ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **10**, Number 4, 2007, 361–379

Accelerating The Computation of The Physical Parameters Involved in Transcranial Magnetic Stimulation Using FPGA Devices

O. CREŢ[1], I. TRESTIAN[1], F. DE DINECHIN[3]
L. DARABANT[2], R. TUDORAN[1], L. VĂCARIU[1]

[1] Computer Science Department, Technical University of Cluj-Napoca,
[2] Electrotechnics Department, Technical University of Cluj-Napoca,
26 Bariţiu street, Cluj-Napoca, Romania
E-mail: Octavian.Cret@cs.utcluj.ro

[3] LIP, École Normale Supérieure de Lyon
UMR CNRS/INRIA/ENS-Lyon/Université Claude Bernard Lyon 1
46 allée d'Italie, 69364 Lyon cedex 07, France
E-mail: Florent.de.Dinechin@ens-lyon.org

Abstract. In the last years the interest for magnetic stimulation of the human nervous tissue has increased, because this technique has proved its utility and applicability both as a diagnostic and as a treatment instrument. Research in this domain is aimed at eliminating some disadvantages of the technique: the lack of focalization of the stimulated human body region and the reduced efficiency of the energetic transfer from the stimulating coil to the tissue. Designing better stimulation coils is so far a trial-and-error process, relying on very compute-intensive simulations. In software, such a simulation has a very high running time (several hours for complicated geometries of the coils). This paper proposes and demonstrates an FPGA-based hardware implementation of this simulation, which reduces the computation time by 2-3 orders of magnitude. Thanks to this powerful tool, some significant improvements in the design of the coils have already been obtained.

1. Introduction

The preoccupation for improving the quality of life, for persons with different handicaps, led to extended research in the area of functional stimulation. Due to its advantages compared to electrical stimulation, magnetic stimulation of the human nervous system is now a common technique in modern medicine [1].

A difficulty of this technique is the need for accurate focal stimulation. Another one is the low efficiency of power transfer from the coil to the tissue. To address these difficulties, coils with special geometries must be designed.

Because of the diversity of the medical applications involved, the design of the coils requires testing a large number of geometries in order to find an adequate solution for the desired application [2].

One of the major problems that appear in the design phase of such coils is the computation of the inductivity of the stimulating coil. For simple shapes of the coils (circular), one can determine analytical computation formulas [3] which are extremely complicated. When, however, the shape and the spatial distribution of the coil's turns do not belong to one of the known structures, a numerical method needs to be used for determining the inductivity of the coils.

The idea of the computation method is to divide the coils in small portions. Starting from this method, two computation systems are presented in the paper:

The first one is classical and it just consists of a software implementation (Matlab);

The second one consists of realizing a hardware architecture that exploits the intrinsic parallelism of the problem. The physical support of this architecture is an FPGA device.

The problem with the software implementation is its running time. Coils are designed by trial-and-error, and this approach is impractical if each trial requires half a day of computation. Besides, as this time grows with the complexity of the coil, it prevents designing complex coils. This paper shows that FPGA-based hardware acceleration is able to solve this bottleneck.

The rest of the paper is organized as follows: Section 2 presents the inductivity computation process. In Section 3 some considerations are made about the software implementation. Section 4 deals with the hardware implementation and presents the global architecture as well as the architecture of the main building blocks. Section 5 makes a comparison between the two implementations in terms of running speed and results obtained, and Section 6 presents the main conclusions of this work.

2. Inductivity computation

The simulation of magnetic stimulators with complex forms requires dividing their coils in several parts. The self-inductance of the circuit, divided in n parts, can be computed with formula (1). This mainly adds up the self-inductivities of the separate segments with the mutual inductivities of all the involved segments. The method is described in [4] and in more detail in [5].

$$L = \sum_{k=1}^{n} L_k + \sum_{k=1}^{n} \sum_{i=1}^{n} M_{ki}, \quad for \ \ (i \neq k) \tag{1}$$

The self-inductivity of a short straight conductor, with round cross-section, for low frequencies, is given below:

$$L = \frac{\mu_0 l}{2\pi} \left(\ln \frac{2l}{r} - \frac{3}{4} + \frac{128}{45\pi} \frac{r}{l} - \frac{r^2}{4l^2} \right), \tag{2}$$

where l is the conductor's length, and r the radius of its cross-section. The mutual inductivity between two straight conductors converging into a point is evaluated as:

$$M = \frac{\mu_0}{4\pi} \cos\varphi \left[a \ln \frac{a+b+c}{c+a-b} + b \ln \frac{a+b+c}{c+b-a} \right]. \tag{3}$$

The given quantities are represented in Fig. 1, with a and b representing the length of the conductors, and ϕ the angle between them.

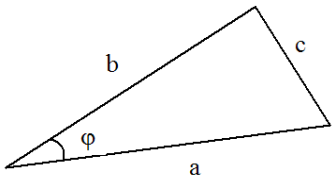

Fig. 1. Computing the mutual inductivity between two converging conductors.

For the general case, we consider two conductor segments in space. The first segment is delimited by the points of coordinates (x_a, y_a, z_a) and (x_b, y_b, z_b), while the second segment is delimited by the points of coordinates (x_c, y_c, z_c) and (x_d, y_d, z_d), as shown in Fig. 2.

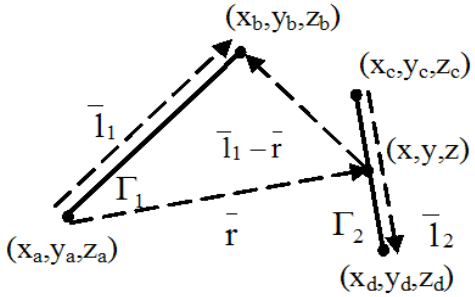

Fig. 2. Two segments in space.

