# Approximate Computing Techniques for FIR Filters Implementation

Tiago Viegas
tiago.viegas@tecnico.ulisboa.pt
Instituto Superior Técnico
Universidade de Lisboa

***Abstract*—Approximate computing is a computation technique which produces non exact results, but that are sufficiently approximated for the purpose they were intended. Applications in the areas of audio or video allow good enough approximations to be used, since for the human being can not distinguish them from the exact result or it can extract useful information from the resulting noisy auditory or visual stimuli. Approximate computing has as its main objective to trade the accuracy of the output for savings in resources, such as energy, computation time or circuit area.**

**In this article some existing techniques of approximate computing are presented, which result of alterations of the system at three levels: physical level, logic level and architectural level. For the project of FIR filters with shift-adds architectures two different approximate computing methods and algorithms are proposed and developed, that aim to reduce the FIR filters implementation (circuit area) and operation (power) costs. In the first method the cost reduction is obtained by removing adders from a base architecture and substituting them by other existing partial sums. With this method gains in area and power up to 33% and 22.6%, respectively, were obtained with a signal-to-noise ratio of 30 dB. In the second method resources are saved by utilizing approximate adders, which introduce errors in $k$ bits. In this method reductions in area up to 56.7% and in power up to 45.6% were obtained, for a signal-to-noise ration of 60 dB.**

***Index Terms*—approximate computing, finite-impulse response (FIR) filters, energy efficiency, approximate adders.**

## I. Introduction

THE increasing density of transistors by area unit at each new CMOS technology node leads to an increasing power consumption of the circuits, and that changed the digital circuit design main goal from performance to energy efficiency [1]. In a wide variety of applications, e.g. multimedia or image and audio processing, the result may contain some errors without affecting too much its quality. For audio and video applications, this happens as the human being is tolerable to errors simply because it can extract useful information from noisy stimuli, therefore in such applications it can be good enough an approximate value rather than an completely exact one.

The resilience to error of some applications can be explored by using some techniques that are referred as approximate computing. Approximate computing techniques that the accuracy of the result of a computing system by saving some resources, such as energy, implementation area and/or delay [2]. These techniques can be applied at different abstraction levels, such as the physical level, e.g. reducing the operating voltage of a circuit, the logic level, e.g. modifying the logic functions of an adder, and the architectural level, e.g. removing components and wires along a path in a circuit.

Not all applications can make use of approximate computing techniques, e.g. in a RISC processor the arithmetic blocks only consume 6% of the total energy consumed by the processor [3], which means that using approximating computing in such a scenario will not yield significant savings in energy. Moreover, approximate computing may not be adequate to general or programmable processors due to the fact that they are design to execute generic applications which may not be tolerable to errors [4]. However, there are some applications which use intensively arithmetic operations and can be tolerant to errors, i.e. it can be enough that the results of these applications are an approximation of the real value. Some functions, implemented by DSP blocks, such as digital filtering, by FIR or IIR filters, or FFT, fall into this category of applications. These functions use extensively multiplication and sum operations, so by using approximate computing techniques in these basic operations it can be possible to implement more energy-efficient DSP blocks, knowing that their output can have some error. It is important to refer that since approximate computing techniques introduce some kind of error, it is necessary to quantify this error, as much as possible, and the resulting circuits should be evaluated in the context of the application that they are design to execute.

In this paper two approximate computing methods for the implementation in hardware of FIR filters with shift-adds architectures are presented. One method works at the architectural level an the other at the logic level, and both have the main goal to reduce the FIR filter's implementation and operations costs. The approximate computing method at the architectural level reduces the number of adders needed to implement a FIR filter by removing some adders from an already existing architecture, redoing after the necessary connections in order to obtain the best approximations of the values computed by the removed adders. The approximate computing method at the logic level reduces the area needed to implement a FIR filter by replacing some adders, also in an already existing architecture, by simple adders that perform an approximated computation, while maintaining a Signal to Noise Ratio (SNR) close to a desired value.

The remainder of this paper is organized as follows: Section II introduces approximate computing and presents some existing techniques of this area of investigation, divided in three

different categories of approximation: physical level, logic level, and architectural level. Section III describes the different stages of implementing FIR filters, from their specifications to their implementation in hardware. In this section, architectures based on shifts, additions and subtractions are addressed in particular. Section IV addresses the approximate computing method at the architectural level, explaining its implementation and presenting the experimental results for some FIR filters. Section V introduces the approximate computing method at the logic level. This method is based on a modified adder/subtracter which errors are characterized. An algorithm to use such approximate operators was developed for FIR filters implementation and its result are evaluated on a set of filters. Finally, Section VI concludes this paper.

## II. Approximate computing

Approximate computing refers to an area of investigation which includes an wide range of activities, which have the common goal of finding solutions that allow computing systems to trade resources, e.g. implementation area or energy, for the quality of the computed result [2].

Since the appearing of this area of investigation various techniques in different levels of abstraction were proposed [2], [5]. Methods at the physical level of abstraction include methods that change the conditions of operation of a circuit, such as the operating voltage, in order to reduce its power consumption. Methods at the logic level of abstraction includes techniques that alter a computing block, e.g. an adder or a multiplier, logic functions in order to reduce the circuit complexity, and therefore the area required to implement them, or the energy consumption. The change of logic functions can be performed at gate-level or at transistor-level. Modifications of a circuit at a transistor-level, such as removing transistors, changes the logic function performed by the circuit, and for this reason these techniques are included in the logic level. Methods at the architectural level refers to methods that changes an architecture of a system, usually by removing components and wires, in order to save resources.

### A. Methods at the physical level

The first techniques of approximate computing that appeared used the method of *voltage overscalling* (VOS) [5]. Using the VOS method the computing elements are operated at a lower voltage than the minimum voltage that ensures a correct operation.

Lowering the operating voltage leads to a decrease in energy consumption, since in CMOS circuits the energy consumed is proportional to the operating voltage. The operating voltage that ensures a correct operation is determined by the operating frequency, which in turn is determined by the total delay of the circuit. If the operating frequency remains the same but the operating voltage is lowered, the result generated by this circuit may be incorrect, due to the fact that some bits may not have time to propagate to all components of the circuit within a clock cycle, e.g. for a ripple carry adder (RCA) a carry may not propagate to all necessary full adders (FA), leading to a inexact result. There exists different VOS methods. Methods that lower equally the operating voltage in the entire circuit are called uniform VOS. Other methods called non-uniform VOS operate more important components, i.e. the ones that compute the most significant bits of the result, at higher voltages than the components that generate the less significant bits, in order to decrease the magnitude of the error introduced. Non-uniform VOS methods are limited to the number of distinct voltage levels available [6].

### B. Methods at the logic level

Changing logic functions of a circuit can decrease its complexity, leading to a simpler circuit, or decrease its power consumption by lowering the switching activity. However, this may resulte in some combinations of the inputs to generate a incorrect result, but if the circuit complexity or power consumption is decreased significantly, these circuits may be of interest to applications resilient to errors. Next we will give some examples of this kind of methods for a multiplier and several adders.

