# Exploiting Partial Reconfiguration through PCIe for a Microphone Array Network Emulator

Bruno da Silva, An Braeken and Abdellah Touhafi
Dept. of Industrial Sciences (INDI),
Vrije Universiteit Brussel (VUB),
Brussels, Belgium

Federico Dominguez
Escuela Superior Politecnica del Litoral (ESPOL),
Guayaquil, Ecuador

## ABSTRACT

The current Micro-Electro-Mechanical Systems (MEMS) technology enables the deployment of relatively low-cost wireless sensor networks composed of MEMS microphone arrays for accurate sound-source localization. However, the evaluation and the selection of the most accurate and power-efficient network's topology is not trivial when considering dynamic MEMS microphone arrays. Although software simulators are usually considered, they consist of high-computational intensive tasks, which require hours to days to be completed. In this paper, we present an FPGA-based platform to emulate a network of microphone arrays. Our platform provides a controlled simulated acoustic environment, able to evaluate the impact of different network configurations such as the number of microphones per array, the network's topology or the used detection method. Data fusion techniques, combining the data collected by each node, are used in this platform. The platform is designed to exploit the FPGA's partial reconfiguration feature to increase the flexibility of the network emulator as well as to increase performance thanks to the use of the PCI-express high-bandwidth interface. On the one hand, the network emulator presents a higher flexibility by partially reconfiguring the nodes' architecture in runtime. On the other hand, a set of strategies and heuristics to properly use partial reconfiguration allows the acceleration of the emulation by exploiting the execution parallelism. Several experiments are presented to demonstrate some of the capabilities of our platform and the benefits of using partial reconfiguration.

## 1 INTRODUCTION

Wireless sensor networks (WSN) composed of microphone arrays are becoming popular [1], [2] thanks to the relatively low cost of Micro-Electro-Mechanical Systems (MEMS) sensors. However, validation and verification of these networks, using simulations, are time consuming procedures. Furthermore, before the deployment of a WSN composed of microphone arrays, the network must be tested in adapted environments such as anechoic chambers to avoid undesired reflections, possible distortions or acoustic artifacts. Simulators offer usually a solution since they quickly provide information about the capabilities of a network. For instance, they can be used to explore the effects of different node's architectures, network topologies or network synchronization strategies. However, simulation processes are computationally intensive tasks which usually require hours or days to complete. Due to the inherent parallelism that microphone arrays present, we believe that FPGAs can accelerate the simulation of such type of networks. Here we present an extended version of the microphone array network emulator (*NE*) presented in [3], [4], which mimics the node's response, combines the response of the network's nodes and provides an estimation of the network's response under a certain acoustic scenario. Therefore, instead of a pure software-based *NE* like the one presented in [1], the proposed *NE* uses an FPGA to accelerate the node's computation by implementing exactly the same HDL code that is going to be deployed in the nodes of a real network. From one side, an improved version of the sound-source locator proposed in [5] and accelerated in [6] is used as nodes of the network. From the other side, the *NE* uses partial reconfiguration (*PR*) to adapt the network topology and the node's configuration to increase accuracy of the sound source location. As a result, the functionalities of our *NE* are distributed between the host and the FPGA, using a high-bandwidth PCI-express (*PCIe*) interface for the communication and *PR*.

This paper extends the work and results presented in [3], [4]. On the one hand, this paper presents a more detailed description of the *NE* platform, by providing low-level details of the node's architectures under evaluation and detailed use of *PR* through *PCIe*. On the other hand, the use of *PR* through *PCIe* is exploited to not only extend the capabilities of the *NE* but also used to accelerate the emulation of multiple network's topologies. The main contributions of this work can be summarized as follows:

- A fully detailed architecture description of an FPGA-based *NE*, providing low-level details of the platform and how the *PR* via *PCIe* is used to reconfigure the network's characteristics.
- Strategies and heuristics to exploit the use of *PR* through *PCIe* to further accelerate the computations on the FPGA.

This paper is organized as follows. The motivation of including *PR* as part of our system is done in Section 2. Section 3 presents related work. The description of the *NE* architecture, the data fusion technique and the *PR* is done in Section 4. How *PR* is used to expand the supported nodes' configurations of the *NE* and how to accelerate executions of the emulator by using *PR* is described in Section 5. In Section 6, the proposed *NE* is used to evaluate certain network's configurations. Finally, our conclusions are presented in Section 7.

## 2 WHY PARTIAL RECONFIGURATION?

*PR* is a unique feature of FPGAs which allows us to change the functionality of a part of the reconfigurable logic in runtime. This results not only in a better reuse of area but also in a potential increment of performance when properly applied. Streaming applications demanding multiple individual computations of similar

tasks but with different configurations are the ideal candidates. Furthermore, the execution of streaming applications on FPGAs can exploit parallelism by means of pipelining.

Reconfigurable resources are divided into static and dynamic parts when applying *PR*. The resources for the communication interfaces or for the *PR* control are usually in the static part. The dynamic part supports *PR* at runtime to allocate different mutually exclusive functionalities, known as reconfigurable modules (*RM*). The reserved logic resources for the dynamic part are denoted as reconfigurable partitions (*RP*). Thus, each *RP* is dimensioned to provide enough logic resources to support several *RMs*.

## 2.1 Beyond the Available Resources

*PR* allows a higher level of resource reuse because functionality can be multiplexed in time on the reconfigurable logic. Such feature allows the allocation of different tasks in the same *RP*. The only condition is that their associated *RMs* must be multiplexed in time by partially reconfiguring the *RP*.

The benefit of *PR* for an efficient resources' management has been exploited in different ways in recent years. For instance, the architecture presented in [7] supports 80 distinct hardware architectures, with different levels and precisions, of DCT computations. Other applications, instead, use *PR* to switch between applications' modes in order to reduce the consumed FPGA resources and the overall power consumption. An example is presented in [8], where different image processing operations are switched in runtime to more power-efficient modes. The runtime management of the FPGA's resources through *PR* allows self-adaptive or self-repairing systems such as the one presented in [9]. Further examples of how *PR* improves the resource utilization and increases the flexibility of the system are detailed in [10].

## 2.2 Performance Opportunities

*PR* has area and time cost. On the one hand, additional area is dedicated to support *PR*. On the other hand, the *PR* of a *RP* requires a certain amount of time, which is directly related to the size of the *RP* and the available *BW* to load the bitmap. Everything is slightly different when considering *PR* over *PCIe*. *PR* is usually exploited for small FPGAs or FPGA-based *SoC*. In such devices, the logic resources are very limited, which evidences the benefit of *PR* by multiplexing in time the available resources. An FPGA board with *PCIe* is typically a high-end FPGA offering a large amount of resources to support high-performance applications. Thus, the sizes of the *RPs* are usually larger to exploit the available resources and/or to allocate complex applications demanding many logic resources. As a consequence, the corresponding bitmaps of the *RMs* are bigger, consuming more external on-board memory. Fortunately, since *PR* over *PCIe* is supported, the bitmap files can be located on the host side and be loaded through *PCIe*. The use of a high-bandwidth interface such as *PCIe* not only allows the reduction of the *PR*'s area overhead but also provides new opportunities to the use of FPGAs as hardware accelerators. However, a proper placement and scheduling of the tasks to be executed on the FPGA is mandatory to compensate the remaining time overhead.

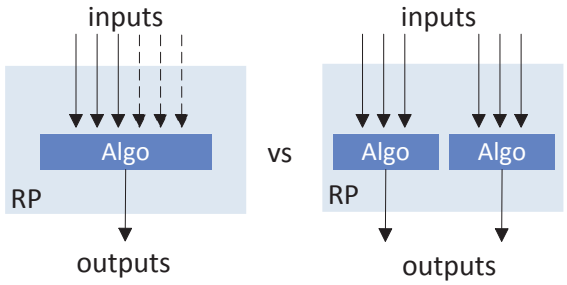



**Figure 1:** ***The use or PR allows us to exploit the available resources for different algorithm's configurations. For instance, when not all the input data need to be processed (dash arrows) the RP can be reconfigured to allocate more instances of the algorithm, doubling the throughput in this example.***

## 2.3 Our approach

*PR* offers several benefits in the context of a microphone array network emulator. As mentioned above, the reuse of the logic resources increases the flexibility of the *NE* by modifying the functionalities in runtime. For instance, *PR* allows the evaluation of different node's architectures in runtime as has been shown in [3], [4]. Thanks to *PR*, the area reuse not only increases the system's flexibility but can be also used to increase performance when exploiting the level of parallelism of the tasks of the *NE* accelerated on the FPGA. Figure 1 depicts the main idea, where a *RP* is configured to support a scalable algorithm, which processes several inputs in parallel (e.g. a convolutional filter with several kernel sizes). The performance doubles thanks to partially reconfiguring the *RP* to support two instances of the algorithm while consuming the same area. Of course, the overall performance by using *PR* through *PCIe* only increases if the time overhead is reduced. As extension of the platform presented in [3], we propose several heuristics to properly place and schedule the nodes' configurations. The overall result is a flexible platform optimized to achieve the highest area and performance efficiency.

# 3 RELATED WORK

In this section, we provide an overview of similar previous work, and explain the relations and differences compared to our work.

