>163</td>
<td>164</td>
<td>165</td>
<td>166</td>
<td>167</td>
<td>168</td>
<td>169</td>
</tr>
<tr>
<td>144</td>
<td>145</td>
<td>146</td>
<td>147</td>
<td>148</td>
<td>149</td>
<td>150</td>
<td>151</td>
<td>152</td>
<td>153</td>
</tr>
<tr>
<td>128</td>
<td>129</td>
<td>130</td>
<td>131</td>
<td>132</td>
<td>133</td>
<td>134</td>
<td>135</td>
<td>136</td>
<td>137</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
<td>120</td>
<td>121</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
<td>72</td>
<td>73</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

$A$ $B$ $C$ $D$
5.6.5. Inter-Action between Hardware and Software for Pre-Computation

Figure 5.22 presents the hardware and software interaction. It shows that many pre-compiled coefficients from software can be passed to the hardware to accelerate hardware processing and reduce the logic used. Hardware data can be passed back to the software to compare the results against those expected.

![Hardware/Software Systems Interaction](image)

Table 5.6 shows the advantage of pre-computing a huge part of the design in software before hardware implementation. It shows the benefit of pre-computations of the weight (W) and data shift (SH) on the speed and resources of the FPGA. The “Hardware + Software” approach of Table 5.6 shows that the system runs at a higher speed compared to the Hardware only approach because the only limitation in that approach is the BlockRAM speed. For Virtex-5, it is around 550MHz. The resources utilization are also reduced as they are pre-computed at compile time. The experiment was done on a 9 points FOV and four microphones (see Figure 5.16). No multiplications or divisions are necessary in the hardware + software co-design, as those values are pre-computed in software, which explains the zero number of DSP48 resources used in TABLE 5.6.
Table 5.6: Weight and Shift Computation using ML505 BOARD

<table>
<thead>
<tr>
<th>Resources</th>
<th>Hardware + Software (%)</th>
<th>Hardware (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice Registers</td>
<td>8</td>
<td>1%</td>
</tr>
<tr>
<td>SLICE LUTs</td>
<td>8</td>
<td>1%</td>
</tr>
<tr>
<td>RAM</td>
<td>2</td>
<td>3%</td>
</tr>
<tr>
<td>DSP48</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>Estimate Speed</td>
<td>450 MHz</td>
<td></td>
</tr>
</tbody>
</table>

5.7. Summary and further work

This work has shown that the main contribution of this research, which was to accelerate sound source localization, is better suited when using a single sound source. Although the approach presented in Chapter-4 was expanded in this Chapter to accelerate multiple sound source localization, it scores poorly to reduce interference between speakers. One possible avenue to improve this research and reduce interferences between speakers is to create multiple sub-array microphone system as presented in Figure 5.23.

In Figure 5.23, multiple microphone arrays are used to detect the both sound source A and B. The first microphone array is composed of microphone \( \{1, 2, 3, 4\} \) with its center at \( O_1 \), the second is composed of the microphone \( \{1, 2, 3, 4, 5, 6, 7, 8\} \) with its center at \( O_2 \) and the last one is composed of microphone \( \{7, 8\} \) with its center in \( O_3 \). The difference in distance between the sound source different sub-arrays center improve the detection of the speakers A and B.

Although the localization of the sound source DOA is more directive using the very well known MUSIC algorithm (see Figure 5.8) compared to the algorithm proposed in this
work (see Figure 5.4), the necessity to know the number of speakers beforehand in the MUSIC algorithm is a colossal drawback. Equation (5.19) or (5.20) used to approximate the number of speakers can hardly be computed in real-time.

An alternative approach to detect the number of speakers prior to computing the MUSIC algorithm is suggested below. This could improve the overall algorithm. In [5.25] the cross-correlation function of the Hilbert envelope (HE) of the linear prediction (LP) residual signals derived from the multi-speaker signals is used to determined the number of speakers. The computation of the preprocessed HE is defined as in equation (5.63).

\[
    g_i(n) = \frac{h_i^2(n)}{1/(2Ns + 1) \sum_{m=n-Ns}^{n} h_i[m]} \quad i \in \{1, 2, \ldots, N\}
\]

(5.63)

Where \( g_i(n) \) is the preprocessing HE of the LP residual of multi-speakers collected at \( i^{th} \) microphone, \( Ns \) is the number of samples corresponding to one frame and \( N \) the number of microphones. The Hilbert envelope \( h(n) \) of the LP \( e(n) \) is defined as in equation (5.64).

\[
    h(n) = \sqrt{e^2(n) + e_H^2(n)}
\]

(5.64)

Where \( e_H(n) \) is the Hilbert transform of \( e(n) \). The number of prominent peaks of the cross-correlation function \( \eta_2(l) \) between the preprocessed HEs \( g_1(n) \) and \( g_2(n) \) should correspond to the number of speakers.
CHAPTER 6 Low Latency Hybrid Speaker Localization Algorithm Implemented in an FPGA Architecture

6. DECLARATION AND PRESENTATION

I hereby declare that this chapter is entirely of my own doing and that it has not been submitted as an exercise for a degree at any other University.

6.1. Purpose of this Chapter

This chapter will present the architecture of the overall system, as designed by the author, as well as the modules that implement the novel hybrid Algorithm and Beamforming presented in Chapter 4 and Chapter 5 to achieve real-time operation. This chapter will also elaborate a strategy to reduce the system power consumption as described in Chapter 1 by increasing module re-usability, while controlling bit growth and dataflow. Finally, this chapter will present the partition of the novel algorithm to suit Partial Reconfiguration (PR) and Multi-Boot (MB) as described in Chapter 2.

6.2. Introduction

The improvement of throughput in speaker’ localization algorithms require that the most suitable architecture that can guarantee the best dataflow and flexibility be chosen. Good coding technique and bit growth control can slightly reduce resource utilization without impacting throughput. The main concern when implementing a localization algorithm is the signal integrity between the microphone and the FPGA. The choice of the FPGA used to implement the design is determined by the criteria below:
1. The FPGA storage resources (RAM), computational resources (DSP48, IP-cores, Primitive) and logic (slice).
2. FPGA Clock speed to achieve the desired throughput.
3. FPGA Interface to the external world.

Point (1) influences the extent of the parallelization that can be applied to the algorithm. Point (2) affects the computation speed independently of the processing approach (sequential or parallel). Point (3) decides whether an interface is necessary between the FPGA and the sensors (MEMS Microphones).

Reducing the logic resources and switching frequency \( f_c \) presented in points (1), (2) and (3) will impact the power consumption by limiting the nets’ switching activities denoted as \( \alpha \) in equation (2.9) of Chapter 2.