*1) Approximate multiplier:* If we analyse the logic table of a 2x2 bits multiplier, we notice that the only result that needs four bits to be represented is the one corresponding to $11_2 \times 11_2$, which result is $1001_2$. If we change this result to a three bit one, being $111_2$ the closest one to the correct result, we are allowing a error, of magnitude 2 ($|7-9|$), to occur, however the circuit that implements this modified multiplier is much simpler (having only 4 AND gates and 1 OR gate) than the conventional one (that haves 5 AND gates and 2 XOR gates).

In order to build larger multiplier blocks, the modified 2x2 bits multiplier can be used to calculate partial products and then use conventional adders to sum these partial products. It was shown that using these modified multiplier blocks, a dynamic power reduction of 45.4% can be achieved, comparing to the conventional multiplier blocks, and using these approximate multipliers in a FIR filter a energy consumption reduction of 18.3% can be obtained [7].

*2) AMA (Approximate Mirror Adder):* The AMA adders result of 5 different modifications to the implementation of a FA, called the mirror adder, composed by 24 transistors. Removing transistors from the mirror adder, and consequently changing the logic functions, in such a way that short-circuits and open-circuits do not occur, five different approximations for a FA were obtained. Four of the approximate FA derived have 8, 10 and 13 (two approximations have the same number of transistors) fewer transistors than the conventional mirror adder and one of the approximations uses only buffers. These approximations present errors in the sum and carry-out bits truth tables. The approximation that only uses buffers has the most errors (4 in the sum bit truth table and 2 in the carry-out truth table).

Using the approximation with only buffers, in the FAs that generate the nine less significant bits of the result, for an image compression application resulted in power savings of $\approx 60\%$ with a PSNR (peak signal-to-noise ratio) of 25.46 dB, while using conventional adders resulted in a PSNR of 31.16 dB [4].

*3) AXA (Approximate XOR/XNOR-based Adder):* Another example of FA approximations obtained by removing transistors of conventional FA implementations are the AXA adders. These are derived of a XOR- and a XNOR-based implementations of a FA that are both realized with 10 transistors. Three different AXA adders exist, AXA1, AXA2 and AXA3, having 8, 6 and 8 transistors, respectively. AXA1 is the one with most errors in sum and carry-out truth tables (4 in the sum bit and 4 in the carry-out bit), AXA2 only has 4 errors in the sum bit truth table and AXA2 only has 2 errors in the sum bit truth table.

When compared to the XNOR-based implementation, AXA1 has 76.09% less delay, AXA2 has 65.45% less static power consumption and AXA3 has 30.57% less dynamic power consumption [8].

*4) LOA (Lower-part-OR Adder):* Removing transistors is not the only way of obtaining approximate computing blocks. LOA adders are obtain by modifying the lower part, i.e. the part that computes the less significant bits of the result, of a conventional adder. A $p$-bit LOA adder has two different parts, an exact part (composed by a n-bit conventional adder) and an approximate part (composed by one AND gate and $m$ OR gates), where $p = m + n$. Each of the $m$ less significant bits of the result are generated by an OR gate and the carry-in bit of the exact part is generated by an AND gate connected to the $m - 1$ bits of the operands.

Using LOA adders, along other approximate multiplier blocks, for a neural network used in facial recognition showed a 54% improvement in area, while maintaining equal behaviour to the neural network implemented with precise computing blocks, only differing in the number of training epochs [9].

*5) ETA (Error-Tolerant Adder):* The area needed to implement a computing block may not be the only resource that approximate computing blocks tend to save. Sometimes is desired that energy consumption is reduced, even if it implies an increase in the area needed to implement such computing blocks. ETA adders allows a decrease in energy consumption by reducing the carry propagation within the adder. To do that, a different addition arithmetic is introduced, which is divided in two different parts, an exact part and an inexact one. The exact part is performed in the conventional manner, while the inexact part is an entirely different process. Addition in the inexact part is done from the most significant bit (within this part) to the less significant one. The addition in the inexact part is performed as usual until both bits of the operands are "1", when this happens the remaining bits (in the less significant bit direction) are set to "1".

An 32-bit ETA adder that performs this addition arithmetic, which 20 bits are approximated, has a power-delay product (PDP) 66.29% better than a conventional 32-bit RCA, with only a 12.3 % increase in the number of transistors [10].

Other adders, ETAII, ETAIIM and ETAIV, were also proposed, which divide a conventional adder (RCA or carry-select adder) in several sub-adders, in order to improve the accuracy and the power consumption. ETAIV revealed the best accuracy and its PDP is 13.33% better than the ETA, however it needs 43% more transistors when compared to ETA [11], [12].

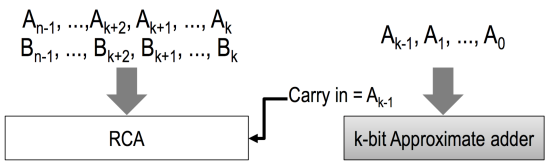


Fig. 1. $n$-bit copy of operand adder with $k$ approximated bits [1].

*6) Copy of operand adder:* Energy consumption of a circuit is often associated with the area that a circuit needs to be implemented, since normally the more components a circuit has the more energy it consumes. So one way of reducing energy consumption is to simply reduce the area needed for the implementation of a circuit. The copy of operand adder proposes just that, and it reduces the area needed to implement an RCA by completely removing FAs in the least significant bits. A $n$-bit copy of operand adder is composed by a $(n-k)$-bit RCA and a $k$-bit approximate adder, which simply copies the $k$ least significant bits of one of the operands to the $k$ least significant bits of the result. The carry-in bit of the $(n-k)$-bit RCA corresponds to the $k-1$ bit of the operand that has its bits copied, as shown in the Fig. 1.

FIR filters implemented with this adder showed an area reduction up to 18.8% and energy reduction up to 15.5%, with signal-to-noise ratios (SNR) higher than 60 dB [1].

### C. Methods at the architectural level

Another approach of approximate computing is to work at a higher abstraction level, such as the architectural level. In this level the entire architecture of a system can be modified in order to achieve reductions of resources. One example of an approximate method that works by changing a computing system architecture is one that removes components and wires along the path that has the lowest probability of being active during the circuit operation [13]. This method uses simulations in order to compute the paths probabilities of being active and iteratively removes components and wires along the path that as the lowest probability until a desired error limit, imposed by the application that the circuit was design to execute, is achieved. The error measurement is different for each application, so this method is highly tied to the kind of application it is applied to.

Another method for realizing architectures for approximate computing, which was design for FIR filters is the algorithm NAIAD [14]. This algorithm finds a set of coefficients for the filter which yields architectures with a reduced number of adders by allowing a small change in the filter specifications, more specifically in the pass-band and stop-band ripples.

## III. FIR FILTERS

Finite-impulse response (FIR) filters are usually used in digital signal processing applications because they have desired features, such as stability and linear phase [15]. The design process of FIR filter implemented in hardware can be divided in two different steps: i) coefficients are found given the filter specifications; ii) the hardware architecture is chosen and optimized.

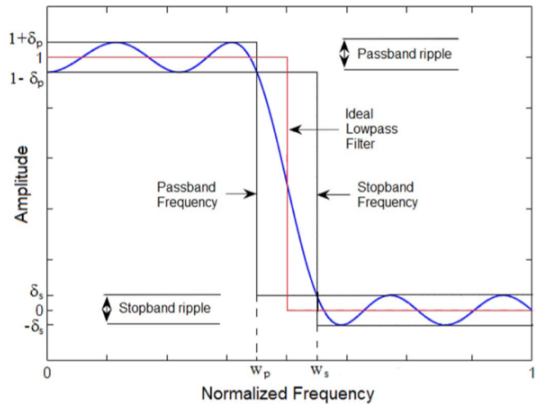


Fig. 2. Frequency response magnitude of a low-pass filter [14].

