Accelerators for Breast Cancer Detection

DANIELE JAHIER PAGLIARI, Politecnico di Torino, Dipartimento di Automatica e Informatica
MARIO R. CASU, Politecnico di Torino, Dipartimento di Elettronica e Telecomunicazioni
LUCA P. CARLONI, Columbia University, Department of Computer Science

Algorithms used in microwave imaging for breast cancer detection require hardware acceleration to speed up execution time and reduce power consumption. In this article, we present the hardware implementation of two accelerators for two alternative imaging algorithms that we obtain entirely from SystemC specifications via high-level synthesis. The two algorithms present opposite characteristics that stress the design process and the capabilities of commercial HLS tools in different ways: the first is communication bound and requires overlapping and pipelining of communication and computation in order to maximize the application throughput; the second is computation bound and uses complex mathematical functions that HLS tools do not directly support. Despite these difficulties, thanks to HLS, in the span of only 4 months we were able to explore a large design space and derive about 100 implementations with different cost-performance profiles, targeting both a Field-Programmable Gate Array (FPGA) platform and a 32-nm standard-cell Application Specific Integrated Circuit (ASIC) library. In addition, we could obtain results that outperform a previous Register-Transfer Level (RTL) implementation, which confirms the remarkable progress of HLS tools.

CCS Concepts: • **Hardware** → **Hardware accelerators**; **High-level and register-transfer level synthesis**; *Arithmetic and datapath circuits*; • **Computer systems organization** → *System on a chip*; *Embedded systems;*

Additional Key Words and Phrases: Microwave imaging for breast cancer detection

ACM Reference Format:
Daniele Jahier Pagliari, Mario R. Casu, and Luca P. Carloni. 2017. Accelerators for breast cancer detection. ACM Trans. Embed. Comput. Syst. 16, 3, Article 80 (March 2017), 25 pages.
DOI: http://dx.doi.org/10.1145/2983630

1. INTRODUCTION

Innovations in medical imaging are key to increasing the effectiveness of health care as well as to reducing its cost. New medical imaging techniques quite often involve the solution of complex mathematical problems or the repetitive application of simple algorithms to a multitude of data points (e.g., for 3D images). As a consequence, it is not easy to enclose the required computational capacity in medical equipment with a constrained form factor and/or a limited power budget. Some level of hardware customization is then required to obtain power-efficient and high-performance implementations of innovative medical imaging algorithms [Cong et al. 2011]. These

This work is partially supported by the NSF (Award #: 1219001), by C-FAR (Contract #: 2013-MA-2384), one of the six SRC STARnet centers, and by the Italian MIUR (project MICENEA).
Authors' addresses: D. J. Pagliari, Dipartimento di Automatica e Informatica, Politecnico di Torino, Corso Duca degli Abruzzi, 24 - 10129 Torino, Italy; email: daniele.jahier@polito.it; M. R. Casu, Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Corso Duca degli Abruzzi, 24 - 10129 Torino, Italy; email: mario.casu@polito.it; L. P. Carloni, Department of Computer Science, Columbia University in the City of New York, 1214 Amsterdam Avenue, New York, NY 10027-7003, USA; email: luca@cs.columbia.edu.

DOI: http://dx.doi.org/10.1145/2983630

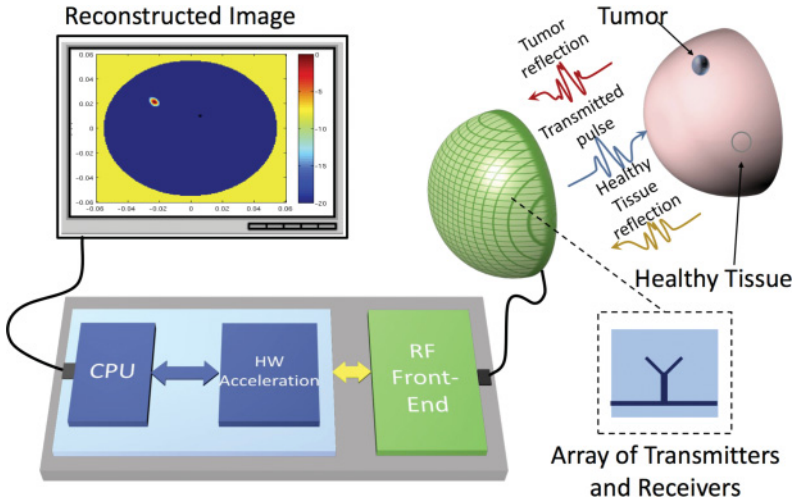


Fig. 1. Microwave-imaging breast cancer detection system.

specialized hardware modules, commonly referred to as *accelerators*, speed up the execution of the critical sections of an application, while limiting power consumption and form factor with respect to multicore or GPU-based solutions.

One of the new promising areas of medical imaging is Microwave Imaging (MI) for breast cancer detection. This method is considered a valid alternative to overcome the limitations of X-ray mammography, such as the danger due to ionizing radiations, and the high rate of false negatives and false positives [Bond et al. 2003]. MI uses an array of antennas to irradiate the breast of a patient with microwave signals. The corresponding reflections are sampled and processed digitally, to produce a map of the breast with lesions highlighted. The effectiveness of the method is due to the high dielectric contrast between tumors and surrounding tissues in the microwave spectrum.

Like in other medical imaging techniques, the execution time of a pure software implementation of MI algorithms would be inappropriate for a clinical scenario, hence calling for a custom hardware implementation. Figure 1 illustrates a possible embedded architecture implementation, in which a CPU executes the noncritical parts of the computation and offloads the critical ones to a dedicated hardware accelerator.

Thanks to High-Level Synthesis (HLS), designers can obtain an effective accelerator implementation at Register-Transfer Level (RTL) starting from a high-level specification (e.g., a C/C++ or SystemC description) [Coussy and Morawiec 2008; Martin and Smith 2009]. The RTL is then the input to a standard design flow, either targeting an Field-Programmable Gate Array (FPGA) or an ASIC technology. Current commercial HLS tools allow designers to quickly explore the design space and synthesize in a shorter time span many more alternative microarchitectures than they could if they started at RTL. By doing so, designers can easily obtain the Pareto frontier in the cost/performance space of the solutions, and prune those that are not Pareto optimal [Liu et al. 2012; Prost-Boucle et al. 2013].

In this work, which extends Jahier Pagliari et al. [2015], we examine two alternative methods for breast cancer MI and use HLS to accelerate the critical sections of the corresponding imaging algorithms. These algorithms stress the design process and the capabilities of HLS tools in different ways. The first method, called *MIST Beamforming* [Li et al. 2005], requires the execution of a set of simple operations on a massive amount of input data. The second, *MUSIC-Inspired (MUSIC-I)* [Ruvio et al. 2013], processes a relatively smaller amount of data but requires the execution of a set of

operations that are significantly more complex. Consequently, the performance of the corresponding accelerator-based implementations are likely communication bound for MIST Beamforming and computation bound for MUSIC-I. Other differences include the memory access patterns and the control logic complexity.

For both methods, after determining the sections to accelerate, we wrote a design specification in SystemC. We used this specification to derive a set of Pareto-optimal RTL implementations, by means of a state-of-the-art commercial HLS tool. We completed a comprehensive design-space exploration by varying HLS knobs like parallelism, pipelining, resource sharing, mapping of the memory elements and, in the case of MUSIC-I, the implementation of complex mathematical operators. To fully leverage the potential of the accelerators, we aimed at microarchitectures that optimize the pipeline between communication and computation. During synthesis, we targeted both FPGA and ASIC technologies. We verified and validated the designs on a Xilinx Zynq board, which is the emulation platform closest to a final SoC implementation.

The following summarizes our findings and contributions:

—This is the first time that HLS is used for the acceleration of MI algorithms for breast cancer detection. Furthermore, while a manually designed RTL implementation of MIST Beamforming has been reported in the literature [Colonna et al. 2013; Casu et al. 2014], this article presents the first accelerator design for MUSIC-I.
—The comparison with the manually designed RTL implementation of MIST Beamforming shows that the HLS design is superior, thus demonstrating both the quality of the results that can be achieved with modern HLS tools and the quality of our system-level design effort.
—For both MUSIC-I and MIST, our designs provide a significant acceleration over a software implementation running on an embedded system with a limited use of area resources and power. Thanks to the configurable degree of parallelism and the high scalability of the microarchitectures, the acceleration benefits are ultimately limited by either the power/area budget or the communication bandwidth.
—With HLS we could design, validate, and evaluate more than 100 implementations of the algorithms in less than 4 months. This contrasts quite strikingly with the 3 months required for a single RTL implementation of MIST.

The article is organized as follows. In Section 2, we describe the main features of the two MI algorithms. In Section 3, we illustrate the architecture of the accelerators. The results of HLS and the speedup versus a software implementation are presented in Section 4, which also analyzes the best solutions in the design space not only in terms of sheer performance but also in terms of energy and power efficiency, and use of hardware resources. The related work is analyzed in Section 5. Finally, Section 6 concludes the article.

2. MI BREAST CANCER DETECTION

Tomography and linear scattering are the most common approaches for MI breast cancer detection [Nikolova 2011]. Tomography solves an ill-posed problem to reconstruct the entire dielectric profile of the breast. If the goal is to detect only the presence of tumors, simpler linear-scattering (or *radar*) algorithms can be used to identify the locations of the most dominant scatterers in the breast.

We focus on two linear-scattering methods: the well-established Microwave Imaging via Space-Time (MIST) Beamforming [Li et al. 2005], and a new approach based on a MUltiple SIgnal Classification-Inspired (MUSIC-I) strategy [Ruvio et al. 2013]. Both methods use Ultra Wide-Band (UWB) pulses to irradiate the breast. An array of transceivers generates UWB pulses and then collects the corresponding reflections to be processed. Next, we provide a succinct description of the two methods. We focus

specifically on the computational aspects of the two algorithms that are relevant from the point of view of their hardware acceleration. A complete theoretical treatment of the two algorithms can be found in the original papers [Li et al. 2005; Ruvio et al. 2013].

2.1. MIST Beamforming

The bulk of the processing of MIST Beamforming consists of synthetically focusing the reflections scattered by a particular point of a volume, called a *voxel*. The scattered UWB pulse is sampled and digitized by N_{ant} antennas. These data are first preprocessed to eliminate clutter caused by the air-skin interface. Then, the so-obtained signals, which we denote as $(x_1[n], x_2[n], \ldots, x_{N_{\text{ant}}}[n])$, undergo the actual beamforming step, in which they are delayed and filtered, in order to align the scattering contributions due to the given voxel on the signal received by different antennas.

For each voxel location v, the first step is a coarse alignment of the N_{ant} signals, obtained shifting each signal in time by a different number of samples. For the a-th signal $x_a[n], 1 \leq a \leq N_{ant}$, the shift is computed as $n_a(v) = N_A - d_a(v)$, where $d_a(v)$ is the round-trip delay between antenna a and location v rounded to the nearest sample, and N_A is a reference chosen greater than or equal to the maximum delay, that is, $N_A \geq \max_a(d_a(v))$.