On the second segment, we consider a point of coordinates (x, y, z). The parametric equation of the second segment is:

$$\begin{cases} x = x_c + (x_d - x_c)\ t, \\ y = y_c + (y_d - y_c)\ t\ , \\ z = z_c + (z_d - z_c)\ t\ . \end{cases} \tag{4}$$

With the above geometrical coordinates, we can find the mutual inductivity between these segments (using Neumann formula). For two circuits, Γ_1 and Γ_2, in a homogenous media with μ permeability, the mutual magnetic flux Φ_{21} is:

$$\Phi_{21} = \int\limits_{S_{\Gamma_2}} \overline{B}_{21}\overline{dS} = \int\limits_{\Gamma_2} \overline{A}_{21}\overline{dl}_2. \tag{5}$$

Since circuits Γ_1 and Γ_2 are shaped like two straight segments, the mutual flux can be evaluated by integrating the magnetic vector potential created by the first segment along the second one. Considering the magnetic vector potential generated by a conducting segment, the mutual inductivity can be computed using the following equation:

$$L_{21} = \frac{\mu_0}{4\pi} \cdot \oint\limits_{\Gamma_2} \ln \frac{\left|\bar{l}_1 - \overline{r}\right| + l_1 - \frac{\bar{l}_1 \cdot \overline{r}}{l_1}}{r - \frac{\bar{l}_1 \cdot \overline{r}}{l_1}} \cdot \frac{\overline{l_1}}{l_1} \cdot \overline{dl}_2. \tag{6}$$

The vectors in equation (6) are:

$$\begin{aligned}
\overline{dl}_2 &= \left(x'(t) \cdot \bar{i} + y'(t) \cdot \bar{j} + z'(t) \cdot \bar{k}\right) \cdot dt = \\
&= \left((x_c - x_d) \cdot \bar{i} + (y_c - y_d) \cdot \bar{j} + (z_c - z_d) \cdot \bar{k}\right) \cdot dt\,, \\
\overline{dl}_2 &= \left(x'(t) \cdot \bar{i} + y'(t) \cdot \bar{j} + z'(t) \cdot \bar{k}\right) \cdot dt = \\
&= \left((x_c - x_d) \cdot \bar{i} + (y_c - y_d) \cdot \bar{j} + (z_c - z_d) \cdot \bar{k}\right) \cdot dt\,, \\
\bar{l}_1 &= (x_b - x_a) \cdot \bar{i} + (y_b - y_a) \cdot \bar{j} + (z_b - z_a) \cdot \bar{k}, \\
\overline{r} &= (x - x_a) \cdot \bar{i} + (y - y_a) \cdot \bar{j} + (z - z_a) \cdot \bar{k}, \\
\bar{l}_1 - \overline{r} &= (x_b - x) \cdot \bar{i} + (y_b - y) \cdot \bar{j} + (z_b - z) \cdot \bar{k},
\end{aligned}$$

while coordinates x, y, z are expressed as a function of parameter t, according to (4). The limits of the integral in equation (6) are given by $t \in [0, 1]$.

As can be seen in the above formulas, the operations involved in computing the inductivity of a coil are vector operations. A logarithm, some division, addition and multiplication operations can also be observed in equation (6). Equation (1) is mainly an accumulation of mutual inductivities. The integral according to t will be evaluated using the trapezoidal method.

3. Presentation of the algorithm

A coil is made up of a certain number of turns rolled around a central rod. Each turn can be considered as a perfect circle. The coil is structured on several vertical stages. On each stage there are more turns (horizontal turns). The outer radius of the coil will be further denoted by a. Other parameters are the diameter of the metallic turn and the distance (insulation) between consecutive turns.

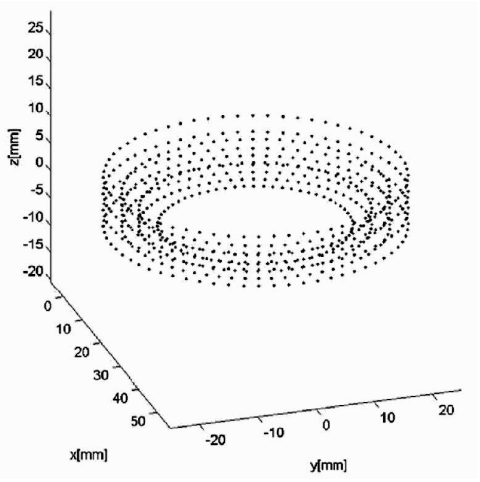

Fig. 3. Coil approximation using a finite number of points.

It is possible to have a different number of turns on every vertical stage. It is also possible to have a variable number of vertical stages, as shown in Fig. 3, where one can also notice the different number of turns on each stage.

A complete magnetic stimulation device contains a Slinky coil. Considering a coil with N turns, the "Slinky-k" coil is generated by spatially locating these turns at successive angles of $i \times 180/(k-1)$ degrees, were $i = 0, 1, \ldots, k-1$. If the current passing through this coil is I, then the central leg carries the total current $N \times I$. These coils are shown in Fig. 4, where each rectangle represents a leaf of the coil, viewed in perspective.

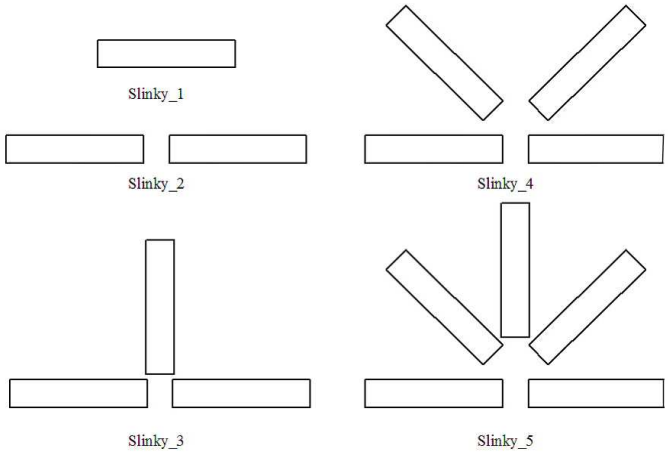

Fig. 4. Magnetic coil structures of the stimulation device.

For computing the inductance, the turns could be approximated by a finite number of points. We considered, after a series of tests, that a suitable amount of points on a turn is 64.