The output of a $N$-tap FIR filter can be calculated using the following expression:

$$y[n] = \sum_{i=0}^{N-1} h_i \cdot x[n-i], \tag{1}$$

where $N$ is the filter order, $h_i$ is the $i$th filter coefficient and $x[n-i]$ is the filter input delayed by $i$ samples. The $h_i$ coefficients, with $i = 0, ..., N-1$, are determined according to the filter frequency response specifications. For a low-pass FIR filter the parameters for the frequency response are, in general, the following:

- $\omega_p$ - pass-band frequency.
- $\omega_s$ - stop-band frequency.
- $\delta_p$ - pass-band ripple.
- $\delta_s$ - stop-band ripple.

For different kind of filters, such as high-pass, band-reject or band-pass the parameters are different, however there's little difference on how the coefficients are found for these filters. In order to undestand how these parameters change the filter's frequency response specifications, the magnitude of the frequency response of a low-pass filter is shown in Fig. 2.

Given the filter specifications various methods, such as windowing or Parks-McClellan [16], can be used to compute the filter's coefficients.

Having the filter's coefficients, the next step for the FIR implementation is choosing an architecture to use. The most common architectures used are the direct form, obtained by the straightforward implementation of (1), and the transposed form, depicted in Fig. 3 and Fig. 4, respectively. These two architectures need the same number of multipliers, adders and registers to be implemented, however they differ in the size of the registers, critical path and input capacitance [15].

Knowing that the filter's coefficients are pre-determined and realizing multipliers in hardware leads to an increase in area, delay and energy consumption, more efficient architectures for implementing a FIR filter in hardware are used [17]. These architectures use only adders and shifts to realize the necessary multiplications and use several techniques exists to minimize the number of adders needed to implement the filter. In order to comprehend how a multiplication by a constant can be

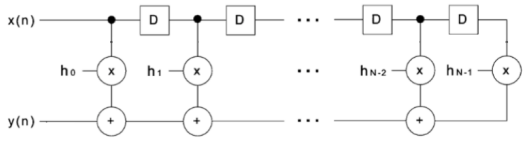


Fig. 3. Direct form of a $N$-tap FIR filter [14].

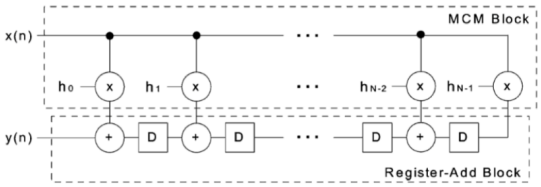


Fig. 4. Transposed form of a $N$-tap FIR filter [14].

realized only by sums and shifts let's take the example of the multiplication $y = 11x$. If we write the constant 11 in a binary representation we get 1011, so the multiplication can be written in the form $y = (1011)x = x \ll 3 + x \ll 1 + x$, which corresponds to two additions and two left-shifts.

Analysing the transposed form of a FIR filter, depicted in Fig. 4, we notice that the input is multiplied, in parallel, by different constants. The block that performs these multiplications is called a MCM (Multiple Constants Multiplication) block. Since these multiplications are performed in parallel and can be realized using only additions and shifts, the partial sums can be shared among the multiplications, thus reducing the number of adders needed to implement this block.

Several different methods that try to minimize the number of adders needed to implement a MCM block have been proposed and can be divided in two categories: CSE (Common Subexpression Elimination) and GB (Graph based) methods [15]. The CSE methods represent the constants under a number representation, such as binary or CSD (Canonical Signed Digit), and then the most common subexpressions are shared between the constant multiplications. The GB methods use graphs to find the intermediate subexpressions that can be shared, to yield the minimum number of operations. GB methods generally find better solutions, however they require more computational resources. An example of solutions given by an CSE and a GB method, for the multiplication by the constants 51 and 77, where only 4 and 3 operations are needed, respectively, is shown in Fig. 5. If the multiplication by 51 and 77 were done in separate 6 operations would be needed.

FIR filters with the architectures that only use additions, subtractions and shifts, known as shift-adds architectures, will be the target for the approximate computing methods proposed in the following sections.

## IV. FIR filters - Approximate Computing at the architectural level

In this section an approximate computing method, at the architectural level, for the implementation in hardware of FIR filters, with shift-adds architectures, is proposed. The

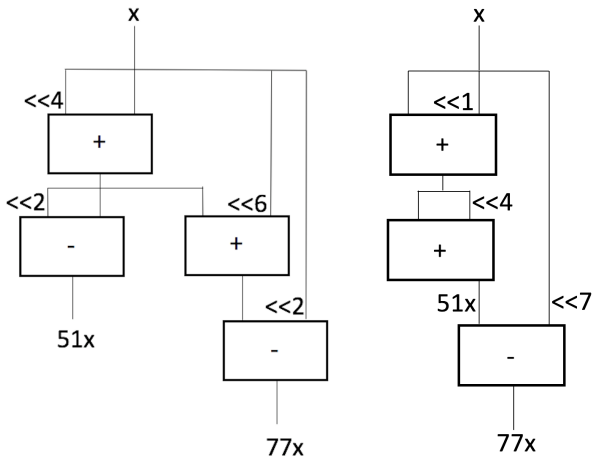


Fig. 5. Solution given for the multiplication by the constants 51 and 77 by a: (a) CSE method, (b) GB method

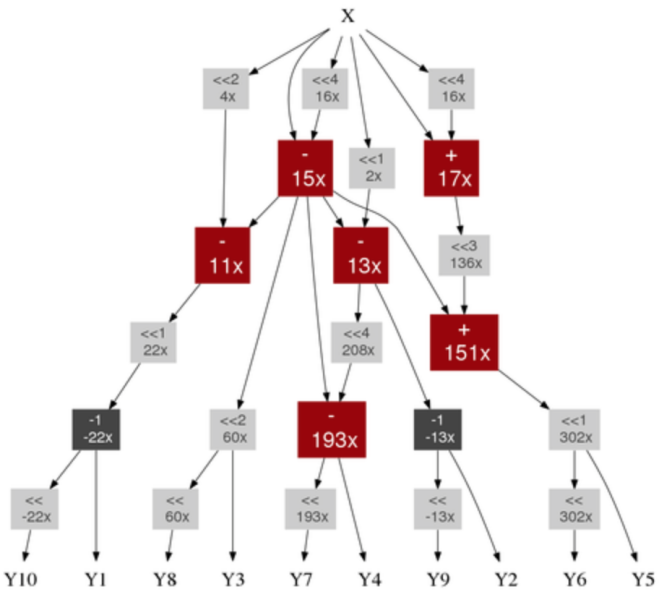


Fig. 6. Shift-adds architecture of the MCM block of the example FIR filter.

main goal of this method is to reduce the area needed for the implementation of a FIR filter, by removing adders and rewire the necessary connections within an existing MCM block, while trying to minimize the overall error at the MCM block outputs. An exhaustive and a heuristic algorithm that implements this method is presented. The results of this method for 10 FIR low-pass filters are also presented within this section.