In the second step, the samples with index smaller than N_A are discarded, since they surely do not contain the reflection from voxel v, given the previous alignment. This step is conceptually equivalent to multiplying the samples with the following function:

$$g[n] = \begin{cases} 1 & \text{if } n \geq N_A \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

In the third step, the signal from each antenna is filtered with a Finite Impulse Response (FIR) filter. The length L of the filter (same length for all antennas) and the arrays of filter weights w_a, with $1 \leq a \leq N_{ant}$ are designed to compensate for path-length-dependent dispersion and attenuation, and to isolate the frequency band of interest. The computation of filter length and weights is not reported for brevity reasons and because, as explained in Section 3, it is not computationally relevant from the point of view of HW acceleration. All details can be found in Li et al. [2005].

The fourth step isolates the part of the signal in which the reflection created by point v is expected. This is done by zeroing all the samples outside a window of length L_H, starting before sample index N_H and ending after sample index $N_H + L_H$:

$$h[n] = \begin{cases} 1 & \text{if } N_H \leq n \leq N_H + L_H \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Finally, all the aligned, filtered, and windowed signals are combined, and the voxel energy is computed by squaring and accumulating the signal samples over the window span. The entire energy evaluation phase for voxel v can then be summarized as

$$en(v) = \sum_n |h[n] \sum_{a=1}^{N_{\text{ant}}} \sum_{l=0}^{L-1} w_a(v,l) \cdot g[n-l] \cdot x_a[n-l-n_a(v)]|^2, \quad \forall v \in [1, N_{\text{vox}}]. \tag{3}$$

Thanks to alignment, contributions from the voxel of interest are added coherently while noise and reflections due to other locations are rejected. This process is repeated for all voxels, that is, $\forall v \in [1, N_{\text{VOX}}]$, after changing alignment and filtering coefficients. The result is a map of reflected energy, usually a 2D slice of a 3D image. Alignment "delays" and filter weights are determined voxel by voxel: they depend on the configuration and characteristics of the antennas and of the propagating medium.

```
for v = 1 : NVOX % loop over voxels
  energy = 0;
  for n = 0 : LH  % loop over samples
    z = 0;
    for a = 1 : NANT % loop over antennas
      for l = 1 : L % loop over filter weights
        idx = n + l + (NH - L);
        if idx > NA % windowing and alignment
          z = z + ( x(a, idx - d(a,v)) * w(a,v,l));
        end
      end
    end
    energy = energy + (z * z);
  end
  en(v) = energy;
end
```

Listing 1. MIST Beamforming computational kernel.

Listing 1 shows a simplified MATLAB implementation of the beamforming kernel. Matrix x contains the N_{ant} signals after clutter removal, while d and w arrays store alignment delays and filter weights, respectively. The content of these last two arrays is computed in a preliminary characterization phase, not reported in the code, as it only depends on the characteristics of the system and not on the input data. Finally, array *en* stores the energy values. Note that we avoid the multiplications by windowing functions (1) and (2) by properly indexing the arrays accessed in the energy computation. A detailed discussion on the determination of parameters (e.g., window threshold, its length, etc.) can be found in the literature [Li et al. 2005; Colonna et al. 2013].

2.2. MUSIC-Inspired Reconstruction

MUSIC-I is a proven robust method when characterization of the antennas and of the heterogeneity in the breast tissue is not possible. Contrarily to MIST, which works in time domain, MUSIC-I processes the samples in frequency domain via Fast Fourier Transform (FFT).

The first step is again clutter removal to eliminate the reflections caused by the skin. This is obtained with a subspace-projection method [Ruvio et al. 2013]. The N_{freq} frequency-domain samples are organized in matrix $\mathbf{B}$ whose columns correspond to N_{ant} antennas. The matrix is transformed via Singular Value Decomposition (SVD), that is, $\mathbf{B} = \mathbf{UDV}^T$; $\mathbf{U}$ and $\mathbf{V}$ contain in each column the left and right singular vectors of $\mathbf{B}$, respectively; $\mathbf{D}$ is a diagonal matrix with the eigenvalues (λ_i) on its diagonal, sorted in decreasing order of magnitude; as usual, the dominant eigenvectors are those associated with the eigenvalues with largest absolute value. Since skin-air reflections are orders of magnitude larger than reflections due to possible tumors in the breast, it is possible to isolate the clutter subspace by removing the first k singular values of $\mathbf{B}$. Therefore, the decluttered sample matrix $\mathbf{S}_d$ is obtained as follows:

$$\mathbf{S}_d = \sum_{i=k+1}^{N_\lambda} \lambda_i u_i v_i^H, \tag{4}$$

where u_i and v_i represent the ith column of $\mathbf{U}$ and $\mathbf{V}$, respectively, and H indicates the Hermitian (conjugate) transpose operator. N_λ is the total number of the eigenvalues obtained as $N_\lambda = \min(N_{\text{ant}}, N_{\text{freq}})$. In Ruvio et al. [2013] it has been shown that a proper value for k is 2, hence we use such value.

According to the original MUSIC theory [Schmidt 1986], the location of the points that scatter most of the incident electromagnetic field can be obtained from the

correlation matrix of the decluttered samples, which is computed in the first step of the actual image reconstruction, for each frequency f from 1 to N_{freq}. Let S_d^f be the fth row of $\mathbf{S}_d$; then the fth correlation matrix is computed as follows:

$$\mathbf{R}^f = S_d^f \cdot S_d^{fH}. \tag{5}$$

Similarly to SVD decluttering, MUSIC is a subspace projection method and consists in separating the so-called *signal subspace* from the *noise subspace*. Therefore, the second step of the reconstruction aims to obtain the signal subspace, and consists in computing eigenvalues and eigenvectors of $\mathbf{R}^f, \forall f$. Since the rank of $\mathbf{R}^f$ is proven to be one, the MUSIC theory proposes to consider as signal subspace the singular vector corresponding to the only nonnull eigenvalue.

The third step is the evaluation of the *propagator* that maps each possible *scatterer*, which in this case corresponds to each trial location in the image that is being reconstructed, to the scattered field. The propagator is a vector of the form

$$\mathbf{W}(\mathbf{r}_o, \mathbf{r}, f) = \left(G^2(\mathbf{r}_{o1}, \mathbf{r}, f), G^2(\mathbf{r}_{o2}, \mathbf{r}, f), \ldots, G^2(\mathbf{r}_{oN_{ant}}, \mathbf{r}, f)\right). \tag{6}$$

In this equation, $\mathbf{r}_{oa} = (x_{oa}, y_{oa})$ represents the position of the ath antenna, and $\mathbf{r} = (x, y)$ is the position of the trial location. G is a scalar Green's function:

$$G(\mathbf{r}_{oa}, \mathbf{r}, f) = e^{-jk_b(f)\cdot\|\mathbf{r}_{oa}-\mathbf{r}\|}, \tag{7}$$

where $k_b(f)$ is the wave number at frequency f in the breast, in turn computed as

$$k_b(f) = 2\pi f\sqrt{\frac{\epsilon_b}{c}}. \tag{8}$$

In Equation (8), c stands for the light speed and ϵ_b is the breast average dielectric permittivity, which can be computed by averaging the data of breast tissues from a large database [Burfeindt et al. 2012]. The propagator actually contains the *square* of the Green's function because the signal propagates from one antenna in $\mathbf{r}_{oa}$ to location $\mathbf{r}$, it gets reflected, and travels back to $\mathbf{r}_{oa}$ following the same path.

In the fourth step the propagator $\mathbf{W}(\mathbf{r}_o, \mathbf{r}, f)$ is normalized and multiplied by $\mathbf{u}_1(f)$, the nonnull dominant eigenvector of $\mathbf{R}^f$, as follows:

$$F(\mathbf{r}_o, \mathbf{r}, f) = \left\langle \frac{\mathbf{W}(\mathbf{r}_o, \mathbf{r}, f)}{\|\mathbf{W}(\mathbf{r}_o, \mathbf{r}, f)\|}, \mathbf{u}_1(f) \right\rangle, \tag{9}$$

where $\langle\cdot,\cdot\rangle$ is the Hermitian Inner Product. The fifth step is the evaluation for each frequency of a *detection function*:

$$D(\mathbf{r}_o, \mathbf{r}, f) = \frac{1}{1 - |F(\mathbf{r}_o, \mathbf{r}, f)|^2}. \tag{10}$$

Theoretically, function $D(\mathbf{r}_o, \mathbf{r}, f)$ should have a peak when the trial location $\mathbf{r}$ corresponds to one of the scatterers. However, due to the low rank of $\mathbf{R}^f$, spurious artifacts may appear in the image in the presence of noise [Solimene et al. 2012]. To reduce the artifacts, data at multiple frequencies are mixed in the last step, by computing the product of all the individual detection functions:

$$P(\mathbf{r}_o, \mathbf{r}) = \prod_{m=1}^{N_f} D(\mathbf{r}_o, \mathbf{r}, f_m). \tag{11}$$

In the final image, for each location $\mathbf{r}$, the value of $P(\mathbf{r}_o, \mathbf{r})$ is plotted. Since the peaks corresponding to real scatterers are found in the same position for all frequencies,

```
1  inv_P = 1;
2  for f = 1 : Nfreq % loop over frequencies
3    % correlation matrix
4    R = Sd(:,:,f).' * conj(Sd(:,:,f));
5    % eigenvectors/eigenvalues computation
6    [V,D]=eig(R);
7    [max_val, max_idx] = max(abs(diag(D)));
8    for u = 1 : nx % loop over image rows
9      for v = 1 : ny % loop over image columns
10       % green function computation
11       Wn = (exp(-j*kb(f) * sqrt((x(u)-xo).^2 + (y(v)-yo).^2)).').^2;
12       % hermitian inner product and norm
13       F(u,v,f)= norm((Wn / norm(Wn))' * V(:,max_idx));
14     end
15   end
16   % product over different frequencies
17   inv_P = inv_P.*(1-F(:,:,f).^2);
18 end
19 P = 1/inv_P;
```

Listing 2. MUSIC-Inspired computational kernel.

multiplying the various indicator functions will amplify them, while leaving artifacts untouched, hence ultimately increasing the signal to noise ratio.

More details on MUSIC-I are available in the literature [Ruvio et al. 2013].

The MATLAB code in Listing 2 shows the steps of MUSIC-I image reconstruction for a 2D image. In Listing 2, *nx* and *ny* represent the image size in pixels; $x(nx)$ and $y(ny)$ store the horizontal and vertical coordinates (assuming the origin is the center of the observation domain) of the location under examination; *xo* and *yo* contain the positions of the antennas; k_b is the wave number; W_n is the propagator array; F stores the Hermitian inner products. Finally, matrix P contains the detection function.