We therefore need to compute the inductance of such a coil. We have to take each of the 64 points and combine them into segments made up of one point and the next (consecutive) point in a certain direction. After this, each segment, one by one, is held as a reference. Then, the formula presented in this paragraph is applied using this reference segment and all the other segments on the coil. For each pair of segments a value is obtained. These values must be added in order to obtain the coil's total inductance.

There are two phases for the functioning of the software implementation:

- In *Phase 1*, the coordinates of the points are generated. These are computed using trigonometric functions. The results produced in this phase are also used in the hardware implementation.
- In *Phase 2*, the actual computation of the values is performed. As mentioned before, in order to compute the inductivity, we accumulate the values corresponding to the mutual inductivities.

When we evaluate pairs of segments on the coil, three distinct cases can arise:

- The two segments are actually the same segment. In this case we add the segment's own self-inductivity given by Equation (2). Since all segments have the same characteristics, this value is the same for all segments.
- The two segments are neighboring segments, that is, they have exactly one common point. In this case their mutual inductivity is given by Equation (3). Since the configuration is the same for all pairs of neighboring segments, this too is a constant value.
- The two segments are neither the same nor neighboring; they are two distinct segments in space. For complex configurations consisting of a large number of turns, this is the most general case, which accounts for most of the computation time. This case is evaluated using Equation (4).

The software implementation takes into account these considerations. We evaluate the third case by computing the mutual inductivities of all non-intersecting segments. The self-inductivity of the segments and the mutual inductivities of each segment against its neighboring ones are multiplied by the number of segments and accumulated at the end.

In order to evaluate the mutual inductivity of two separate segments in space we have introduced 5 variables denoted var_1 to var_5 which correspond to vector operations involving the two segments as described in the previous section. These variables will also be used in the next section. The points (x_0, y_0, z_0) and (x_1, y_1, z_1) correspond to the extremities of the first segment, while (x_2, y_2, z_2) and (x_3, y_3, z_3) correspond to the extremities of the second one.

$$var_1 = (x_1 - x_0) \cdot (x_3 - x_2) + (y_1 - y_0) \cdot (y_3 - y_2) + (z_1 - z_0) \cdot (z_3 - z_2),$$
$$var_2 = \sqrt{(x_3 - x_2)^2 + (y_3 - y_2)^2 + (z_3 - z_2)^2},$$

$$var_3 =$$
$$= \sqrt{((x_3 - x_0) + (x_0 - x_1) \cdot t)^2 + ((y_3 - y_0) + (y_0 - y_1) \cdot t)^2 + ((z_3 - z_0) + (z_0 - z_1) \cdot t)^2},$$
$$var_4 = \sqrt{(x_0 + (x_1 - x_0) \cdot t - x_2)^2 + (y_0 + (y_1 - y_0) \cdot t - y_2)^2 + (z_0 + (z_1 - z_0) \cdot t - z_2)^2},$$
$$var_5 = (x_3 - x_2) \cdot (x_0 + (x_1 - x_0) \cdot t - x_2) + (y_3 - y_2) \cdot (y_0 + (y_1 - y_0) \cdot t - y_2) +$$
$$+ (z_3 - z_2) \cdot (z_0 + (z_1 - z_0) \cdot t - z_2).$$

These values are used for further computing the accumulation value:

$$\text{Accumulator} = \text{Accumulator} + \frac{var_1}{var_2} \cdot \log \frac{var_3 + var_2 - \dfrac{var_5}{var_2}}{var_4 - \dfrac{var_5}{var_2}}. \tag{7}$$

In the formulas above there is a variable t, which is a factor occurring in the integral from Equation (6). Therefore several values need to be considered for t and then the integral needs to be computed using the trapezoidal method for values in the interval (0, 1) = [0.1, 0.2, 0.3 ... 0.9]. The process above needs to be repeated several times in order to compute the final inductivity of the coil.

The main drawback of a software implementation is the extremely high running time. It can be in the order of tens of minutes even for simple configurations, while for complex geometries of the coils it can exceed several hours (Example: for a 58-turns coil, about 5 hours computation time on a recent PC).

Once the software simulation had been validated against actual coils, it was decided to try to accelerate it using custom hardware.

4. Hardware implementation

4.1. Field-programmable gate arrays

A Field-Programmable Gate Array (FPGA) is a semiconductor device containing programmable logic components ("logic blocks"), and programmable interconnects. Logic blocks can be programmed to perform simple or more complex functions. In most FPGAs, the logic blocks also include memory elements, from flip-flops to more complete blocks of memories.

A hierarchy of programmable interconnects allows logic blocks to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. Logic blocks and interconnects can be programmed by the customer/designer, after the FPGA is manufactured, to implement any logical function. Their advantages include a shorter time to market, ability to re-program in the field to fix bugs, and lower non-recurring engineering costs [6].

4.2. Floating point operators - FPLibrary

Several libraries of floating-point operators for FPGAs have been published in the last few years. In this work, we use FPLibrary, developed at École Normale Supérieure

de Lyon and freely downloadable from [7]. Mantissa size and exponent size parameterize each operator in this library, allowing one to choose the precision and the dynamic range of the numbers. It provides operators for addition, subtraction, multiplication, division and square root, some useful conversions and some elementary functions (currently exponential, logarithm and sine/cosine), in combinatorial or pipelined flavor. It is written in portable VHDL. FPLibrary also offers operators for the alternative *logarithmic number system* [8].

The Core Generator tool, which comes with the Xilinx ISE, also offers floating-point operators. FPLibrary was chosen essentially because it offers a logarithm [9] which is not available in the Core Generator. However, it also proved more area-efficient. As our design requires a large number of operators in a tree-like pipeline, latency was not our main concern.

4.3. First system architecture

The hardware implementation implies the same two phases as the software one, but Phase 1 is not computation-intensive and its implementation is kept in software.

In the Fig. 5 below a block diagram of the system is displayed. Three main blocks can be distinguished. The most important block is the pipeline stage, which receives values, computes them, and in a final stage accumulates them. The pipeline will be described afterwards.