The remaining sections of the chapter are organized as follows: section 6.3 presents the proposed systems full architecture. Sections 6.4 presents the system acquisition chain from the filter to the Fast Fourier Transform (FFT). Section 6.5 discusses the logic diagram of the computation chain from after the FFT to the localization of the sound source using the GCC and DSB. Section 6.6 discusses the system partition to implement the reconfiguration flow. Section 6.7 gives a brief presentation of the system verification strategy. Section 6.8 will presents conclusions and possible further work.

6.3. System Full Architecture

The system depicted in Figure 6.1 is the proposed block diagram for the implementation of the beamforming and sound source localization using the novel algorithm developed in this work. Figure 6.1 shows that the main synchronization logic between modules is done by the master state machine (SM) as it is the only logic common to all of the modules (See Figure 6.2). All the modules in pink below the master SM are module SMs that communicate the state of their modules’ computation (in blue) to the master SM (See details in Figure 6.2). The results of the computation of the modules in blue are passed to their corresponding registers (in yellow) or to the interfaces (in yellow) and are then transferred to the corresponding Block.
RAM. The black lines between interfaces (in yellow) are buses to transfer data between modules. A Router (in yellow) can also be used for data transfer between modules. A more accurate representation of a system that uses a router solely for data transfer between module is represented in Figure 6.3. All the modules and their impact on the overall system throughput are explained later in this Chapter.
Figure 6.1: Full System Block Diagram

Figure 6.2: Master State Machine Descriptions
Figure 6.3: Full System using Routers to Communicate between Process Elements (PE)

The Router presented in Figure 6.3 (denoted by R) is a possible expansion of this work as intensive parallel communication may become necessary between the FFT and IFFT (Inverse FFT) if the number of microphones is increased. Therefore an embedded switching network, called a Network-on-Chip (NoC), to interconnect IP modules to the system on chip (SOC) may becomes necessary when the system is expanded to include voice recognition.

The router for a NOC takes more logic resources than a bus based interface [6.1] but the NOC can reduce the routing associated with large buses.

Therefore the connection between the module in this work will be a combination of point to point and point to bus. Two groups of modules are represented in the system shown in Figure 6.1: those representing the acquisition chain and those representing computation chain.
6.4. Acquisition Chain Processing Element (PE)

The purpose of the acquisition chain is to retrieve the modulated signal from the microphone, demodulate it using a filter, then cut it into frames to detect the presence of a voice in each frame. The detection of the voice is performed using a voice activity detector (VAD) and then a Fast Fourier Transform (FFT) is computed. The acquisition chain is shown as a point to point connection in Figure 6.4 and as a point to bus connection in Figure 6.5. This difference will be explained later in this chapter.

![Figure 6.4: System Acquisition Chain using a Point-to-Point Connection](image)

![Figure 6.5: System Acquisition Chain using a Point-to-Bus (Multiplexer) Connection](image)
6.4.1. Filters and Framing modules

Different filters such as FIR (Finite Impulse Response) or CIC (Cascade Integrator-Comb) can be used to demodulate the sigma-delta ($\Delta$-$\Sigma$) signal coming from the microphone (see Figure 6.4). The FIR filter is mathematically modeled as in equation (6.1) and computed using the Block Diagram in Figure 6.6.

\[
Y[n] = \sum_{i=0}^{N-1} H[i]X[n-i]
\]  

where $N$ is the number of coefficients (or taps) of the filter, $X$ is the input signal, $Y$ is the output, $Y[n]$ is the current output sample and $H[i]$ represents the filter coefficients [6.2].

Cascade Integrator-Comb (CIC) filters, also known as Hogenauer filters, are multirate filters often used for implementing large sample rate changes in digital systems. CIC filters have a structure that only requires adders, subtractors, and delay elements. These simple structures make CIC filters appealing for their hardware-efficient implementations of multirate filtering [6.3]. Therefore the CIC filter will be preferred for this work. The general concept of a CIC filter is the low-pass response that results from the filtering of input signals. The system response of such filter is modeled in equation (6.2).

\[
H(Z) = \left(\frac{1-Z^{-R*M}}{1-Z^{-1}}\right)^N
\]  

where $N$ is the number of CIC stages, $R$ is the rate change (decimation or interpolation) and $M$ is the differential delay in the comb section stages of the filter.
There are two sections to the CIC decimator filter: an integrator section with N integrator stages that process input data samples at a sampling rate $f_s$ and a comb section that operates at a lower sampling rate $f_s/R$. This comb section consists of N comb stages with a differential delay of M samples per stage. The down sampling operation decimates the output of the integrator section by passing only the Rth sample to the comb section of the filter. Figure 6.7 represents the CIC computation block diagram.

Figure 6.7 shows why the CIC filters are preferred in this work; it is due to its simple architecture compared to the FIR filters. The CIC decimators gain $G$ at the comb section output is defined as in equation (6.3) [6.3].

$$GAIN = (RM)^N$$ (6.3)

Equation (6.3) is used to calculate the number of bits required for the last comb of the filter due to bit growth. If $Bin$ is the input number bits, then the output size $Bout$ is modeled as in equation (6.4).

$$Bout = \text{ceiling}(N \cdot \log_2(RM)) + Bin$$ (6.4)

In general when ignoring the noise in the computation the input and output widths of the CIC filter are almost in the same range [6.3]. Bit growth control is adopted throughout this work to reduce the logic resources and therefore the power consumption.

Figure 6.4 shows a module called Framing used to cut the signal in different frame sizes. The number of samples in every frame is computed as in equation (6.5)

$$Fram(sample) = X \mod Y = X - (Y \cdot E(X/Y))$$ (6.5)

$E(X/Y)$ is the integer part of $X/Y$. $X \in N$ and is a fixed value which is a power of two that can take any value from [256..4096] while $Y$ varies from [0..X]. $Y$ varies as modeled in equation (6.6).
\[ Y = U_{n+1} = U_n + r \]  

(6.6)

Where \( U_0 = 0 \), \( r = 1 \) and \( 1 \leq n \leq X \).

