Hardware Acceleration of Beamforming in a UWB Imaging Unit for Breast Cancer Detection

Hindawi Publishing Corporation
VLSI Design
Volume 2013, Article ID 861691, 11 pages
http://dx.doi.org/10.1155/2013/861691

Research Article

Hardware Acceleration of Beamforming in a UWB Imaging Unit for Breast Cancer Detection

Francesco Colonna, Mariagrazia Graziano, Mario R. Casu, Xiaolu Guo, and Maurizio Zamboni

Department of Electronics and Telecommunications (DET), Politecnico di Torino, C.so Duca degli Abruzzi, 24 I-10129 Torino, Italy

Correspondence should be addressed to Mario R. Casu; mario.casu@polito.it

Received 14 September 2012; Accepted 15 May 2013

Academic Editor: Lazhar Khriji

The Ultrawideband (UWB) imaging technique for breast cancer detection is based on the fact that cancerous cells have different dielectric characteristics than healthy tissues. When a UWB pulse in the microwave range strikes a cancerous region, the reflected signal is more intense than the backscatter originating from the surrounding fat tissue. A UWB imaging system consists of transmitters, receivers, and antennas for the RF part, and of a digital back-end for processing the received signals. In this paper we focus on the imaging unit, which elaborates the acquired data and produces 2D or 3D maps of reflected energies. We show that one of the processing tasks, *Beamforming*, is the most timing critical and cannot be executed in software by a standard microprocessor in a reasonable time. We thus propose a specialized hardware accelerator for it. We design the accelerator in VHDL and test it in an FPGA-based prototype. We also evaluate its performance when implemented on a CMOS 45 nm ASIC technology. The speed-up with respect to a software implementation is on the order of tens to hundreds, depending on the degree of parallelism permitted by the target technology.

1. Introduction

Prescreening tests aimed at breast cancer diagnosis dramatically reduce mortality. Mammography, the technique currently used for screening, is very effective but has a few significant shortcomings: its high cost prevents a widespread diffusion, thus limiting *de facto* the organization of pervasive screening campaigns; its rate of false positives in young patients is very high; its use of ionizing radiations does not allow a frequent use; the obtained images are not tridimensional. Other techniques, like ultrasound or magnetic resonance, partially solve these problems but raise other issues. None of these techniques has the characteristics required to promote a widespread diffusion and to permit frequent screening campaigns on large ensembles of individuals.

One promising alternative is based on the radar principle and operates at microwave frequencies with Ultrawideband (UWB) pulses [1–3]. A set of antennas is placed around the patient's breast, and UWB pulses are sent to the breast target. The breast tumor typically exhibits a large dielectric contrast with the surrounding fatty tissue and thus reflects more the incident signal. The detection of the tumor requires, however, a significant amount of processing of the reflected signals. Figure 1 is an overview of a full UWB system for breast cancer detection.

The UWB technique does not use ionizing radiations and proved capable of detecting tumors as small as 2 mm [4]. So far, the technique has been demonstrated using bulky RF apparatus (e.g., a network analyzer connected to antennas placed around the patient's breast) and standard general purpose processors for the elaboration of the UWB signals and the generation of 2D or 3D maps of the reflected energy.

Our aim is to demonstrate the feasibility of an integrated and compact system, which we outline in Section 2. Our previous works were focused on the UWB probe system, particularly on the transmitter and the receiver [5–8]. Here, we focus instead on the processing part, which we refer to as Imaging Unit (see Figure 1). So far, software-only versions of the complex processing algorithms have been proposed

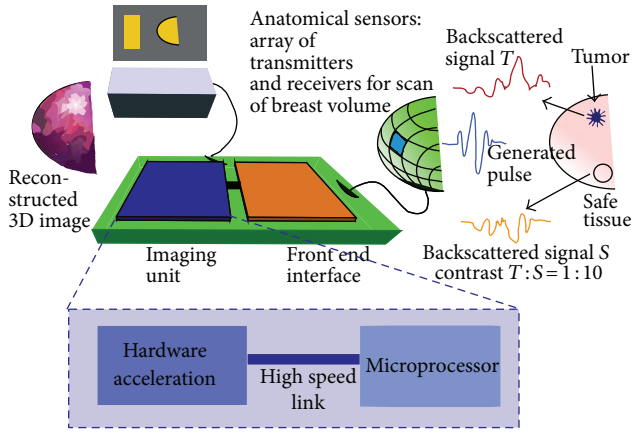


Figure 1: The UWB-based breast cancer imaging system. The inset is a high-level view of the imaging unit, which consists of a processor for the execution of the less demanding tasks, and a hardware accelerator for the most computationally-intensive processing tasks.

[4, 9], which get typically executed offline by general purpose processors. For the system to be effectively and efficiently exploited in screening, the imaging unit should be able to process UWB data almost in *real-time*, that is, with a negligible delay compared to the whole examination time as perceived by the patient. The imaging unit should be organized as in the inset in Figure 1. A microprocessor executes the control tasks and the part of the processing that does not require specialized hardware, whereas a hardware accelerator, connected to the processor through a high-speed bus, performs all the timing-critical tasks.

In this paper, we report the results of our investigation on the imaging unit. These are our *contributions*.

(i) We are the first to profile the various software tasks a complete UWB imaging algorithm for breast cancer detection consists of [10]. Based on the profiling, we are able to decide which part of the imaging software needs to be accelerated by a specialized hardware. As we illustrate in Section 3, *Beamforming* is the most critical task.

(ii) We propose for the first time a complete implementation of a hardware accelerator for the beamforming algorithm in synthesizable VHDL. The architecture is described in Section 4.

(iii) We prove the functionality of the accelerator with a FPGA-based prototype, as shown in Section 5.

(iv) We evaluate performance and resource utilization when the accelerator is implemented both in a Xilinx Virtex-4 and in a standard-cell ASIC with a 45 nm CMOS technology. These results are reported in Section 6.

We end the paper in Section 7 with a recapitulation of the main results alongside a final discussion about our achievements.

2. The System

The overall UWB imaging system we aim at is sketched in Figure 1. An array of transceivers generates and receives UWB signals. Figure 2(b) reports a typical UWB pulse. The literature suggests that the frequency range between 0.5 GHz and 10 GHz should be covered to achieve both high contrast (low frequency range) and high resolution (high frequency range). A variety of signals can be used, like Gaussian derivative or modulated and modified Hermite polynomials (MMHP).

The contrast between the damaged (possibly a tumor) and healthy regions is due to an abrupt dielectric constant variation and can be as high as 10 : 1.

The entire breast volume is scanned by sequentially activating the transceivers. In the *monostatic* configuration [11], each antenna transmits and receives a scattered signal. Transceivers activation and data collection are coordinated by a control unit through a synchronous serial bus, as shown in the block diagram in Figure 2(a). All sampled data are collected and delivered to a memory.

Samples in memory are processed by the *imaging unit*, which in turn produces a 2D or 3D map of *reflected energy* values, each related to a different breast volume location (voxel). The maps are finally sent to a personal computer and displayed.