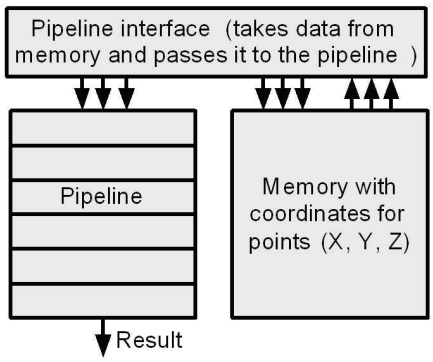

Fig. 5. Architecture of the hardware system.

The coordinates are stored in a Block RAM memory. There are 3 memories, one for each coordinate, X, Y, and Z. The synchronization logic, which gives the data to the pipeline, is implemented in a special interface. This interface consists of counters and latches. The counters are orchestrated to generate the proper addresses, while the latches are needed to implement a caching logic, which saves some of the memory used.

The pipeline stage consists of several sub-stages, based on the computations involved:

- A first stage computes the variables var_1, var_2, var_3, var_4 and var_5 (the formulas were given in the previous section). There are several operations, which are

common to the 5 variables. The pipeline reuses the corresponding intermediate values.

- The second stage computes the value to be accumulated and it is presented in Fig. 6. The latencies in this part of the pipeline are pretty large, up to 14 cycles, corresponding to waiting for a square root and about 29 cycles waiting for the logarithm and other operations to finish.
- The third stage is the accumulator. Because FPLibrary doesn't provide such a component one needed to be improvised using the existing resources. Special considerations were made in order to work around the specific latency that a simple adder introduces.

Not only the specific operators, but also buffer stages are used, which compensate the latency introduced by some of the operators. For instance, the addition operator always introduces three cycles of latency and this must be compensated with three buffers. This also goes for the multiplication operator, which introduces four cycles of latency.

The design of the Accumulator is the most important part of the pipeline's architecture, since it computes intermediary values and at the end provides the final result. As mentioned above, special considerations need to be made with regard to the accumulator because of the latencies introduced by the adders in the FPlibrary (3 cycles).

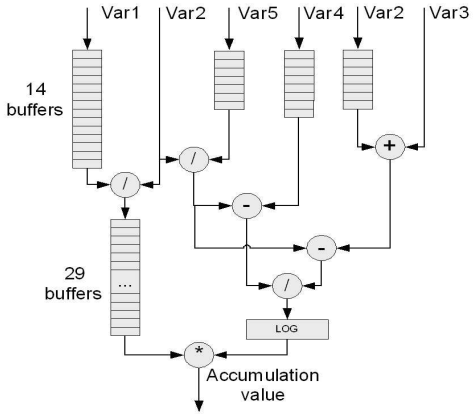

Fig. 6. Second stage. Computing accumulation value.

The values that will be accumulated come and enter the accumulator stage on the pin located to the left. The accumulator has a classical structure, using a feedback input. The process can be seen below in Fig. 7.

Input	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8	N_9				
Output				N_1	N_2	N_3	$N_1 + N_4$	$N_2 + N_5$	$N_3 + N_6$	$N_1 + N_4 + N_7$	$N_2 + N_5 + N_8$	$N_3 + N_6 + N_9$	

Fig. 7. Adder latency issues.

The input numbers come serially, N_1, N_2, N_3, N_4, etc. The adder at the beginning of the accumulating stage adds these numbers. Because of the 3 cycles clock latency, when N_1, N_2, N_3 have been inserted in this adder, the next numbers that come are to be added to the numbers that have been inserted 3 cycles before. For example N_1 is added with N_4, N_5 with N_2 and N_6 with N_3 respectively. The same goes for the next numbers, N_7, N_8 and N_9.

All the numbers are added but at the end three sums are generated: one for the numbers N_1, N_4, N_7 ... N_{3k+1}, the next for the numbers N_2, N_5, N_8 .... N_{3k+2}, and the last for the numbers N_3, N_6, N_9 ... N_{3k} (as shown in Fig. 8).

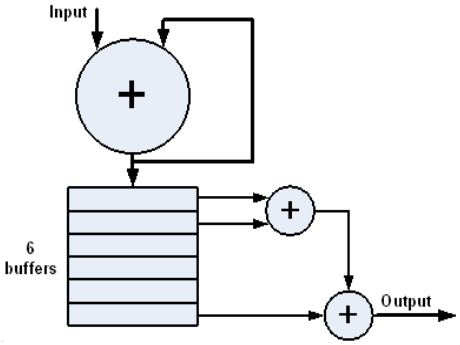

Fig. 8. Architecture of the accumulator.

These three sums are added in the final stage of the accumulator to generate the exact result. This final stage consists of three registers that delay the three corresponding sums. We add these three registers together in order to generate the final result.

One idea regarding the accumulating stage of the computation is to keep the other stages in a Simple Precision Format (32 bits) and enhance only the accumulator (64 bits, or only with a larger mantissa). Such a design would greatly limit the error losses corresponding to the accumulation of numbers of different ranges.

4.4. Hardware implementation issues

The performance and feasibility of the hardware implementation largely depends on its physical support. Our hardware platform was a Digilent Inc. board populated with a Xilinx Virtex2PRO30 FPGA device. The problem with this implementation was that it is quite large: it depleted the space of the FPGA device we had available at this moment. To estimate the total space needed, we synthesized the design for a larger FPGA device (a Virtex4 160LX). A report of the device utilization is shown below:

Selected Device:	4vlx160ff1148-12	
Number of Slices:	23656 out of 67584	35%
Number of Slice Flip Flops:	20834 out of 135168	15%
Number of 4 input LUTs:	44515 out of 135168	32%

The maximum frequency for this implementation was reported as 137.552 MHz.

As we can see the implementation fits without problems on this Virtex4 board. Regarding an implementation on our Virtex2Pro board two options were available.

The first option was to reduce the precision at which the pipeline operated. This ensured a reduction of both the buffer stages that provided the synchronization between the stages and a reduction in size of the operators.