### 6.4.2. VAD and FFT Modules

Voice activity detection (VAD) is used to detect the presence of speech in an audio signal. VAD plays an important role as a preprocessing stage in numerous audio processing applications. For example, in voice over IP (VoIP) and mobile telephony applications, VAD can reduce bandwidth usage and network traffic by transmitting audio packets only if speech is detected. Furthermore, the performance of speech recognition, speaker recognition, and source localization can be improved by applying these algorithms only to parts of the audio that are identified as speech. Video conferencing is another prominent source localization application where VAD is important. In such an application, source localization is performed, and the video camera is steered in the direction of the audio source when speech is detected using VAD [6.4]. Actual speech activities normally occupy 60% of the time of a regular conversation in a telecommunication system. VAD enables reallocating resources during the periods of speech absence [6.5]. The VAD module is used to compute noise mean and variance. These values are used to determine a threshold equation (6.7) a mean equation (6.9) and a variance equation (6.10). Each incoming frame is compared to that threshold. If the computed value is lower than the threshold, it is possible to conclude that a noise is present, otherwise it may be concluded that the signal is a voice signal.

\[ N_{\text{thresh}} = \mu + \text{cst} \times \sigma \]  

(6.7)

A good statistical model of a noise signal in the time domain is the zero-mean gaussian process \( N(\mu_N, \sigma_N^2) \) with a given variation \( \sigma_N^2 \), mean \( \mu_N = 0 \), and normal probability density function (PDF) given by equation 6.8 presented in Figure 6.8 [6.6].

\[ P(x|\mu_N, \sigma_N) = \frac{1}{\sigma_N\sqrt{2\pi}} \exp\left(\frac{-(x-\mu_N)^2}{2\sigma_N^2}\right) \]  

(6.8)

The constant (cst) that will be multiplied by \( \sigma \) in equation (6.7) is 3 as shown in Figure 6.8 to assure that above the threshold the signal contains a valid sound source.
The mean ($\mu$) defined in equation (6.8) is modeled as in equation (6.9).

$$\mu = E(x) = \frac{\sum_{i=0}^{N_s} X_i}{N_s}$$  \hspace{1cm} (6.9)

The variance ($\sigma$) is defined as equation (6.10).

$$\sigma^2 = E(x^2) - E(x)^2 = E(x) = \frac{\sum_{i=0}^{N_s} x_i^2}{N_s} - \mu^2$$  \hspace{1cm} (6.10)

Equation (6.8) and (6.9) can be implemented in an FPGA using the block diagram presented in Figure 6.9 and Figure 6.10.
The series of letters in Figure 6.9 are intermediary nets sizes in the bit control growth of the mean computation. Table 6.1 shows bit growth per state. The main idea of this concept is to provide to the net only the size necessary to hold the result of the previous computation. The purpose is to limit power consumption.

Table 6.1: Bit Growth per State in the Mean Computation

<table>
<thead>
<tr>
<th>STATE</th>
<th>BIT GROWTH</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$N_b$</td>
</tr>
<tr>
<td>B</td>
<td>$N_b + (N_s - 1)$</td>
</tr>
<tr>
<td>C</td>
<td>$N_b + (N_s - 1) - \log_2 N_s$</td>
</tr>
<tr>
<td>D</td>
<td>$N_b + (N_s - 1) - \log_2 N_s + (N - 1)$</td>
</tr>
<tr>
<td>E</td>
<td>$N_b + (N_s - 1) - \log_2 N_s + (N - 1) - \log_2 N$</td>
</tr>
<tr>
<td>F</td>
<td>$N_b + (N_s - 1) - \log_2 N_s + (N - 1) - \log_2 N + (N_c - 1)$</td>
</tr>
<tr>
<td>G</td>
<td>$N_b + (N_s - 1) + (N - 1) + (N_c - 1) - \log_2 N_s - \log_2 N - \log_2 N_c$</td>
</tr>
</tbody>
</table>

Figure 6.10: Block Diagram of the Standard Deviation Computation over Multiple Frames
Along with the bit growth control described above, this work proposes to implement Figure 6.9 and Figure 6.10 as presented in Figure 6.11. Although the fully combinatorial approach shown in Figure 6.11.x represents the fastest system, the output x and all the intermediary stages will be exposed to glitches from the inputs \{A, B, C, D\}. The switching of these nets increases the power consumption and does not guarantee the stability of the output result. Figure 6.11. y registers the output to guarantee the stability of the output results and reduces the glitches, thereby reducing the power consumption. Leaving the system inputs without registers may be necessary depending on the system connected to them (combinatorial or sequential). The fully registered approach of Figure 6.11.z will always be preferred whenever possible to reduce data glitches and improve timing constraints.

To detect a voice in a signal, the VAD algorithm (See Figure 6.9 and Figure 6.10) is required to perform some multiplications with a constant as shown in equation 6.7. Figure 6.12 presents the author's proposed methods for multiplying variables by a constant. Figure 6.12.a shows that the multiplication of a variable by 6 can be divided in two simple multiplications depicted in Figure 6.12.b. Figure 6.12.b still presents a problem as the multiplication by 3 is not handled so well in a FPGA. Therefore the multiplication by 3 can be expressed as in Figure 6.12.c which shows that a multiplication can then be transformed to
series of additions and shift registers. Figure 6.12.d can also be considered to compute Figure 6.12.a at the cost of more shift registers and additions. The methodology used in this work reduces the use of dedicated multipliers in the FPGA.

![Diagram](image)

Figure 6.12: Implementation of Constant Factor Multiplication using Shift Register and Addition [6.9]

The concept illustrated in Figure 6.12 is very important especially when using an FPGA with a limited number of dedicated multipliers as shift and addition operations are in general less costly in term of logic resources. Figure 6.13 shows the distribution of logic resources in the Virtex-5 (ML505). It shows a limited number of DSP48 blocks capable of performing multiplication and accumulation operations, therefore DSP48 should be used with caution. Thus the knowledge of the resources distribution on the hardware selected is crucial. It is also necessary when floorplanning the design to place and route close to other modules which interact together to avoid any timing issue and to utilize hardware resources efficiently.

Figure 6.14 depicts a DSP48 block diagram which could be used to perform complex number operations such multiplication, accumulation after the FFT computation. To minimize the use of DSP48 blocks they should be allocated only for operations such as in equation (6.11). These computation rules can saves up to 25% of the DSP multipliers.

\[(a+bj)(c+dj) = ac - bc + j(ad + bc) \quad (6.11)\]
Another resource that is very important for the implementation of this work is the number of Block Rams in the FPGA. The limited number of BlockRAM (BLKRAM) in some FPGA as shown in Figure 6.13 can be compensated by distributed BlockRAMs (ie a Block Ram made of logic resources) if necessary. All the other FPGA resources are sufficient for this work. This illustrates the importance of resource and layout planning.

Figure 6.13: Resource Distribution in a Virtex-5 (ML505) FPGA
Figure 6.5 shows that every microphone is connected to two Block Rams. This architecture is explained by the fact that the microphone continuously records and sends data. When the data in the first Block Ram is being read to detect a sound source location the second Block Ram is receiving the data from the microphone. The stop condition in the state machine in Figure 6.15 is a blocking condition that should be avoided. It occurs in the following condition: The first Block Ram is full and the reading of the second Block Ram is not terminated and vice versa, resulting in the loss of data. The simplest way to address this issue is to create a faster clock dedicated to the reading of data while the data writing in the Block Ram is done with a slower clock. The novel algorithm architecture will therefore require a multiple clocks domain system approach.
The timing diagram of the acquisition chain is presented in Figure 6.16. It shows that data latency when reading the Block Ram is similar to the FFT (Fast Fourier Transform) computation that shows that FFT Processing Element (PE) does the reading of the Block Ram.