The total time necessary to execute a scan of the whole volume is typically much less than one second [12], even though it may change depending on design choices like the number of antennas N_{ANT}, the sampling frequency, the number of samples acquired N_s, the type of UWB signal selected, and the methods used to compensate noise. We then assume 1 s as a reference value to which we compare the time required by the imaging unit. In particular, we assess the need for hardware acceleration when the software processing largely exceeds this timing reference, resulting in an unacceptable performance for a realistic clinical scenario.

The imaging unit is then the focus of this work, and its data-flow diagram is shown in Figure 3. It performs the image reconstruction in two steps, according to the flow proposed in [9, 11].

Calibration (Skin Artifact Removal Algorithm (SKAR) Block in Figure 3). This step removes unwanted scattering contributions determined by the large dielectric contrast between air and skin. These artifacts are orders of magnitude greater than useful contribution and need to be removed.

Beamforming (Beamforming (BEAF) Block in Figure 3). This step reconstructs a map of scattered energy in a 2D or 3D volume. This is done by the microwave imaging via space-time (MIST) beamforming method, which coherently shifts in time the various received signals in order to focus the energy analysis on a single voxel [4].

The following section details the Imaging Unit organization and provides a profiling of the time needed to execute the various subtasks of calibration and beamforming.

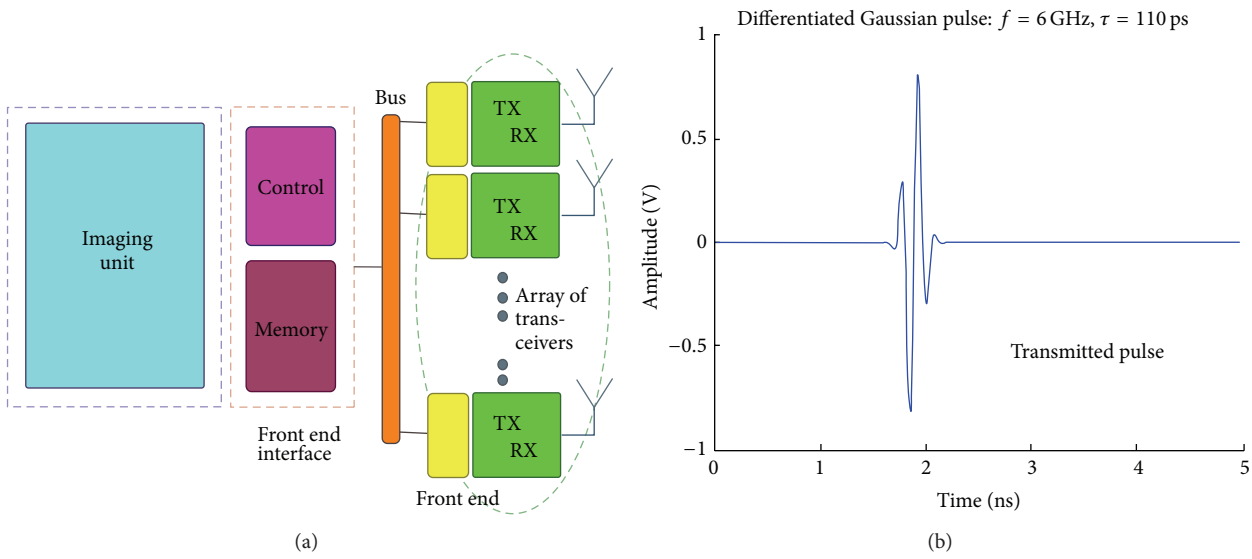


FIGURE 2: The system block diagram. (b): an example of a UWB transmitted pulse, central frequency 6 GHz, total duration time 110 ps.

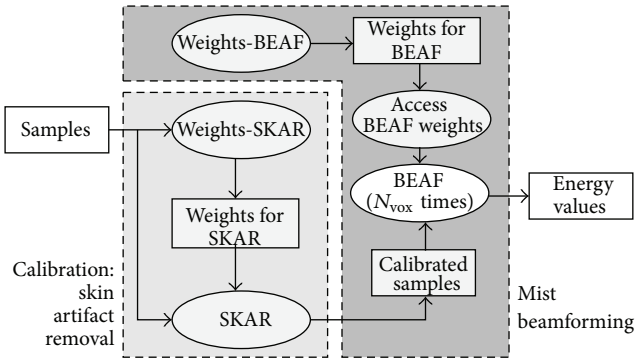


FIGURE 3: Data-flow diagram of the imaging unit computation tasks.

3. The Imaging Unit

The imaging unit performs sequentially the following subtasks (see Figure 3).

Weights-SKAR. The skin artifact removal algorithm (hereinafter SKAR) needs weight coefficients that are functions of the received signals and the position of the antennas. These data are then processed once for each scan of the breast to generate the *weights for SKAR*.

SKAR. By removing the artifact of the large skin reflection from the received signal samples, as well as the contribution of the direct path between transmitter and receiver, the SKAR sub-task generates a set of *calibrated samples*.

Weights-BEAF. Beamforming (hereinafter BEAF) also needs weights coefficients, but these only depend on the transmitted signals and not on the received ones. Therefore, the *weights for BEAF* are computed once and for all at the beginning, when the characteristics of the transmitted signal are selected.

Access BEAF Weights. In this phase, BEAF weights are read from a memory. This operation is repeated for each voxel. We separate it from the actual BEAF to permit a correct profiling of performance.

BEAF. This phase applies a space-time beamformer to the signal samples received by each antenna, a step that is done specifically for each voxel. To focus on a specific voxel, the signals are shifted in time, filtered with the *weights for BEAF*, and finally coherently added (see Section 4 for a more detailed formulation of the algorithm). The weighting operation enhances the contribution in each signal of the reflection determined by the voxel under analysis and deemphasizes at the same time the scattering contributions of the other voxels. By iterating on the number of voxels, BEAF produces an energy map for the whole volume.

We first implemented all the subtasks in software using Octave [13]. We also built a finite-difference time-domain (FDTD) electromagnetic solver to simulate the transmission of the signals, and the reflections that occur in a realistic numerical breast phantom [14]. The output data of the FDTD simulation are the received signals that we process with our Octave routines, according to the tasks outlined above.

3.1. Profiling. SKAR and BEAF have a very different impact on the overall execution time. This difference depends on the difference in complexity, but mostly on the different number of times they get executed. SKAR is performed once, and so is the computation of the SKAR weights, whereas BEAF is repeated as many times as there are voxels. In the largest maps found in the breast phantom repository [14], the voxels can be as many as $N_{vox} = 4.25 \cdot 10^6$, which clearly makes BEAF the bottleneck for the whole computation. The weights required for the BEAF operation, however, do not depend on the specific breast but only on some specific design choices such as the shape of the UWB pulse and the location of transmitters and receivers. The computation of these values is thus performed only once at the beginning of the breast examination.

Table 1: Software runtime of SKAR and BEAF subtasks for a case with $N_{vox} = 4.25 \cdot 10^6$.