This option was first implemented. We reduced the mantissa of the operands by 10 bits. Instead of a large mantissa having 23 bits, the mantissa now had only 13 bits. Although the design fitted on a Virtex-II Pro board at about 98% of its capacity, the results obtained with this method were discouraging. They were more then 30% off from the actual result provided by Matlab. Therefore another method needed to be found.

The next option was to reduce the frequency at which the pipeline stage operates and time-multiplex some of the resources (square root – three occurrences in design, some of the adders). This has the advantage of preserving the pipeline's precision, the cost being a reduction in speed.

Selected Device:	2vp30ff896-6	
Number of Slices:	13380 out of 13696	97%
Number of Slice Flip Flops:	15350 out of 27392	56%
Number of 4 input LUTs:	24156 out of 27392	88%

Of course we have a reduction in operating frequency: 85.714 MHz related to the weaker characteristics of the FPGA device and the more precise timing requirements.

4.5. Error analysis of the first architectures

The numerical results from this first architecture always stayed within 3–4% of the Matlab results, considered as a reference. The Matlab computations are performed in double precision floating point (53 bits of mantissa). Our computations are performed in a format of a much lower precision (16 bits of mantissa). Of course, the discrepancies come from this difference. Let us now look at it in more details. In such an accumulation of a very large number of floating-point summands, there are two sources of error which cumulate their effects:

- Firstly, the datapath of Fig. 6 computes each elementary summand with an error which is typically in the order of 2-16. since we have 16 bits of precision. This is a relative error, which means that it affects the last bits of the mantissa of each summand.

- Secondly, all these elementary values are accumulated, and again this accumulation involves rounding errors.

Let us first consider the first source of error. In the very worst case, if we sum N summands of similar magnitudes, each holding a positive error ε, the sum holds an error $N\varepsilon$. In our largest coil test, $N = 10.1920^2$: if ε is an error on the last bit of the mantissa, the cumulative rounding error may invalidate up to $\log_2(10.1920^2) = 25$

bits of the result, when the mantissa of a single-precision number holds 24 bits only. However, this is a worst-case situation: in an actual simulation, rounding errors may be positive and negative, so that they compensate each other in average. Some models predict an expected cumulated error equal to the square root of the worst case, which would invalidate in our case 12 bits only. Besides, summands do not all have the same order of magnitude, therefore the distribution of errors with respect to the final result is more favorable than the worst case situation. Finally, all the summands are much smaller than the result: we instrumented the software to measure that the largest summand is 2^{15} times smaller in absolute value than the final sum. As a consequence, relative to the final sum, an error on the summands is also 2^{15} times smaller.

Still, this shows that we may require increasing the precision of the floating-point format to use this architecture on much larger coils. The only limit in this work was the target FPGA, which is small by current standards: porting our architecture on a larger FPGA will allow us to improve the working precision of the pipeline to 24 bits, which should be more than enough. Now let us consider the accumulation error. Its cumulated effect is much more dramatic than the effects of the summand errors: First, as we know that the final result is much larger in magnitude than the largest summand, this will soon be the case in the accumulation: most of the additions will add a large current sum with a much smaller summand. In floating-point, in such cases, the mantissa of both numbers are aligned, and this means discarding the lower bits of the smaller result – and sometimes all the bits as seen on Fig. 9. Of course, discarding many bits of a summand brings in a much larger error than the error which was carried by the summand itself.

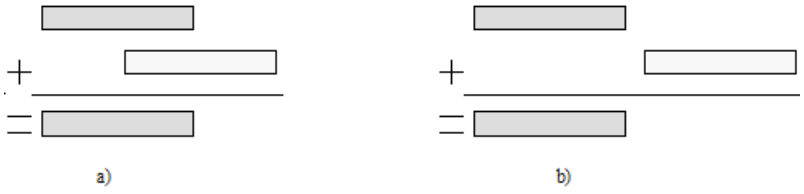

Fig. 9. Accumulation with very different exponents: a) only part of the bits are discarded; b) all the bits are discarded.

4.6. An improved architecture with a large accumulator

As shown in subsection 4.5, in the process of accumulating large amounts of numbers, of very different exponents, a significant error can appear. In order to eliminate it, a fixed-point giant accumulator was designed. The numbers which must be added are aligned with respect to the giant accumulator's fixed point (as shown in Fig. 10), and therefore there will be no losses if this accumulator's width is set to be large enough so that no numbers are shifted outside its range.

The giant accumulator is conceived to receive a floating point number which, according to its exponent, will be properly aligned for the accumulating step. The

final result will be a fixed-point binary number, where the position of the radix point is previously set by the user.

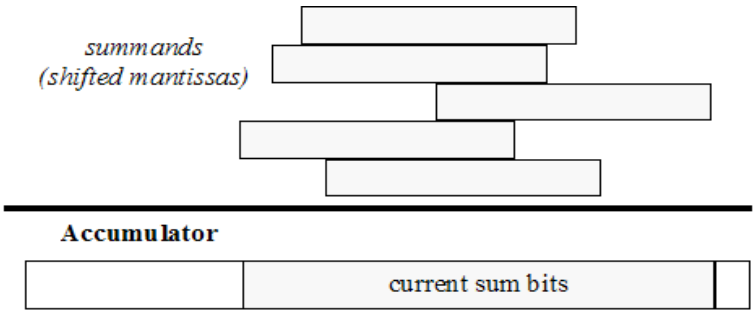

Fig. 10. The Giant Accumulator's principle.

In addition, the large accumulator is easier to pipeline: compared to Fig. 8 (the old accumulator) it removes all the shifts that were on the critical path of the loop by keeping the current sum as a large fixed-point number (typically much larger than a mantissa, see Fig. 10). There is still a loop, but it is now a fixed-point accumulation for which current FPGAs are highly efficient. Specifically, the loops uses fast-carry logic and involves only the most local routing. For illustration, for 64-bits (11 bits more than the DP mantissa), a Virtex4 with the slowest speed grade (–10) runs such an accumulator at more than 200 MHz, while consuming only 64 CLBs.

The shifters now only concern the summand (see Fig. 11), and can be pipelined as deep as required by the target frequency.