FPGAs have been already used as emulators for WSN. The authors in [11] propose an FPGA-based WSN emulator for the design, simulation and evaluation of WSNs. Similarly, the authors in [12] present an FPGA-based wide-band wireless channel emulator able to generate white Gaussian noise, multi-path and Doppler fading effects. Both works are complementary to ours as we focus on a detailed emulation of a network node while simplifying the wireless communication aspect. Furthermore, the *PR* of our emulator provides a higher dynamism and flexibility, which is not exploited in the mentioned emulators.

The use of *PR* has been thoroughly explored and proposed during the last decades. From one side, the use of *PR* to change the configuration of a node in a network has been already considered. For example, a LUT-based *PR* is proposed in [13] as part of an adaptive beamformer and in [5] to obtain a dynamic angular resolution of their acoustic beamforming. Our *NE*, however, considers the

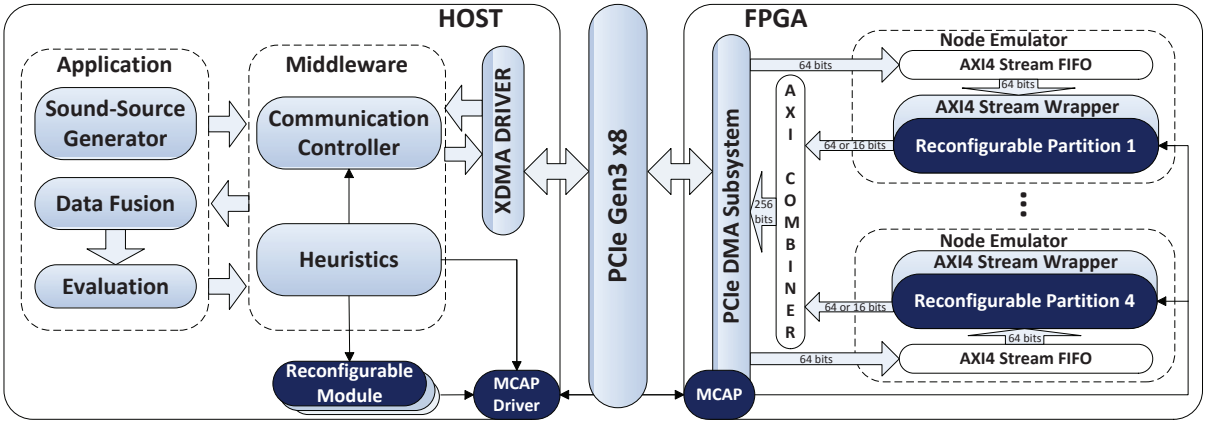



**Figure 2:** ***Distribution of the $NE$'s components. The $PR$ of the nodes and the data communication between the host and the FPGA are done via PCIe. A middleware abstracts the application from the heuristics for the merging and scheduling of the nodes' configurations, the host-FPGA communication and the $PR$ control. The dark blue boxes represent the components involved in the PR.***

complete reconfiguration of the node and not only a minor component. Thanks to this additional flexibility different architectures can be evaluated on the nodes.

From the other side, the use of *PR* induces certain area and time overhead. Specially critical is the time overhead, which must be overcome in order to achieve high performance. Several authors have proposed strategies to mitigate the impact of this overhead. The approach proposed in [14] minimizes the total reconfiguration time when distributing the tasks onto the target architecture, through a proper placement and scheduling. Despite the authors exploiting task's similarities, they do not use such characteristic to further exploit the *RP* resources. The optimal placement and scheduling is an NP-problem which is usually solved as an Integer-Linear Programming (*ILP*) problem. Thus, the authors in [15] present their *ILP* model together with an heuristic to exploit *PR* techniques such as *module reuse* to reduce the number of reconfigurations. However, their approach does not consider the use of *PR* to increment the resource sharing of the *RPs*. Our approach, instead, considers not only the *module reuse* during execution time but takes advantage of task's similarities to share logic resources of *RP*s. A more similar work to the one presented here is presented in [16]. The authors propose the resource sharing of *RPs* by merging tasks of streaming applications thanks to identifying similarities between tasks. Although our approach addresses similar applications, our strategy prioritizes the maximum area reuse of *RP*s while reducing the number of reconfigurations on a PCIe-based FPGA. Despite the fact that none of the mentioned works uses *PCIe*, *PR* through *PCIe* has been already targeted in [17] and [18]. Our proposed *NE* presents a more complex application which benefits from the current state-of-the-art technology [19]. As far as we are aware, the presented *NE* is one of the first applications using the recently introduced Xilinx MCAP [20] to partially reconfiguring the FPGA through PCIe.

## 4 NETWORK EMULATOR DESCRIPTION

The main purpose of the *NE* (Figure 2) is to mimic the functionality of a network composed of microphone array nodes and to evaluate the network's response for certain acoustic scenarios. This network increases the accuracy of the sound-source location by combining the response of each node. This information is used as an early estimation about how the network would react in real-world scenarios and allows a fast design space exploration in order to target priorities like overall power consumption or the accuracy of the sound-source localization. Our *NE* is flexible enough to support multiple network topologies, different sound-source detection methods or a variable number of nodes and sound sources.

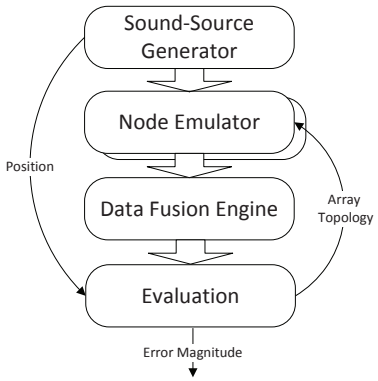
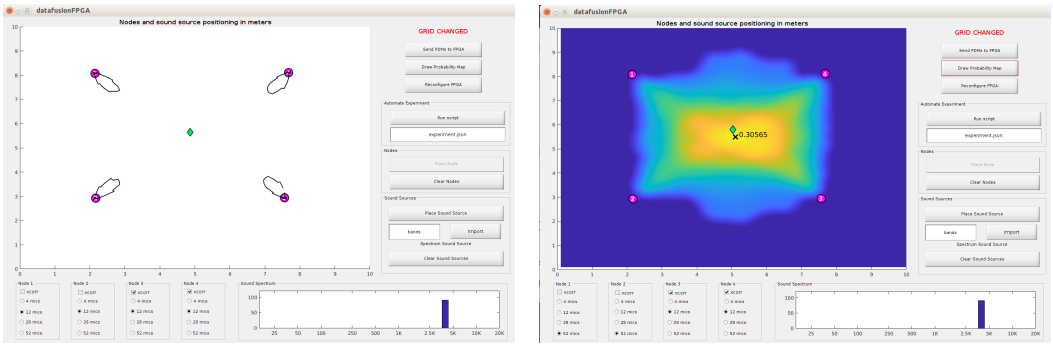


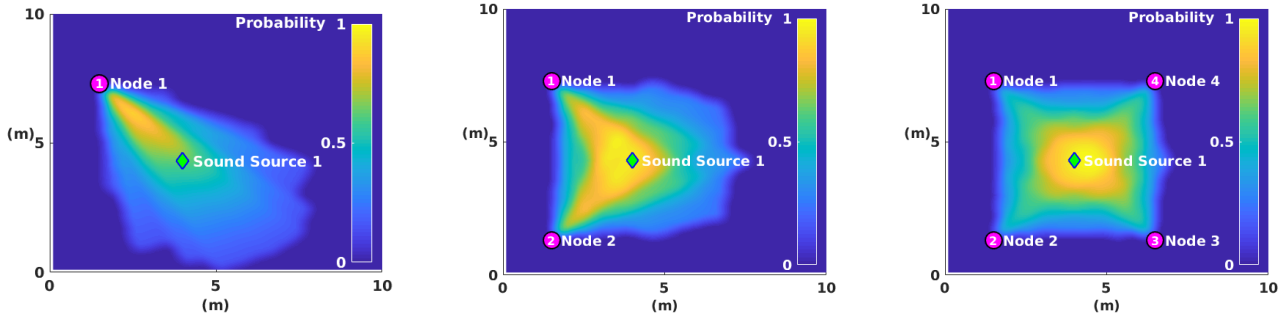

**Figure 3:** ***Execution steps of the $NE$. The sound sources are generated and processed by the nodes. The nodes of the network are reconfigured based on the error obtained after the evaluation of the data fusion.***

Figure 3 summarizes the execution steps of the *NE*. One or multiple sound sources are generated for a target scenario composed of a variable number of nodes. Each execution consists of several iterations to compute all the necessary nodes on the available *RPs*. The data collected from the nodes, the *polar steering response maps*, are fused and used to estimate the position of the sound sources. An evaluation of the error is done by considering the estimated position where the sound source is located and the known position of




(a) Fetching *polar steering response map* from the FPGA (b) Fusing the data to locate the sound source

**Figure 4:** ***The data fusion front-end is capable of simulating a sound field with multiple sound sources (green diamond) and multiple nodes (red circles). The front end generates PDM signals for each microphone in each node that are then sent to the FPGA back end. The FPGA generates the corresponding polar steering response map (a) which is then fed to the data fusion algorithm to generate a probability map (b) and estimate the localization error.***




(a) One node (b) Two nodes (c) Four nodes

**Figure 5:** ***Our data fusion technique combines the polar steering response map produced by each node to generate a probability map that estimates the location of the observed sound sources. As more nodes are used, the localization accuracy is improved. This technique has been adapted from [1].***

the sound source. Then, based on the target strategy under evaluation, the network is readjusted by partially reconfiguring the nodes with different configurations. The overall network power consumption or the accuracy of the sound source location are examples of potential strategies.