Task	Runtime
Weight-SKAR	16.4 s
SKAR	0.25 s
Weights-BEAF	2.17 ms/voxel
	2.6 h
Access BEAF	7.32 μs/voxel
weights	31 s
BEAF	4.88 ms/voxel
	5.8 h

We performed software simulations of the complete system with an Intel CPU (Core i5-430M, 3 MB L3, 2.26 GHz clock frequency), 4 GB DRAM, and Linux Ubuntu 11.10 with kernel 3.0 and instrumented our code to accurately determine the CPU runtime. The results are in Table 1. For the BEAF sub-tasks, results are detailed for one voxel and for the whole breast volume.

For all sub-tasks, with the exception of SKAR, runtime exceeds by far the 1 s bar. However, all tasks, except weights-BEAF and BEAF, are executed in less than one minute, which can still be judged acceptable especially because a further optimization of the code can significantly reduce this time. As for weights-BEAF, 2.6 h is certainly a long time, but the weights could be computed offline, once and for all, and accessed from memory when needed. As expected, the BEAF algorithm has the largest impact on timing performance due to the large number of iterations. Based on this analysis, we decided to focus on the hardware acceleration of the BEAF sub-task, while all the other tasks are executed in software. The target is an imaging unit like the one sketched in Figure 1: a processor is coupled with a dedicated hardware unit (ASIC) that accelerates the timing critical parts of the computation.

4. A Hardware Implementation of BEAF

4.1. The MIST Beamforming (BEAF) Algorithm. We report a succinct description of the MIST beamforming and refer to the literature for all the details [4, 10].

Let us consider the backscatter contribution from a single breast location $\mathbf{r}_0$, assuming that the signal received by the ith receiver contains only information from that source, for simplicity. We denote the sampled version of that signal as $x_i[n]$ and its Fourier transform as

$$X_i(\omega) = I(\omega) S_{ii}(\mathbf{r}_0, \omega), \quad 1 \le i \le N_{\text{ANT}}, \tag{1}$$

where $I(\omega)$ is the Fourier transform of the transmitted signal $i(t)$. N_{ANT} is the number of antennas, which is also the number of transmitters and receivers. $S_{ii}(\mathbf{r}_0, \omega)$ is an analytical model of the monostatic frequency response associated with the propagation from the ith antenna to $\mathbf{r}_0$ and back. The distance between each antenna and the point $\mathbf{r}_0$ is not the same for all the antennas, and so the round-trip times of the signals differ. We then need to time align the signals by delaying them by an integer number of samples, $n_i(\mathbf{r}_0) = n_a - \tau_i(\mathbf{r}_0)$. In this equation, τ_i is the round-trip propagation delay in the ith channel for location $\mathbf{r}_0$ and is obtained by dividing the round-trip path length by the average propagation speed and rounding to the nearest sample. The term n_a is the reference time to which every signal is aligned and is chosen to be the worst-case delay over all channels and locations:

$$n_a \ge \text{round}\left(\max_{i,\mathbf{r}_0} \tau_i(\mathbf{r}_0)\right). \tag{2}$$

The signal is windowed to remove components before sample n_a that are not important for energy estimation. The following window function $g[n]$ is used:

$$g[n] = \begin{cases} 1, & n \ge n_a \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The algorithm also allows to equalize path-length dependent dispersion and attenuation, to interpolate any fractional time delays remaining after coarse time alignment, and to bandpass-filter the signal. These operations are done in time domain using a bank of FIR filters, each associated with a vector of L weights $\mathbf{w}_i = [w_{i0}, w_{i1}, \ldots, w_{i(L-1)}]$. Increasing the number of weights improves accuracy but increases complexity, too, and so must be carefully chosen.

By summing the output of the filters, we obtain

$$z[n, \mathbf{r}_0] = \sum_{i=1}^{N} \sum_{l=0}^{L} w_{il} \cdot g[n-l] \cdot x_i[n - l - n_i(\mathbf{r}_0)], \tag{4}$$

where w_{il} is the lth weight of the ith filter and $n_i(\mathbf{r}_0)$ is the delay value associated with the ith filter.

We need to define another window:

$$h[n, \mathbf{r}_0] = \begin{cases} 1, & n_h \le n \le n_h + l_h \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

The interval between n_h and $n_h + l_h$ is the window of samples in which we expect to find the received pulse. Even if we know the shape of the transmitted signal, the scattering from the tumor is frequency-dependent. Therefore, the backscattered pulse is a distorted version of the transmitted pulse, and defining the exact timing window is not straightforward. The values of these parameters suggested in [4] are meant to maximize the contrast for small lesions.

The energy scattered by $\mathbf{r}_0$ can be computed as the sum of the square of the windowed values:

$$p(\mathbf{r}_0) = \sum_n \left| z[n] h[n, \mathbf{r}_0] \right|^2. \tag{6}$$

Many parameters in the previous equations have to be assigned a value before proceeding with the hardware design. The value of n_h can be computed based on the time shift n_a, which is easily computed as shown in (2), on the delay of the FIR filter $\tau_0 = (L-1)/2$ and on the first significant sample of the transmitted pulse N_{start}:

$$n_h = N_{\text{Start}} + n_a + \tau_0. \tag{7}$$

As for l_h, an optimal value is suggested in [4] and is $l_h = 5$. As for L, a reasonable value for a Gaussian pulse like the one in Figure 2, which we use for our experiments, is $L > 50$, and we chose $L = 55$.

Table 2: Maximum difference of energy values with respect to ideal computation for three approximation cases.

Calibration	Tumor Pos.	Max error		
		Approx-I	Approx-II	Approx-III
IDEAL	Low	26.78%	23.28%	0.25%
	Med	14.64%	4.94%	0.07%
	Deep	17.97%	6.70%	0.16%
SKAR	Low	33.60%	25.02%	0.44%
	Med	15.61%	8.67%	0.16%
	Deep	30.68%	34.46%	0.48%

4.2. Accuracy and Approximation. One of the crucial design choices is data representation. We assume that received signals are sampled and represented as 16-bit 2's complement numbers [12]. These data are processed according to (4) and (6), in which multiplications and additions are the most frequent operations. The number of bits for the intermediate and final results is one of the designer's knobs, who has to find the best trade-off between precision and complexity. We explored three possible approximation strategies, which we discuss in the following.

Approx-I. Here, we keep data parallelism as low as possible. In the first multiplication in (4), the 16-bit inputs are multiplied and a 32-bit output is produced. Before the addition in the same equation, we truncate the multiplication results and discard the 16 least significant bits. Then, to avoid overflow, the additions are performed over 32 bits by extending the remaining 16 most significant bits. This parallelism ensures no overflow for up to 16 levels of addition. Before the final multiplication in (6), the 16 least significant bits are discarded, and the multiplication result is a 32-bit number.

Approx-II. In this case, we do not discard the 16 least significant bits in the square computation in (6), obtaining a 64-bit result. The 32 least significant bits of the result are then cut away. This solution requires of course larger multipliers and so implies a larger complexity.

Approx-III. In this case, we keep the maximum level of accuracy and perform all the operations with 64 bits.