3. HIGH-LEVEL DESIGN OF HARDWARE ACCELERATORS

3.1. Software Profiling

The profiling and hardware acceleration of MIST Beamforming have been studied before in the literature [Casu et al. 2014; Colonna et al. 2013]. These works showed that filter weights and alignment delays can be computed offline and that the air-skin decluttering step can be implemented sufficiently fast in software. Based on these results we focused our effort on accelerating the beamforming step, which is computationally the heaviest [Casu et al. 2014]. In contrast with prior works, however, we designed our accelerator at a level of abstraction higher than RTL.

To identify the critical sections of MUSIC-I, we wrote an optimized version of the original MATLAB code in C language and profiled a software execution divided in five steps: FFT, declutter (SVD), eigenvalues extraction, Hermitian Product (matrix F in Listing 2), and computation of the detection function (matrix P in Listing 2). We evaluated the execution time for reconstructing a 2D image with 200×200 pixels, starting from samples obtained at 20 different frequencies from 18 antennas.

Table I reports the execution times obtained with a *single-thread* version of the C code on an Intel Xeon E5-2630 CPU (2.40GHz, 16 cores, 32 threads, 128-GB RAM, Linux kernel 2.6.32-624). The loop over all voxels that computes F (lines 8–23 of Listing 2) is the obvious target for hardware acceleration. Even though this section completes in a few seconds on a small 2D image, the loop over a 3D image with $200 \times 200 \times 200$ voxels would require about 12 minutes for each of the two breasts of a patient,[1] a time that

[1]3D images in MUSIC-I, like in MIST, are obtained by combining multiple 2D "slices." Hence, the execution time of each phase scales linearly with the number of slices.

Table I. Profiling of MUSIC-Inspired Algorithm

Subtask	Runtime (s)
FFT	0.0003322
Declutter (SVD)	0.0008422
Computation of Eigenvalues	0.0026025
Computation of F	3.4667474
Computation of P	0.0028863

is not acceptable for a clinical scenario. When 20 threads compute in parallel (using PThreads), each a different frequency component of F, we obtain a speedup around $10.5\times$. Thus, the computation of F remains the bottleneck of the execution time.[2]

3.2. Comparison Between MIST and MUSIC-I Critical Sections

Having identified the critical sections in the two algorithms, it is interesting to compare them, by pointing out the issues that each of them raises for the design of the corresponding hardware accelerator. In particular, we analyze the complexity of datapath implementations and the storage requirements.

3.2.1. Datapath Complexity. From the viewpoint of designing a HW datapath, the critical section of MIST can be reduced to an advanced filter, and only requires additions and multiplications on real numbers, as is clear from Listing 1. Once an appropriate data representation is chosen, commercial HLS tools support these operations natively, and can automatically synthesize an optimal datapath for a given target technology.

A first remarkable difference between the two algorithms is that MUSIC-I requires operations on complex numbers rather than on real numbers. This approximately requires a duplication of the hardware resources necessary to represent real and imaginary parts. Other than multiplications and additions on complex numbers, as shown in Listing 2, MUSIC-I also performs the following operations:

—exponential function;
—sine and cosine (required for the complex part of the exponential);
—division;
—square root.

Many algorithms have been reported in the literature for these functions [Koren 2002], with different characteristics in terms of the performance-complexity tradeoff to be considered for their implementation. For instance, some of these internally require multiplications; others only use additions and shifts, but pay the reduction in complexity with a larger number of clock cycles to obtain a result. Even though commercial HLS tools do not support these functions natively, the large number of possibilities for their implementation fits perfectly the characteristics of these tools.

In summary, the design of a datapath for the critical section of MUSIC-I is definitely more complex than the one for MIST and is expected to result in a more costly hardware implementation.

3.2.2. Storage Requirements. The exact amount of storage elements required by the two accelerators cannot be determined before completing the architecture definition. For a preliminary comparison, however, we consider the case of an image that is processed atomically, that is, under the hypothesis that the entire input and output datasets are stored in the accelerator internal memories. Even if, as we shall see, this is not

[2]Notice that these results refer to a server-class processor, whose power consumption and form factor may be unsuitable for medical equipment. A better reference would probably be an embedded processor, which is inevitably slower due to the limited number of cores/threads and the slower clock frequency.

Table II. Input Requirements for the Accelerated Sections of the Two Reconstruction Algorithms, and Example in Bytes for a 200 x 200 Image

MIST		
Input	**No. of Elements**	**Example**
x	$N_s \cdot N_{ant}$	4,608
w	$N_{vox} \cdot N_{ant} \cdot L$	39,600,000
d	$N_{vox} \cdot N_{ant}$	720,000
Total		40,324,608 ($\approx$ 8MB)
MUSIC		
Input	**No. of Elements**	**Example**
(xo, yo)	$2 \cdot N_{ant}$	36
(x, y)	$N_X + N_Y$	400
kb	$2 \cdot N_{freq}$	40
$u1$	$2 \cdot N_{ant} \cdot N_{freq}$	720
Total		1,196 ($\approx$ 4KB)

Table III. Output Requirements for the Accelerated Sections of the Two Reconstruction Algorithms, and Example for a 200 x 200 Image

MIST		
Output	**No. of Elements**	**Example**
en	N_{vox}	40,000 ($\approx$ 320KB)
MUSIC		
Output	**No. of Elements**	**Example**
F	$N_X \cdot N_Y \cdot N_{freq}$	800,000 ($\approx$ 3.2MB)

a realistic scenario, it simplifies the understanding of some of the key architectural choices that we made, as described in the following sections. Tables II and III report input and output requirements of the two accelerated sections. For each variable, the number of elements is expressed as a function of the system parameters. We consider the case of a 2D image with 200×200 pixels, under the following realistic conditions:

—number of antennas (for both algorithms): $N_{ant} = 18$;
—number of frequencies (for MUSIC): $N_{freq} = 20$;
—number of samples per channel (for MIST): $N_s = 256$;
—filter tap length (for MIST): $L = 55$;
—number of voxels (pixels for a 2D image): $N_{vox} = N_X \cdot N_Y = 200 \cdot 200$.

Values in bytes in Tables II and III refer to a fixed-point representation that we present later in Section 3.3. The multiplication by a factor 2 in k_b and u_1 of MUSIC-I is due to the fact that they are complex numbers. Tables II and III show that MIST Beamforming requires a much larger input than MUSIC-I needs, greater than three orders of magnitude ($\approx$2000 : 1). On the other hand, MUSIC-I produces more output data, but in this case the difference is less ($\approx$10 : 1).

The total input requirements of MIST and the output requirements of MUSIC-I are relatively large even for this small 2D example. Due to the dependency on system parameters, these storage requirements increase significantly when a larger (or 3D) image is generated. Since the size of on-chip memories in accelerators has some inherent limitations [Cota et al. 2015], the processing must be decomposed in a sequence of computations performed on many smaller portions of the image.

To summarize, the critical section of MIST is much simpler than that of MUSIC-I in terms of complexity of the involved operations, but it manages a much larger dataset. As a result, the two kernels have a very different ratio of computation and communication. These differences are reflected in our hardware implementations of

the two critical sections. As we shall see, the execution time of our MIST accelerator is limited by the I/O bandwidth, whereas that of the MUSIC-I accelerator is limited by the actual parallelism obtained given a constraint on the number of synthesizable hardware resources.

3.3. Fixed-Point Approximation

While software versions of the two algorithms operate on double-precision floating-point values, a fixed-point representation is more convenient for a custom hardware implementation.[3] To determine the best data representation for the two algorithms, we ported the MATLAB code of Listings 1 and 2 to SystemC, and we built a flexible simulation environment in which all data widths of the two models could be set by means of C++ *traits*. With this framework, we performed an accuracy study, experimenting with various data widths and fractional point positions. As input data, we used a set of numerical breast models obtained from electromagnetic simulations.

We selected the minimum data widths that produce an error less than 1% in all pixels of output images, with respect to references obtained with 64-bit floating-point operations. This approximation is acceptable because radar imaging just aims to highlight the points that scatter the incident electromagnetic field the most. The value of 1% was set based on the experience that we gained working with specialists in this field. For MIST, we achieved the required precision by representing inputs as 16-bit signed words and outputs as 64-bit values. MUSIC-I requires more fractional digits to avoid underflow in the computation of trigonometric functions and square roots. Thus, we chose a 32-bit input/output width. For both algorithms, intermediate calculations use as many bits as needed to avoid overflow.[4] These widths are obtained by analyzing the propagation of worst-case scenarios through the datapath. SystemC eases the implementation of mixed-width datapaths because both integer and fixed-point classes in this language include methods to manage width extension/reduction as well as overflow/underflow, thereby abstracting the hardware details for the user [IEEE 2011].

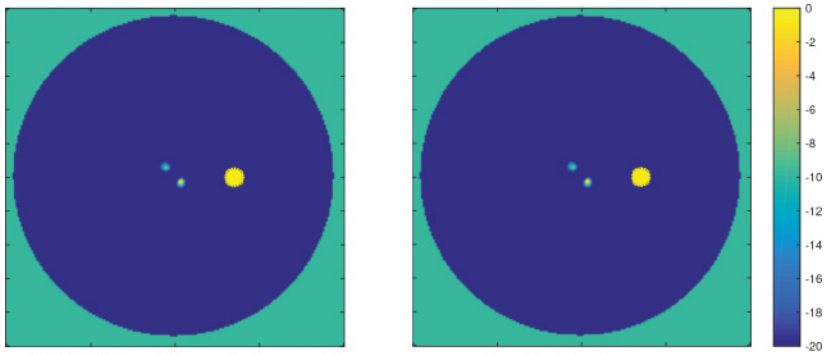
The accuracy study has been performed manually, but it can be easily scripted by gradually increasing the bit widths, until all simulations reach the required precision. The execution time of this characterization phase depends on the size of the available dataset of images. However, thanks to the efficiency of untimed SystemC simulation, the time required to compute the fixed-point output for each image has the same order of magnitude of the software execution time of the two algorithms, detailed in Section 3.1.

Figure 2 reports an example of two MUSIC-I images reconstructed with floating-point and fixed-point operations, respectively; notice that there is no appreciable difference between the two cases. Similar results are obtained for MIST, not reported for brevity.

3.4. Microarchitecture

Figures 3 and 4 show high-level views of the accelerators. The two microarchitectures that we devised exploit the intrinsic parallelism that both algorithms offer at the voxel level, as evident in Listings 1 and 2. The accelerators are composed of a series of parallel subblocks (blue rectangles in Figures 3 and 4) that operate concurrently on different sets of input data. Computations in subblocks are scheduled by a top-level control unit, composed of several concurrent SystemC threads, which also manage I/O

[3]We also verified that a fixed-point software implementation of the two algorithms is not faster than a double-precision one. Thus, moving to fixed-point does not remove the need for hardware acceleration.
[4]As will be clear in Section 3.6, since all the operations are replaced by combinations of additions, subtractions, shifts, and multiplications, it is easy to compute the number of bits needed to avoid overflow.