### A. Proposed method

In order to reduce the area needed to implement the MCM block, with a shift-adds architecture, adders within this block can be removed. When we remove an adder the node where its result was connected is left unconnected, so this node must be rewire to somewhere else. One way of reconnecting this node is to connect it to one of the remaining adders or to the input of the MCM block, which may compute similar values, considering shifts, to the adder that was removed.

This modifications may alter the final result presented at the outputs of the MCM block, i.e. it can introduce some error when comparing to the unchanged MCM block. In order to choose the best adders to remove and the best re-routing of the unconnected nodes, i.e. the ones that lead to a smaller error in the MCM block outputs, the total error $E$ was chosen to compare the different options and it was defined as:

$$E = \sum_{i=1}^{N} |\tilde{y}_i - y_i|, \tag{2}$$

where $\tilde{y}_i$ is the $i$th output of the modified MCM block, with input equal to 1, and $y_i$ is the $i$th output of the original MCM block, also with input equal to 1.

In order to find a solution which yields a small value of $E$, given a number of adders to remove, an exhaustive and a heuristic algorithm were developed and will be presented in the following subsections. Both algorithms start with an existing MCM block architecture which was obtained by computing the filter coefficients with the Parks-McClellan method, then quantifying those coefficients with a chosen number of bits and finally applying a GB method (Hcub [18]) to obtain the shift-adds architecture.

### B. Exhaustive algorithm

The exhaustive algorithm searches for the solution which yields the smallest $E$ value, given a number of adders to remove, $N_{sub}$. It tries every possible combination with $N_{sub}$ adders and keeps the one with the smaller $E$ value.

For every combination of adders tested the following steps are performed: i) the adder, belonging to the combination tested, with the smallest depth is removed; ii) the node left unconnected is connected to the adder, with a depth smaller to the adder removed, or to the input which has the closest value, considering shifts to the left, to the one computed by the removed adder; iii) the values computed by the remaining adders are updated; iv) steps i), ii) and iii) are repeated for every adder in the combination to test; v) $E$ value is calculated.

In order to comprehend better how this algorithm works let's take an example for a MCM block generated for a 10-tap filter with $\omega_s = 0.1\pi$, $\omega_p = 0.5\pi$ and coefficients, quantified with 10 bits, $-22, -13, 60, 193, 302, 302, 193, 60, -13, -22$. The shift-adds architecture of this MCM block is shown in Figure 6, where the free in hardware shift operations are shown in light grey and adders/subtractors in dark red, and the value computed by each operation is indicated.

Suppose we want to remove 2 adders from the MCM block and the combination being tested is removing the adder that computes $15x$ and the adder that computes $193x$. The algorithm starts by removing the $15x$ adder since it's the one with the smaller depth, then it checks what's the best way to reconnect the node left unconnected. Since the $15x$ adder has depth 1 it can only use the input to connect it, and in this case it chooses a left shift by 4 bits which results in $16x$, which is the closest value of $15x$ that is available. Next the values computed by the remaining adders are updated due to

the previous modification. These values are updated to the following (starting from the top left adder and excluding the removed $15x$ adder): $17x$, $12x$, $14x$, $152x$ and $208x$. Finally the same process is applied to the $193x$ adder (now $208x$), where the node which is left unconnected after the removal of the $193x$ adder is connected to the $12x$ adder (former $11x$) and shifted to the left by 4 bits, resulting in a computed value of $192x$. Completed the removal of the two adders the $E$ value is computed, which in this case is 20.

The same process is performed to all other combinations and it is kept the one which yields the lowest value of $E$.

### C. Heuristic algorithm

When the number of adders in the starting MCM block architecture increases, the number of combinations that need to be tested can be too much for the exhaustive algorithm to give a solution in a reasonable time. To address this issue a heuristic algorithm was developed, and it speeds up the search for a solution by reducing the number of combinations to test.

By running the exhaustive algorithm for MCM blocks with a few adders it was noticed that increasing the number of adders to remove, $N_{sub}$, by one, the solution found generally included all the adders removed in the previous solution for the same MCM block, i.e. with a $N_{sub}$ value smaller by one. So the heuristic algorithm, in order to decrease the number of combinations to test, starts with $N_{sub} = 1$, finds a solution, exactly in the same way as the exhaustive algorithm, and keeps this solution. Next, it increases $N_{sub}$ by one and finds a solution, but now only searching combinations that includes the adder removed for $N_{sub} = 1$, and saves this new solution. The same process is applied, searching only combinations that includes all adders removed in the previous solution, until $N_{sub}$ reaches the desired value, i.e. the number of adders to remove.

### D. Results

The method proposed was tested, using the heuristic algorithm, in 10 different FIR low-pass filters. The specifications, as well the synthesis results, of the filters tested are presented in Table I, where $\omega_p$ and $\omega_s$ are the pass-band and stop-band frequencies, $N$ the filter order, # bits the number of bits used to represent each coefficient, $\delta_p$ and $\delta_s$ the pass-band and stop-band ripples, $MA$ the number of adders in the MCM block and $TA$ the total adders in the FIR filter architecture.

These filters where described in VHDL, using the two's complement representation, defining the input with the same number of bits used to represent the coefficients and the output with twice the bits of the input. Next, these filters were synthesised using the synthesis tool Synopys Design Vision ™ version C-2009.06-SP3 and the generic cell library Faraday ™ UMC L180. The area was obtained by the synthesis tool and the dynamic power was calculated, also by the synthesis tool, using a static probability of 50% and a toggle-rate of 50% for all inputs (except for the clock and clear inputs).

The next step was to obtain the approximate filters by choosing values for $N_{sub} = 1, .., MA - 1$ and applying the heuristic algorithm for each filter. The frequency response

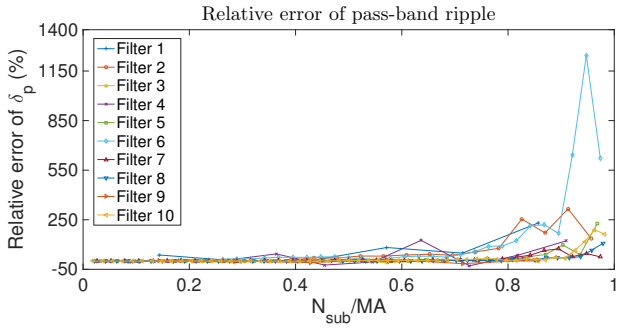


Fig. 7. Relative error of the pass-band ripple.

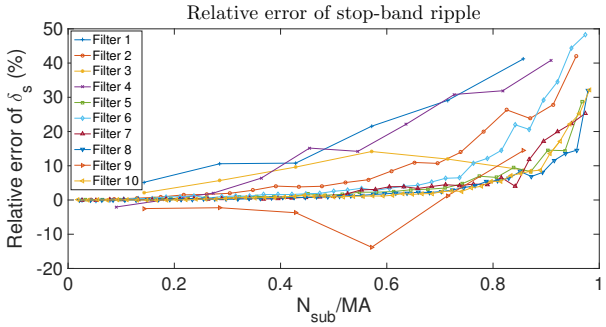


Fig. 8. Relative error of the stop-band ripple.

of each filter was obtained and the pass-band and stop-band ripples were compared with the ripples in the respective exact filter. Figures 7 and 8 show the results for the relative errors of the pass-band and stop-band ripples. In a general way the relative error of $\delta_p$ increases as the percentage of adders removed in the MCM block increases. Removing less than 20% of adders of the MCM block, introduces a relative error of $\delta_p$ smaller than 7% for all filters (except for filter 1 which has a 36% error with one adder removed). When the percentage of adders removed is lower than 50% the relative error of $\delta_p$ is kept lower than 50% for all filters. The filters which have the highest relative error of $\delta_p$ are the ones, for these set of filters, with lower value of $\delta_p$ in the exact filter, as expected. The maximum relative error for $\delta_s$ is lower than 50% for all filters. For less then 70% removed adders the relative error of $\delta_s$ it is lower than 6.5% for filters 5, 6, 7, 8 and 10, which are the filters with more adders in the exact MCM block. The filters that have the largest relative error for $\delta_s$ for a percentage of adders removed lower than 70% are the ones with less adders in the exact MCM block.