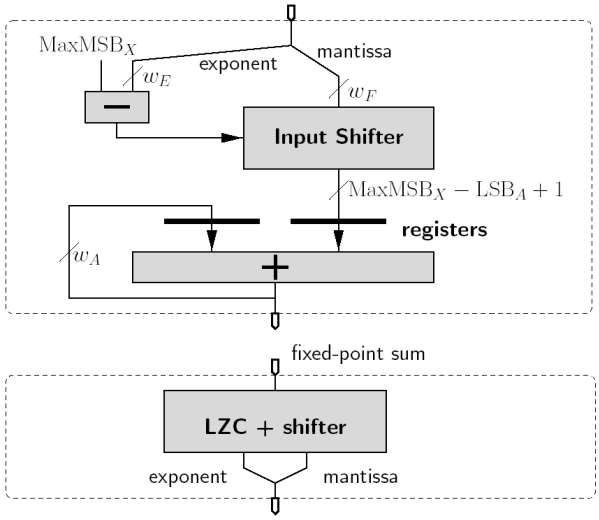

Fig. 11. The proposed accumulator (top) and postnormalisation unit (bottom).
Only the registers on the accumulator itself are shown.
The rest of the design is combinatorial and can be pipelined arbitrarily.

The normalization of the result may be performed at each cycle, also in a pipelined manner. However, most applications won't need all the intermediate sums: they will output the fixed-point accumulator (or only some of its most significant bits), and the final normalization may be performed offline in software, once the sum is complete, or in a single normalizer shared by several accumulators (case of matrix operations). Therefore, it makes sense to provide this final normalizer as a separate component, as shown by Fig. 11.

For clarity, implementation details are missing from these figures. For example, the accumulator stores a two's complement number, so the shifted summand has to be sign-extended. The normalization unit also has to convert back from two's complement to sign/magnitude. All this isn't on the critical path of the loop either. Table 1 shows the results obtained from using these two types of accumulators, for some coils configurations. The configurations are given in the following format: number of turns per leaf, each leaf being structured on several vertical levels (for instance, configuration 7,6,6,6 – 5,5 – 5,5 – 6,6,6,7 has 25 turns on the first and fourth leaf, arranged on 4 levels; the first level has 7 horizontal turns, and the next ones 6 turns; the second and third leaves have 10 turns).

Table 1. The values generated by the two accumulators

Nr.	Configuration	Classical FP Accumulator	Giant Accumulator
1	5,5,5,5,5 – 5,5 – 5,5 – 5,5,5,5,5	84.8	85.7
2	5,5,5,5,5 – 4,3,3 – 3,3,4 – 5,5,5,5,5	87.8	88.8
3	7,6,6,6 – 5,5 – 5,5 – 6,6,6,7	77.9	78.7
4	7,6,6,6 – 4,3,3 – 3,3,4 – 6,6,6,7	80.6	81.7
5	8,7,5,5 – 5,5 – 5,5 – 5,5,7,8	75.9	74.6
6	8,7,5,5 – 4,3,3 – 3,3,4 – 5,5,7,8	78.9	77.7
7	8,7,6,4 – 5,5 – 5,5 – 4,6,7,8	75.4	74.2
8	8,7,6,4 – 4,3,3 – 3,3,4 – 4,6,7,8	78.6	77.1
9	8,7,7,3 – 5,5 – 5,5 – 3,7,7,8	73.9	72.6
10	8,7,7,3 – 4,3,3 – 3,3,4 – 3,7,7,8	76.9	75.5

The results from the giant accumulator are more accurate, and therefore this one is more reliable in situations where a big precision is needed. For even more accurate results, one could set a larger width of the giant accumulator (currently this width is set to 64 bits).

5. Experimental results

The main achievement of the hardware implementation over the software one is the reduction in computation time. By performing one accumulation per clock cycle the hardware solution is indeed efficient and can be used even for the most complex magnetic stimulation systems.

In terms of complexity, both implementations, in software and in hardware, have the same complexity, $O(n^2)$ with n being the number of distinct segments. As we have

said the specific hardware structure performs one accumulation per clock cycle. That means that each clock cycle, a mutual inductivity between two segments is evaluated. The software implementation performs the same computations in a longer time.

We have analyzed our software and hardware implementations using three distinct configurations (Fig. 12). The values produced by the implementations are given in Table 2, where we can see the comparison between the results provided by the software and the hardware solutions.

First we analyzed simpler cases, 1 to 4 turns. The outer turns are the widest turns on the coil while the inner turns are the neighbors of the outer turns located closer to the center. Then the results for the given configurations are presented. The analyzed quantity was the inductivity. We give also the number of segments, which determines the complexity.

In an actual simulation, the rounding errors compensate each other, which explains why our results are still accurate. However, it shows that we will require increasing the precision of the floating-point format to use this architecture on larger coils. Fortunately, this extra precision is mostly useful in the final accumulator.

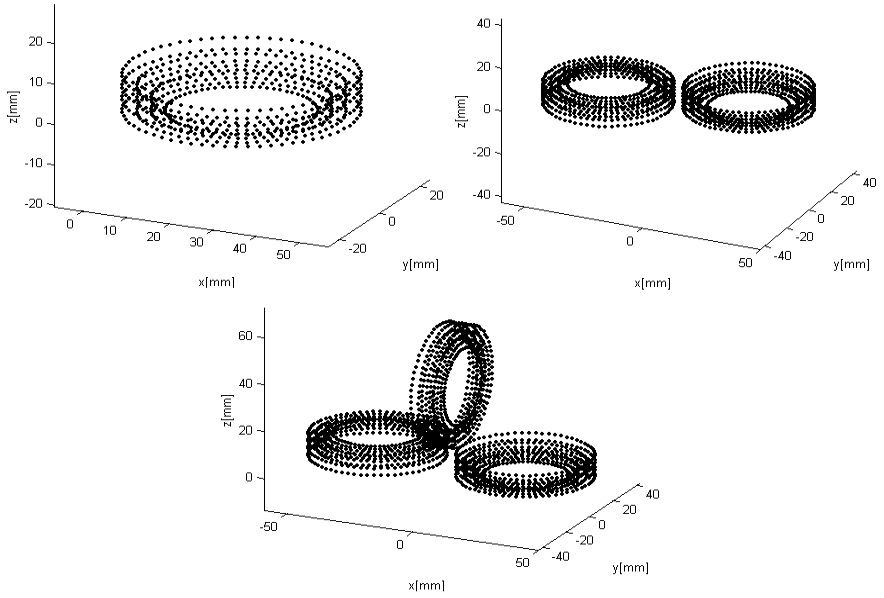

Fig. 12. Analyzed configurations.