(a) 64-bit floating-point operations. (b) Fixed-point operations.

Fig. 2. (a) Floating-point vs. (b) fixed-point approximation for a MUSIC-I breast image. The color gradient shows the most reflecting points and highlights the presence of a highly reflective tumor.

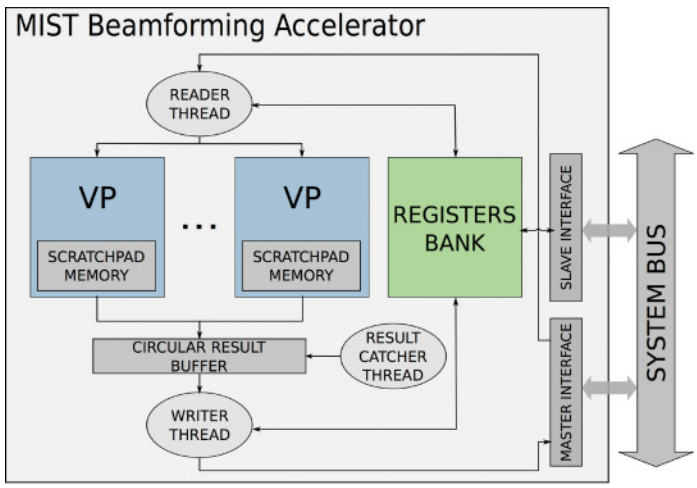


Fig. 3. Accelerator architecture for MIST Beamforming.

interfaces between the accelerator and the rest of the system (Figure 1), as detailed in Section 3.5.

3.4.1. Implicit versus Explicit Parallelism. In behavioral modeling, the ideal way to achieve parallelism would be to describe the entire processing part of the accelerators as a single sequential SystemC loop, and then use the *partial unrolling* features of the HLS tool to unfold some of the iterations during scheduling. However, commercial HLS tools still present some limitations when this type of *implicit* parallelism is pursued.

In particular, limitations are related to memory conflicts, which happen when *shared data structures* are accessed in parallel loop iterations. If these data structures are mapped to on-chip Static Random-Access Memory (SRAM) during synthesis (i.e., the optimal choice for area and power reduction in large arrays), parallelism is limited by the number of ports of the available memory IPs. Theoretically, a higher degree of parallelism can be achieved if data structures are *partitioned* and stored in a number of smaller SRAMs (when parallel accesses are *nonoverlapping* by construction), or *replicated* (when accesses instead *overlap)*. Commercial HLS tools, however, can automatically infer partitioning and replication only in straightforward cases. On the other

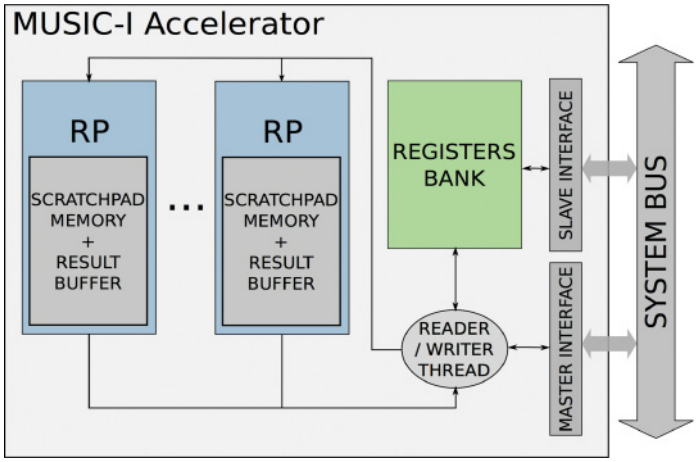


Fig. 4. Accelerator architecture for MUSIC-I.

Table IV. Data Structures of Listings 1 and 2 Classified in Terms of Access Concurrency

	Overlapping Access	Nonoverlapping Access
MIST Beamforming	`x`	`w,d,en`
MUSIC-I	`xo, yo, kb, u1`	`x,y,F`

hand, forcing *every* memory element to be mapped to flip-flop registers to maximize the access parallelism, ruling out the possibility to use SRAM blocks, would not only severely limit the design-space exploration, but also most likely result in suboptimal implementations.

To avoid these issues, we specified parallelism *explicitly* in the two accelerators, by instantiating *subblocks* in the high-level description (VP and RP blocks in Figures 3 and 4). The resulting SystemC code is *structural* at the top level, due to the explicit instances of the parallel subblocks, and *behavioral* at the bottom level, due to the untimed specification of the internal microarchitecture of each subblock.

The number of subblocks can be set at synthesis time by modifying proper preprocessor constants. Therefore, this number effectively becomes an additional knob for design-space exploration, complementary to those provided by the HLS tool such as pipelining, fine-grain parallelization (inside the subblocks), and resource sharing.

3.4.2. Parallel Subblocks Granularity. Besides computational logic, each subblock contains private *scratchpad memories* to store the input and output data relative to the computations scheduled to it. To determine the granularity of the task performed by each subblock, we considered the trade-off between scratchpad memory size and complexity in the top-level controller of the accelerators: typically, a larger scratchpad footprint corresponds to a lower control overhead at the top level, and vice versa.

Scratchpad memory size depends not only on the input and output datasets required by the two algorithms to process a voxel, previously presented in Table II, but also on the data sharing among different voxels. As pointed out in Section 3.4.1, Listings 1 and 2 contain data structures that are accessed during the computation of multiple voxels. Some are accessed in a *nonoverlapping* way, that is, different elements in the arrays are related to different voxels. These data can be split and distributed in chunks to the subblocks. Some data structures, on the contrary, are accessed in an *overlapping* way: these must be replicated and stored in each subblock to avoid conflicts. Table IV provides a summary of data sharing in the two algorithms.

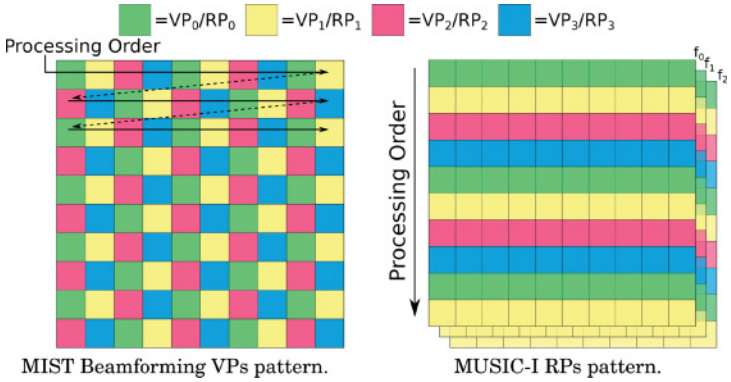


Fig. 5. Example of processing patterns in MIST Beamforming and MUSIC-I, for a 10×10 image. Both accelerators have four parallel elements ($VP/RP_{0:3}$), and MUSIC-I operates on three different frequencies ($f_{0:2}$).

In MIST, weights w and delays d are the largest inputs, and they are not shared among voxels. Samples x, instead, have a smaller size and are shared among all voxels. The large size of w and d calls for a voxel-level partitioning among parallel elements, therefore named *Voxel Processors (VPs)*. To avoid access conflicts, each VP receives a copy of the samples at the beginning of computation. Since the samples size is small, the overhead due to data replication is negligible. Weights and delays, instead, are distributed cyclically to VPs, any time a new voxel must be computed.

In MUSIC-I, the smaller size of the dataset and the higher data sharing call for a coarser granularity in the assignment of processing tasks to parallel elements. Therefore, the MUSIC-I accelerator is composed of a series of *Row Processors (RPs)*, each dedicated to computing one or more *rows* of matrix F. Each RP receives a copy of shared arrays xo, yo, kb, and V at the beginning of the computation. Since every block operates on separate row ranges, a suitable partitioning in subarrays is possible for the remaining inputs x and y, thereby avoiding data duplication. The larger size of the atomic processing unit in MUSIC-I reduces the control overhead in the top-level controller of the accelerator. An even coarser granularity (e.g., computational blocks dedicated to computing the entire F matrix, each on a different frequency) would require an excessively large output scratchpad memory for each subblock.

Figure 5 shows the sequence of processing tasks for the two accelerators, for a small 2D example. Arrows indicate the order in which locations of the output image/frequency plane are processed. Colors indicate assignment of computations to parallel processing elements. For each assigned task, the top-level control unit of the accelerator writes the corresponding input dataset in the VP/RP scratchpad memory, triggers the VP/RP execution, and collects the result(s) at the end of the computation.

3.5. Processor-Accelerator Communication

As shown in Figures 3 and 4, the accelerators have two interfaces to the main system bus. Through the *slave* interface, the embedded processor accesses a bank of memory-mapped registers inside the accelerators, in which the processor writes commands and reads the execution status. Through the *master* bus interface, the two accelerators transfer data to and from the main system memory in Direct Memory Access (DMA).

3.5.1. TLM Modeling. We developed the bus interfaces using a synthesizable Transaction-Level Modeling (TLM) library. Compared to signal-level modeling, TLM offers the advantage of abstracting the bus protocol, by modeling transactions as generic

read and write operations on blocking FIFO queues. Besides simplifying the design phase, abstract primitives also enable the portability of the designs to a different bus (or to a network-on-chip) with minimal or no code rewriting.

In particular, for the experiments described in Section 4, we mapped the agnostic TLM protocol to a signal-level implementation of the AMBA AXI3 standard, which is the interface adopted in our FPGA prototyping platform.

3.5.2. Sequence of Operations. Computation and I/O are orchestrated by the top-level controller of the two accelerators. In particular, the controller reacts to changes in the contents of the register bank. Before activating the accelerators, the system CPU writes in the appropriate registers the following information:

—the number of items (voxels or rows) to be processed;
—the physical *source address* in memory from which inputs are fetched via DMA;
—the *destination address* where outputs have to be saved, also via DMA;
—A command, written in the *control register*, indicating relevant details on the type of computation to be performed (e.g., if the inputs refer to new voxels of a previous image, or to a new image).

When the *start bit* of the control register is asserted, the SystemC threads of the controller perform the following actions in sequence: (1) start fetching input data from the main memory using the master bus interface; (2) distribute the workload among VPs/RPs, stalling the I/O whenever all processing elements are busy; (3) write back the results in main memory as soon as they are available. As we explain in Section 3.5.3, these actions are pipelined so that computations and DMA transfers perfectly overlap.

When all the computed outputs have been saved, the controller alerts the CPU with an interrupt. In case an error occurs, this information is made available to the CPU in an additional *status register*. This "batch" scheme limits the interaction between CPU and accelerator to the initial and final phases of the computation.