We evaluated these three alternatives with a software implementation of the algorithm in Octave, before moving to the hardware design. The algorithm performance depends on the relative position of any high-permittivity tissue and the position of the antennas. Therefore, we compared the performance considering an ideal breast model uniformly filled with fat tissue for three possible tumor positions in the breast: a small depth (*low*), an intermediate depth (*med*), and a considerable depth (*deep*).

We also evaluated the effect on the performance of the SKAR algorithm. In particular, we compared the performance of the actual SKAR with an *ideal* calibration, in which we remove any artifact by artificially subtracting the response without tumor from the response with tumor.

The three maps in Figure 4 are obtained with SKAR calibration and with the three approximation strategies for the case of *deep* position. In the first case, in particular, the low resolution of the internal computations creates a large artificial clutter response and so a clear loss in accuracy. Nevertheless, the tumor position is clearly and correctly found. In the second case, the removal of the least 32-bit of the energy result eliminates the clutter response but may end up in inaccuracies should a point of interest have a low energy and be therefore eliminated. The third approximation is of course the one with the greatest accuracy. In fact, the results in the third map in Figure 4 are identical to those obtained from a simulation with double precision floating-point numbers.

In Table 2, we report the maximum error of the energy values with respect to an ideal computation (with no approximation). The maximum error is calculated as follows:

$$\text{MaxError} = \max\left(\left|\frac{\text{EM}}{\max(\text{EM})} - \frac{\text{EM}_{\text{fp}}}{\max\left(\text{EM}_{\text{fp}}\right)}\right|\right). \quad (8)$$

EM represents the energy map without approximations (double precision), and EM_{fp} is the energy map with approximation (fixed point). The error is determined by comparing energy values normalized to the maximum values obtained with or without approximation, respectively. The maximum error is consistent with the graphical results of the maps.

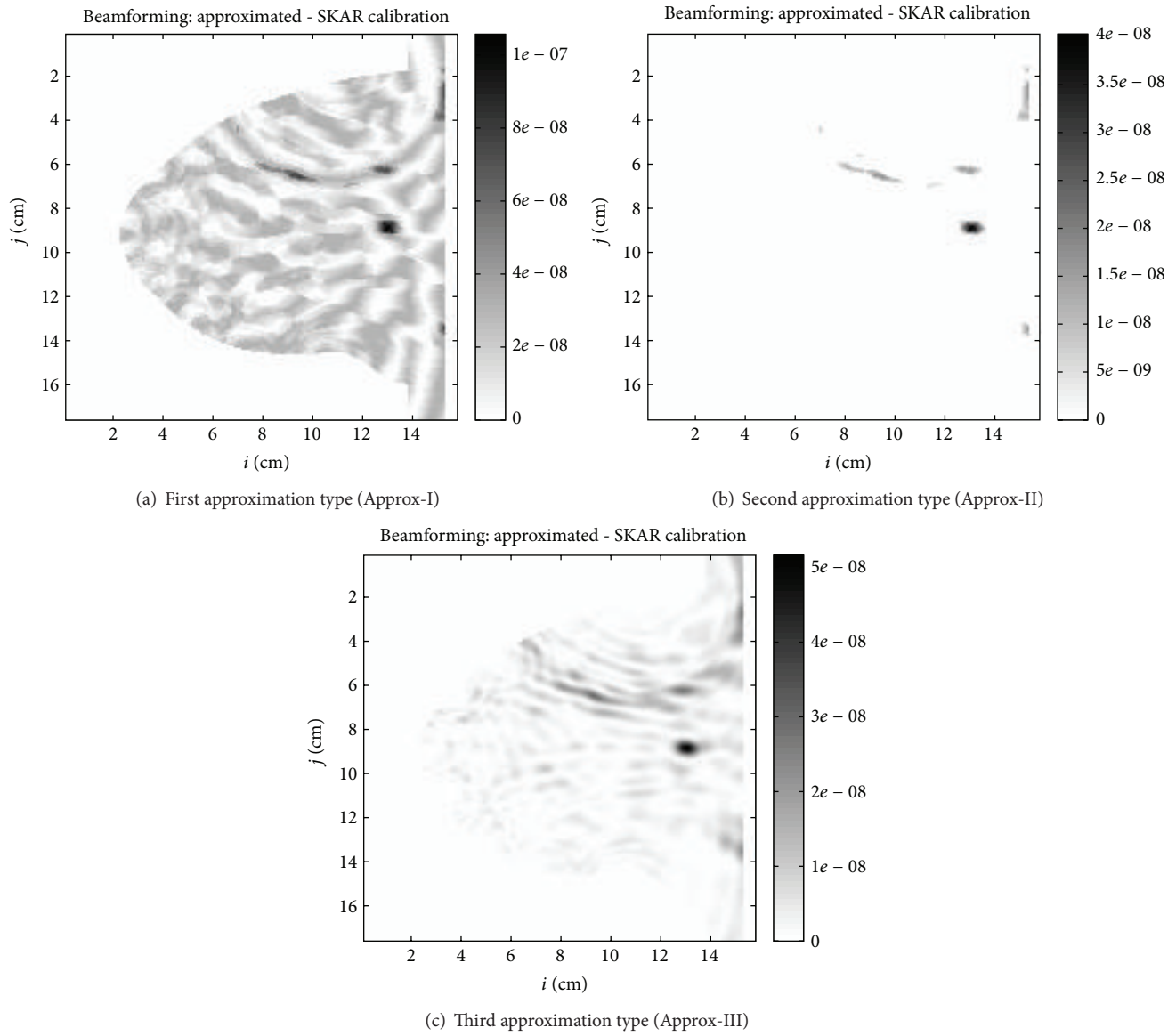
To keep the maximum level of accuracy, then, we decide to maintain a 64-bit data representation. This decision does not affect the design as much as one may think. To store 64-bit energy values, we need twice as much memory locations than we need for the 32-bit case. However, the memory size is mostly determined by the many weights we have to store, and both the number and the bit size of the weights do not depend on the representation of the intermediate data.

4.3. Architecture. After having decided the parallelism for internal data representation, we moved on to the RTL hardware design. We described our system in synthesizable VHDL. Design-time parameters were used to keep the design flexible. The block diagram in Figure 5 represents the BEAF architecture that we described in RTL. In the following, we describe the behavior of each component.

Weights & Delays Shift Register. When the image reconstruction process starts, after the acquisition phase, *calibrated samples*, *weights for BEAF,* and delays values are available each in their own memories. For each energy value, an entire window of samples is processed in every FIR filter, together with the corresponding weights. Before the filter computation, these weights are sequentially read from memory and loaded in a serial-in parallel-out shift register included in the *Weights & Delays shift register* block. The delay values are also read from memory at the same time. They are sent to the τ blocks to determine the correct time-shift amount necessary to align the signals, according to the beamforming algorithm described in Section 4.1.

Sample Memory. The sample values stored in this memory are read one at a time for each channel and fed to the filter blocks.

Filters. A bank of *filters* receives samples from the *sample memory* and delay values from the *Weights & Delays shift register*. In a fully parallel implementation, the number of *filters*



(a) First approximation type (Approx-I)

(b) Second approximation type (Approx-II)

(c) Third approximation type (Approx-III)

Figure 4: Energy map of the breast after BEAF data approximation for ideal breast model and SKAR calibration. A tumor is present in a *deep* position.