Comparing the results gave by this method for filters 1,4 and 9 with the algorithm NAIAD [14], is shown that this method yields filters with a lower number of total adders, as seen in Table II. However NAIAD can generate filters with significantly lower ripples (except for $\delta_p$ in filter 4).

Another way to evaluate the quality of the approximate filters obtained is to use the signal-to-noise ratio (SNR) metric. The SNR was measured by generating a white Gaussian noise signal with 2000 samples and filtering this signal by the exact and the approximate filters. The noise then corresponds to the difference of the filtered signal using the approximate filter

TABLE I
SPECIFICATIONS AND SYNTHESIS RESULT OF THE TESTED FILTERS.

| FIR filter | $\omega_p/\pi$ | $\omega_s/\pi$ | $N$ | # bits | $\delta_p$ (dB) | $\delta_s$(dB) | $MA$ | $TA$ | Area ($\mu m^2$) | Dynamic power ($mW$) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.300 | 0.50 | 30 | 9 | 0.0923 | -44.13 | 7 | 28 | 21436 | 35.60 |
| 2 | 0.150 | 0.25 | 40 | 16 | 0.1027 | -38.66 | 23 | 62 | 70055 | 102.04 |
| 3 | 0.100 | 0.15 | 50 | 8 | 0.7710 | -22.73 | 7 | 52 | 33262 | 56.28 |
| 4 | 0.042 | 0.14 | 60 | 10 | 0.0758 | -42.53 | 11 | 66 | 52021 | 88.47 |
| 5 | 0.150 | 0.20 | 60 | 16 | 0.2343 | -31.51 | 31 | 90 | 101008 | 146.98 |
| 6 | 0.120 | 0.18 | 80 | 16 | 0.0476 | -45.11 | 38 | 117 | 135096 | 204.97 |
| 7 | 0.120 | 0.15 | 80 | 16 | 0.3751 | -27.45 | 38 | 117 | 137718 | 203.01 |
| 8 | 0.180 | 0.20 | 100 | 16 | 0.5219 | -24.67 | 47 | 146 | 168647 | 257.97 |
| 9 | 0.200 | 0.24 | 105 | 8 | 0.9157 | -21.70 | 7 | 69 | 56225 | 109.98 |
| 10 | 0.200 | 0.22 | 120 | 16 | 0.3463 | -28.13 | 54 | 173 | 201525 | 310.90 |

TABLE II
COMPARISION WITH NAIAD [14]

| | NAIAD | | | Aprox. Method | | |
|---|---|---|---|---|---|---|
| Filter | $TA$ | $\delta_p$(dB) | $\delta_s$(dB) | $TA$ | $\delta_p$(dB) | $\delta_s$(dB) |
| 1($N_{sub}=3$) | 30 | 0.0274 | -50 | 25 | 0.9530 | -39.37 |
| 4($N_{sub}=2$) | 72 | 0.1036 | -60 | 64 | 0.0575 | -42.52 |
| 9($N_{sub}=5$) | 109 | 0.0864 | -40 | 64 | 0.7739 | -21.43 |

and the filtered signal using the exact filter. This measure was taken 10 times for each approximate filter and an average SNR value was obtained. Next the approximate filters with an average SNR value close to 90, 80, 70, 60, 50, 40 and 30 dB were described in VHDL and synthesised in the same way as the exact filters. The area and dynamic power were obtained for each synthesised approximate filter and compared to the respective exact filter, which yields the results presented in Table III.

The higher savings in area and power occurs for filters with $MA > 11$, for instance at a 40 dB SNR level these filters have an average reduction in area and power of 21.8% and 12.6%, respectively, while other filters only have an average reduction of 3.4% and 1.7% in area and power, respectively. For SNR higher than 65 dB area and power savings are lower than 5% and 3%, respectively.

With this method significant area and power reductions can be achieved at the expense of the filter's quality, however because it is a method at the architectural level which removes adders, and subsequently rewires the unconnected nodes, the filter's quality degradation is significant for a small reduction in area and power. So in the next section an approximate computing method at the logic level, is presented in order to reduce the error introduced at each partial sum.

## V. FIR FILTERS - APPROXIMATE COMPUTING AT THE LOGIC LEVEL

In the previous section, a method that removes entire adders from the MCM block, in order to reduce the area needed for the FIR filter implementation, was introduced. In this section a method that has the same goal, i.e. reduce the area needed for FIR filter to be implemented in hardware, but follows an entire different approach from the previous method is presented.

TABLE III
SYNTHESIS RESULTS OF APPROXIMATE FILTERS.

| $N_{sub}$ | SNR (dB) | Area Red. (%) | Power Red. (%) |
|---|---|---|---|
| Filter 1 | | | |
| 1 | 46.9 | 2.16 | 1.52 |
| 3 | 41.0 | 8.20 | 4.23 |
| 6 | 28.0 | 15.54 | 6.42 |
| Filter 2 | | | |
| 1 | 76.8 | 1.73 | 0.98 |
| 3 | 67.9 | 4.95 | 2.72 |
| 6 | 60.1 | 8.71 | 4.70 |
| 9 | 52.1 | 12.66 | 7.24 |
| 17 | 40.1 | 24.58 | 12.97 |
| 20 | 29.8 | 34.96 | 22.59 |
| Filter 3 | | | |
| 1 | 35.7 | 1.09 | 0.16 |
| 3 | 29.8 | 1.34 | 1.48 |
| Filter 4 | | | |
| 1 | 46.2 | 0.80 | 1.19 |
| 3 | 40.2 | 3.29 | 2.29 |
| 6 | 31.2 | 3.31 | 3.31 |
| Filter 5 | | | |
| 1 | 79.4 | 0.73 | 0.66 |
| 3 | 69.0 | 2.46 | 1.73 |
| 8 | 59.8 | 6.25 | 3.64 |
| 18 | 49.4 | 14.66 | 4.51 |
| 25 | 40.5 | 22.05 | 11.37 |
| 29 | 31.0 | 32.09 | 19.71 |
| Filter 6 | | | |
| 1 | 78.8 | 0.46 | 0.38 |
| 5 | 71.0 | 2.70 | 1.92 |
| 16 | 59.7 | 9.06 | 5.92 |
| 27 | 50.3 | 19.00 | 12.18 |
| 30 | 41.7 | 21.39 | 13.29 |
| 34 | 29.3 | 25.10 | 13.54 |
| Filter 7 | | | |
| 1 | 84.3 | 0.58 | 0.49 |
| 4 | 71.4 | 2.67 | 1.55 |
| 10 | 59.7 | 6.12 | 3.42 |
| 19 | 50.9 | 12.72 | 11.03 |
| 30 | 39.4 | 20.02 | 13.51 |
| 33 | 27.5 | 28.02 | 15.60 |
| Filter 8 | | | |
| 1 | 86.0 | 0.53 | -0.03 |
| 5 | 79.1 | 2.14 | 0.98 |
| 10 | 69.4 | 4.48 | 2.81 |
| 18 | 59.9 | 8.55 | 4.54 |
| 30 | 49.4 | 15.56 | 8.19 |
| 39 | 39.3 | 21.40 | 11.80 |
| 45 | 28.5 | 32.97 | 20.84 |
| Filter 9 | | | |
| 1 | 38.1 | 1.09 | 0.16 |
| 3 | 32.2 | 1.34 | 1.48 |
| Filter 10 | | | |
| 1 | 86.5 | 0.52 | 0.45 |
| 4 | 80.5 | 1.64 | 1.13 |
| 12 | 70.8 | 5.14 | 3.00 |
| 20 | 60.4 | 8.63 | 4.61 |
| 37 | 50.3 | 15.65 | 8.53 |
| 45 | 39.8 | 21.74 | 11.52 |
| 50 | 32.1 | 26.02 | 12.64 |