The multiplexing of multiple Block RAMs to one FFT PE as described in Figure 6.5 is justified by the fact that the writing latency of the Block RAM (the time taken to write a data frame) is longer than the time to read the Block RAM.
The IDLE time in the timing diagram, indicate that there is a window of time during which the Block Ram could be used for another function.

![Acquisition Chain Timing Diagram](image)

**Figure 6.16: Acquisition Chain Timing Diagram**

The multiplexer connected to the FFT in Figure 6.5 could be replaced by a small system that directs the FFT computation of each Block Ram connected to the microphone starting with the first Block Ram then the second. A simple state machine such as the one depicted in Figure 6.17 can replace the multiplexer shown in Figure 6.5.

![State Machine Replacing the Figure 6.5 Multiplexer for FFT Computation](image)

**Figure 6.17: State Machine Replacing the Figure 6.5 Multiplexer for FFT Computation**

The Discrete Fourier Transform (DFT) is one of the most important tools in the field of digital signal processing. Due to its computational complexity, several fast Fourier transform (FFT) algorithms have been developed over the years. It has been shown that the decimation-in-time (DIT) algorithms provide better signal-to-noise-ratio than the decimation-in-frequency (DIF) algorithms when finite word length is used [6.8]. The most common representation of
the signal as a sum of sinusoidal signals is the Fourier Transformation which in is continuous
variant has the form presented in equation (6.12) [6.6].

\[ X(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft}dt \] (6.12)

where \( x(t) \) is the input signal, \( X(f) \) the spectrum with each frequency \( f \) linked to a
magnitude and phase respectively as in equation (6.13) and (6.14).

\[
\left| X(f) \right| = \sqrt{\Re(X(f))^2 + \Im(X(f))^2} \] (6.13)

\[
\phi(f) = -\arctan\left(\frac{\Im(X(f))}{\Re(X(f))}\right) \] (6.14)

Although many approaches such as the Taylor approximation of equation 6.15 could be
used to compute the square root or the angle of arrival of the sound source model as in
equation 6.13 or equation 6.14 [6.9], this approach is resource and time costly as shown by
the polynomial development of sine and cosine in equation (6.15, 6.16). Therefore an
approach based on the cordic algorithm is preferred.

\[
f(x_0) = \sum_{k=0}^{K} \frac{df^{(k)}(x-x_0)}{dx} x^k \bigg|_{x=x_0} \] (6.15)

Cosine polynomial approximation

\[
\cos(X) = 1 - \frac{X^2}{2} + \frac{X^4}{4!} - \frac{X^6}{6!} + \ldots + (-1)^n \frac{X^{2n}}{(2n)!} + \ldots = \sum_{k=0}^{\infty} (-1)^n (X)^{2n} \] (6.16)

Sine polynomial approximation

\[
\sin(X) = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots + (-1)^n \frac{X^{2n+1}}{(2n+1)!} + \ldots = \sum_{k=0}^{\infty} (-1)^n (X)^{2n+1} \] (6.17)

The Cordic algorithm is well discussed in the literature [6.10 , 6.11 ,6.12] and the
mathematical model as in Table 6.2 is preferred to the Taylor’s expansion due to the
simplicity of the algorithm architecture. The Cordic solution is composed only of shift register
adders and subtractors as shown in Figure 6.18. Many previously verified Cordic
implementations are available as public domain Intellectual Property (IP) modules, and they
can be used to perform this computation.
Table 6.2 presents the form of the equations used in the computations of the CORDIC algorithm while Figure 6.18 shows a diagram of a typical CORDIC implementation.

### Table 6.2: CORDIC Algorithm Computation Equation [6.13]

<table>
<thead>
<tr>
<th>CORDIC – Rotation/Vector Modes</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rotation Mode</strong></td>
<td></td>
</tr>
<tr>
<td>( x_{i+1} = x_i - y_i d_i 2^{-i} )</td>
<td>( x_n = A_n [x_0 \cos z_0 - y_0 \sin z_0] )</td>
</tr>
<tr>
<td>( y_{i+1} = y_i + x_i d_i 2^{-i} )</td>
<td>( y_n = A_n [y_0 \cos z_0 + x_0 \sin z_0] ), ( z_n = 0 )</td>
</tr>
<tr>
<td>( z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) )</td>
<td>( A_n = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} )</td>
</tr>
</tbody>
</table>
| \( d_i = \begin{cases} 
-1 & \text{if } z_i < 0 \\
+1 & \text{otherwise} 
\end{cases} \) |  |
|  |  |
| **Vector Mode**               |  |
| \( x_{i+1} = x_i - y_i d_i 2^{-i} \) | \( x_n = A_n \sqrt{x_0^2 + y_0^2} \) \( y_n = 0 \) |
| \( y_{i+1} = y_i + x_i d_i 2^{-i} \) | \( z_{i+1} = z_i + \tan^{-1}\left(\frac{y_i}{x_i}\right) \) |
| \( z_{i+1} = z_i - d_i \tan^{-1}(2^{-i}) \) | \( A_n = \prod_{i=0}^{n} \sqrt{1 + 2^{-2i}} \) |
| \( d_i = \begin{cases} 
+1 & \text{if } y_i < 0 \\
-1 & \text{otherwise} 
\end{cases} \) |  |
This design can use Cordic Intellectual property (IP) for the computation of operations such as: trigonometric functions, logarithms, exponential and square root. To localize the sound source the signal received by the microphone is discretized before computing the FFT. The FFT is then modeled as in equation (6.18).

\[
X(k) = \sum_{n=0}^{N-1} x(n)W_{N}^{nk}
\]  

(6.18)

Where \(W\) is the twiddle factor modeled as in equation (6.19).

\[
W_{N}^{nk} = e^{-j\left(\frac{2\pi nk}{N}\right)} = \cos\left(\frac{2\pi nk}{N}\right) - j\sin\left(\frac{2\pi nk}{N}\right)
\]  

(6.19)

Where \(x(n)\) is the time-domain discrete input signal and \(X(k)\) is the signal spectrum. \(n\) represents the discrete time-domain index, while \(k\) is the normalized frequency-domain index.

Since FFT processors using a radix-4 architecture have fewer multiplications than processors using radix-2 [6.14], they are preferred in order to reduce the memory access rate and arithmetic workload, and hence, power consumption. The difference in the number of
multiplications and additions for a 64 points FFT between Radix-2 and 4 are presented in Table 6.3 below. In addition, the higher parallelism of radix-4 compared to the radix-2 architecture shown in Figure 6.19 and Figure 6.20 respectively provides the radix-4 approach with a better throughput.