The results of the two methods analyzed for the three configurations mentioned always stayed in the range of 3–4% of each other, with the Matlab result being slightly bigger than the result given by the hardware implementation. This can be attributed to the fact that Matlab uses by default double precision while in our system we have used only single precision operations. Indeed, a rough worst-case error analysis tells us that the accumulation, in the largest coil test, of 10.1920^2 floating-point numbers introduces a cumulative rounding error that may invalidate up to $\log_2(10.1920^2) = 25$ bits of the result, when the mantissa of a single-precision number holds 24 bits only.

This is a worst-case situation: in an actual simulation, these rounding errors compensate each other, which explains that our results are still accurate. However, it shows that we will require increasing the precision of the floating-point format to use this architecture on larger coils. Fortunately, this extra precision is mostly useful in the final accumulator.

Table 2. Comparison of results

Configuration	Inductivity (hardware) [μH]	Inductivity (software) [μH]	Duration (hardware) [no. of clock cycles]	Running speed* (hardware) [seconds]	Running speed (software) [seconds]
Slinky_1 coil (1 outer turn)	0.097	0.097	40 960	0.00047	4.2
Slinky_1 coil (2 outer turns)	0.30	0.30	163 840	0.00194	18
Slinky_1 coil 4 turns (2 out. 2 in.)	0.92	0.93	655 360	0.00764	72
Slinky_1 coil (10 turns)	3.81	3.9	4 096 000	0.04705	420
Slinky_2 coil (10-10)	8.4	8.6	16 384 000	0.19411	1 680
Slinky_3 coil (10-10-10)	13.32	13.6	36 864 000	0.42941	3 600

* at 85.714 MHz, clock period 11.66 ns.

The flexibility of FPGAs allows us to use different precisions in different parts of the architecture. Besides, a format intermediate between single and double precision may be used. It should be noted that a more accurate pipeline requires more hardware, but the same execution time: it still computes one accumulation per cycle. As a global comparison, the hardware solution runs approximately two-three orders of magnitude faster then the software one. The frequency is related to the physical board we had available, but for a more recent FPGA chip (i.e. Virtex4 or Virtex5), the device's capacity as well as the working frequency will increase, thus leading to an improved performance.

As mentioned above, an adequate geometry of the stimulation coil can lead to a better focality of the stimulus (the ability of a coil to stimulate a small area of the tissue) and it can also improve the efficiency of the energy transfer from the coil to the target tissue. The form and size of the turns, their position inside the coil, and the insulation gap between turns are all important parameters that should be considered when designing a magnetic coil. Therefore, in order to establish the most suitable coil geometry for a specific medical application, a large number of structures have to be tested, making of coil design a trial-and-error process, even if the risk involved is only computation time.

In a very recent paper [10], we analyzed the influence that space distribution of the magnetic coils' turns has on the efficiency of energy transfer from the stimulator to the target tissue. The analysis was performed for a Slinky_3 coil configuration (see Fig. 12), with applications on transcranial magnetic stimulation (TMS). It turned out that the electrical energy dissipated in the circuit of the stimulator - required in order to achieve the activation threshold - is 25% lower for the most efficient configuration than for the less efficient one, and the coil heating per pulse is also 35% smaller.

However, the tested coils only had 18 turns, because larger structures could not be efficiently analyzed using the software implementation. This is a major drawback, since recent references in this field [2] stipulate that the inductivity of the stimulating coil should be larger than 30 μH for an improved efficiency of the energy transfer from the coil to the target tissue.

Therefore, based on the inductivity calculus described in this paper and performed with a less time-consuming computation technique - the hardware implementation, we were able to estimate the inductivity of 25 different configuration of a Slinky_4 coil with 70 turns. The coil has an outer radius of each leaf equal to 30 mm, an insulation gap between turns of 0.2 mm and the wire radius is 1 mm.

This estimation, never performed before, allows further investigations regarding the importance of coil configuration and geometry on the focality and efficiency of the stimulation (it goes without saying that the hardware technique implemented can be applied to the inductivity computation of any other structure of coil).

Table 3 contains the values of the inductivity computed for the given configurations of coils. The first 10 values have already been presented in Table 1. For a better understanding, Fig. 13 represents configuration 2 of Table 3.

Table 3. Computed inductivity for the given configurations of coils

Nr.	Inductivity L [μH]	Configuration
1	75.050885	9,8,5,4 – 5,5 – 5,5 – 4,5,8,9
2	78.079665	9,8,5,4 – 4,3,3 – 3,3,4 – 4,5,8,9
3	72.23413	10,5,5,5 – 5,5 – 5,5 – 5,5,5,10
4	75.135825	10,5,5,5 – 4,3,3 – 3,3,4 – 5,5,5,10
5	70.11591	10,6,6,3 – 5,5 – 5,5 – 3,6,6,10
6	73.19011	10,6,6,3 – 4,3,3 – 3,3,4 – 3,6,6,10
7	63.69439	11,8,6 – 5,5 – 5,5 – 6,8,11
8	66.33864	11,8,6 – 4,3,3 – 3,3,4 – 6,8,11
9	64.692085	10,8,7 – 5,5 – 5,5 – 7,8,10
10	64.16614	10,8,7 – 6,4 – 6,4 – 7,8,10
11	63.9505	10,8,7 – 4,3,3 – 3,3,4 – 7,8,10
12	65.897165	9,8,8 – 5,5 – 5,5 – 8,8,9
13	68.82137	9,8,8 – 4,3,3 – 3,3,4 – 8,8,9
14	63.62399	10,9,6 – 5,5 – 5,5 – 6,9,10
15	66.443895	10,9,6 – 4,3,3 – 3,3,4 – 6,9,10