3.5.3. Concurrent Computation and Communication. To minimize the execution time, we focused on the optimization not only of the accelerators datapath, but also of the control and the synchronization. In particular, the pipelining between computation and communication phases has been thoroughly crafted to minimize idle phases.

The first optimization aims to avoid stalling the VPs and RPs computation during the long I/O transfers of the large input dataset in MIST and the large output dataset in MUSIC-I. Since our accelerators have a single master interface, these data transfers happen *serially*, and are managed by the top-level controller. To cope with this issue, we added the optional inclusion (to be decided at synthesis time) of double buffers (aka *ping-pong buffers*) to be instantiated in VPs input and RPs output scratchpad memories, respectively. These buffers, which are shown in Figure 6 with a 0/1 index, are only useful for those data that have to be renewed every time a new computation starts (i.e., data that are *distributed* among subblocks, as explained in Section 3.4.1). Since their insertion is enabled or disabled by a SystemC preprocessor constant, the presence of ping-pong buffers becomes another design-space exploration knob.

Another improvement consists in buffering the output data and transferring them to the main memory in *bursts*. In MIST, transferring the single energy output produced by each VP would be suboptimal (due to the bus overhead). Therefore, results are stored in a circular buffer and transferred to the bus as soon as a sufficient number of elements for a burst is ready. On the other hand, output buffering is not needed in MUSIC-I: since the number of pixels in a row of matrix F is typically larger than the maximum AXI burst length, there is no need to group outputs before a DMA write.

The previously described solutions affect the complexity of the control part in the two accelerators. In MIST, two main concurrent threads are needed to manage computation

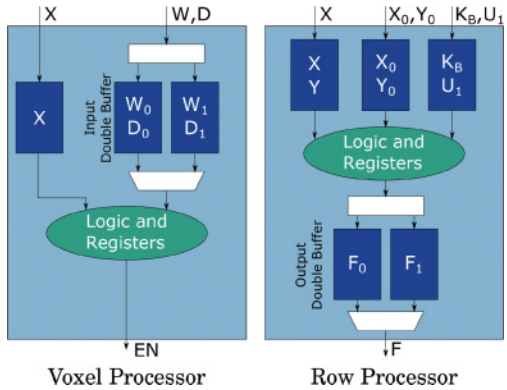


Fig. 6. Internals of the parallel computing elements of the two accelerators.

and I/O. The *reader* logic copies new input data to the VPs scratchpads via DMA, and triggers their execution; the *writer* logic gathers VP results and performs output bus transfers. Internal signal-level synchronization avoids data loss or overwriting when the output buffer fills up or when VPs are not ready to receive new data (both with and without a ping-pong scheme). Two threads are needed because both input and output transfers happen continuously during the operation of the accelerator. Burst length and circular buffer size are generic parameters to be set at synthesis time.

Control and synchronization are simpler in MUSIC-I, because all rows of matrix F for a given frequency are obtained with the *same* small set of input data. Consequently, after a single initial input burst, all RPs can start computing together. As we explained, there is also no need for buffering output transfers. In summary, a single control thread is sufficient to manage both I/O directions.

In conclusion, MUSIC-I RPs are internally more complex than MIST VPs, but MIST uses a more advanced communication and synchronization scheme. This difference is reflected in the experimental results reported in Section 4.

Simplified timing diagrams displaying the typical sequence of operations performed in the two accelerators are shown in Figures 7 and 8. The diagrams summarize the interaction between the system's processor (CPU), the accelerators controllers (CTL), and parallel processing units ($RP_{0:3}$/$VP_{0:3}$), focusing in particular on the beginning and on the end of the hardware operations. Arrows indicate some of the causality relations among operations. Two cases are shown: with and without double buffering.

The advantages of doubling critical scratchpad memories are evident from the pictures. In the MIST Beamforming case shown in Figure 7, without double buffers the long DMA transfers needed to fetch the large input data (recall Table II) can start only when at least one VP has completed its assigned computation. Otherwise, the input data on which VPs are working would be overwritten, creating errors. In the example of Figure 7, after the first four sets of inputs have been transferred to VPs scratchpads, the controller has to wait for VP_0 to finish its processing, before starting a new DMA read. This reduces the utilization of VPs, as shown by the empty time intervals between two successive processing phases. (Output transfers in MIST Beamforming can happen concurrently with input reads, and do not introduce further time overheads.) On the contrary, with double buffers the controller can start fetching the inputs relative to voxel no. 5 while VP_0 is still processing voxel no. 0, copying these new inputs in the "inactive" window of its double buffer. Consequently, the idle phases of all VPs are either reduced or completely eliminated, depending on the latencies of processing and I/O, which are a consequence of synthesis choices.

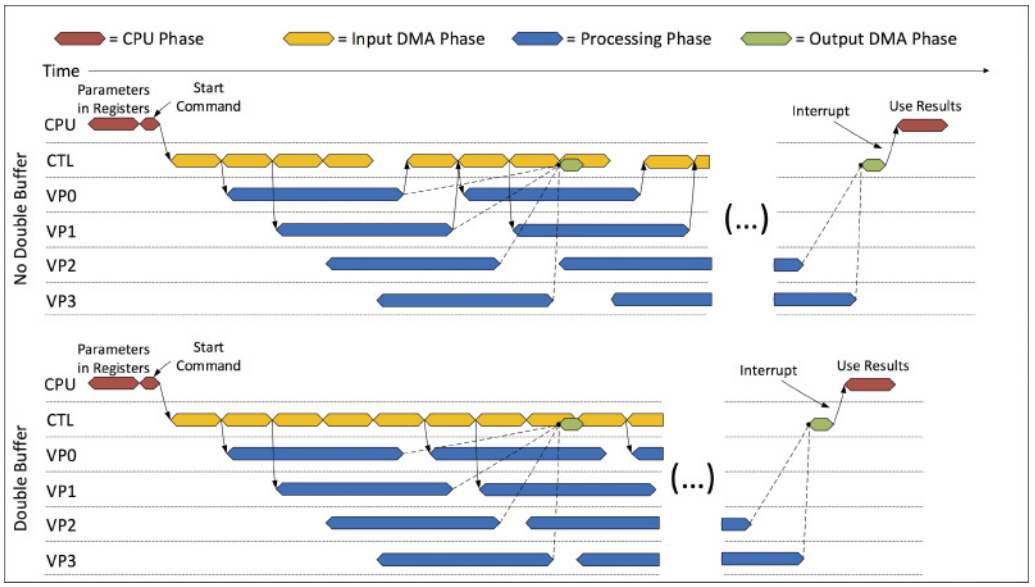


Fig. 7. Timing diagram of the execution of the MIST Beamforming accelerator.

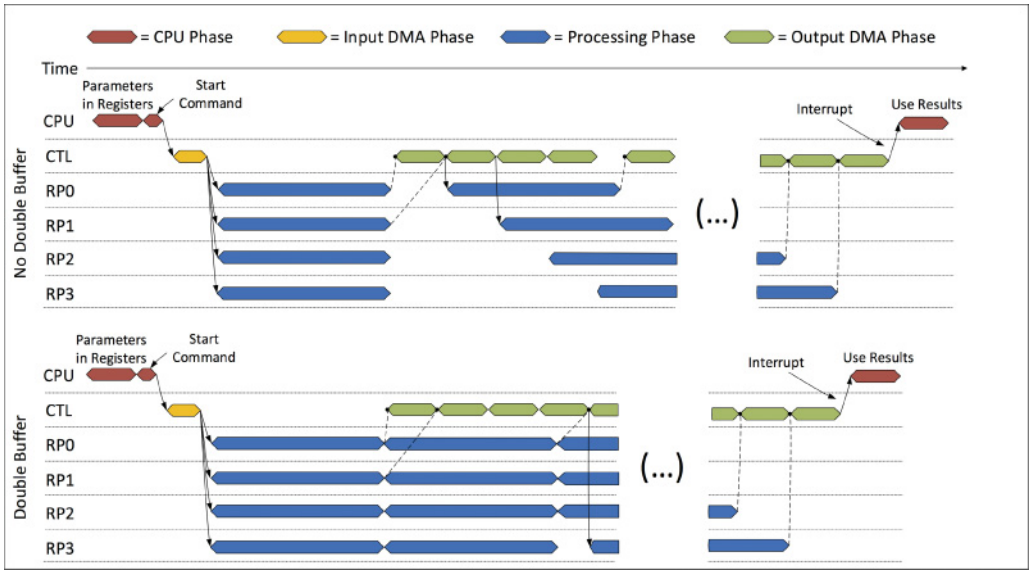


Fig. 8. Timing diagram of the execution of the MUSIC-I accelerator.

In the case of MUSIC-I, shown in Figure 8, the benefits of double buffers are even more evident. Here, the critical DMA phases are related to output. The difference with respect to MIST is that all RPs start processing together (since inputs are small and mostly shared). However, when one RP without double buffers has completed its processing, it must wait until the controller has transmitted all outputs to the main system memory before restarting, to avoid overwriting results. Since the accelerator has a single DMA interface, the controller serializes output transfers related to different

RPs, which creates very long idle phases. In the example, the worst case is for RP_3, which has to wait for a time equal to the sum of all output transfers on all RPs. With double buffers, instead, each RP immediately starts producing a second row of outputs by switching the buffer used for writing results. Meanwhile, the controller can free the first buffer with an output DMA burst. Again, depending on the relation between processing and I/O latencies, this reduces or eliminates idleness in RPs.

3.6. Hardware Implementation of Mathematical Kernels in MUSIC-I Critical Section

The math of the critical section of MUSIC-I is nontrivial from a hardware-implementation viewpoint. Most commercial HLS tools do not provide implementations of the operations required (division, square root, exponential, and trigonometric functions). Therefore, we implemented a SystemC library that maps each operation to one or more synthesizable specifications of known algorithms. When multiple specifications are available, a preprocessor constant allows the user to select among them. After this selection, the implementation can be further customized using standard HLS strategies (e.g., loop unrolling/pipelining, inlining/sharing of the function, etc).

This solution provides more flexibility than using a "fixed" RTL IP for each operation. For example, we can easily make different optimization choices for our two different targets (FPGA and ASIC), by trading off resource usage with the operation latency.

For trigonometric functions (required by the complex exponential in the critical section of MUSIC-I), we used the CORDIC algorithm [Parhami 2010], which we extended to accept any input angle, since in our application the input is not limited to $[-\frac{\pi}{2}, \frac{\pi}{2}]$. This extension can be achieved using trigonometric equalities to convert every angle to one that stands in the validity range of CORDIC, and then modifying the result accordingly. For instance, for an angle θ between $\frac{\pi}{2}$ and π, it is sufficient to use $\theta' = (\theta - \pi)$ as input for CORDIC, since $\cos(\theta') = -\cos(\theta)$. In implementing this extension, we made sure to only use loops with a fixed number of iterations, independent from the input angle. The reason is that unbounded loops limit the exploration capabilities of HLS tools, preventing unrolling and pipelining. To extend CORDIC to *any* angle with a fixed number of operations we had to use a multiplication, whereas the classic version of the algorithm only contains sums, shifts, and accesses to a Look-Up Table (LUT). However, the additional cost can be amortized by other multiplications performed in the accelerator. In fact, resource sharing between a mathematical kernel and the rest of the hardware is possible, as long as the corresponding function call is inlined.