Table 6.3: 64-POINT FFT ALGORITHM COMPLEXITY [6.15]

<table>
<thead>
<tr>
<th>ALGORITHM</th>
<th>Number of Real Multiplications</th>
<th>Number of Real Additions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RADIX-2</td>
<td>264</td>
<td>1032</td>
</tr>
<tr>
<td>RADIX-4</td>
<td>208</td>
<td>976</td>
</tr>
</tbody>
</table>

Figure 6.19: Radix-4 Arithmetic Unit
The FFT is very well documented in the literature and multiple papers propose architectures for its implementation [6.15, 6.16, 6.17].

6.5. Computation Chain Processing Element (PE)

The sound source location is computed in both the frequency and time domains. In the frequency domain, the GCC and the DSB $\beta$-PHAT are computed and in the time domain, the DSB SRP and the GCC direction of arrival (DOA) are computed.

6.5.1. Frequency Domain Computation

The DSB $\beta$-PHAT is computed for every microphone channel after the FFT computation as shown in Figure 6.21. The DSB $\beta$-PHAT is modeled as in equation 6.20 and computed with the logic resources presented in Figure 6.22. For four microphones 4 modules such Figure 6.22 would be necessary for a full parallel implementation of the DSB $\beta$-PHAT.

\[
W(f) = \frac{X(f)}{|X(f)|^\beta} \quad \text{(6.20)}
\]
The GCC $\beta$-PHAT is computed on all possible combinations of the FFT channel taken two by two as shown in Figure 6.23. This figure demonstrates the asymmetry of the GCC algorithm as the number of IFFTs has increased compared to the number of FFTs. Equation 6.21 is the mathematical model of the GCC computation in the frequency domain that is computed using the logic resources presented in Figure 6.24.

$$C(k) = \left( \frac{FFT(x_i(t)).FFT(x_j^*(t))}{|FFT(x_i(t)).FFT(x_j^*(t))|^\beta} \right)$$  \hspace{1cm} (6.21)
Table 6.4 shows an estimation of the number IP cores that can be used for the implementation of the Author’s new-hybrid algorithm (GCC-DSB). The asymmetry between number of FFTs and IFFTs presented above can be predicted using the $C^2_N$ Formula (See Equation 4.2 for more details). One way to reduce this asymmetry is to change the connection between modules. Looking at Table 6.4 the point-to-point connection approach cannot be the only connection used for the implementation of the GCC-DSB as it requires many IP Cores. The point-to-bus connection compare to the point-point reduce the number IP Cores as it increases modules re-usability. The point-to-bus can easily be implemented due to the different PE computation speed in the system [4.14].

**Table 6.4: Combined GCC-DSB IP Cores Resources**

<table>
<thead>
<tr>
<th>Resources</th>
<th>Resources IP Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Point-To-Point</td>
</tr>
<tr>
<td>CIC</td>
<td>4</td>
</tr>
<tr>
<td>FFT</td>
<td>4</td>
</tr>
<tr>
<td>SQRT</td>
<td>10</td>
</tr>
<tr>
<td>DIV</td>
<td>10</td>
</tr>
<tr>
<td>IFFT</td>
<td>10</td>
</tr>
</tbody>
</table>
The GCC and the DSB fetch their data from the same FFT result Block RAM. The sequence in which the data is fetched from the Block Ram impacts many system parameters, including the throughput, the system re-usability, the system flexibility and the necessary number of accesses to the Block Ram. Looking at Figure 6.22 it is seen to be similar to Part (2) of Figure 6.24 therefore the modules re-usability could be implemented. Figure 6.24 represents the logic necessary to implement one cross-correlation between two microphones. A system of 4 microphones would need 6 modules such as Figure 6.24 for a full parallel implementation.
Figure 6.25 expands the example of cross correlation of two microphones presented in Figure 6.24 to 4 microphones. It discusses the sequence in which the data need to be fetched from the 4 different Block RAMs to optimize the DSB \( \beta \)-PHAT and GCC \( \beta \)-PHAT computation criteria. The main criteria being:

1. The system flexibility
2. Reduction of the number of switches between Block RAMs
3. System re-usability

Figure 6.25 (a) represents the most flexible approach as it can be adapted to any change to the number of microphones in the system. It is a flexible approach due to its adaptability and predictability. The fetch sequences of Figure 6.25 (a) can be modeled with a simple double for loop with variable \((I, J)\) varying as \(I = 1\) and \(I < J \leq N\) for the branch A. For the branch B the loop is represented with variable \((k, p)\) varying as \(I < k \leq N-1\) and \(k < p \leq N\) where \(N\) is the number of microphones. The flexibility in Figure 6.25(a) comes at the cost of more logic resources. The state machine to control the fetching of data from branch A or B of Figure 6.25(a) is depicted in Figure 6.26. So although Figure 6.25(a) also reduces the amount
of switching between Block RAMs, the drawback of this approach remains the amount of logic necessary to implement it.

Figure 6.26: State Machine of Figure 6.25.a Fetch Sequence

Figure 6.25 (b) and Figure 6.25 (c) are the fastest configurations and represent the least logic resources configurations. Their data fetching sequences from different BlockRAMs along with Figure 6.25 (d) must be hard coded in logic or a Block RAM as they are not predictable. Figure 6.25 (d) is the only sequence where all microphones (N = 4) are fetched at the first computation sequence allowing the $\beta$-PHAT block diagram of Figure 6.22 to be reused for the GCC computation of Figure 6.24 part (2) after the first data fetch.

Figure 6.27 shows the impact on the throughput of the order in which the Block RAMx are read to compute the GCC and the DSB $\beta$-PHAT. It shows that Figure 6.25 (b) and Figure 6.25 (c) are the fastest. The longest throughputs are in Figure 6.25 (a) and Figure 6.25 (d) but they are the best in term of flexibility and module re-usability respectively.
The last computation in the Frequency domain is the IFFT Inverse (FFT). It is mathematically modeled as in equation (6.22) which can be implemented by reusing the FFT PE as shown in Figure 6.28. This methodology also increases the modules re-usability.

\[ w(n) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} W(f)e^{2\pi ft} df \]  

(6.22)

6.5.2. Time Domain Computation after the IFFT

After the IFFT the computation of the direction of arrival (DOA) of the sound source is straightforward as shown in Figure 6.29. The number of Block RAM accesses is \( C^p_N * N_s \) where \( N_s \) is the number of samples in the Frame, \( N \) is the number of microphones and \( P = 2 \). The number of Arcosine functions is \( C^p_N \), the number of additions is \( N*N_s+1 \). The computation of the GCC DOA angle is not costly in resources and time.
Figure 6.29: GCC Angle Computation Block Diagram

The main computation cost in sound source localization comes from the delay and sum beamforming and the SRP algorithm shown in Figure 6.30. Figure 6.30 shows that a certain numbers of adders and multipliers can be used in parallel to provide a hardware contribution to accelerate the hybrid algorithm developed to compute the DSB in real-time. The novel hybrid algorithm reduces the number of small square (NOSS) to be searched to localize the sound source and can be combined to parallel computation and pipeline approach presented in Chapter 2 to improve the system throughput. This approach is inspired by Amdahl strategy presented in Chapter 2.
Figure 6.30: Delay and Sum and Steered Power Response Algorithm Block Diagram
Table 6.5 also be seen in Chapter 4 as Table 4.6 shows the contribution of the novel hybrid algorithm, reducing the number of NOSS depending on the sound source direction of arrival (DOA) in this case $\phi = 53.5^\circ$, $\phi = 43.5^\circ$ and $\phi = 20^\circ$.