## 4.1 Distributed Functionality

The *NE* is built using the node's architecture described in the previous section. Thanks to the scalability and flexibility of the architecture, each node of the network can present a different configuration. The network is designed to preserve this flexibility in order to adapt its response for the variances in the acoustic environment. Therefore, the *NE* must support multiple node's possible configurations [6]. Some configurations are supported thanks to control signals, to disable certain microphones, or through *PR*, specially when evaluating the use of different architectures. Further details regarding the supported node's configurations are provided in Section 4.4.

Figure 2 depicts the main components of the *NE*, which are distributed between the host and the FPGA:

*4.1.1 Host.* The host contains the sound source generator, the data fusion of the polar maps and the evaluation of the data fusion. A graphical user interface (Figure 4) abstracts the user from these computations and from the host-FPGA communication and *PR*. The graphical user interface consists of a front-end generated in Matlab that communicates with the FPGA back-end through a *middleware*. The front-end is capable of simulating a sound field with multiple sound sources and nodes. Each sound source can have different frequency bands and each node can have different array configurations and calculation methods. Multiple sound sources are converted to PDM format in order to be compatible with the expected input data format of the node. The front-end is also capable of generating probability maps with the *polar steering response map* produced by the nodes on the FPGA.

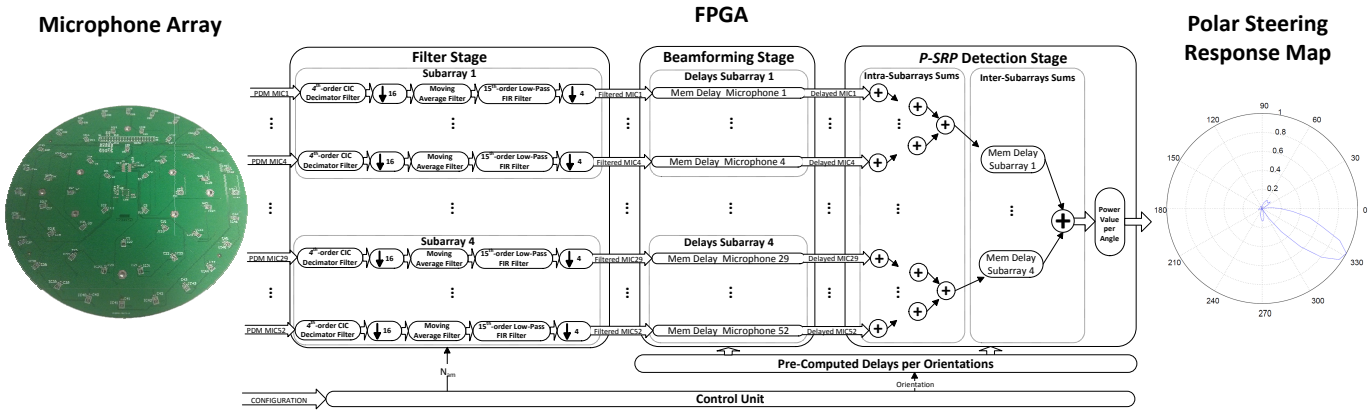



**Figure 6:** ***Node's design emulated in the $NE$ using P-SRP detection method. Each node is composed of a MEMS microphone array, a filter stage, a beamforming stage and a detection stage.***

The front-end uses data fusion to locate sound sources. Data fusion techniques combine the information gathered by different sensors measuring the same process to enhance the understanding of that process. In the context of this article, data fusion is performed by aggregating and combining the acoustic directivity information, represented as a *polar steering response map*, gathered by each node to produce a probability map of the location of the observed sound sources in a two-dimensional field. This technique is originally presented in [1] and has been used to validate the capacity of their microphone array design to locate sound sources (Figure 5).

Based on the data fusion results, the front-end calculates three error parameters: the localization error (in meters), the number of undetected sound sources and the number of phantom sound sources.

The front-end allows us to create a network of nodes and to validate our architecture with a permutation of different scenarios: array architecture, detection method, sound spectrum, and sound-source positions.

Finally, a *middleware* abstracts the application from the acceleration on the FPGA by managing the nodes' configurations, the FPGA's *PR* and the host-FPGA communication. Further details about the heuristics for the placement and schedule of the nodes are detailed in Section 5.

*4.1.2 PCIe Communication.* The communication between the host and the FPGA uses the Xilinx PCIe DMA driver available in [21]. This driver enables the interaction of the software running on the host with the DMA endpoint IP via PCIe.

*4.1.3 FPGA.* On the FPGA side, the *NE* uses the IP core DMA subsystem for PCIe [22]. This IP core provides support for different types of reconfiguration through PCIe, such as Tandem, Tandem with Field Updates and PR. In our case, the IP core DMA subsystem for PCIe with PR support is used. The DMA capability of this core is configured to act like an AXI4 bridge, operating at 125 MHz and with an AXI4 stream interface of 256 bitwidth. The HDL code of each node is encapsulated in an AXI4-Stream Wrapper in order to be compatible with AXI4-stream. This AXI4-Stream Wrapper

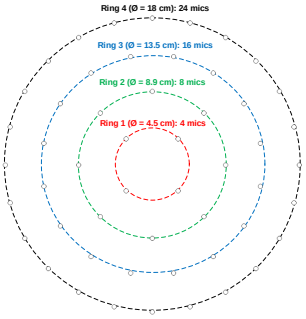



**Figure 7:** ***Sound-source localization device composed of 4 digital MEMS microphone subarrays.***

interfaces an input AXI4-Stream FIFO, both integrated in a Node Emulator entity. The *NE* is composed of 4 Node Emulators operating at 62.5 MHz and with a 64-bits AXI4-Stream interface each. Finally, the output AXI4-streams of the Node Emulators are combined in a 256-bits AXI4-Stream to interface the PCIe DMA Subsystem IP core.

## 4.2 Node Description

The original architecture proposed in [5] has been improved in [6] by rearranging the detection method (*DM*) in a modular fashion and by reducing the control management. The filter stage has been also modified to operate uninterruptedly during the beamed orientation transition. The implementation of the node's architecture on the FPGA (Figure 6) is done in VHDL and designed to process in stream fashion. Moreover, the nodes of the *NE* are composed of several cascaded stages operating in pipeline for a fast sound-source location.

*4.2.1 Microphone Array.* The audio data is acquired by the microphone arrays and expressed as a multiplexed pulse density modulation (PDM). The microphone array is composed of four concentric subarrays of 4, 8, 16 and 24 digital MEMS microphones [23]

mounted on a 20-cm circular printed board (Figure 7). Each subarray is dynamically activated or deactivated in order to facilitate the capture of spatial acoustic information using a beamforming technique. The distributed geometry of the subarrays allows the adaptation of the sensor to different sound sources. Therefore, the computational demand is adapted to the surrounding acoustic field, making the sensor array more power efficient if only a necessary number of subarrays are active. The emulation of the microphone array is partially done at the host side by the sound generator. The sound wave corresponding to the sound sources are generated in a PDM format for each microphone based on the node to be emulated and the position of the microphone in the node. The frequency band of the audio sources ranges from 100 Hz up to 15 kHz. In order to respect the technical specifications of the ADMP521 MEMS microphones, the generated audio signals are oversampled at 2 MHz.

*4.2.2 Filter Stage.* The single-bit PDM signal needs to be filtered to remove the high-frequency noise and to be downsampled to retrieve the audio signal in a Pulse-Code Modulation (PCM) format. The removal of the undesired high frequency noise and the downsampling is done at the filter stage. Thus, each microphone signal has one cascade of filters to downsample and to remove the high-frequency noise. The first filter is a $4^{th}$ order low pass Cascaded Integrated-Comb (CIC) decimator filter with a decimation factor of 16. This type of filter only involves additions and subtractions [24], which significantly reduces the resource consumption. The CIC filter is followed by a 32-bits running average block to reduce the microphone DC offset and by a 15th order serial low-pass FIR filter with a cut-off frequency of 12 kHz and a decimation factor of 4 completes the filter chain. The serial design of the FIR filter drastically reduces the resource consumption, but limits the maximum order of the filter, which must be equal to the decimation factor of the CIC filter. The data representation is a signed 32-bits fixed point representation with 16 bits as fractional part. The filter's coefficients are represented with 16 bits. However, the bitwidth is higher in the filter to keep the proper data resolution due to some internal filter operations, but the inter-filter data representation is set to constant by applying the proper adjustment at the output of each filter.

*4.2.3 Beamforming Stage.* Beamforming techniques focus the array to a specific orientation by amplifying the sound coming from the pre-defined direction, while suppressing the sound coming from other directions. Therefore, the directional variations of the surrounding sound field are measured by continuously steering the focus direction in a 360° sweep. Our Delay-and-Sum based beamformer is applied to 64 orientations, which represents an angular resolution of 5.625 degrees.