For the real part of the exponential, we used a recursive algorithm that converges in a number of iterations that is logarithmic in the operands width [Koren 2002]. The basic version of this algorithm requires additions, shifts, and LUT accesses. However, like for CORDIC, MUSIC-I data cannot be bound in the convergence range of the recursion. The extension to any bit width, presented in Koren [2002], requires multiplications.

For division and square root, we implemented two alternative algorithms. The basic, "slow" versions use shift-and-subtraction algorithms [Parhami 2010]; the "fast" solutions are based on a convergent recursion equation known as the Goldschmidt algorithm [Koren 2002]. Slow algorithms only use additions and shifts, but they require a number of operations linear in the operands width. Fast versions, instead, use multiplications and LUTs, but the execution has logarithmic time complexity.

It must be noted that the number of iterations in the behavioral specifications of the mathematical kernels does not necessarily correspond to the clock cycles latency in the resulting hardware. Since specifications are untimed, the HLS tool allocates clock cycles to operations only at synthesis time, and can obtain very different results depending on the target technology. Moreover, latency, as well as the amount of resource sharing,

also depend on the scheduling and binding directives (e.g., unrolling, pipelining, etc.) set by the library user at synthesis time as well as other HLS optimizations.

4. EXPERIMENTAL RESULTS

4.1. Setup

We experimented with two different technology targets, a Xilinx Artix-7 FPGA and a 32-nm CMOS standard-cell ASIC library. The FPGA target was chosen primarily for validating the design on a prototype platform based on the Xilinx Zynq XC7Z702 SoPC, which is composed of a dual-core ARM Cortex-A9 and the Artix-7 Programmable Logic (PL) [Xilinx 2013]. The Zynq system bus implements the AXI standard, and has slave and master ports accessible from the PL. Therefore, we could validate all FPGA implementations of our designs from a full-system perspective, connecting the accelerated parts with the rest of the reconstruction algorithm, implemented in software on the Cortex-A9. To complete the HW/SW stack, we wrote a custom device driver for each accelerator on the Linux Operating System, kernel version 3.13.0.

By using Cadence CtoSilicon, a commercial HLS tool, we performed a design-space exploration (DSE) as detailed in the following subsections to determine the Pareto set of optimal implementations. Then, starting from the HLS-generated RTL code of the two accelerators, we obtained gate-level netlists through logic synthesis.

For the FPGA technology, we used Xilinx Vivado Design Suite. The different use of PL resources by the two kernels, especially DSP blocks, affected the place-and-route step. Consequently, we could synthesize the MIST accelerator at a frequency of 75MHz and MUSIC-I at 50MHz. As a cost metric, we used the percentage of occupation of the most critical resource type on the PL. In most MUSIC-I implementations these resources are DSP blocks, while in MIST Beamforming they are LUTs. To measure performance, we profiled the execution time of the complete reconstruction algorithm on the Zynq, including software parts and driver overheads. Finally, we also gathered power consumption data provided by Vivado.

With standard-cell ASIC experiments, we aimed to evaluate the performance of a hypothetical SoC implementation. We used Synopsys Design Compiler (DC) for the logic synthesis step, and we successfully reached a 1-GHz clock frequency for both accelerators. Cost was measured as the silicon area (in mm^2) and execution time was obtained via logic simulation. Power data were also estimated with Synopsys DC.

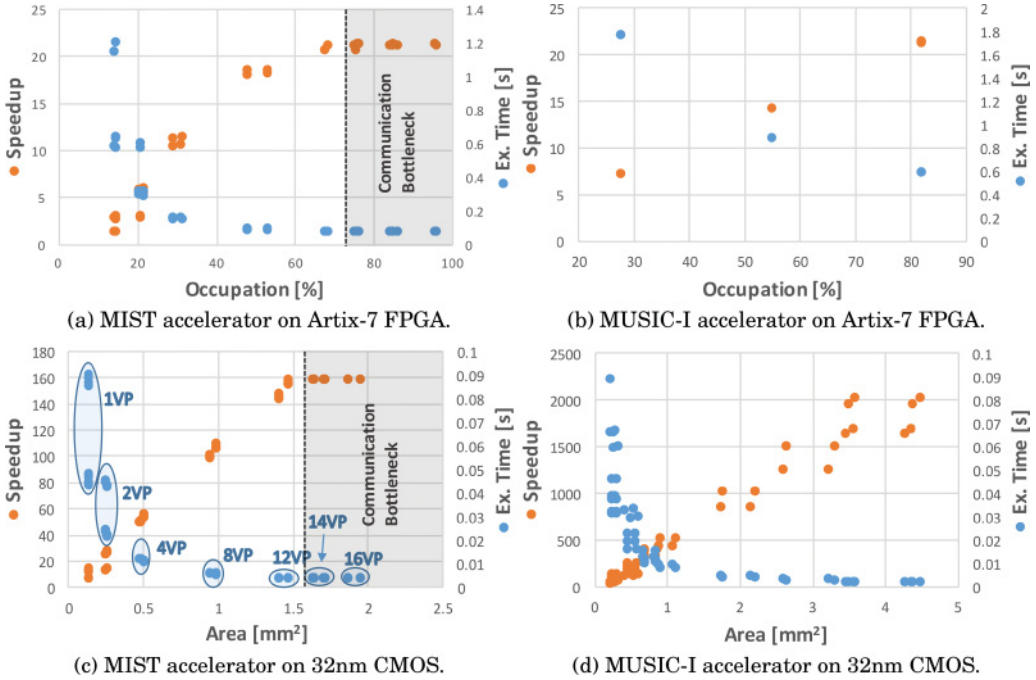
The quality of the accelerators design can be conveniently evaluated with metrics such as execution time and power consumption. Rather than absolute values, however, it is more meaningful to evaluate them in relative terms using the nonaccelerated software version as a baseline.

For the execution time, FPGA implementations have been compared with a software running on the Zynq Cortex-A9 CPU at 667MHz. ASIC results were evaluated against the execution time of an identical implementation of the Cortex-A9 in a 32-nm CMOS technology. To determine the clock frequency for this case, we used the tool McPAT [Li et al. 2013], which returns estimates on power, area, and timing given a processor description in terms of technology, architecture, and workload. McPAT already contains a 32-nm model of the Cortex-A9 that is estimated to run at 2GHz.

To estimate the software power consumption, we first gathered workload statistics needed by McPAT by profiling the compiled binaries of the software versions of MIST and MUSIC-I. Thanks to the detailed power consumption breakdown provided by McPAT, we could isolate the contribution of each algorithm, removing the common power that the processor would consume anyway, also when accelerators are used. This contribution is the baseline to which accelerators power consumption is compared. For the FPGA implementation, we rescaled the McPAT model parameters to the 28-nm

Table V. Software Execution Time and Average Power. MIST Parameters: $N_A = 9$, $L = 55$, 152×176 2D Image. MUSIC-I Parameters: $N_A = 18$, $N_{FREQ} = 20$, 200×200 2D Image

	FPGA Cortex-A9 @667MHz		ASIC Cortex-A9 @2GHz	
Algorithm	**Exec. time (s)**	**Power (mW)**	**Exec. time (s)**	**Power (mW)**
MIST	1.80	42.10	0.60	161.82
MUSIC-I	12.72	41.80	4.24	160.70



(a) MIST accelerator on Artix-7 FPGA.

(b) MUSIC-I accelerator on Artix-7 FPGA.

(c) MIST accelerator on 32nm CMOS.

(d) MUSIC-I accelerator on 32nm CMOS.

Fig. 9. Accelerators performance vs. area design space; same parameters of software versions (see Table V).

technology in which the Xilinx Zynq is fabricated, and set the clock frequency to 667MHz. For the ASIC implementation we used directly the 32-nm model.

Table V reports baseline execution time and power consumption of the software version of the algorithms, for both FPGA and ASIC. The algorithm parameters are reported in the table caption. Note that the relatively short execution time is a consequence of the small image size used for these experiments, and that differences between MIST and MUSIC-I are in part due to the values of the parameters.

4.2. Design-Space Exploration

For both technologies, we performed a designer-driven DSE by setting the HLS knobs (primarily loop breaking, unrolling, or pipelining, function inlining, memory flattening on registers or SRAM instantiation) as well as the previously mentioned custom architectural configurations (e.g., amount of VP/RP parallelism, choice between fast and slow math kernels, and double-buffer insertion). After pruning Pareto-dominated implementations, we obtained 80 Pareto-optimal design points for MIST Beamforming and 104 for MUSIC-I. *All these alternative implementations were obtained from a single SystemC specification for each accelerator.*

Figure 9 reports some of these results in the performance versus cost design space. The vertical axes report both the absolute execution time and the speedup with respect

Table VI. Minimum and Maximum Design Metrics Obtained on FPGA from a Single SystemC Specification

	Exec. Time (ms) (HW/SW Speedup)		Occupation (%)		Power (mW) (HW/SW Ratio)		Energy (mJ) (HW/SW Ratio)		EDP (Js·10^{-3}) (HW/SW Ratio)	
Algorithm	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
MIST	84.43 (21.42)	1,207.63 (1.49)	13.9	95.9	25 (0.59)	387 (9.19)	7.87 (0.1)	32.08 (0.42)	0.76 (0.006)	38.74 (0.28)
MUSIC-I	595.07 (21.38)	1,765.92 (7.20)	27.3	100	55 (0.24)	309 (7.39)	125.38 (0.24)	138.53 (0.26)	78.26 (0.01)	221.41 (0.03)

to software. The horizontal axes report occupation (in percentage) for the FPGA implementations, and silicon area (in mm^2) for the ASIC ones. Parameters used for the experiments are the same as those used for the software evaluation reported in Table V. Note that operations performed in both accelerators are data independent (Listings 1 and 2), so these graphs are valid for any set of input samples of the same size.

The FPGA Pareto curve of MIST Beamforming in Figure 9(a) is much denser in points than the MUSIC-I one in Figure 9(b). The reason is that VPs are simpler and smaller than RPs, hence it is possible to allocate from 1 to 16 of them without exceeding the FPGA resources. On the contrary, all MUSIC-I implementations with more than three RPs exceed the available DSP blocks and LUTs. Moreover, since the FPGA area metric considers a single resource type, most solutions were discarded as Pareto dominated: for instance, solutions without ping-pong buffers were pruned because they use the same number of critical resources (LUTs/DSPs) but execute more slowly. The maximum achieved speedup is similar for the two FPGA accelerators, close to 25×.