Table 6.5: DSB Computation Throughput with N=4, Ns=512

<table>
<thead>
<tr>
<th>Localization</th>
<th>$N_{\text{oss}} = 256$ (DSB)</th>
<th>$N_{\text{oss}} = 32$ (DSB+GCC)</th>
<th>$N_{\text{oss}} = 25$ (DSB+GCC)</th>
<th>$N_{\text{oss}} = 12$ (DSB+GCC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blk$_{\text{read}}$</td>
<td>$[(N_s \cdot N)<em>{\text{data}} + (N_s \cdot N)</em>{\text{PHAT}} \cdot N_{\text{oss}} + (2N_s \cdot N)<em>{\text{S_R}} + (2N_s \cdot N)</em>{\text{S_R}} \cdot N_{\text{oss}}]$</td>
<td>1575108</td>
<td>203040</td>
<td>159969</td>
</tr>
<tr>
<td>MULT</td>
<td>$[(N_s \cdot N)<em>{\text{data}} + (N_s \cdot N)</em>{\text{PHAT}} \cdot N_{\text{oss}} + (2N_s \cdot N)<em>{\text{S_R}} + (2N_s \cdot N)</em>{\text{S_R}} \cdot N_{\text{oss}}]$</td>
<td>2228480</td>
<td>296993</td>
<td>236038</td>
</tr>
<tr>
<td>ADD</td>
<td>$[(N_s \cdot N)<em>{\text{data}} + (N_s \cdot N)</em>{\text{PHAT}} \cdot N_{\text{oss}} + (2N_s \cdot N)<em>{\text{S_R}} + (2N_s \cdot N)</em>{\text{S_R}} \cdot N_{\text{oss}}]$</td>
<td>1572864</td>
<td>206853</td>
<td>163845</td>
</tr>
<tr>
<td>DIV</td>
<td>$(2N_s \cdot N)<em>{\text{S_R}} \cdot N</em>{\text{oss}}$</td>
<td>1048576</td>
<td>137217</td>
<td>108545</td>
</tr>
<tr>
<td>SQRT</td>
<td>$(N_s \cdot N)<em>{\text{S_R}} \cdot N</em>{\text{oss}}$</td>
<td>524288</td>
<td>68808</td>
<td>54272</td>
</tr>
</tbody>
</table>

Assuming that all operations are performed in one clock cycle, except the division and square root (8 clock cycles using Xilinx IP Cores), the number of clock cycles per frame (NCCF) is given by equation (6.23).

$$NCCF = Mult + Add + Blk_{\text{read}} + 8.(Div + Sqrt) \quad (6.23)$$

Table 6.6 shows that computing the DSB solely does not meet the requirement for real-time performance as the computation throughput must be in the range [0..11.6 ms] as discussed in Chapter 4. When using the Novel algorithm proposed in this work, real-time operation is achieved for a 400 MHz clock with a NOSS = 256. Increasing the number of multipliers and adders working in parallel as suggested by the Amdahl approach presented in Chapter 2 is a Hardware contribution to improve performance. Using more multipliers and adders than the number of microphones would not accelerate the DSB and SRP in serial mode (See Figure 6.30 as a reference).
Table 6.6: DSB Throughput using the Hybrid Algorithm with a Hardware Contribution N=4

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speed</th>
<th>200 MHz</th>
<th>400 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSB (Algorithm) with 1 (multipliers, adders)</td>
<td></td>
<td>90 ms</td>
<td>45 ms</td>
</tr>
<tr>
<td>GCC-DSB (Algorithm) with 1 (multipliers, adders)</td>
<td></td>
<td>11.75 ms</td>
<td>5.9 ms</td>
</tr>
<tr>
<td>GCC-DSB Hardware (Contribution) with 2 (multipliers, adders)</td>
<td></td>
<td>5.9 ms</td>
<td>3 ms</td>
</tr>
<tr>
<td>GCC-DSB Hardware (Contribution) with 4 (multipliers, adders)</td>
<td></td>
<td>3 ms</td>
<td>1.5 ms</td>
</tr>
</tbody>
</table>

6.6. System Partitioning for a Reconfigurable Architecture

The new Hybrid Algorithm presented by the author could benefit from PR or MB mainly for logic reduction therefore power consumption. Prior knowledge can define the likelihood of the presence of one or multiple sound source at a certain time. In addition, the voice activity detector can be used to awake the system from its sleep mode when a voice is sensed. In that scenario the system will increase the number of microphones and corresponding logic to improve the localization of the sound source.

Depending of the end application which can be:

To localize the sound source, to listen to the speaker or voice recognition and with the knowledge of the number of speaker and the dominant frequency of their voice the DSB or MVDR can be selected for Beamforming.

Finally the prior knowledge of the motion speed of the speaker after the first detection can be used to change the detection from the GCC-DSB to the DSB-DSB as shown in Figure 4.20. That change will replace all the GCC by a Blank Bitstream which represents an important logic resource saving.

The main characteristic of the sound source localization case study compared to the capsule case study presented in Chapter 2 is speed. In Chapter 2 the slow system presented allowed the system to indiscriminately use the Multiboot (MB) or the Partial Reconfiguration (PR). In the sound source localization case study, the system throughput is around 11.6 ms which is 10% of the Virtex-5 or Virtex-6 board full configuration time approximately. Therefore applying a strict real-time constraint added to a reconfiguration approach is challenging due to the extra time necessary for system reconfiguration.
A workaround to this dilemma will be proposed, namely the partition of the system for a reconfigurable approach, with the presentation of candidate-modules for reconfiguration along with the events that trigger reconfiguration.

Potentially every module could be reconfigurable either because the functionality of that module needs to be replaced by another task or just by a blank bitstream to reduce power consumption. In this work a few modules can remain in the static region, among them are the FFT and IFFT used to compute the GCC presented in Figure 6.31, as a change in the number of microphones does not impact their functioning, but all the storage elements such as Block Ram need to be changed at run-time therefore put in the reconfigure region.

![Figure 6.31: Block Diagram from the FFT to the IFFT for GCC Computation](image)

The points represented in Figure 6.31as M or 1/M are details in Figure 6.24 and could remain in the static region of the FPGA. Another module that needs to stay in the static area due the PR flow restriction is the clocking system which includes the free running clock of the FPGA and the pulse enabling clock to synchronise data across clock domain shown in Figure 6.32. The multi-boot flow do not suffer from any restriction.
Figure 6.32: One Pulse Enable Clock Block Diagram

Figure 6.33 shows a very simplified version of the structure of the block diagram of the novel algorithm proposed in this work to better explain the reconfiguration strategy.