The filtered signal of each microphone is delayed by a specific amount of time determined by the focus direction, the position vector of the microphone and the speed of sound. All possible delays are precomputed, grouped based on the supported beamed orientations and stored in block RAMs (BRAM) during the compilation time. Therefore, the 32-bits filtered audio of each microphone is delayed based on the precomputed values and grouped following their subarray structure to support a variable number of active microphones. Thus, instead of implementing one simple beamforming operation of 52 microphones, there are four beamforming operations in parallel for the 4, 8, 16 and 24 microphones. Only the beamforming block linked to an active subarray is enabled, while the disabled beamformers are set to zero.

*4.2.4 Detection Stage.* The *polar steering response maps* are obtained at this stage. The output data is normalized based on the maximum output value for each complete loop. The normalized outputs need to be represented with at least 16 bits to avoid errors due to the data representation.

We distinguish here 2 different *DMs*. Both methods, already proposed in [3], can be available by partially reconfiguring the node's architecture based on the active subarrays:

*Polar Steered Response Power:* The original architecture in [1] proposed the Delay-and-Sum beamforming technique. This technique uses the added beamformed values to calculate the output power of the signal per orientation in the time domain. The computation of this output power for different beamed orientations defines the *Polar Steered Response Power* (*P-SRP*). The *P-SRP* informs in which direction the sound-source is located since the maximum power is obtained when the focus corresponds to the location of a sound source.

*Cross Correlation:* The cross-correlation (*CC*) method is based on the cross-correlated pairs of microphones. Thus, the time-differences-of-arrival (*TDoA*) is the lag associated with the maximum measured correlation. The *P-SRP* method is more robust to reverberation and noise effects [25] since it considers all available information. Nevertheless, we propose the alternative implementation of *CC* where all the global information is used and the difference between beamed orientations is amplified. Once the audio is beamformed, the audio data of all microphones are cross-correlated with each other and accumulated. Thus, once the audio data is properly delayed, the maximum of the positive values determines the location of a sound source. This *CC* method, however, demands a high number of multiplications because all possible pairs of microphones need to be correlated. The total number of multiplications ($M_{am}$) depends on the number of active microphones ($N_{am}$) and is expressed as follows:

$$M_{am} = \frac{N_{am} \cdot (N_{am} - 1)}{2} \quad (1)$$

Unfortunately, $M_{am}$ drastically increases when increasing the number of active microphones. For instance, $M_{am} = 6$ when only the inner subarray, composed of 4 microphones, is active. Furthermore, $M_{am}$ increases from 66 to 1326 when activating the first two inner subarrays (12 microphones) or all subarrays (52 microphones) respectively. This fact has a significant impact in the resource consumption, since not only a large number of DSPs are consumed but also the LUTs used to keep the fixed point precision. Each multiplication extends the 32-bits fixed point data representation to 64 bits, which is adjusted to 32 bits again before the next multiplication in the multiplication tree [3].

The *CC* method promises better accuracy when using a lower number of microphones. The theoretical implementation needs 66 multiplications in order to reach all possible combinations of the 12 active microphones. However, in order to save resources

| Perspective | Evaluation |
|---|---|
| Node (FPGA) | Detection Method |
| | Number of active microphones |
| | Number of orientations |
| | Sensing time |
| Network (Host) | Data Fusion techniques |
| | Power efficiency topology |
| | Data desynchronization |

**Table 1:** ***The proposed platform enables the exploration of multiple node's and network's configurations.***

| Parameter | Definition | Range |
|---|---|---|
| $DM$ | Detection Method | [P-SRP, CC] |
| $N_{am}$ | Number of active microphones | [4, 12, 28, 52] |

**Table 2:** ***The experimental results detailed in Section 6 are obtained by evaluating different node's configurations.***

while maintaining the maximum flexibility, the implementation under evaluation only considers the combinations between the microphones of a subarray and not the combinations between the microphones of different subarrays. Therefore, the number of multiplications is reduced to 32, with 6 and 28 multiplications for the 4 microphones of the inner subarray and the 8 microphones of the second subarray respectively. Because this modular *CC* promises higher accuracy, it is an interesting candidate to replace the *P-SRP* method when a low number of microphones is active. The analysed *CC* method in our experiments only considers the use of the two inner subarrays.

### 4.3 Accuracy

The effective dynamic range of the floating point data representation provides a high accuracy at the cost of a high resource consumption and a performance cost, which discourages the use of floating point operations in the node's architecture. The alternative fixed point data representation, however, induces undesired errors [26], [27]. The most sensitive blocks of the node's architecture are located in the filter and the detection stages. A variable fixed point representation is applied at each node's stage to minimize the errors induced by this type of data representation. The internal operations in the filters are scaled in order to provide enough bits for the data representation. However, in order to reduce the overall resource consumption, the output of each block is rescaled to signed 32-bits fixed point representation with 16 bits of fractional part. The evaluation of the impact in the accuracy of the node's response has been performed for each supported frequency by comparing the results with our reference model programmed in Matlab which mimics the node's architecture and is already used in [3], [4], [5] and [6]. As a result, the fixed point data representation at each stage of the node's architecture guarantees an average relative error of $2.42 \times 10^{-5}$, compared to floating-point data representation.

### 4.4 Design Space Exploration

Table 1 summarizes the most relevant parameters that can be evaluated in our platform. Some parameters are related to the node's architecture while others are relevant at the network perspective. Although the node's parameters like the impact of the number of orientations or the sensing time have been discussed in [6] from the performance point of view, the *NE* allows the evaluation of the network's configurations. For instance, different data fusion techniques or network topologies for a lower network's power consumption can be evaluated. Moreover, the error induced by the data desynchronization from the nodes can be estimated. Notice that, due to the distribution of the functionality, the node's parameters affect to the FPGA design while the network's parameters affect to the code running on the host. Therefore, in order to focus on how *PR* can be exploited by our platform, only a couple of node's parameters are considered for the design space exploration.

The node's architecture permits the modification of $N_{am}$ in runtime (Figure 6) to adapt the node's response to the dynamic behavior of acoustic environments. However, $N_{am}$ has a significant impact in the area consumption and in the node's power consumption as detailed in [6]. This fact makes $N_{am}$ an interesting parameter to be evaluated from a network point of view because while a lower $N_{am}$ leads to a lower node's power consumption, a lower accuracy is the price to pay. Further details about the node's architecture, demanded hardware resources and the impact of the supported configurations are detailed in [6].

Although different node's configurations are supported in runtime, this does not occur with the *DMs*, since only one of the proposed *DMs* can be allocated on the FPGA. As a consequence, the switch between the proposed *DMs* is only possible when using *PR*. Nevertheless, the evaluation of the *DMs* is fully supported in our *NE* for the two inner subarrays. Both parameters are used to evaluate networks composed of nodes with variable values of $N_{am}$ and *DMs* (Table 2). The experimental results are presented in Section 6.

## 5 STRATEGIES FOR EXPLOITING PARTIAL RECONFIGURATION

The *middleware*, located in the host side, is not only in charge of the host-FPGA communication through *PCIe* but also to control and to optimize the *PR*. This layer abstracts the front-end from the back-end configuration. While the user only needs to configure the topology of the network, the nodes' configuration and the sound sources, the *middleware* optimizes the execution of the *NE* by merging and scheduling the nodes in the available *PRs*. Thus, the user does not need any knowledge about the *RPs* configurations or in what *RP* a particular node is emulated.

One execution of the *NE* consists of the emulation of a certain number of nodes under an acoustic context. The *middleware* distributes the nodes between the available *RPs*. Several iterations are needed in case the number of nodes to be emulated ($n_{node}$) is higher than the number of *RPs* ($n_{RP}$). Thus, the number of iterations ($n_{iter}$) is defined as:

$$n_{iter} = \left\lceil \frac{n_{node}}{n_{RP}} \right\rceil \quad (2)$$

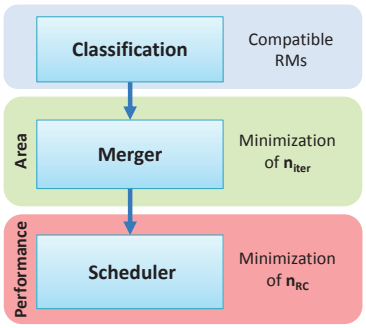



**Figure 8:** ***Our approach consists of several heuristics to optimize the area reuse and to improve performance by reducing*** *PR*. ***Firstly, the nodes' configurations are classified and sorted based on their compatible*** *RMs*. ***Secondly, the increment of the area reuse is possible by merging similar nodes' configurations to increase the overall*** *LP*. ***Finally, an optimized scheduler minimizes the*** *PR* ***by properly allocating the nodes.***

Thanks to the independence of the nodes, there are no data dependencies. This fact simplifies the *middleware* decisions. Thus, the *middleware* uses some heuristics to optimize the merging of compatible node's configurations in order to exploit the available resources in one *RP*, and also optimizes the scheduling of the computation of the merged node's configurations. As a result, the *middleware* not only allows the abstraction of the user about of the *NE* internal configuration but also to exploit *PR* for increasing performance.