ASIC implementations better highlight the flexibility and scalability of our mixed structural-behavioral description. In Figure 9(c) we grouped together MIST solutions with the same number of VPs, to show that by changing this number we can easily reach the desired portion of the design space. Then, we can achieve a finer tuning by modifying the internals of each VP with HLS tool directives. In MUSIC-I, as shown in Figure 9(d), the number of solutions is even larger because the higher complexity of RPs enables more flexibility. Compared to the FPGA case in Figure 9(b), many more ASIC solutions are not Pareto dominated (e.g., those without ping-pong buffers).

The MIST Beamforming speedup curves in Figures 9(a) and 9(c) show an interesting behavior, highlighted by gray regions in the graphs: beyond a certain threshold, adding more VPs or reducing their latency does not improve performance. The performance limiter is the bandwidth of the communication bus: beyond a certain number of VPs, it becomes impossible to fetch data at a rate sufficiently high to keep them all busy. The threshold depends on the chosen bus protocol, the clock frequency, and the target technology. With our selected targets and the AXI bus, we found that the maximum useful number of VPs is 14 for FPGA and 12 for ASIC (Figures 9(a) and 9(c), respectively).

The speedup in MUSIC-I, instead, increases linearly with the number of RPs. Clearly, a similar threshold exists but is not likely to be reached in any realistic case due to the much lighter I/O requirements. The I/O bound limits MIST speed up to 160×, whereas MUSIC-I with 16 RPs reaches the 2,000× mark. Interestingly, the speedup for a given area is five times larger for MUSIC-I (e.g., 500× at $1mm^2$ vs. 100× of MIST).

Tables VI and VII summarize the DSE results reporting the minimum and maximum values of the main metrics over the whole set of Pareto-optimal implementations. Besides execution time and area occupation, these metrics include average power consumption, total energy consumed during the execution time, and the product of energy and execution time (energy-delay product, aka EDP metric). Power and energy values refer to the accelerator in its entirety, including control logic and I/O interfaces.

The metrics show that performance and area vary by more than one order of magnitude between minimum and maximum, for both accelerators and both target

Table VII. Minimum and Maximum Design Metrics Obtained on a 32-nm ASIC from a Single SystemC Specification

	Exec. Time (ms) (HW/SW Speedup)		Area (mm^2)		Power (mW) (HW/SW Ratio)		Energy (mJ) (HW/SW Ratio)		EDP ($Js \cdot 10^{-3}$) (HW/SW Ratio)	
Algorithm	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
MIST	3.79 (159.12)	90.51 (6.66)	0.13	1.95	17.58 (0.11)	173.68 (1.07)	0.49 (0.005)	1.59 (0.016)	0.002 ($3.2 \cdot 10^{-5}$)	0.14 (0.002)
MUSIC-I	2.1 (2,023.61)	89.49 (47.40)	0.206	4.47	37.31 (0.23)	1,353.2 (8.42)	2.09 (0.003)	4.80 (0.007)	0.005 ($1.8 \cdot 10^{-6}$)	0.43 ($1.5 \cdot 10^{-4}$)

Table VIII. Value of Main HLS Knobs Required to Obtain the Optimal Solutions

MUSIC-I ASIC	**Parall.**	**Buffer**	**Memory**	**Main Loop**	**Math**
Best Performance	16 RP	double	registers	pipelined	fast
Best Power	1 RP	single	SRAM	sequential	slow
Best Energy	8 RP	single	SRAM	pipelined	fast
Best EDP	16 RP	single	SRAM	pipelined	fast
Best Area	1 RP	single	registers	sequential	slow
MUSIC-I FPGA	**Parall.**	**Buffer**	**Memory**	**Main Loop**	**Math**
Best Perf./EDP	3 RP	double	BRAM	pipelined	fast
Best Area/Pow./En.	1 RP	single	BRAM	sequential	slow
MIST ASIC	**Parall.**	**Buffer**	**Memory**	**Main Loop**	
Best Perf/En./EDP	12 VP	double	SRAM	pipelined	
Best Pow./Area	1 VP	single	SRAM	pipelined	
MIST FPGA	**Parall.**	**Buffer**	**Memory**	**Main Loop**	
Best Perf.	14 VP	double	BRAM	pipelined	
Best Pow./Area	1 VP	single	BRAM	pipelined	
Best En./EDP	8 VP	double	BRAM	pipelined	

technologies. This is an indicator of how different the nondominated solutions are, thereby offering a rich set of options for the design reuse of each accelerator across different systems. Particularly remarkable is that all these implementations were obtained starting from the same SystemC specification.

Moreover, although the largest implementations of the two accelerators consume more power than the software versions, the total *energy* consumed is always less than 50% for FPGA and 2% for ASIC, thanks to the reduced execution time. The quality of the performance-energy trade-off using accelerators is summarized by the very low values of the EDP metric. These results confirm that, although high performance implementations could be obtained with a high-end server-class processor or a GPU, a hybrid architecture with an embedded CPU and accelerators is particularly suitable for applications that need cost-efficient and energy-efficient solutions.

The radar charts in Figure 10 are obtained selecting five implementations that are optimal for at least one of the considered design metrics (execution time, power, area, energy, and EDP). All quantities are normalized so that the optimal value for a given dimension (i.e., maximum performance, minimum area, power, energy, and EDP) corresponds to the edge of the chart. The aim of the chart is to help identify the best overall solutions by graphically highlighting those implementations that tend to stay on the outer edges of the chart, being optimal or close to optimal in more than one metric. Table VIII shows the main HLS knobs used to obtain these five optimal solutions.

As expected, accelerators with minimum area or power are also those with lower performance (low parallelism, no pipelining), and vice versa. Interestingly enough, the charts and the values of the knobs in Table VIII reveal that solutions with high

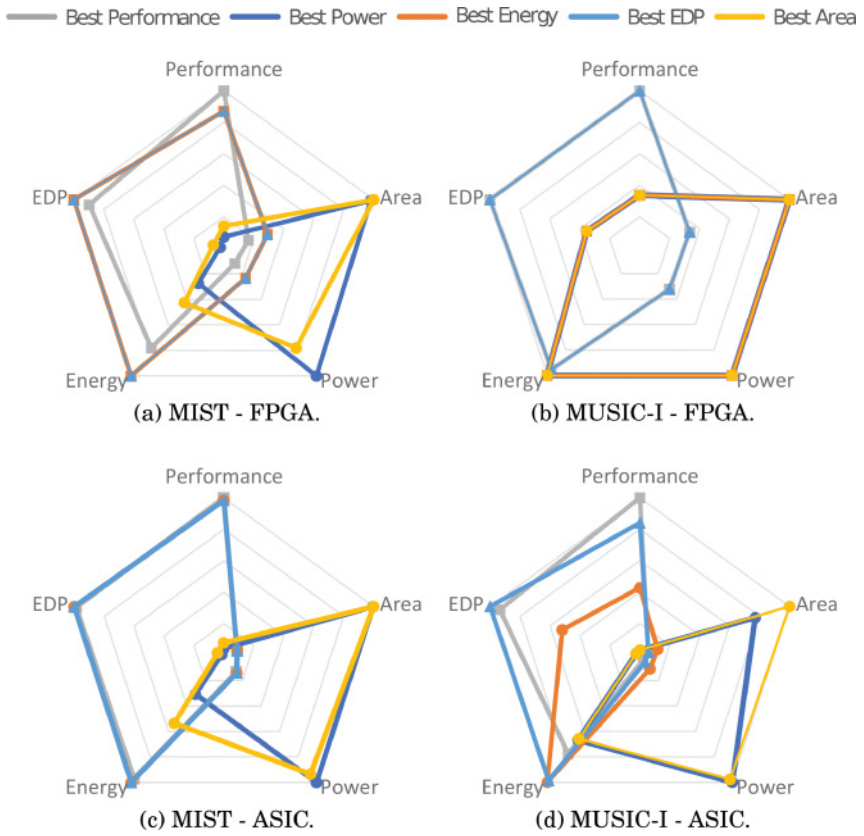


Fig. 10. "Optimal" implementations of the two accelerators for each optimization metric.

Table IX. Best MIST Execution Time Achieved on Zynq Technology with RTL and SystemC Accelerators

RTL Accelerator (s)	SystemC Accelerator (s)	Speedup
0.257	0.08443	3.04

parallelism, double buffers, aggressive loop pipelining, and fast math functions (for MUSIC-I) are not only optimal performancewise, but also optimal or close to optimal in terms of energy and EDP. We conclude that accelerators with the highest parallelism (and, therefore, the largest size) are the most energy-efficient solutions for this particular application: compared to solutions with lower parallelism, they can efficiently utilize the larger silicon area and power to increase the performance at low energy and EDP.

4.3. Comparison with RTL Accelerator for MIST Beamforming

As a final experiment, we evaluated the effectiveness of the HLS methodology against the RTL MIST Beamforming accelerator described in Casu et al. [2014]. Since the RTL code was kindly made available to us by the authors, we could use it to compare the results after resynthesizing it for the same FPGA adopted in our experiments. We obtained that for similar occupation metrics the design synthesized with HLS has a speedup larger than 3× with respect to the RTL design, as reported in Table IX.

Clearly, this result does not mean that it is not possible to obtain an equally good solution directly in RTL. Rather, it demonstrates how HLS makes it possible to perform a broad design-space exploration in a way that is much more efficient than RTL design. In fact, we could explore, evaluate, execute on FPGA, and synthesize on ASIC, 80 (MIST) and 104 (MUSIC-I) alternative implementations in only 4 months. The comparison with the time to design and test a single RTL implementation of MIST Beamforming, which required 3 months as the authors of Casu et al. [2014] communicated to us, confirms the progress reached by HLS tools.

5. RELATED WORK

Since microwave imaging for breast cancer detection is a relatively new field, so far researchers have mostly focused on developing effective detection algorithms rather than on their fast or power-efficient implementation on dedicated hardware. There are, however, a few noteworthy exceptions. In Xu et al. [2012], Shahzad et al. [2014], and Elahi et al. [2015], the authors recognize the need for hardware acceleration, both for nonlinear (i.e., tomography) and linear (i.e., radar) scattering inversion algorithms, as a key factor toward practical microwave-imaging systems for breast cancer detection. They take different routes toward that goal: In Xu et al. [2012], researchers accelerate the tomography algorithms on a Cell broadband engine processor, whereas in Shahzad et al. [2014] and Elahi et al. [2015] a GPU accelerator is targeted.

Although a GPU-based acceleration may lead to an acceptable execution time, it is not the best implementation choice for power-constrained or form-factor limited embedded environments. A dedicated hardware accelerator, based on either an FPGA or an ASIC technology, is more suited for these scenarios. For the MIST Beamforming algorithm, the authors of Casu et al. [2014] show that an implementation on a high-end FPGA outperforms those on a multicore CPU and on a GPU.