Figure 6.33: Simplified Block Diagram of the Full System to Explain Reconfiguration

The reconfiguration strategy in this work is mainly linked to the system application accuracy. The accuracy is improved with the number of sensors involved in the tracking of sound sources. Therefore a pre-study deciding when a higher number of microphones is necessary should be carried out. Basically the system will increase or reduce the number of microphone in the system depending on time. Figure 6.34 shows some of the modules that can be reconfigured and will be used as a base to elaborate the reconfiguration approach depending on the system requirement.
Figure 6.34 shows 4 reconfigurable regions in green. The acquisition chain partial reconfigurable module (ACCPRR1) can be used in conjunction with the computation chain module (CCPRR3) while the ACCPRR2 and CCPRR4 are filled with the blank bistream. If it becomes necessary to increase the number of microphone to 4 then both the ACCPRR1 and ACCPRR2 will be used in the acquisition chain with the CCPRR4 while CCPRR3 is filled with blank bitstream to disable it. The advantage of this approach is that during the configuration no data is lost as both systems can work in parallel before one or the other stops. The drawback of this approach is the loss of FPGA resources when one or the other configuration is chosen.

To workaround the loss of resources one idea is to merge ACCPRR1 with ACCPRR2 for the acquisition chain and CCPRR3 with CCPRR4 for the computation chain. The drawback of this approach is that during the reconfiguration a frame of around 100 ms could be lost which does not represent a problem for speaker localization. To work around that issue the CIC filters should be removed from the reconfigurable region and a huge reconfigurable Block Ram should be used to store data during reconfiguration.

The multi-boot (MB) equivalent of the Figure 6.34 is deployed in two FPGAs. The first FPGA contains the same logic as in Figure 6.34 minus the modules {ACCPRR3 and
CCPRR4} and the second FPGA is also the same as Figure 6.34 minus module { CCPRR4}. More details on the PR conversion to MB is presented in Chapter 2.

6.7. Brief Presentation of the System Verification

Figure 6.35 shows the block diagram used to validate the system signal integrity. It is composed of a Miniature Electro-Mechanical System Microphone (MEMS) [6.19] to capture the speaker voice and a Future Technology Device international (FTDI) [6.20] which is a USB to parallel FIFO interface to connect the FPGA to a personal computer (PC). A Field Programmable Gate-Arrays (FPGA) will be used to provide the required flexibility (scalability and data synchronization) for the implementation of the developed hybrid algorithm while the PC is useful to re-hear the speaker recording with software such as Adobe Edition.

The FTDI is only necessary to verify Figure 6.35 connection signal integrity. Therefore, the data rate through the USB has no impact on this work. Once the signal capture by the microphones can be replayed on the PC then the connection between the microphones and the FPGA is validated.

Sound source localization using GCC-DSB requires a lot of resources to store data that needs to be processed or is being processed. Therefore the Block RAM primitive needs to be tested to guarantee that all data written in the Block RAMs is not altered when read back. The strategy used to verify the FPGA storing hardware consists of adding some Chipscope Pro IP (Intellectual property ) cores as shown in Figure 6.36. Adding Chipscope Pro cores such as the VIO (virtual input/ouptut), ILA (Integrated Logic Analyzer) and ICON (Integrated
Control) whose functions are explained in a work published by the author [6.21] increases logic resource that might slow the system. The solution to that problem is shown in another paper published by the author [6.22]. It consists of bringing the logic resources closer depending on their interactions as shown in Figure 6.37 using the XILINX PlanAhead tool.

Figure 6.36: FPGA Block Ram Testing using Chipscope Pro Tools
6.8. Summary and Further Work

This chapter has shown a selection of an architecture with simple operations such as {addition, subtraction, and shift} to adapt the implementation of the algorithm to the FPGA structure. That explains the choice of the CIC compared to the FIR filters. It also presented its approach to multiple variables with constant (see Figure 6.12) and the impact on the throughput on the sequence of the algorithm execution (see Figure 6.25).

Finally, this chapter has proposed an architecture to implement a low latency based FPGA system for a flexible speaker localization algorithm that combines GCC and DSB. Although the timing constraint imposed by real-time application would make it challenging to use a reconfigurable flow to improve system flexibility, a work around has been proposed at the cost of logic duplication to implement both the PRR and MB flow. Figure 6.38 shows a possible improvement of Figure 6.34. It shows a CCPRR3 with a question mark. Reconfiguring CCPRR3 will require the use of a Partial Reconfiguration Block RAMs (BLK PR) which is used to store data while the GCC is being configured.
The MVDR (Minimum Variance Distortion-less Response)/ASR (Automatic signal Recognition) are two direct applications of this work. Once the speakers is localized, his conversation can be heard using the MVDR as presented in Chapter 5 or used for tracking a specific speaker amongst multiple speakers using automatic signal recognition which will be represented as an extension of this work in Chapter 7. Both application can time share the CCPP4 module.

The drawback of Partial Reconfiguration come from the fact that he reconfiguration of region can only be done sequentially. This drawback can also be solved by the multi-boot approach proposed in this work.

A parallel reconfiguration can be useful to solve this issue. The state of the art today requires partial reconfiguration to be done sequentially.
CHAPTER 7 Conclusions and Perspectives

7. DECLARATION AND PRESENTATION

I hereby declare that this chapter is entirely of my own doing and that it has not been submitted as an exercise for a degree at any other University.

7.1. Purpose of this Chapter

This chapter presents a summary of the achievements described in the thesis as well as the conclusions about the work presented. The main contributions of the work are summarised in turn and some possibilities for future work are outlined.

7.2. Conclusion

At the algorithm level, this thesis has presented a novel sound source localization algorithm and has investigated ranges of applications of speech processing technique acquired by microphone. This topic is immense and cannot be covered only by one research work, but this work represents a solid contribution to the field in the specialized area of speaker or sound source localization. In particular, at the algorithm level, this research has replied to the following questions:

1. Speech Detection: Is there someone speaking? This work replies to this question using a Voice Activity Detector (VAD), which has been covered in Chapter 6. The VAD is a widely used module in most of the applications dealing with voice. It can be computed in the time or frequency domains. In this research, the choice was made to compute the VAD in the acquisition chain in time domain while keeping the
computation chain of the system in sleep mode until a frame with a voice signal was detected. This methodology in addition to a strict bit growth control of the design net was applied to reduce power consumption in the design.