## 5.1 Increasing Network Capabilities

The dynamism required to reduce the estimation error under unpredictable acoustic scenarios is enhanced thanks to *PR*. A clear example occurs when minimizing the overall network power consumption while offering accurate sound sources location. The overall power consumption and the accuracy are directly related to the $N_{am}$. Thus, a trade-off is needed in order to get the highest accurate estimation with the minimum power consumption. *PR* has a role when considering alternative architectures to enhance the quality of results. That is the case of the *CC* method, which is only applicable for a lower number of microphones where it promises better accuracy. In that case, *PR* allows the dynamic modification of the network configuration in runtime to satisfy power constraints. Such evaluation in the *NE* would not be possible without *PR*. Otherwise, the platform had to be completely reprogrammed and a reboot would be needed in order to let the operating system identify the reconfigured *PCIe* device. Our experimental results in Section 6 cover this example.

## 5.2 Heuristics to Increase Performance

Figure 8 shows the heuristics used by the *middleware* to increase performance. *PR* can only be exploited for higher performance by properly placing and scheduling the nodes to be accelerated. Furthermore, we propose the use of *PR* to exploit the configurations' compatibilities in order to better use the available resources.

An existing cost table (*CT*), like Table 3, is used to decide at each step of the heuristics. Such table is elaborated at design time and contains information about the relative area cost of the configurations, related to the most area demanding configuration, and information about configurations' compatibilities. From one side, the relative area cost reflects the configuration's level of parallelism (*LP*), which is used to merge compatible node's configurations. In fact, *LP* is the inverse of the relative area cost since it represents the number of nodes with a certain configuration that can be executed in parallel per *RP*. From the other side, the configurations' compatibilities relate a certain configuration with its supported *RMs*. Thanks to the flexible architecture of the nodes, the number of active microphones varies from 4 to 52 MICs. Their activation is in runtime through control signals and does not require any type of *PR*. As a consequence, certain *RMs* support multiple node's configurations depending on the dedicated logic resources. Thus, the $RM_{52}$ not only supports 52 microphones but also 28, 12 or 4. Such flexibility is used to reduce the number of *PR* in order to achieve higher performance. A summary of the abbreviations used for the *NE* description is done in Table 4.

| Configuration | Time Cost | Area Cost | Compatibility |
|---|---|---|---|
| 52 Mics | $1.0834 \pm 0.0029$ | 1 | $RM_{52}$ |
| 28 Mics | $1.0753 \pm 0.0024$ | 1/2 | $RM_{52}$, $RM_{28}$ |
| 12 Mics | $1.0679 \pm 0.0023$ | 1/4 | $RM_{52}$, $RM_{28}$, $RM_{12}$ |
| 4 Mics | $1.0677 \pm 0.0023$ | 1/4 | $RM_{52}$, $RM_{28}$, $RM_{12}$ |

**Table 3:** ***Cost table of the node's configurations for the second experiment with the NE. The time values are expressed in seconds.***

| Parameter | Definition |
|---|---|
| *RP* | Reconfigurable Partition |
| *RM* | Reconfigurable Module |
| *CT* | Cost table |
| *LP* | Level of parallelism. Inverse of the area cost |
| *P-SRP* | Polar steered response power |
| *CC* | Cross-Correlation |
| *DM* | Detection Method |
| $N_{am}$ | Number of active microphones |
| $M_{am}$ | Number of multiplications |
| $n_{node}$ | Number of nodes' configurations per execution |
| $n_{iter}$ | Number of iterations needed by one execution |
| $n_{RP}$ | Number of *RPs* |
| $n_{RC}$ | Number of *PR* |
| $N$ | Initial nodes' configuration list |
| $N_I$ | Sorted and classified nodes' configuration list |
| $N_M$ | Merged nodes' configuration list |
| $N_T$ | Temporal nodes' configuration list |
| $N_S$ | Scheduled nodes' configuration list |
| $S_{temp}$ | Temporal set of *RP*'s configurations |
| $S_{node}[i]$ | Set of *RP*'s configurations on iteration $i$ |

**Table 4:** ***Abbreviations used for the description of the NE and the presented heuristics.***

*5.2.1 Classification.* The nodes are classified based on an existing *CT* like Table 3. This classification identifies the compatible *RMs* and the *LP* that can be achieved based on their type. Algorithm 1

**Algorithm 1:** *Classification of nodes.*

```
input  : Nodes' configuration list N, CT
output : Sorted and classified node list N_I
1  begin
2      N_I ← Sort nodes based on their area cost (N, CT);
3      N_I ← Find Compatible RM (N_I, CT);
4  end
```

details the operations involved during the nodes' classification. $N$ is composed of multiple nodes' configurations, which can be optimally parallelized and scheduled to minimize the execution time. The relative area cost of each node is used as reference to sort the nodes' list in decreasing order. Lately, the nodes' list is evaluated to identify the compatible *RMs* per configuration.

**Algorithm 2:** *Merging of the nodes' configurations.*

```
input  : Nodes' configuration list N_I
output : Merged configuration list N_M
1  begin
2      N_M ← ∅;
3      N_T ← ∅;
4      for i ∈ N_I do
5          if AreaCost(i) = 1 then
6              N_T ← config(i);
7              AccAreaCost(N_T) ← AreaCost(config(i));
8              CompatibleRMs(N_T)← SmallestRM(N_T,config(i));
9              N_M ← InsertIn(N_M,N_T);
10         end
11         else
12             if AreaCost(i)+AccAreaCost(N_T)> 1 then
13                 N_M ← InsertIn(N_M,N_T);
14                 N_T ← config(i);
15                 AccAreaCost(N_T)← AreaCost(config(i));
16                 CompatibleRMs(N_T)← SmallestRM(N_T,config(i));
17             end
18             else
19                 N_T ← InsertIn(N_T,config(i));
20                 AccAreaCost(N_T)←AccAreaCost(N_T) +
                     max(AreaCost(config(i)),AccAreaCost(N_T));
21             end
22         end
23     end
24 end
```

*5.2.2 Merging.* The merging of the nodes' configurations consists of grouping the maximum number of compatible configurations in one *RP* in order to exploit the otherwise unused resources. For instance, in case $N_I = [52, 52, 52, 28, 28, 12, 12, 12, 12]$ the nodes with $N_{am} = 28$ and $N_{am} = 12$ can be computed in parallel in *RPs* configured with $RM_{28}$ and $RM_{12}$ respectively (Table 3). Since the *RMs* are associated to the nodes' configurations after the merging heuristic, $N_M$ for the previous example results in $N_M = [RM_{52}, RM_{52}, RM_{52}, RM_{28}, RM_{12}]$. Notice that $n_{iter}$ is reduced in one unit thanks to this merging (Eq. 2). In fact, the reduction of $n_{iter}$ is the main objective of this heuristic.

Algorithm 2 shows how nodes are merged based on their *LP* to place in each *RP* as many nodes as possible. This merging intends to reduce $n_{iter}$ and, potentially, the overall execution time. Algorithm 2 starts scanning the list of configurations in increasing relative area cost order. There are three possibilities:

- If the configuration consumes a complete *RP*, which occurs when its cost equals 1, the configuration cannot share any *RP*. Thus, this configuration is allocated to the largest compatible *RM* in order to maximise the reuse of this *RP*. Otherwise, if the demanded relative area cost of the configuration is lower, it can be evaluated for sharing a *RP*.
- In case the addition of the configuration's cost and the accumulated area cost of the already allocated nodes is higher than one *RP*, this *RP* is locked. Firstly, the grouped configurations are moved to the configurations' list $N_M$ since they cannot longer share the resources of one *RP*. Secondly, the new unassigned node's configuration is assigned to the smallest compatible *RM* to limit the sharing to the most constrained situation.
- In case area cost of the configuration allows the addition of this node's configuration to the existing configurations' group, the accumulated area cost (which represents the percentage of occupancy of the *RP*) is incremented by the maximum area cost of new node's configuration. In this way, the area cost of the largest node's configuration dominates and thus unfeasible situations are avoided.

As a result, the configurations are categorized in the compatible *RMs* which maximize the area reuse and potentially increment the overall performance by decreasing $n_{iter}$.

**Algorithm 3:** *Scheduling of the merged nodes.*

```
input  : Merged configuration list N_M
output : Scheduled configuration set S_node
1  begin
2      N_S ← ∅ ;
3      S_temp ← ∅ ;
4      S_node[0] ←InitialRPsConfig;
5      for i ∈ n_iter do
6          S_temp ← S_node[i − 1];
7          for j ∈ n_RP do
8              for k ∈Size(N_M) do
9                  if Config(S_temp(j)) = Config(N_M(k)) then
10                     S_node[i] ←InsertConfigInRP(N_M(k),j);
11                     S_node[i] ←MarkAsConfigured();
12                     N_M ← RemoveConfigfromList(N_M, k);
13                     break;
14                 end
15             end
16         end
17         if NofElements(S_node[i]) < n_RP then
18             for j ∈ NotConfigured(S_node[i]) do
19                 if NofElements(N_M) > 0 then
20                     H_M ← CalcHistogram(N_M) ;
21                     idx_node ← FindMostFreqConfig (H_M) ;
22                     S_node[i] ←InsertConfigInRP(N_M(idx_node),j);
23                     S_node[i] ←MarkAsConfigured();
24                     N_M ← RemoveConfigfromList(N_M, idx_node);
25                 end
26                 else
27                     S_node[i] ←Config(S_node[i − 1]);
28                     S_node[i] ←MarkAsConfigured();
29                 end
30             end
31         end
32     end
33 end
```

*5.2.3 Scheduling.* Once the nodes have been merged, they need to be properly scheduled in order to minimize the number of reconfigurations ($n_{RC}$). The strategy consists in maximizing the reuse of the *RP's* previous configurations between iterations of one execution.