(N_{FIR}) corresponds to the number of transceivers/antennas in the UWB probe. As we discuss momentarily, sequential implementations that use less filters than there are antennas are possible. Each filter includes three main components. The τ blocks are programmable delay lines, which shift the input samples of the right time shift to ensure that all FIR inputs are time aligned. The maximum delay is n_a. The *Window1* blocks perform the $g[n]$ windowing functions: they feed the *FIR* blocks only with samples that come after n_a, and replace all the samples that come before n_a with zero values. In the FIR blocks, the samples are processed according to (4). Every FIR is a fully-parallel structure with L multipliers. Each multiplier is fed with one of the weights, which do not change during the whole voxel computation and with the samples. The latter are sequentially introduced in the FIR blocks and then are shifted from one multiplier to the next via an internal shift register. The outputs of all the multipliers are then summed via an adder tree, producing one single FIR output. This whole computation block is pipelined, with registers at the input of every adder and multiplier.

Sum. The outputs of all the filters are summed together with a adder tree to obtain a single value, which represents the result of (4).

Window2. The samples we are interested in are those in which the signal pulse occurs. *Window2* lets input data pass unaltered if they correspond to the time interval specified by n_h and l_h. Otherwise, the output value is set to zero.

Energy Computation. This block computes the total voxel energy according to (6). A multiplier and an accumulator are

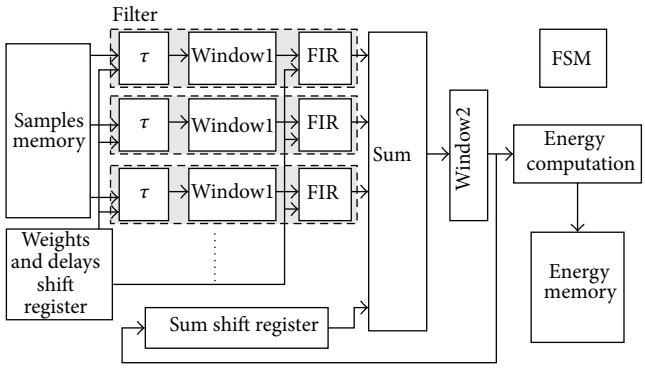


Figure 5: RTL block diagram of the BEAF unit.

used for this purpose. Finally, the energy of the current voxel gets written into the *energy memory*.

FSM. To coordinate the whole computation process, a finite state machine (FSM) is used. A set of counters in it also keeps track of the number of computation steps performed at any control step.

The filtering functions need not to be necessarily performed in parallel. Two extreme solutions are possible. A parallel implementation, in which all the data are processed concurrently, and a serial implementation in which one filter is iteratively used. In the serial implementation, an accumulator is inserted at the output of the filter, since the FIR outputs must be summed to compute the energy value.

Between the two extremes, hybrid solutions are also possible. We can instantiate a number of filters (and thus a number of FIR blocks) N_{FIR} that matches our timing and resource requirements. We can use them iteratively to process the data from N_{ANT} antennas. In case of a hybrid solution, at each iteration, the results of the partial sum of the FIR outputs are fed back to the *sum shift register* block in Figure 5 and added coherently to the current value at a given iteration. When all the antennas sets have been processed, the final result is fed to the energy computation block. The finite state machine is designed in a parameterized way so as to correctly handle all possible solutions with different degrees of parallelism.

It is worth mentioning that a choice of degree of parallelism determines not just the number of FIR blocks, but also the number of τ, *Window1* and *Window2* blocks. In the following, when we refer to N_{FIR} as a parameter for the degree of parallelism, we mean that all these blocks are instantiated N_{FIR} times.

We verified our parameterized VHDL code by means of extensive logic-level simulations based on Mentor Graphics' Modelsim [15] varying the degree of parallelism. We validated the output obtained with the RTL code against the results obtained with a software implementation and refined our design until they perfectly matched.

4.4. Timing Performance of BEAF. In our architecture, each voxel's energy is sequentially computed, therefore the beamforming execution time is

$$T_{\text{BEAF}} = N_{\text{vox}} \cdot T_{\text{vox}}, \tag{9}$$

where N_{vox} is the number of voxels and T_{vox} is the time required to compute each voxel.

The number of clock cycles necessary to execute the beamforming step for each voxel depends on the number of FIR filters in Figure 5. As we discussed above, the spectrum of possible solutions spans from a fully sequential implementation with a single FIR iteratively used for all the antennas to a fully parallel solution with as many filters as there are antennas. The number of iterations depends on the ratio between number of antennas and number of filters:

$$N_{\text{iter}} = \left\lceil \frac{N_{\text{ANT}}}{N_{\text{FIR}}} \right\rceil. \tag{10}$$

T_{vox} is the sum of three contributions:

$$T_{\text{vox}} = T_w + T_{\text{comp}} + T_{wr}. \tag{11}$$

T_w is the time necessary to write the FIR weights in the memory, once they are received from the link that connects the processor with the accelerator, to read the weights from the memory and to store them in local registers within the filters:

$$T_w = \left(2\left(L+1\right)N_{\text{ANT}} + 2N_{\text{iter}}\right)T_{\text{CLK}}. \tag{12}$$

The first term within parenthesis in (12) is the time for writing and reading from the memory L weights. The second term is an overhead for enabling the memory and for asserting a few control signals. T_{CLK} is the clock period.

T_{comp} is the actual computation time and is given by

$$\begin{aligned} T_{\text{comp}} = \Big(&L_{\text{wp}} + \lceil \log_2(L) \rceil + n_h + l_h + 7 \\ &+ \lceil \log_2\left(N_{\text{FIR}} + 1\right) \rceil \Big) N_{\text{iter}} T_{\text{CLK}}. \end{aligned} \tag{13}$$

In (13), L_{wp} is the time distance, evaluated in time samples, that corresponds to the worst path between two different antennas, and is necessary for the time alignment of the various signals. Term $\lceil \log_2(L) \rceil + n_h + l_h$ is the number of clock cycles needed to load the samples in the filters, with n_h and l_h representing the first sample in a window of interest and l_h the size of that window, according to equation (5). Seven clock cycles are needed for window traversal and various register writing. The term that depends on the logarithm of N_{FIR} is related to the final addition in Figure 5.

The final term in (11), T_{nrj}, is given by

$$T_{\text{nrj}} = 3N_{\text{iter}} \tag{14}$$

and represents the time needed to write the voxel's energy value back in the memory.