This is the first article to present a hardware acceleration of a MUSIC-based microwave-imaging algorithm. We are also the first to use HLS for implementing microwave-imaging algorithms. Ours is not the first article, however, to propose the adoption of HLS in the broader field of medical imaging. For image denoising, the authors of Hannig et al. [2010] propose a deeply pipelined and parallel FPGA architecture synthesized starting from a high-level description. In Cong et al. [2011] and Bui et al. [2012], a larger set of medical-imaging benchmarks (denoising, registration, segmentation, etc.) are used to determine the best domain-specific hardware platform and a HLS methodology is used for the implementation on an FPGA target platform. The same medical imaging benchmarks have been used in the literature on the development of accelerator-rich architectures, in which the medical-imaging accelerators are obtained from a HLS description [Cong et al. 2012]. In the context of ultrasound imaging for medical applications, the authors of Amaro et al. [2015] present an FPGA implementation of a synthetic-aperture imaging accelerator, which is obtained entirely with HLS. Another interesting application of HLS to the biomedical field is presented in Oppermann et al. [2015]: here the authors focus on using FPGA technology to accelerate the simulation of biological systems described by systems of differential equations.

For applications outside the biomedical field, HLS has been proven effective for obtaining hardware implementations of DSP algorithms, as shown in Liu et al. [2016] for a complex H.264 decoder, or in Bhatnagar et al. [2013] for a software-defined radio platform. In this field, designers often use a model-based approach using Simulink to first describe and then synthesize via HLS their hardware accelerators. For complex blocks (e.g., for FFT or DCT computation) the standard design flow using Simulink does not let designers fully explore the microarchitectural design space via HLS, a problem faced and solved by the authors of Butt et al. [2016].

The HLS limitations with respect to dealing with kernels that have complex memory access patterns have been addressed by other researchers in the literature. In Butt et al. [2014], the authors elegantly solved the problem of overlapping input data fetch and computation in their HLS-designed pseudolog image transform kernel. Similarly to our solution, a controller external to the computational kernel fetches data into on-chip buffers while computation is performed so as to fully exploit the I/O bandwidth.

Other recent applications of HLS include acceleration of pricing algorithms (using Black-Scholes or Heston model) [Inggs et al. 2015], database analytics [Malazgirt et al. 2015], and control algorithms for power electronic converters [Navarro et al. 2013].

6. CONCLUSIONS

We used HLS to complete a comprehensive design-space exploration of two microwave-imaging algorithms for breast cancer detection. Our results show that while MIST Beamforming remains an interesting solution for lower cost implementations, the recently proposed MUSIC-I method [Ruvio et al. 2013] is generally superior, offering higher scalability (due to the absence of a communication bottleneck) and higher flexibility to system parameter changes. More generally, our work shows the potential that HLS offers in terms of design-productivity gains: in the span of about 4 months one designer in our team with no prior knowledge of the two algorithms was able to specify, synthesize, and evaluate more than 100 alternative implementations (across FPGA and standard-cell technologies). These include implementations that outperform a previous RTL design of MIST.

REFERENCES

J. Amaro, B. Y. S. Yiu, G. Falcao, M. A. C. Gomes, and A. C. H. Yu. 2015. Software-based high-level synthesis design of FPGA beamformers for synthetic aperture imaging. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 62, 5 (May 2015), 862–870.

V. Bhatnagar, G. S. Ouedraogo, M. Gautier, A. Carer, and O. Sentieys. 2013. An FPGA software defined radio platform with a high-level synthesis design flow. In *Proceedings of the 2013 IEEE 77th Vehicular Technology Conference (VTC Spring)*. 1–5.

E. J. Bond, Xu Li, S. C. Hagness, and B. D. Van Veen. 2003. Microwave imaging via space-time beamforming for early detection of breast cancer. *IEEE Transactions on Antennas and Propagation* 51, 8 (Aug. 2003), 1690–1705.

A. Bui, Kwang-Ting Cheng, J. Cong, L. Vese, Yi-Chu Wang, Bo Yuan, and Yi Zou. 2012. Platform characterization for domain-specific computing. In *Proceedings of the 17th ASP-DAC*. 94–99.

M. J. Burfeindt, T. J. Colgan, R. O. Mays, J. D. Shea, N. Behdad, B. D. Van Veen, and S. C. Hagness. 2012. MRI-derived 3-d-printed breast phantom for microwave breast imaging validation. *IEEE Antennas and Wireless Propagation Letters* 11 (2012), 1610–1613.

S. A. Butt, S. Mancini, F. Rousseau, and L. Lavagno. 2014. Design of a pseudo-log image transform hardware accelerator in a high-level synthesis-based memory management framework. *Journal of Electronic Imaging* 23, 5 (2014), 053012.

S. A. Butt, M. Roozmeh, and L. Lavagno. 2016. Designing parameterizable hardware IPs in a model-based design environment for high-level synthesis. *ACM Transactions on Embedded Computing Systems* 15, 2, Article 32 (Feb. 2016), 28 pages.

M. R. Casu, F. Colonna, M. Crepaldi, D. Demarchi, M. Graziano, and M. Zamboni. 2014. UWB microwave imaging for breast cancer detection: Many-core, GPU, or FPGA? *ACM Transactions on Embedded Computing Systems* 13, 3s, Article 109 (March 2014), 22 pages.

F. Colonna, M. Graziano, M. R. Casu, X. Guo, and M. Zamboni. 2013. Hardware acceleration of beamforming in a UWB imaging unit for breast cancer detection. *VLSI Design* 2013 (2013), 11.

J. Cong, V. Sarkar, G. Reinman, and A. Bui. 2011. Customizable domain-specific computing. *IEEE Design & Test of Computers* 28, 2 (2011), 6–15.

J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. 2012. Architecture support for accelerator-rich CMPs. In *Proceedings of the 49th Design Automation Conference (DAC'12)*. ACM, New York, 843–849.

E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. 2015. An analysis of accelerator coupling in heterogeneous architectures. In *Proceedings of the 52nd Annual Design Automation Conference (DAC'15)*. ACM, New York, Article 202, 6 pages.

P. Coussy and A. Morawiec. 2008. *High-Level Synthesis: From Algorithm to Digital Circuit*. Springer.

M. A. Elahi, A. Shahzad, M. Glavin, E. Jones, and M. O'Halloran. 2015. GPU accelerated confocal microwave imaging algorithms for breast cancer detection. In *Proceedings of EuCAP 2015*. 1–2.

F. Hannig, M. Schmid, J. Teich, and H. Hornegger. 2010. A deeply pipelined and parallel architecture for denoising medical images. In *Proceedings of the 2010 International Conference on Field-Programmable Technology (FPT)*. 485–490.

IEEE. 2011. *IEEE Standard for SystemC Language Reference Manual*. IEEE Computer Society.

G. Inggs, S. Fleming, D. B. Thomas, and W. Luk. 2015. *FPGA Based Accelerators for Financial Applications*. Springer International Publishing, Cham, 97–115.

D. J. Pagliari, M. R. Casu, and L. P. Carloni. 2015. Acceleration of microwave imaging algorithms for breast cancer detection via high-level synthesis. In *Proceedings of the 2015 33rd ICCD*. 475–478.

I. Koren. 2002. *Computer Arithmetic Algorithms*. AK Peters.

S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2013. The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. *ACM Transactions on Architecture and Code Optimization* 10, 1 (April 2013), 5:1–5:29.

X. Li, E. J. Bond, B. D. Van Veen, and S. C. Hagnes. 2005. An overview of ultra-wideband microwave imaging via space-time beamforming for early-stage breast-cancer detection. *IEEE Antennas and Propagation Magazine* 47, 1 (Feb. 2005), 19–34.

H.-Y. Liu, M. Petracca, and L. P. Carloni. 2012. Compositional system-level design exploration with planning of high-level synthesis. In *Proceedings of DATE*. 641–646.

X. Liu, Y. Chen, T. Nguyen, S. Gurumani, K. Rupnow, and D. Chen. 2016. High level synthesis of complex applications: An H.264 video decoder. In *Proceedings of the ACM/SIGDA FPGA (FPGA'16)*. 224–233.

G. A. Malazgirt, N. Sonmez, A. Yurdakul, O. Unsal, and A. Cristal. 2015. Accelerating complete decision support queries through high-level synthesis technology (abstract only). In *Proceedings of the ACM/SIGDA FPGA (FPGA'15)*. ACM, New York, 277–277.

G. Martin and G. Smith. 2009. High-level synthesis: Past, present, and future. *IEEE Design & Test of Computers* 26, 4 (2009), 18–25.

D. Navarro, Ó Lucía, L. A. Barragán, I. Urriza, and Ó Jiménez. 2013. High-level synthesis for accelerating the FPGA implementation of computationally demanding control algorithms for power converters. *IEEE Transactions on Industrial Informatics* 9, 3 (Aug. 2013), 1371–1379.

N. K. Nikolova. 2011. Microwave imaging for breast cancer. *IEEE Microwave Magazine* 12, 7 (Dec. 2011), 78–94.

J. Oppermann, A. Koch, Ting Yu, and O. Sinnen. 2015. Domain-specific optimisation for the high-level synthesis of CellML-based simulation accelerators. In *Proceedings of the 2015 25th International Conference on Field Programmable Logic and Applications (FPL'15)*. 1–7.

B. Parhami. 2010. *Computer Arithmetic: Algorithms and Hardware Designs* (2nd ed.). Oxford University Press.

A. Prost-Boucle, O. Muller, and F. Rousseau. 2013. A fast and autonomous HLS methodology for hardware accelerator generation under resource constraints. In *Proceedings of the Euromicro Conference on Digital System Design (DSD'13)*. 201–208.

G. Ruvio, R. Solimene, A. Cuccaro, and M. Ammann. 2013. Comparison of noncoherent linear breast cancer detection algorithms applied to a 2-d numerical model. *IEEE Antennas and Wireless Propagation Letters* 12 (2013), 853–856.

R. O. Schmidt. 1986. Multiple emitter location and signal parameter estimation. *IEEE Transactions on Antennas and Propagation* 34, 3 (Mar. 1986), 276–280.

A. Shahzad, M. O'Halloran, M. Glavin, and E. Jones. 2014. A novel optimized parallelization strategy to accelerate microwave tomography for breast cancer screening. In *Proceedings of the 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC'14)*. 2456–2459.

R. Solimene, A. Dell'Aversano, and G. Leone. 2012. Interferometric time reversal music for small scatterer localization. *Progress In Electromagnetics Research* 131 (2012), 243–258.

Xilinx. 2013. Zynq-7000 All Programmable SoC First Generation Architecture. Xilinx. (December 2013).

M. Xu, P. Thulasiraman, and S. Noghanian. 2012. Microwave tomography for breast cancer detection on cell broadband engine processors. *Journal of Parallel and Distributed Computing* 72, 9 (2012), 1106–1116.

Received March 2016; revised July 2016; accepted July 2016