The method introduced in this section tries to replace the conventional adders, in the MCM block and in the register-add block of an existing FIR filter architecture, with approximate adders, which need a smaller area to be implemented. The approximated adder used is introduced as well the algorithm implementation of the method and results are shown for 10 different low-pass FIR filters.

### A. Approximate adder

The approximate adder used corresponds to the copy of operand adder [1]. This adder was chosen because it is based

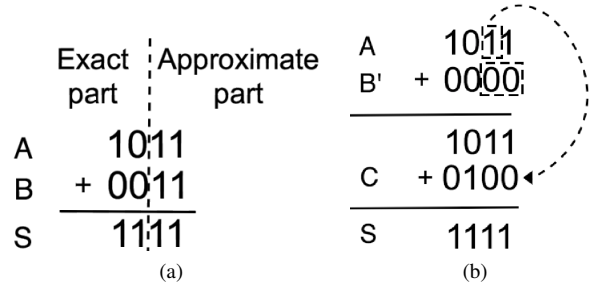


Fig. 9. Operation perform by a 4-bit copy of operand with $k = 2$.

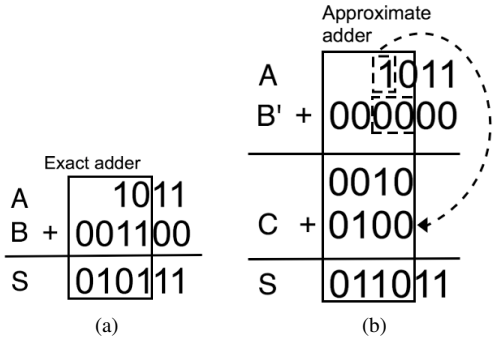


Fig. 10. Sum of 11 with 3, where the $B$ operand was left shifted by 2 bits with a: (a) Exact adder, (b) Approximate adder with $k = 2$.

on a RCA, which is one of the most area-efficient conventional adders, and the approximate part of this adder does not need any logic gates to be implemented. In Fig. 1 a $n$-bit copy of operand adder with $k$ approximated bits is shown. This adder calculates the result by copying the $k$ less significant bits (LSB) of one of the operands to the $k$ LSB of the result and then it uses a conventional RCA to compute the remaining bits of the result. The RCA has a carry-in bit that corresponds to the $k-1$ bit of the operand that has its LSB copied to the result. Let's assume from now on that the operand that has its bits copied is the operand $A$.

Mathematically, the result of the operation realized by this approximated adder can be computed in the following manner: i) the $k$ LSB of the operand $B$ are subtracted from the operand $B$, resulting in $B'$; ii) $A$ and $B'$ are summed; iii) the $k-1$ bit of the operand $A$, shifted by one bit to the left, is added to the previous result. An example of the addition of two 4-bit numbers, $A = 1011$ and $B = 0011$, done by a 4-bit copy of operand adder with $k = 2$ is shown in Fig. 9(a) and the decomposition of this operation is shown in Fig. 9(b). For this example the error made by the approximate adder is 1, because the correct result would be 14 instead of 15.

To use this approximate adder in the MCM block of a FIR filter, we need to consider that often one of the operands is a value that was previously left shifted. This causes the LSB of that operand to be always "0", which means that it is not necessary an adder with the same bit-length as the largest operand. An example of an exact and an approximate sum for the case where the $B$ operand was previously left shifted by 2 bits is depicted in Fig. 10. For this example the error made by the approximate adder is 4 $(27 - 23)$.

The approximation is now done at the bits after the left shift in order to ensure that area is saved, because in the exact sum there's no need to have an adder to compute the LSB corresponding to the shift. For simplification purposes it was defined that the operand that has its bits copied to the result would be the operand that was not left shifted.

In MCM blocks there's also the need of having subtractors, which are implemented, normally, with conventional adders by negating one of the operands and setting the carry-in bit to "1". Notice that approximate adders don't have a carry-in input, so the subtractors are realized only by negating one of the operands, assuming in this way a -1 error. For subtractors the $B$ operand is always the one to be negated and the $k$ bits that are copied to the result correspond to the $k$ LSB of the operand that was not previously left shifted.

Mathematical models of the approximate adder and approximate subtractor were derived and described as follows:

**Adder**:

$$sum = a + b + e(A, B), \tag{3}$$

$$e(A, B) = A_{k+s_B-1}2^{k+s_B} - \sum_{i=s_B}^{k+s_B-1} B_i 2^i. \tag{4}$$

**Subtractor**:

$$sub = a - b + e^-(A, B), \tag{5}$$

$$e^-(A, B) = \begin{cases} -2^{s_B} + A_{k+s_B-1}2^{k+s_B} - \sum_{i=s_B}^{k+s_B-1} \overline{B}_i 2^i, \\ \text{if } s_B \neq 0, s_A = 0; \\ -1 + \overline{B}_{k+s_A-1}2^{k+s_A} - \sum_{i=s_A}^{k+s_A-1} A_i 2^i, \\ \text{if } s_A \neq 0, s_B = 0. \end{cases} \tag{6}$$

where $a = \sum_{i=0}^{n-1} A_i 2^i$, $b = \sum_{i=0}^{n-1} B_i 2^i$, $sum$ is the result of the adder, $sub$ the result of the subtractor, $s_A$ the number of bits operand $A$ was left shifted, $s_B$ the number of bits operand $B$ was left shifted, $e(A, B)$ the error function of the adder and $e^-(A, B)$ the error function of the subtractor.

Having defined how the approximate adders will be used in the architecture of FIR filter it is necessary to find where to use the approximate adders and how many approximate bits they should have, in order to not degrade too much the filter's quality. In the next subsection is presented a search algorithm, which uses the models derived for the adder and subtractor, that was developed for finding a solution to this problem.

### *B. Search algorithm*

The approximate adder and subtractor presented in the previous subsection can be used to replace conventional adders/subtractors in an existing FIR filter architecture in order to reduce the area needed to implement it, however they also introduce some error in the filter's response.