We rewrite the final expression of T_{vox} obtained by wrapping up all previous equations:

$$\begin{aligned} T_{\text{vox}} = \ &\left(2\left(L+1\right)\cdot N_{\text{ANT}} + 2\cdot N_{\text{iter}}\right. \\ &+ \left(L_{\text{wp}} + \lceil \log_2(L) \rceil + n_h + l_h + 7\right. \\ &\left.\left. + \log_2\left(N_{\text{FIR}} + 1\right)\right)\cdot N_{\text{iter}} + 3\cdot N_{\text{iter}}\right) T_{\text{CLK}}. \end{aligned} \tag{15}$$

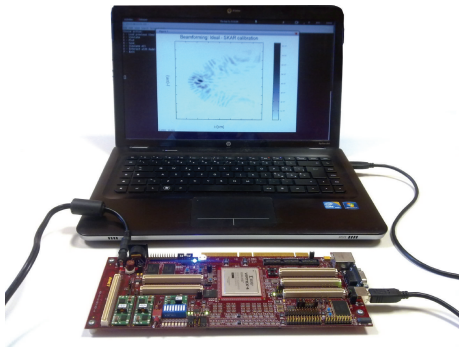


Figure 6: Experimental setup for the FPGA prototype: the personal computer executes in software (Octave) part of the algorithm and allows energy maps visualization. A USB link transfers data to and from the FPGA board where part of the beamforming algorithm is accelerated.

Equation (15) can be used to determine the overall execution time of the beamforming algorithm in (9) given the total number of voxels and applies both to a 2D and a 3D map. The accuracy of (15) was tested with VHDL simulations using Mentor Graphics' Modelsim simulator and experimentally verified with the FPGA prototype described in the next section.

5. Emulation with an FPGA-Based Prototype

We built an emulation prototype of the imaging unit. This prototype relies on a general purpose microprocessor to perform the portions of processing that we decided to execute in software, while the hardware part of the BEAF algorithm was deployed on an Xilinx Virtex-4 XC4VLX160-FF1513 FPGA. In particular, we were able to connect the Intel Core-i5 of a laptop with the FPGA located in an AVNET Virtex-4 LX development board. The Cypress FX2 USB 2.0 chip integrated in the development board was used to communicate with the FPGA. The USB chip behaves simply as a conduit between the USB and the external data-processing logic. The FX2 clock was set to 48 MHz, the data parallelism to 8 bits, and the packet size to 512 bytes.

The prototype has multiple purposes. It serves as an accelerator of the system-level simulation, because it permits a much faster execution of the part of the algorithm executed by the FPGA, compared to a logic-level simulation. The overall architecture, with the software and the hardware running *concurrently,* is clearly a close approximation of the final system, even though the processor and the digital hardware will be different in the final target. The RTL VHDL code that gets synthesized on the FPGA is the same that we will eventually implement in an ASIC; therefore, we can fully verify it before fabrication. Finally, we can anticipate what the *exact* performance will be in terms of clock cycles. The only difference in performance will be then determined by the different clock frequency in FPGA and ASIC.

Both the software and the hardware code are fully instrumented to permit a cycle accurate evaluation of timing performance. The USB communication time is also correctly estimated and decoupled from the overall performance, as a much faster link will be used in a realistic setting like the one depicted in Figure 1.

The software running on the microprocessor is layered, and so the application part and the communication part are independent and portable. The communication part is also further partitioned in a set of low-level, hardware-specific USB primitives (we use the *libusb* library available in Linux) and a set of higher-level, technology-independent APIs for data organization and communication. When the hardware target changes, as it happens when moving from the prototype to the final target, the developers replace only the low-level communication primitives and reuse both the application code and the communication APIs.

Figure 6 is a photograph of the prototype and shows an energy map elaborated in the FPGA, sent to the host computer and displayed in the host screen.

6. Experimental Results on FPGA and ASIC

We ran several experiments, first with Xilinx's ISE software [16] for the FPGA target of our prototype, then with Synopsys Design Compiler [17] for an ASIC target. We were able to determine the resource requirements and the actual timing performance which depends on the actual clock frequency as found by the FPGA and ASIC design CAD tools. The various bitstream files generated after FPGA synthesis, mapping, and place-and-route experiments were each tested in our prototype. They refer to a specific design case with nine antennas located around the numerical breast phantom number 012204 from the repository made available by the University of Wisconsin [14]. This phantom represents in the database an average case in terms of type of tissue and number of voxels. We have run finite-difference time-domain (FDTD) electromagnetic simulations to determine the signal received by the nine antennas, and we fed the imaging unit with the received samples after 16-bit quantization. We also extrapolated our performance results to a case of fifty antennas, which we assume it to be more or less than maximum number of antennas that can be placed around a patient's breast.

6.1. FPGA Experiments. The FPGA resources, the main part of which consists of look-up tables (LUT) and registers, are grouped in slices. Depending on the complexity of the design, and particularly on the number of filters N_{FIR}, a higher or lower amount of LUT and registers is used. When mapping the design, the strategy of ISE does not consist in filling the slices one by one. Instead, to meet timing requirements, the logic gates are spread over a high number of slices, which often results in a poor utilization. Sometimes a slice is declared as occupied even if only one of its LUTs is used. This fact limits the number of filters that we were able to put in our Virtex-4 to a maximum of five FIRs. The results of resource utilization are reported in Table 3.

We notice that the number of occupied slices increases rapidly with the number of filters. With six filters, it exceeds the number of available slices, and we were not able to

Table 3: Area allocation with different number of FIR filters.

N_{FIR}	Slice registers	4 input LUTs	Occupied slices
1	9,081 (6%)	8,038 (5%)	7,188 (10%)
2	17,347 (12%)	20,072 (14%)	16,115 (23%)
3	25,551 (18%)	44,290 (32%)	30,899 (45%)
4	33,851 (25%)	67,247 (49%)	45,139 (66%)
5	42,055 (31%)	91,461 (67%)	59,936 (88%)

Table 4: Hardware execution time over 14485 voxels, with 9 antennas and 16 ns clock period, and speed-up with respect to a software execution requiring $T_{SW} = 12.8$ s.

N_{FIR}	N_{ITER}	T_{exe} [ms]	Speed-up
1	9	806.39	15.87
2	5	553.36	23.13
3	3	425.81	30.06
4	3	426.50	30.01
5	2	362.49	35.31
6	2	362.49	35.31
7	2	362.49	35.31
8	2	362.95	35.27
9	1	298.71	42.85

test that configuration in our prototype, even though we could estimate the performance in simulation. The allocated resources follow a trend that is almost linear with the number of filters, which is in accordance with our expectations.

The timing performance depends proportionally on the number of voxels in the energy map. This value is derived from the 012204 phantom breast at 1 mm resolution, which we also used for the Octave simulations. The 2D map grid consists of 152 rows and 176 columns. The Octave simulation code was optimized to consider in the computation only those voxels which happened to be located in the elliptical space described by the antennas, that is, the area actually irradiated by the transmitters. With such optimization the number of voxels is $N_{VOX} = 14495$.

With such number of voxels, the time required for the software execution of the BEAF on the Intel i5 core of our prototype (with Ubuntu 11.10, kernel 3.0) was approximately $T_{SW} = 12.8$ s. The clock period for the FPGA implementation is $T_{CLK} = 16$ ns, which we use in our experiments with the prototype. Table 4 reports the time required to execute the BEAF routine in hardware and the speed-up with respect to a software implementation. Timing results with N_{FIR} in range 6–9 refer to logic simulations only, because only five filters could be placed in the FPGA. However, since all the results are consistent with (15), they are all fully accurate. Table 5 refers to a hypothetical case with fifty antennas and reports only simulated data, which are also fully accurate, compared to a software execution that requires $T_{SW} = 71$ s.