Algorithm 3 details how the merged configurations in $N_M$ are scheduled based on the configuration of the *RPs* in each iteration.

| Resources | Available | Static | RP 0 | RP 1 | RP 2 | RP 3 |
|---|---|---|---|---|---|---|
| **Slice Registers** | 663360 | 18413 | 93600 (51.42%) | 102400 (46.97%) | 103200 (46.64%) | 102400 (47.00%) |
| **Slice LUTs** | 331680 | 16209 | 46800 (78.81%) | 51200 (72.05%) | 51600 (71.50%) | 51200 (72.07%) |
| **LUT-FF Pairs** | 331680 | 7026 | 46800 (50.97%) | 51200 (47.39%) | 51600 (46.01%) | 51200 (47.24%) |
| **BRAM** | 1080 | 47 | 170 (28.24%) | 170 (28.24%) | 170 (28.24%) | 170 (28.24%) |
| **DSPs** | 2760 | 0 | 460 (24.35%) | 460 (24.35%) | 460 (24.35%) | 460 (24.35%) |
| **Bitmap Size** | - | - | 4,762 MB | 4,762 MB | 4,764 MB | 4,762 MB |
| **Clearing Time** | | | 0.0826± 0.0047s | 0.0836 ±0.0044s | 0.0848 ± 0.0031s | 0.0836 ± 0.0040s |
| **Reconfig. Time** | | | 1.0908± 0.0042s | 1.0988± 0.0056s | 1.0947± 0.0050s | 1.0945± 0.0054s |

**Table 5:** ***Resource consumption of the static and dimensions of the $RPs$, including their highest percentage of occupancy.***

The $RP's$ configuration of the previous iteration is used as initial $RP's$ configuration of the iteration under scheduling. Thus, the list $N_M$ is traverse looking for the same $RM$ loaded in the target $RP$. If found, the node's configuration is assigned to that $RP$ at that particular iteration. The process continues for the next $RP$ until all the available $n_{RP}$ are assigned. If there is no compatible node's configuration with the available $RPs$ at a certain iteration, it could be possible that either all the nodes have been allocated or $PR$ is needed. Based on the number of unallocated nodes, it is possible to distinguish how to proceed. On the one hand, if $PR$ is needed, the most frequent configuration of the unallocated nodes is selected. This configuration is obtained through the calculation of a histogram and maximizes the potential reuse of this configuration over the remaining iterations. On the other hand, the remaining unassigned $RPs$ keep their configuration from the previous iteration to avoid additional and unnecessary $PR$ if there are no more unallocated nodes. The process continues this way until no compatible tasks are available. The $PR$ of some $RPs$ is then mandatory to compute the remaining tasks. Finally, it might be possible that the computation of some $RPs$ is not required. This occurs when $n_{RP}/n_{node}$ is not an integer number. In that case, the $RPs$ maintain their configuration from the previous iteration to avoid additional and unnecessary $PR$.

For the sake of understanding, the scheduling heuristic is applied to the previous example. We assume some initial $RPs'$ conditions and the previous values of $N_M$:

$$InitalRPsConfig = [RM_{28}, RM_{52}, RM_{12}, RM_{52}]$$
$$N_M = [RM_{52}, RM_{52}, RM_{52}, RM_{28}, RM_{12}]$$

The scheduling heuristic distributes the elements of $N_M$ between the required $n_{iter}$ based on the previous iterations $RPs'$ configurations. Therefore, the execution order to minimize $n_{RC}$ results as follows:

$$S_{node}[1] = RM_{28},\ RM_{52},\ RM_{12},\ RM_{52}$$
$$S_{node}[2] = -,\ RM_{52},\ -,\ -$$

where – represents an unused $RP$ in one particular iteration. Thanks to both heuristics, $n_{iter}$ has been reduced to 2 iterations, multiple configurations are computed in parallel and there is no need for $PR$.

## 5.3 Partial Reconfiguration over PCIe

Despite the minimization of $PR$ between iterations due to our scheduler, $PR$ might be unavoidable due to the initial configuration of the $RPs$ and the list of nodes to be executed. Our $PR$ uses the Media


**Figure 9:** ***FPGA floorplanning when all nodes have their subarrays active. The four reconfigurable partitions of the $NE$ are framed into purple boxes.***

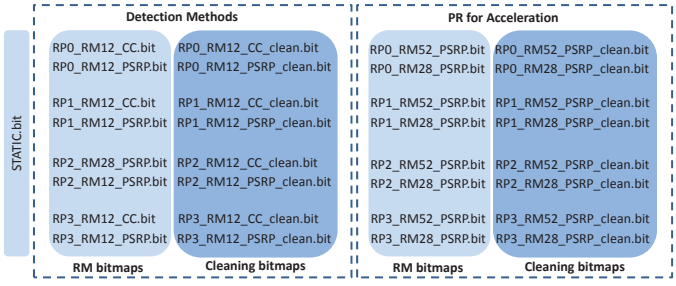



**Figure 10:** ***List of bitmap files required for the experiments evaluating $DM$ and the use of $PR$ for acceleration detailed in Section 6.***

Configuration Access Port (MCAP) [19], which is a new configuration interface available for UltraScale devices that provides a dedicated connection to the $ICAP$ from one specific $PCIe$ block per device. This interface is integrated into the $PCIe$ hard block and provides access to the FPGA configuration logic through the $PCIe$ hard block when enabled. The bitstream loading across the $PCIe$ to

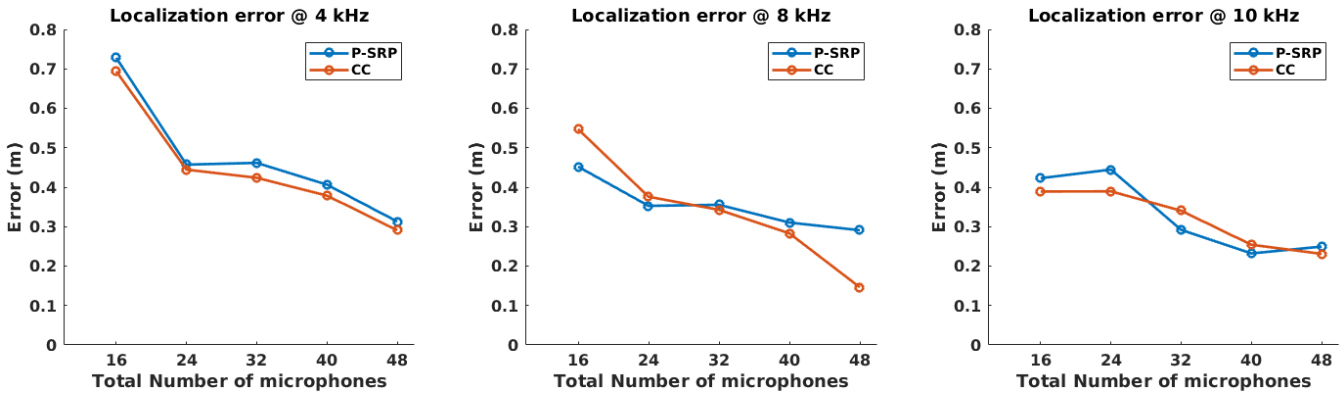



**Figure 11:** ***The localization error, using one sound source and four nodes at three different frequencies, improved as the total number of microphones in the network increased. Both methods, P-SRP and CC, performed equivalently.***

configure the *RPs* of the *NE* is detailed in [20]. The detailed process is only applicable for UltraScale architectures since these architectures need clearing bitstreams in order to prepare the *RP* for the new configuration. Consequently, each new reconfiguration of one *RP* of the *NE* requires a clearing operation before being reconfigured. Otherwise, the subsequent *RM* cannot be initialized [19]. It demands a knowledge of what *RM* is configured at each *RP*. This task is done at the host side by the middleware, which monitors the status of the nodes and applies the proper clearing reconfiguration before each PR of a node.

*5.3.1 Cost Table.* The *CT* of the *NE* shown in Table 3 has been partially defined at design time. It lists the different node's configurations to be executed on the FPGA, their relative area cost and the compatibility between the defined *RMs*. The current *CT* considers 4 different configurations of the nodes based on the active microphones and using the *P-SRP* method. Thus, the microphones of all subarrays are active when $N_{am} = 52$, which is the most area demanding configuration. Thanks to the flexibility of the nodes, $N_{am}$ can be modified at runtime without the need of *PR*. Therefore, the largest configuration is able to support all considered node's configurations. To exploit the unused resources of a *RP* when considering low area-demanding node's configurations ($Nam < 12$), several node's configurations are placed in parallel per *RP*. For instance, when $N_{am} = 12$ up to 4 instances can be placed in a *RP* dimensioned for the $N_{am} = 52$ node's configuration. Here is where *PR* has a role for performance acceleration. Of course, such acceleration is determined by the overhead due to partially reconfigure a *RP* and the time cost of each configuration. The time cost shown in Table 3 is the averaged execution time experimentally measured after 100 executions.