This error needs to be quantified in order to evaluate the filter's output quality. We could try to compare the approximate filter frequency response with the exact one, however the

approximate adder/subtractor is not a linear operator, which makes the approximate filter also non-linear. Another way of measure the approximate filter's output quality is to use the SNR, as explained in Subsection IV-D. The algorithm presented in this subsection uses the SNR metric in order to evaluate the quality of a possible solution.

If we consider all the possible combinations of which adders should be approximate and how many approximate bits they should have, we can infer that an algorithm that considers all these possibilities would be computationally demanding, even for small filters. In order to have a smaller search space, adders were grouped in different groups. Adders within the same group have the same number of approximate bits. The first group includes all adders in the register-add block of the filter and other groups are created with adders of the MCM block, in which adders of the same group have the same depth in the MCM block's adder-tree.

The first step of the algorithm is to create a simulation model of the starting FIR filter architecture, using the models presented in the previous subsection, which has as parameters the number of approximate bits , $k$, of each group of adders. This will allow to compute the SNR for various approximate filters. If $k = 0$ in a group it means that the adders of that group are exact.

The second step is to find a set of $k$s (one for each group) that maximizes the area saved and keeps the SNR value close to a desired value. The set of $k$s is found by applying the following iterative process: i) all groups start with $k = 0$; ii) the value of $k$ is increased by one in one of the groups, keeping the $k$ value in the other groups; iii) the SNR value is computed; iv) repeat ii) and iii) for each group and save the set of $k$s that yield the highest SNR value; v) starting with the set of $k$s saved in iv) repeat ii), iii) and iv) until the desired SNR value is reached. The highest $k$ value possible of each group is equal to the number of bits the filter's coefficients were represented, since this corresponds to the smallest number of bits an adder can have. When a group reachs the highest $k$ possible it is ignored in the subsequent iterations.

### C. Results

The approximate computing method at the logic level was tested for the same 10 low-pass FIR filters as the method introduced in Section IV, where its specifications as well synthesis results are presented in Table 2. For each exact filter the method was applied for the desired SNR values of 80, 70, 60, 50, 40 and 30 dB. The obtained approximate filters were described in VHDL, using two's complement number representation, realizing the existing negations of the MCM block using exact adders, defining the input with the same number of bits as the coefficients and the output with twice the number of bits of the input. Then these filters were synthesized and area and dynamic power values were obtained in the same way as for the exact filters.

Table IV presents the area and dynamic reductions obtained, where it is possible to see that area reductions up to 63.15% and power reductions up to 52.22% were achieved. Filters 2,5,6,7,8 and 10, which its coefficients were represented with 16 bits, obtained the highest area and power reductions. In these filters for a SNR close to 80 dB the average area and power reductions were 29.7% and 21.2%, respectively. In filters 1,3,4 and 9 the highest SNR obtained was inferior to 70 dB, and the average area and power reductions for a SNR close to 40 dB was 23.3% and 17.2%, respectively. It is possible to notice some negative values for the power reduction, which may occurred due to the way the synthesis tool calculated the power consumption.

TABLE IV
SYNTHESIS RESULTS OF APPROXIMATE FILTERS.

| SNR (dB) | Area Red. (%) | Power Red. (%) |
|---|---|---|
| Filter 1 | | |
| 68.7 | 1.06 | -2.78 |
| 59.0 | 15.54 | 14.00 |
| 49.1 | 21.17 | 18.88 |
| 40.0 | 32.06 | 29.70 |
| 30.5 | 39.44 | 37.29 |
| Filter 2 | | |
| 79.2 | 31.32 | 21.69 |
| 68.2 | 37.82 | 29.06 |
| 59.6 | 56.65 | 44.90 |
| 49.5 | 56.67 | 43.93 |
| 39.7 | 54.98 | 43.06 |
| 29.6 | 54.94 | 42.97 |
| Filter 3 | | |
| 51.1 | 2.01 | -2.87 |
| 40.7 | 6.67 | 1.14 |
| 31.7 | 25.53 | 23.51 |
| Filter 4 | | |
| 72.8 | 1.30 | -6.42 |
| 64.8 | 4.97 | -2.64 |
| 51.9 | 19.16 | 12.25 |
| 37.3 | 39.36 | 27.50 |
| 29.6 | 46.20 | 34.92 |
| Filter 5 | | |
| 76.0 | 32.12 | 26.98 |
| 68.2 | 37.05 | 25.27 |
| 58.4 | 42.57 | 31.53 |
| 49.9 | 46.56 | 37.09 |
| 38.9 | 47.37 | 35.96 |
| 31.0 | 48.21 | 37.00 |

| SNR (dB) | Area Red. (%) | Power Red. (%) |
|---|---|---|
| Filter 6 | | |
| 77.8 | 31.20 | 22.15 |
| 66.3 | 44.18 | 33.73 |
| 58.6 | 56.35 | 45.55 |
| 48.8 | 58.38 | 48.08 |
| 37.3 | 57.32 | 46.91 |
| 29.5 | 55.20 | 45.10 |
| Filter 7 | | |
| 75.2 | 32.06 | 20.69 |
| 67.7 | 35.71 | 24.76 |
| 57.6 | 44.46 | 35.95 |
| 49.0 | 47.02 | 39.54 |
| 39.5 | 48.04 | 40.54 |
| 29.9 | 49.16 | 41.66 |
| Filter 8 | | |
| 81.8 | 27.13 | 21.17 |
| 75.3 | 31.29 | 23.17 |
| 63.2 | 43.84 | 31.19 |
| 48.4 | 63.15 | 52.22 |
| 39.8 | 60.60 | 49.67 |
| 29.5 | 57.41 | 46.78 |
| Filter 9 | | |
| 40.0 | 15.19 | 10.49 |
| 28.8 | 28.11 | 23.58 |
| Filter 10 | | |
| 84.5 | 24.51 | 14.77 |
| 74.1 | 30.70 | 20.95 |
| 62.1 | 38.10 | 30.65 |
| 47.5 | 57.64 | 47.69 |
| 39.2 | 57.77 | 47.34 |
| 29.7 | 57.56 | 47.18 |

In Table V, where are represented the number of approximate bits used in each adder group of each approximate filter synthesized, it is possible to notice that, generally, the group which has the highest number of approximate bits is the one that includes the register-add block adders. We then can conclude that the most area savings occured within this block, since this block is generally the one which contains the highest number of adders. We can also notice that the groups which contains the adders with the lowest depth (second group) in the MCM block adder-tree is generally the one with least number of approximate bits, which means that errors in these adders are amplified due to left shifts along the adder tree, and thus generating errors that affects the filter's quality too much.