In both Tables 4 and 5, we observe a similar trend. As we expected, the effect of changing the number of filters mainly depends on the actual number of iterations $N_{ITER} = \lceil N_{ANT}/N_{FIR} \rceil$. Increasing the number of filters boosts performance only when it allows to perform the computation with

Table 5: Hardware execution time over 14485 voxels, with 50 antennas and 16 ns clock period, and speed-up with respect to a software execution requiring $T_{SW} = 71$ s.

N_{FIR}	N_{ITER}	T_{exe} [s]	Speed-up
1	50	4.48	15.86
2	25	2.89	24.53
3	17	2.38	29.79
4	13	2.13	33.31
5	10	1.94	36.61
6	9	1.88	37.86
7	8	1.81	39.19
8	7	1.75	40.59
9	6	1.68	42.14
10–12	5	1.62	43.81
13–16	4	1.56	45.62
17–24	3	1.49	47.56
25–49	2	1.43	49.71
50	1	1.36	52.05

less iterations. For example, in the case with nine antennas we would not get a significant advantage from using eight filters: if we cannot use nine, then we could as well use six and still obtain basically the same performance we would get with eight filters.

The speed-up value grows rapidly for low values of N_{FIR}, whereas its marginal increase tends to diminish in the upper range. The reason is that the part of the algorithm that consists of accessing the memory for reading samples and weights remains sequential and cannot be made parallel.

Considering the same phantom, and iterating the performance evaluation in the 3D case with 1374953 voxels, the resulting trends are similar. To summarize, in the worst case of fifty antennas and $T_{ITER} = 50$, the required time is $T_{exe} = 424.64$ s (around 7 minutes), while in the best case of only one iteration we obtained $T_{exe} = 129.40$ s (around 2 minutes). With respect to a software implementation requiring $T_{SW} = 1.9$ h; the speedup is a factor 16.11 and 52.86, respectively, in the worst and best case.

Referring to data discussed in Table 1 obtained for a worst case breast phantom (in terms of number of voxels), the execution time in the worst case ($T_{ITER} = 50$) would be around 21 minutes (compared to the 5.8 h obtained in the software case).

6.2. ASIC Experiments. For this part of our experiments, we targeted a standard cell library in a 1.1 V 45 nm CMOS technology. We synthesized with Synopsys Design Compiler various instances of our architecture varying the number of filters and determined the relationship between speed-up (with respect to a software execution) and silicon area. The design, when synthesized with the maximum optimization effort available, can run at a clock period of 2 ns. Therefore, compared to the FPGA results reported in the previous section, all the values of speed-up can be multiplied by a factor 8 (16 ns/2 ns).

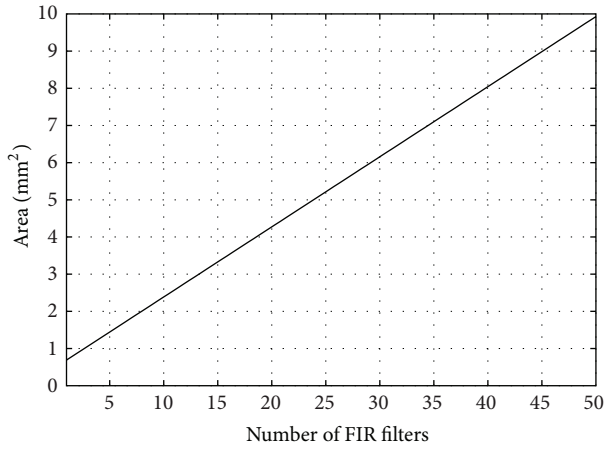


Figure 7: Postsynthesis ASIC area versus number of FIR filters for a 50 antennas system.

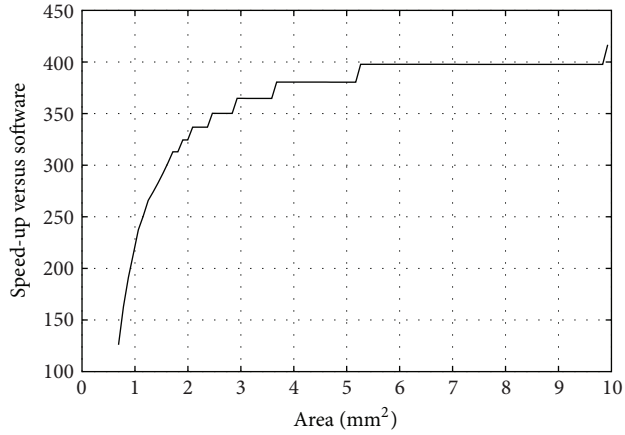


Figure 8: Speed-up versus area in an ASIC implementation of a 50 antennas system.

Figures 7 and 8 report silicon area as a function of the number of filters and speed-up as a function of the silicon area, respectively.

The area curve in Figure 7 grows linearly with N_{FIR}, with a slope of 0.1885 mm^2, which is in fact the area of a single FIR filter. The area depends also on the memory size. The memory area was obtained by considering the use of 100 memory blocks (two for each antenna), each consisting of 256 bytes for the signal samples (25.6 kB over 0.457 mm^2), and a dual-port memory block with 2048 bytes for the weights and delays values (4 kB over 0.0153 mm^2). The memory area weighs on the total area in a significant way only when a few filters are used, because it does not depend on N_{FIR}.

The speed-up, that is the ratio of software execution time and hardware execution time, as a function of the silicon area shows a behavior similar to what we found for the FPGA case, but the acceleration factor is of course much bigger: from around 50 to around 400 in the case of fifty antennas in the ASIC case. This result would be notably improved in case a more scaled technology would be used [18]. As we

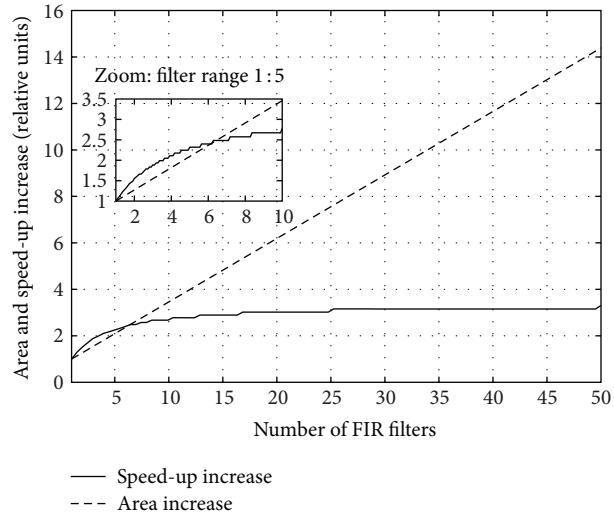


Figure 9: Trade-off between speed-up and area when changing the number of filters in a 50 antennas ASIC implementation. Cost-effective solutions are those with $N_{\text{FIR}} \leq 6$.

utilize more and more area to make room for the filters, the speed-up first increases in a steep way, then it bends because the sequential part of the algorithm starts dominating the execution time.