*5.3.2 Defining Reconfigurable Partitions.* Figure 9 depicts the 4 *RPs* available on the *NE*. The sizes of the *RPs* are determined by the nodes' configuration sizes and the dedicated I/O channels. Firstly, the *RPs'* size must be large enough to support different node's architectures based on $N_{am}$ or *DM* as detailed in Table 2. Nevertheless, the *RPs* have a fixed dimension independent of the node's parameter under evaluation since hierarchical *PR* is not supported [28]. Thus, our *RPs* are designed to fit the most demanding resource node's configuration, which is $T_{52Mics}$ based on Table 3. Secondly, the 64-bits dedicated AXI4-Stream interface also limits the maximum *LP* per *RP*. Due to the characteristics of the node emulation, each normalized output needs to be represented with at least 16 bits, which limits to 16 the number of nodes simultaneously allocated on the FPGA. Finally, notice that the *RPs* better adjust the available resources to the most area demanding node's configuration with respect to [3], [4].

Figure 10 depicts the list of supported *RMs* based on the experiments presented in Section 6. Notice that each RP supports the same number of *RMs*. For instance, there are 4 different *RMs* based on the number of active subarrays when using the *P-SRP* method. Table 5 details the dimensions of the *RPs* and their percentage of occupancy when configured with $RM_{52}$. Their values slightly vary since not all *RPs* have exactly the same size, containing a different number and type of slices.

*5.3.3 Partial reconfiguration Overhead.* The reconfiguration time per *RP*, including the cleaning operation and the *PR* rounds to 1.09 s. It represents a relatively slow *PR* when considering the *PCIe* theoretical bandwidth and the size of the *RP* bitstream files, which rounds to 4.762 MB. Nevertheless, the time values have been experimentally obtained at the *MCAP* driver side.

## 6 EXPERIMENTAL RESULTS

A couple of experiments are detailed in this section to demonstrate some of the capabilities of the *NE* and the benefits of *PR*. The sound field simulation used in the front end has been optimized for a two dimensional open field where sound attenuation, caused by propagation, has been assumed to be negligible. All the experiments demand a *PR* involving one or more *RMs*. The first experiment demonstrates how *PR* is used to evaluate different *DMs* for several node's configurations and sound source profiles. The main purpose is to show how the capabilities of the *NE* are extended thanks to *PR*. Thus, different *DMs* can be evaluated in runtime, which could not be possible without *PR*. Our strategies and heuristics are evaluated in the second experiment, which exemplifies how the use of *PR* can lead to a significant performance improvement. The increment in performance comes from the better resource

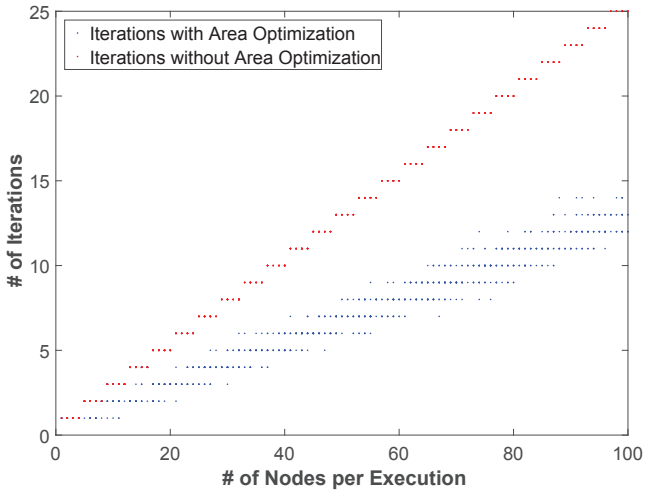



**Figure 12:** ***The $n_{iter}$ is reduced by merging nodes.***

exploitation. As a result, *NE* executions are accelerated when the network is composed of a minimum number of nodes.

All the supported node's configurations (Table 2) are implemented and stored in the host side. As a result, the bitmaps required to run the experiments detailed in this section (Figure 10) are loaded to the *NE* by using *PR* through *PCIe* when needed. Because no bitmap compression technique has been applied, all the bitmaps associated to one *RP* have the same size (Table 5). The bitmaps are grouped based on the experiment where they are used. However, some bitmaps like *RP_RM*12*_PSRP* are used in both experiments. Finally, a static bitmap file contains the static logic and the initial *RMs*' configuration.

The FPGA card used for the implementation of the *NE* is a Xilinx Kintex Ultrascale from Alpha Data (ADM-PCIE-KU3), whose available resources are detailed in Table 5. It provides a Gen3 *PCIe* connection, supporting up to two *PCIe* x8 controllers. Vivado 2016.4 has been used to develop the *PCIe* DMA Subsystem and the *PR* through design flow. The system has been implemented in an Ubuntu 14.04.1 machine that uses C/C++ code, bash scripts and Matlab 2016b.

## 6.1 Impact of the Detection Methods

The *PR* feature of the *NE* provides the capability to evaluate different node's configurations. The following experiment intends to demonstrate how the *PR* allows the comparison of two different *DMs* from the network point of view.

Figure 11 shows the average error in the estimation of the sound source when applying data fusion of 4 nodes using the two inner subarrays. The values have been obtained for a random position of a standalone sound-source at 3 different frequencies (4, 8 and 10 kHz). The RMs are partially reconfigured to switch between both methods. The evaluation also considers the permutations of all possible combinations of the two inner subarrays. Thus, the top left error value corresponds to the 4 nodes with only the inner subarray active while the top right corresponds to all the nodes with two inner subarrays active. The results show that the *CC* method does not offer a significant improvement compared to the *P-SRP* method. Despite offering a lower estimation error, its implementation in a distributed network of microphone arrays is not completely justified considering the additional resource consumption due to the required multiplications. Nevertheless, further experiments must be done with different sound-source frequencies, with real measurements and in an anechoic room before discarding completely the advantages of the *CC* method.

## 6.2 Partial Reconfiguration for Higher Performance

The use of *PR* to increment performance is evaluated through multiple executions with $n_{node}$ and random $N_{am}$ per node. All the experimental results explained here have been obtained after 10000 executions of up to 100 random nodes per execution. Only the *P-SRP* method is considered in this experiment. The only difference in the node's configuration is $N_{am}$, which directly affects the node's resource consumption. Therefore, the performance increases thanks to an increment in the number of node's configurations executed in one iteration, which is done by allocating on each *RP* multiple node's configurations with small $N_{am}$.

Figure 12 depicts the required $n_{iter}$ based on the $n_{node}$ with and without merging nodes. The $n_{iter}$ is, by default, expressed in Eq 2. This value can be decreased thanks to exploiting the unused resources per *RP*. Thus, the merging of nodes to share resources of one *RP* leads to a lower $n_{iter}$ needed per execution. Since the $n_{iter}$ determines the execution time, the merging of the nodes is expected to directly benefit the performance. Both graphs are stepped because at least $n_{RP}$ nodes are executed per iteration.

Figure 13 depicts the average execution time and the overall speedup. The non-heuristic strategy (*None*) is used as reference. This strategy does not need to partially reconfigure the *RP*, and therefore, does not benefit from the use of the heuristics. Each *RP* is configured with the same *RM*52 in order to support all the node's configurations under evaluation. Consequently, the *None* strategy time-multiplexed the nodes in the available *RPs* without any merging or scheduling. The other two strategies under evaluation consider the proposed heuristic for merging of the node's configurations as standalone (*Merge*) and combined with the proposed heuristic for scheduling (*Merge + Schedule*). Both strategies have a random initial configuration of the *RPs*, which are randomly asserted after each execution when using *PR*. This is the expected behavior of the *NE* since the final configuration of the *RPs* after one execution is unknown in advance, at least not before the execution of the heuristics.

The results depicted in Figure 13 reflect that, although a lower $n_{iter}$ obtained by merging the node's configuration in the same *RP* should lead to a higher performance, the *PR* time overhead decreases the overall performance. Moreover, the use of *PR* without a proper scheduling induces performance decrements, which is reflected in Figure 13. Although the merging of nodes increases the parallelism and diminishes the $n_{iter}$, the *PR* overhead dominates when the nodes are not properly scheduled. Executions demanding a low $n_{node}$ are specially sensitive to this overhead, because the *PR* time overhead represents a large percentage of the overall execution time. As a consequence, the proper scheduling of the nodes is not beneficial unless a certain $n_{node}$ must be computed per execution.