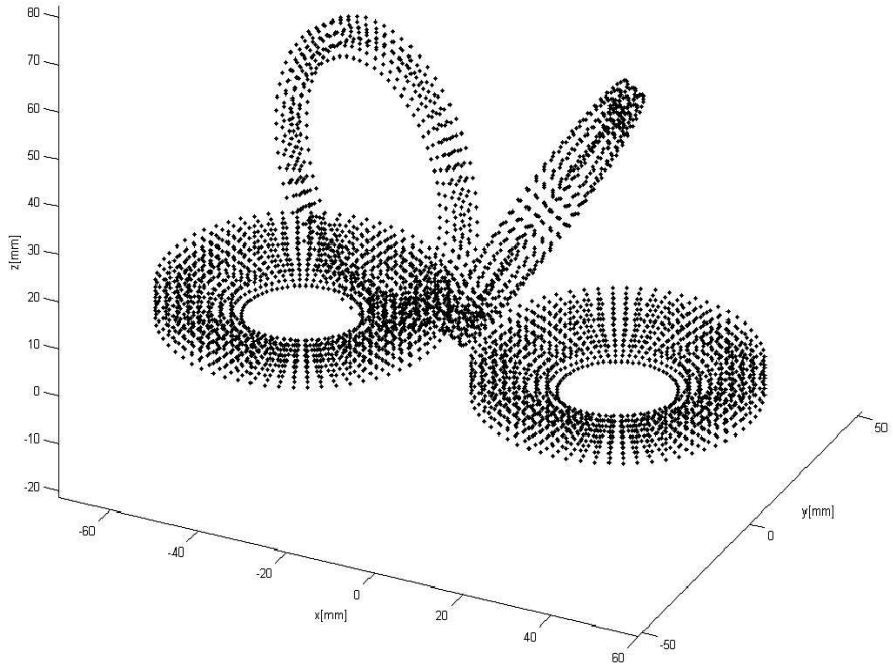

Fig. 10. Configuration 12 of a Slinky_4 coil with 70 turns.

These values are now already in use for the estimation of energetic parameters of stimulation coils and results are to be published in our future works. But it is now important to emphasize the fact that performing a software computation of the inductivities of such a large number of different coils (each one divided in 4 480 segments) would have been practically very difficult to perform.

6. Conclusions and future work

The equipment used in magnetic stimulation of the nervous system is costly and bulky. This is mainly due to the fact that the currents flowing through the stimulation coil are very intense (kA), leading to coil heating and strong electromagnetic forces that might destroy the coil. Therefore, magnetic coil design is one of the most important aspects of the technique of magnetically stimulating the nervous system.

Since every medical application requires its own optimal structure of the magnetic coil, the results emphasized in this paper can play an important role for future work on coil design.

Because of the large amount of operations involved (several tens of millions just for one coil) it is very hard to debug such a hardware system at least at an acceptable level, but the obtained results show an excellent concordance with those obtained in software. Our implementation has the advantage of greatly speeding up the computation time and hence shortening the design process. On larger FPGA devices the

process can achieve a greater speed by accommodating more computational structures in parallel. These structures would evaluate multiple pairs of segments in parallel and accumulate them to the final value.

The development of an optimized norm operator (to be included in FPLibrary) will provide a space efficient alternative to the combination of multipliers and adders we currently use in the implementation and would probably enhance the latency as well.

The next step of this research will consist of a study on 50 cases of coils with large numbers of turns (more than 70 turns, in the different configurations: Slinky_2, Slinky_3, Slinky_4 and Slinky_5). The powerful FPGA-based computational tool described in this paper allows us to compute both the coil's inductivity and the magnetic field's value on a given point in a short time (a few minutes, which is much less than the time that a software run would require - remember that for a 58-turns coil, the necessary time is 3 hours, and the runtime grows quadratically).

References

[1] MOZEG D., FLAK E., *An Introduction to Transcranial Magnetic Stimulation and Its Use in the Investigation and Treatment of Depression*, University of Toronto Medical Journal, vol. **76**, no. 3, 1999, pp. 158–162.

[2] GRIŠKOVAL I., HÖPPNER J., *Transcranial magnetic stimulation: the method and application*, Medicina (Kaunas), 2006, **42**(10), pp. 792–804.

[3] HAN K. S., *Self-inductance of air-core circular coils with rectangular cross section*, IEEE Transactions on Magnetics, vol. **23**, no. 6, November 1987, pp. 3916–3921.

[4] CREŢ L., CIUPA R., *Remarks on the Optimal Design of Coils for Magnetic Stimulation*, *ISEM Proceedings*, Bad Gastein, Austria, 2005, pp. 352–354.

[5] KALANTAROV P., TEITLIN L., *Calculul inductivităţilor*, Ed. Tehnică, Bucureşti, 1958.

[6] GUELL D., EL-GHAZAWI T., GAJ K., KINDRATENKO V., *High-Performance Reconfigurable Computing*, IEEE Computer, vol. **40**, no. 3, March 2007, pp. 23–27.

[7] `http://www.ens-lyon.fr/LIP/Arenaire/Ware/FPLibrary/`

[8] COLLANGE S., DETREY J., DE DINECHIN F., *Floating Point or LNS: Choosing the Right Arithmetic on an Application Basis*, Proceedings of the 9^{th} EUROMICRO Conference on Digital System Design, Dubrovnik, Croatia, August 2006, pp. 197–203.

[9] DETREY J., DE DINECHIN F., *A Parameterizable Floating-Point Logarithm Operator for FPGAs*, Proceedings of the 39^{th} Asilomar Conference on Signals, Systems & Computers, November 2005, pp. 1186–1190.

[10] CREŢ L., PLEŞA M., MICU D. D., CIUPA R., *Magnetic Coils Design for Focal Stimulation of the Nervous System*, Proceedings of EUROCON 2007, 9–12 September 2007, Warsaw, Poland, pp. 1998–2003.