We report in Figure 9 two more curves, the *speed-up increase* and the *area increase*. These two curves are simply the *normalized* versions of the curves reported in Figures 7 and 8. In practice, we divided the area and the speed-up of the solution with N_{FIR} filters by the area and the speed-up of the solution with one filter, respectively. The rationale behind this choice is to evaluate the trade-off between performance and cost. In particular, we judge that a parallel solution is *cost-effective* if, for a given percentage cost increase, the performance increases at least as much as the cost [19, 20]. From the inset in Figure 9, we are able to evaluate for what value of N_{FIR} the two curves cross each other. The cost-effective solutions are those that use no more than six filters.

7. Conclusions

Implementing in hardware the Mist-beamforming algorithm portion of the image reconstruction process determines a remarkable acceleration in terms of execution time. From a practical point of view, this acceleration implies a significant improvement in the applicability of the system: if we consider the case of a 3D image reconstruction with a fifty-antenna system, we move from 1.9 hours of computation in the software-only version to only about 7 minutes for an FPGA implementation and less than one minute for an ASIC implementation, even in the serial hardware configuration (i.e., with $N_{\text{FIR}} = 1$). The FPGA clock frequency was set in our experiments to 62.5 MHz, which is the clock frequency of the IP that communicates with the UWB external chip. However, our results show that the FPGA accelerator alone can run up to 180 MHz, hence providing a further 2.88X speed-up with respect to the software version. Therefore,

even the fully serial configuration gives us a noteworthy acceleration that justifies alone the validity of the hardware design. When using some degree of parallelism, the timing performance improves even more, overcoming by far any possible advantage of a software implementation. We should consider that Octave is an interpreted language, and so the code is executed by an interpreter program that interprets it at runtime. Sometimes, an interpreted code has worse execution time than an optimized compiled code written, for instance, in C. We plan then to translate the Octave code in C as a next step. Even though preliminary results suggest that a 10X reduction of the CPU execution time can be achieved, this will not help reduce the BEAF part of the computation down to an acceptable level (an acceleration of more than 400X would be needed if an ASIC implementation with fifty antennas is the reference point). Therefore, the BEAF task will still require a hardware accelerator. An optimized C code may help, instead, to reduce the other tasks below the 1 s target.

Thanks to the parameterized implementation, our design can be easily adapted to various configurations with different number of transmitters and receivers and for various technology targets (FPGA or ASIC) with different available resources. This gives our design a high degree of flexibility and scalability.

Acknowledgment

This work was partially supported by the Italian Ministry of Education and Research under FIRB 2012 Project "MICENEA."

References

[1] S. C. Hagness, A. Taflove, and J. E. Bridges, "Three-dimensional FDTD analysis of a pulsed microwave confocal system for breast cancer detection: design of an antenna-array element," *IEEE Transactions on Antennas and Propagation*, vol. 47, no. 5, pp. 783–791, 1999.

[2] X. Li and S. C. Hagness, "A confocal microwave imaging algorithm for breast cancer detection," *IEEE Microwave and Wireless Components Letters*, vol. 11, no. 3, pp. 130–132, 2001.

[3] X. Li, S. K. Davis, S. C. Hagness, D. W. Van Der Weide, and B. D. Van Veen, "Microwave imaging via space-time beamforming: experimental investigation of tumor detection in multilayer breast phantoms," *IEEE Transactions on Microwave Theory and Techniques*, vol. 52, no. 8, pp. 1856–1865, 2004.

[4] E. J. Bond, X. Li, S. C. Hagness, and B. D. Van Veen, "Microwave imaging via space-time beamforming for early detection of breast cancer," *IEEE Transactions on Antennas and Propagation*, vol. 51, no. 8, pp. 1690–1705, 2003.

[5] M. R. Casu, M. Crepaldi, and M. Graziano, "A VHDL-AMS simulation environment for an UWB impulse radio transceiver," *IEEE Transactions on Circuits and Systems I*, vol. 55, no. 5, pp. 1368–1381, 2008.

[6] M. Cutrupi, M. Crepaldi, M. R. Casu, and M. Graziano, "A flexible UWB transmitter for breast cancer detection imaging systems," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '10)*, pp. 1076–1081, March 2010.

[7] M. R. Casu, M. Graziano, and M. Zamboni, "A fully differential digital CMOS UWB pulse generator," *Circuits, Systems, and Signal Processing*, vol. 28, no. 5, pp. 649–664, 2009.

[8] M. Crepaldi, M. R. Casu, M. Graziano, and M. Zamboni, "A mixed-signal demodulator for a low-complexity IR-UWB receiver: methodology, simulation and design," *Integration, the VLSI Journal*, vol. 42, no. 1, pp. 47–60, 2009.

[9] S. C. Hagness, A. Taflove, and J. E. Bridges, "Two-dimensional FDTD analysis of a pulsed microwave confocal system for breast cancer detection: fixed-focus and antenna-array sensors," *IEEE Transactions on Biomedical Engineering*, vol. 45, no. 12, pp. 1470–1474, 1998.

[10] S. K. Davis, E. J. Bond, X. Li, S. C. Hagness, and B. D. Van Veen, "Microwave imaging via space-time beamforming for early detection of breast cancer: beamformer design in the frequency domain," *Journal of Electromagnetic Waves and Applications*, vol. 17, no. 2, pp. 357–381, 2003.

[11] X. Li, E. J. Bond, B. D. Van Veen, and S. C. Hagness, "An overview of ultra-wideband microwave imaging via space-time beamforming for early-stage breast-cancer detection," *IEEE Antennas and Propagation Magazine*, vol. 47, no. 1, pp. 19–34, 2005.

[12] P. Rosingana, *Design of a UWB Imaging System for Breast Cancer Detection [M.Sc. thesis]*, Politecnico di Torino.

[13] J. W. Eaton, *Gnu Octave Manual*, Network Theory Ltd., 2002.

[14] UWCEM, "Numerical Breast Phantom Repository," http://uwcem.ece.wisc.edu/home.htm.

[15] ModelSim Reference Manual, v6.5e, Mentor Graphics, 2010.

[16] ISE Design Suite 13: Release Notes Guide, Xilinx, 2011.

[17] Design Compiler Reference Manual: Optimization and Timing Analysis Version D-2010.03, Synopsys, 2010.

[18] A. Pulimeno, M. Graziano, and G. Piccinini, "Udsm trends comparison: from technology roadmap to ultrasparc niagara2," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 7, pp. 1341–1346, 2012.

[19] S. V. Tota, M. R. Casu, M. R. Roch, L. Rostagno, and M. Zamboni, "Medea: a hybrid shared-memory/message-passing multiprocessor NoC-based architecture," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '10)*, pp. 45–50, March 2010.

[20] M. R. Casu, M. R. Roch, S. V. Tota, and M. Zamboni, "A NoC-based hybrid message-passing/shared-memory approach to CMP design," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 261–273, 2011.