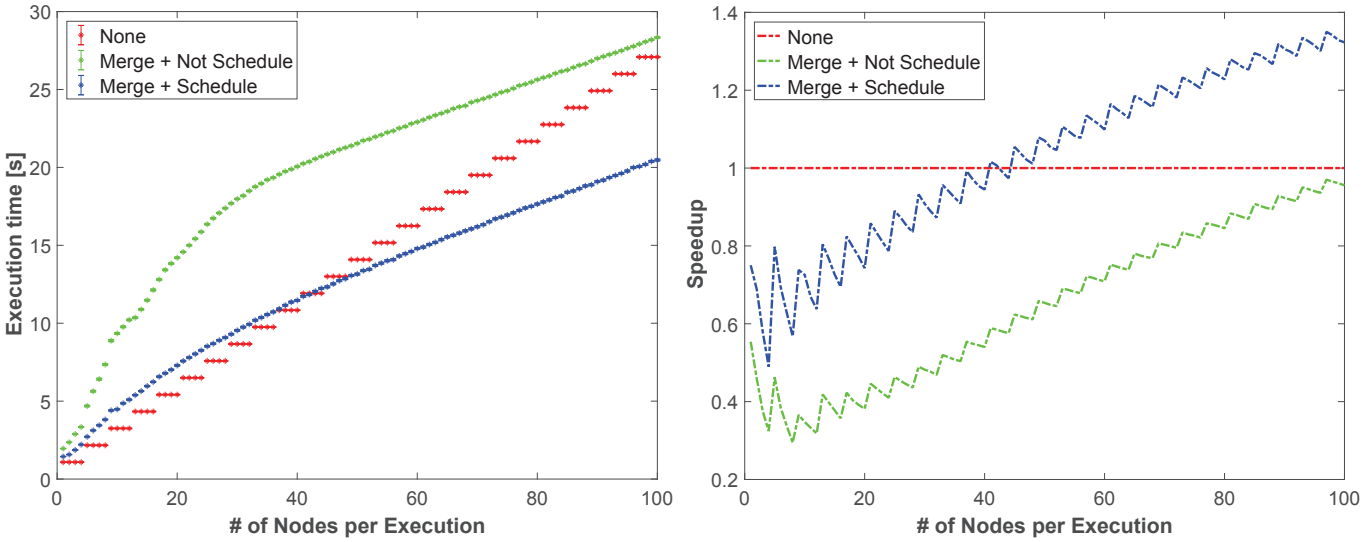



**Figure 13:** ***Average execution time and speedup for different strategies.***

The rightmost figure in Figure 13 represents the overall speedup when only merging or also scheduling. Both graphs are saw waves for the same reason the graphs in Figure 12 are stepped. Because several nodes' configurations are computed in parallel, the $n_{iter}$ remains constant while incrementing $n_{node}$. Thus, the speedup increases when increasing $n_{node}$ computed in the same amount of time, but suddenly decreases when an additional iteration must be computed. The proper merging and scheduling of the nodes' configurations is only beneficial in average when $n_{node}$ is higher than 45.

## 7 CONCLUSION

The presented *NE* has shown the capacity to evaluate different WSN configurations thanks to its ability to mimic the node's response to several sound sources. The use of *PR* through PCIe not only allows us to obtain a flexible *NE* capable of exploring multiple configuration scenarios in runtime but also to accelerate the *NE*'s execution by exploiting the available resources and the inherent parallelism of the node's emulation. We believe our approach for the *NE* not only provides an interesting case study of how *PR* can be used to increment performance but can also be extended for other streaming applications such as video processing, where similar convolutional filters must be applied to different image sources. Nevertheless, it will also be interesting to explore other strategies like bitstream compression in order to further reduce the PR time cost. Finally, in the current version of our emulator, the user is able to select the node and its configuration at every moment. Although the current version of the emulator is managed by the user, we consider that certain level of intelligence can be added in the control automation to determine, in real-time, the best strategies to evolve the network configuration to obtain the lowest power consumption with the lowest estimation error.


## ACKNOWLEDGEMENTS

This work was supported by the European Regional Development Fund (ERDF) and the Brussels-Capital Region-Innoviris within the framework of the Operational Programme 2014-2020 through the ERDF-2020 Project ICITYRDI.BRU. This work is also a result of the CORNET project *"DynamIA: Dynamic Hardware Reconfiguration in Industrial Applications"* [29] which was funded by IWT Flanders with reference number 140389. Finally, the authors would like to thank Xilinx for the provided software and hardware under the University Program donation.


## CONFLICT OF INTEREST

The authors declare that there is no conflict of interests regarding the publication of this paper.

## REFERENCES


[1] Tiete, J., et al. *"SoundCompass: a distributed MEMS microphone array-based sensor for sound source localization"*, Sensors 14.2 (2014): 1918-1949.

[2] Ottoy, G., et al. *"A low-power MEMS microphone array for wireless acoustic sensors"*, Sensors Applications Symposium (SAS), 2016 IEEE. IEEE, 2016.

[3] da Silva, B., et al. *"A partial reconfiguration based microphone array network emulator"*, Field Programmable Logic and Applications (FPL), 2017 27th International Conference on. IEEE, 2017.

[4] da Silva, B., et al. "Demonstration of a partial reconfiguration based microphone array network emulator." Field Programmable Logic and Applications (FPL), 2017 27th International Conference on. IEEE, 2017.

[5] da Silva, B.,et al. *"Runtime reconfigurable beamforming architecture for real-time sound-source localization"*, Field Programmable Logic and Applications (FPL), 2016 26th International Conference on. EPFL, 2016.

[6] da Silva, B., et al. *"Design Considerations when Accelerating an FPGA-Based Digital Microphone Array for Sound-Source Localization"*, Journal of Sensors (2017): 2.

[7] Huang, J., et al. *"Scalable FPGAbased architecture for DCT computation using dynamic partial reconfiguration"*, ACM Transactions on Embedded Computing Systems (TECS) 9.1 (2009): 9.

[8] Avelino, A., et al. *"LP-P$^2$IP: A Low-Power Version of P$^2$IP Architecture Using Partial Reconfiguration"*, International Symposium on Applied Reconfigurable Computing. Springer, Cham, 2017

[9] Reorda, M. S., et al. *"An error-detection and self-repairing method for dynamically and partially reconfigurable systems*, IEEE Transactions on Computers 66.6 (2017): 1022-1033.

[10] Koch, D., et al. *"Partial reconfiguration on FPGAs in practice - Tools and applications"*, ARCS Workshops (ARCS), 2012. IEEE, 2012.
[11] Nasreddine, N., et al. *"Wireless sensors networks emulator implemented on a FPGA"*, In Field-Programmable Technology (FPT), 2010 International Conference on (pp. 279-282). IEEE, 2010.
[12] Val, I., et al. *"FPGA-based wideband channel emulator for evaluation of Wireless Sensor Networks in industrial environments"*, In Emerging Technology and Factory Automation (ETFA), 2014 IEEE (pp. 1-7). IEEE, 2014.
[13] Llamocca, D., et al. *"A Reconfigurable Fixed-Point Architecture for Adaptive Beamforming"*, Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International. IEEE, 2016.
[14] Ghiasi, S., et al. *"Optimal reconfiguration sequence management"*, In Proceedings of the 2003 Asia and South Pacific Design Automation Conference (pp. 359-365). ACM, 2003
[15] Cordone, R., et al. *"Partitioning and scheduling of task graphs on partially dynamically reconfigurable FPGAs"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 28.5 (2009): 662-675.
[16] Wildermann, S.,et al. *"Placing multimode streaming applications on dynamically partially reconfigurable architectures"*, International Journal of Reconfigurable Computing, 2012
[17] Vu, D. V., et all *"Enabling partial reconfiguration for coprocessors in mixed criticality multicore systems using PCI Express Single-Root I/O Virtualization"*, In ReConFigurable Computing and FPGAs (ReConFig), 2014 International Conference on (pp. 1-6). IEEE, 2014.
[18] Kizheppatt, V. et al. *"DyRACT: A partial reconfiguration enabled accelerator and test platform"*, Field Programmable Logic and Applications (FPL), 2014 24th International Conference on. IEEE, 2014.
[19] Xilinx, *Vivado Design Suite User Guide - Partial Reconfiguration*; Xilinx User Guide 909 (v2016.4), https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug909-vivado-partial-reconfiguration.pdf. 2016.
[20] Xilinx, *Bitstream Loading across the PCI Express Link in UltraScale Devices for Tandem PCIe and Partial Reconfiguration*; Xilinx Answer 64761, https://www.xilinx.com/support/answers/64761.html. 2016.
[21] Xilinx, *Xilinx PCI Express DMA Drivers and Software Guide*; Xilinx Answer 65444, https://www.xilinx.com/support/answers/65444.html. 2016.
[22] Xilinx, *DMA Subsystem for PCI Express v2.0*; Xilinx Product Guide 195, https://www.xilinx.com/support/documentation/ip_documentation/xdma/v2_0/pg195-pcie-dma.pdf. 2016.
[23] AnalogDevices. *ADMP521 datasheet "Ultralow Noise Microphone with Bottom Port and PDM Digital Output*, Technical Report, Analog Devices: Norwood, MA, USA, 2012.
[24] Hogenauer, E. *"An economical class of digital filters for decimation and interpolation"*, Acoustics, Speech and Signal Processing, IEEE Transactions on 29.2 (1981): 155-162. IEEE, 1981.
[25] Lima, M. V., et al. *"A volumetric SRP with refinement step for sound source localization"*, IEEE Signal Processing Letters, 22(8), 1098-1102.IEEE, 2015.
[26] Barnes, C. W., et al. *"On the statistics of fixed-point roundoff error."* IEEE Transactions on Acoustics, Speech, and Signal Processing 33.3 (1985): 595-606.
[27] Cmar, R., et al.*"A methodology and design environment for DSP ASIC fixed point refinement."* Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings. IEEE, 1999.
[28] Beckhoff, C., et al. *"Go ahead: A partial reconfiguration framework"*, Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on. IEEE, 2012.
[29] Mentens, Nele, et al. *"DynamIA: Dynamic hardware reconfiguration in industrial applications"*, International Symposium on Applied Reconfigurable Computing. Springer, Cham, 2015.