The method presented in this section obtains better area and power savings than the method presented in [1], where reductions of 18.8% and 15.5% in area and power, respectively, were obtanied for 16-bit coefficients FIR filters, with a SNR of 80 dB. The increase of area and power reductions,

TABLE V
NUMBER OF APPROXIMATE BITS IN EACH ADDER GROUP.

| SNR (dB) | # approximate bits in each group |
|---|---|
| Filter 1 | |
| 70 | [1,0,0,0] |
| 60 | [4,0,0,0] |
| 50 | [5,0,0,3] |
| 40 | [7,0,0,4] |
| 30 | [8,0,2,6] |
| Filter 2 | |
| 80 | [12,0,3,2,6,8,6,8] |
| 70 | [14,0,5,4,8,10,7,11] |
| 60 | [16,2,6,6,10,11,8,12] |
| 50 | [16,4,8,8,11,13,10,14] |
| 40 | [16,6,10,9,13,15,12,15] |
| 30 | [16,7,11,11,15 ,16,14,16] |
| Filter 3 | |
| 50 | [1,0,0] |
| 40 | [2,0,0] |
| 30 | [5,0,0] |
| Filter 4 | |
| 70 | [1,0,0] |
| 60 | [2,0,0] |
| 50 | [5,0,0] |
| 40 | [7,1,0] |
| 30 | [8,2,3] |
| Filter 5 | |
| 80 | [13,0,0,0,2,11] |
| 70 | [14,1,2,0,3,12] |
| 60 | [15,3,3,2,5,14] |
| 50 | [16,4,5,4,7,15] |
| 40 | [16,6,7,6,8,16] |
| 30 | [16,8,8,7,10,16] |

| SNR (dB) | # approximate bits in each group |
|---|---|
| Filter 6 | |
| 80 | [12,0,0,5,5,6] |
| 70 | [14,1,0,6,7,7] |
| 60 | [15,3,2,8,8,9] |
| 50 | [16,4,4,9,10,11] |
| 40 | [16,7,6,11,12,13] |
| 30 | [16,8,7,13,13,14] |
| Filter 7 | |
| 80 | [12,0,2,0,4,7,4,11] |
| 70 | [13,0,4,0,6,8,6,12] |
| 60 | [15,2,5,3,7,10,7,13] |
| 50 | [16,4,7,5,9,11,9,15] |
| 40 | [16,6,9,6,11,13,11,16] |
| 30 | [16,7,10,8,12,15,13,16] |
| Filter 8 | |
| 80 | [11,0,0,0,5] |
| 70 | [12,0,0,2,7] |
| 60 | [14,1,0,3,8] |
| 50 | [16,4,4,7,12] |
| 40 | [16,6,6,8,13] |
| 30 | [16,8,7,10,15] |
| Filter 9 | |
| 40 | [3,0,0] |
| 30 | [5,0,2] |
| Filter 10 | |
| 80 | [10,0,0,2,5,9] |
| 70 | [12,0,0,4,6,11] |
| 60 | [14,0,2,5,8,13] |
| 50 | [16,4,6,8,11,16] |
| 40 | [16,6,8,10,13,16] |
| 30 | [16,7,10,12,15,16] |

while maintaining the same SNR value, is due to the fact that different approximations for the MCM block's adders are considered, while in [1] adders in this block have all the same approximation. It is also worth to mention that the method presented in this section revealed greater area and power reductions, for the same SNR value, than the method presented in Section IV, since it reduces area also in the register-add block (where the biggest savings occured) and allows a better fine-tuning of the error introduced in the filter.

## VI. CONCLUSION

Approximate computing has being presented as a viable option to reduce implementation costs in applications resilient to error, allowing some errors to occur in its outputs. However the results should be maintain qualitatively acceptable, according to the usage they are design for. In this paper two approximate computing methods for FIR filter implementation in hardware were presented, one at the architectural level and another at the logic level.

The first method reduced the area needed for the FIR filter implementation, by removing adders in the MCM block of an existing shift-adds architecture and making the necessary connections to minimize the error at its outputs. This method achieved reductions in area and power up to 33% and 22.6%, respectively, with a SNR of 30 dB. The second method reduced the area by replacing the conventional adders in an existing shift-adds FIR filter architecture by an approximate adder [1], which copies $k$ bits of one of the operands to the result and needs less area to be implemented. This method achieved better area and power savings than the first method, presenting for a SNR of 60 dB savings in area and power up to 56.7% and 45.6%, respectively. This savings result not only from the MCM block but also frim the register-adder block, in which more adders area avalilable an can be replaced without increasing to much the erros on the output. It was also shown that errors in adders with low depth in the MCM block leads to greater errors at its outputs than in highest depth adders, due to left shifts along the adder-tree which magnifies the error.

## REFERENCES


[1] L. B. Soares and S. Bampi, "Approximate adder synthesis for area- and energy-efficient FIR filters in CMOS VLSI," in *Proceedings of* $13^{th}$ *Internacional New Circuits and Systems Conference (NEWCAS)*, 2015.

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proceedings of* $18^{th}$ *IEEE European Test Symposium (ETS)*, 2013.

[3] W. J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. C. Harting, V. Parikh, J. Park, and D. Sheffield, "Efiicient embedded computing," *Computer*, vol. 41, no. 7, pp. 27–32, July 2008.

[4] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Transactions on Computer-Aided Design Of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, January 2013.

[5] K. Palem and A. Lingamneni, "Ten years of building broken chips: The physics and engineering of inexact computing," *ACM Trans. Embed. Comput. Syst*, vol. 12, no. 2s, May 2013.

[6] L. N. B. Chakrapani, K. K. Muntimadugu, A. Lingamneni, J. George, and K. V. Palem, "Highly energy and performance efficient embedded computing through approximately correct arithmetic: A mathematical foundation and preliminary experimental validation," in *Proceedings of IEEE/ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2008, pp. 187–196.

[7] P. Kulkami, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proceedings of* $24^{th}$ *Annual Conference on VLSI Design*, 2011, pp. 356–352.

[8] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, "Approximate XOR/XNOR-based adders for inexact computing," in *Proceedings of* $13^{th}$ *IEEE International Conference on Nanotechnology*, 2013.

[9] H. R. Mahdiani, A. Ahmadi, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems–I: Regular Papers*, vol. 57, no. 4, pp. 850–862, April 2010.

[10] N. Zhu, W. L. Goh, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 8, pp. 1225–1229, August 2010.

[11] N. Zhu, W. L. Goh, G. Wang, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *Proceedings of the* $12^{th}$ *International Symposium on Integrated Circuits*, 2009, pp. 69–72.

[12] ——, "Enhanced low-power high-speed adder for error-tolerant application," in *Proceedings of IEEE International SoChip Design Conference*, 2010, pp. 323–327.

[13] A. Lingamneni, C. Enz, J.-L. Nagel, K. Palem, and C. Piguet, "Energy parsimonious circuit design through probabilistic pruning," in *Proceedings of Design, Automation and Test in Europe Conference*, 2011, pp. 764–769.

[14] L. Aksoy, P. Flores, and J. Monteiro, "Exact and approximate algotithms for the filter design optimization problem," *IEEE Transactions on Signal Processings*, vol. 63, no. 1, pp. 142–154, January 2015.

[15] ——, "A tutorial on multiplierless design of fir filters: Algorithms and architectures," *Circuits, Syst., Signal Process.*, vol. 33, no. 6, pp. 1689–1719, January 2014.

[16] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete- Time Signal Processing*, 2nd ed. Prentice-Hall, Inc., 1999, ch. 7.

[17] L. Aksoy, E. Gunes, and P. Flores, "Search algorithms for the multiple constant multiplications problem: Exact and approximate," *Elsevier J. Microprocessors and Microsystems*, vol. 34, no. 5, pp. 151–162, 2010.

[18] Y. Voronenko and M. Püschel, "Multiplierless multiple constant multiplication," *ACM Transactions on Algorithms*, vol. 3, no. 2, 2007.