2. Speech Localization: Where is the person speaking? This work replies to this question using a Novel Hybrid Algorithm developed by the author that combines the GCC and DSB. This novel algorithm is the product of the following observation: The GCC is a very fast but inaccurate algorithm while the DSB is a very accurate algorithm but with a long computation time. The idea of this work was to accelerate the DSB by computing the GCC prior to the DSB. Then the next step was to create some drivers to transform the GCC result to be exploited by the DSB. This novel approach reduces the DSB computation by more than 80% when computed in sequential and serial operations mode. A higher percentage reduction can be achieved using a parallel and pipelined hardware approach if necessary. Further enhancements to the hardware architecture are also possible, but depend on the end application, as outlined in Section 7.2. In addition, it has been shown that this work could be extended to improve the performance of the MUSIC and MVDR algorithms. This work has been discussed in Chapter 4, Chapter 5, Chapter 6 and well-reviewed papers have been published to support the novel hybrid algorithm [7.1] [7.2].

3. Speech Beamforming: Can the detected voice be purified from any surrounding noise? This work replies to this question using two algorithms the DSB and the MVDR. Both algorithms can be used for sound source localization and Beamforming. For low frequency a signal MVDR performs better compared to the DSB therefore it will be preferred. For high frequency signals, MVDR has the same performance than the DSB therefore the DSB will be preferred overall as it requires less resource logic and it is faster compared to the MVDR. This work has been discussed in Chapter 5 and a paper has been published to support it [7.3].

4. Speaker recognition: What is the speaker’s name? (John, Lisa, etc) [7.4]. This section expands the novel hybrid algorithm to the tracking of a specific speaker among multiple speakers using voice recognition. As this approach does not seek to know what the speaker is saying but rather who is speaker, only half of the acquisition chain
data needs be used to achieve the goal of real-time tracking. The author has presented 4 different commonly used algorithms for voice feature extraction and justified the selection of MFCC as the one proposed. This work has been discussed in Chapter 7 and a paper to support this Chapter published in [7.5].

At the Hardware level, this research has explained how these algorithms could be implemented in an FPGA in order to achieve real-time localization performance at moderate or low levels of power consumption. The author has investigated a number of possible implementations of the algorithms and has found optimal implementation architectures. Furthermore, the author has applied these novel techniques to a number of other topical sound studies and has again shown how this algorithm can be implemented in FPGAs to enhance real-time performance in these domains. This work built upon the investigation done in Chapter 2 which has been published by the author in a series of papers [7.6][7.7][7.8].

At the flow level, in Chapter 2 the thesis has investigated issues relating to the reconfigurability of FPGA designs, and has concluded that the little used of MultiBoot technique offers real advantages over the more commonly used DPR method, especially when power consumption issues are important. A number of relevant publications were generated on the basis of this work [7.9][7.10]. Finally, in most cases, the author has shown that the speech localization novel algorithm may be implemented using MB techniques without loss of real-time performance but with reduced power consumption.

7.3. Perspective

Although the results achieve the objectives set at the beginning of this research, some extension still could be made. At the algorithm level, this research can be extended to the following questions.

At the hardware level:

1. Different configurations of microphone arrays could be investigated to improve the accuracy of the localization of the sound source. Some aspects of this work were presented in Chapter 3.
2. A number of applications can benefit from this work, such as hearing aids, video conferencing applications and telephony. Each chapter dealing with sound source localization has presented a number of new applications that could benefit from this work or from the algorithm developed by the author.


ALTERA Corporation “Stratix V Device Overview” SV51001, June 05, 2013

XILINX Corporation “Zynq-7000 All Programmable SOC” UG926 (v4.0) May 14, 2013.

XILINX Corporation “XC7Z045 All Programmable SOC” UG954 (v1.3) July 2013.


Stephanie Tap “BPI Fast Configuration and iMPACT Flash Programming with 7 Series FPGAs”

Xapp587 (v1.1) May 10, 2013
Xilinx Corporation : LogiCore IP AXI HWICAP (v2.02.a) DS 817 April 24, 2012  Access the 06/11/2013.

Accessed 11/11/2013

Nam Ma, Yinglong Xia, Viktor Prasanna, Task Parallelle Implementation of Belief Propagation in Factor Graphs”, Workshop on Parallel and Distributed Computing for Machine Learning and Inference Problems (ParLearning’12- in conjunction with IPDPS’12), May 2012


Fahad Qureshi, Syed Asad Alam and Oscar Gustafsson 4K-point FFT Algorithms based on optimized twiddle factor multiplication for FPGAs The Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsta), Shanghai, Sept, 22-24, 2010,


Ivan TASHEV “Sound Capture and Processing”


Dr Andrew Greensted “The Lab Book Pages “Online Collection of Electronic Information”

Alex Khenkin & Jerad Lewis “Recommendations for Mounting and Connecting the Analog Devices, Inc Bottom-Ported MEMS Microphones” AN-1003 APPLICATION NOTE ANALOG DEVICES.


Jerad Lewis “Microphone Specification Explained” AN-1112 APPLICATION NOTE ANALOG DEVICES


Hiroshi Umezu and Kenji Suyama “Multiple Sound Source Localization based on Local Existence Property of Speech Signal” World Academy of Science, Engineering and Technology 06 2011


Ivan Tashev, Sound Capture and Processing, Wiley, Ed., 2009

Dr John Mc Donough “Spoken Language System” Saarland University December 7, 2009
Christophe Ris “Développement d’un logiciel de localisation spatiale à l’aide d’un réseau de microphone linéaire” Mons Université Internal Document


Kevin D. Donohue “Audio Systems Array Processing Toolbox” Department of Electrical and Computer Engineering “Audio System Laboratory University of Kentucky website http://www.engr.uky.edu/~donohue/audio/Arrays/MAToolbox.htm


Carlos T and al “Evaluation of a MUSIC-based Real-time Sound Localization of Multiple Sound Sources in Real Noisy Environments” The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems October 11-15, 2009 St. Louis, USA


Vite-Frias Jose Alberto1 Romero-Troncoso Rene de Jesus2 Ordaz-Moreno Alejandro

[6.15] "VHDL Core for 1024-Point Radix-4 FFT Computation" Proceedings of the 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig 2005) 0-7695-2456-7/05-2005 IEEE.


Christian S Ibala, J. Vachaudez Carlos Valderrama “Combining Sound Source Tracking Algorithms Based on Microphone Array to Improve Real-Time Localization” Published in MIXDES LODZ Poland 24-29 May 2012.

Christian S Ibala, J. Vachaudez Carlos Valderrama “Novel interface drivers to combine real-time localization and tracking algorithms” Published in Applied Electronic 2012 Czech Republic 5-7 September 2012

Christian S Ibala, F. Bettens, Carlos Valderama “Faster Sound Source Localization and Tracking with Limited Computation Load”. International Symposium on signal, Image Video and communication (ISIVC2012) July 4-6, 2012 Valenciennes France


APPENDIX

Appendix A source code can be found at the link below:

http://www.engr.uky.edu/~donohue/audio/Arrays/MAToolbox.htm