INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2010, VOL. 56, NO. 4, PP. 345–350
Manuscript received September 27, 2010; revised November 2010. DOI: 10.2478/v10177-010-0045-9

Modified Distributed Arithmetic Concept for Implementations Targeted at Heterogeneous FPGAs

Mariusz Rawski

Abstract—**Distributed Arithmetic (DA) plays an important role in designing digital signal processing modules for FPGA architectures. It allows replacing multiply-and-accumulate (MAC) operations with combinational blocks. The quality of implementations based on DA strongly depends on efficiency of methods that map combinational DA block into FPGA resources. Since modern FPGAs have heterogeneous structure, there is a need for quality algorithms to target these structures and the need for flexible architecture exploration aiding in appropriate mapping. The paper presents a modification of DA concept that allows for very efficient implementation in heterogeneous FPGA architectures.**

Keywords—**Distributed arithmetic, FPGA, heterogeneous programmable structures.**

I. Introduction

DISTRIBUTED ARITHMETIC (DA) is an important technique to implement digital signal processing (DSP) functions in Field Programmable Gate Arrays (FPGAs) [1]. It provides an approach for multiplier-less implementation of DSP systems, since it is an algorithm that can perform multiplication with use of lookup table (LUT) that stores the precomputed values and can be read out easily, which makes DA-based computation well suited for FPGA realization, because the LUT is the basic component of FPGA. DA specifically targets the sum of products computation that is found in many of the important DSP filtering and frequency transforming functions.

The major disadvantage of DA technique is that the size of DA-LUT increases exponentially with the length of input. Several efforts have been made to reduce the DA-LUT size for efficient realization of DA-based designs. In [2] to use offset-binary coding is proposed to reduce the DA-LUT size by a factor of 2. Recently, a new DA-LUT architecture for high-speed high-order has been introduced in [3], where the major disadvantage of the FIR filters is vanished by using carry lookahead adder and the tri-state buffer. On the other side, some structures are introduced for efficient realization of FIR filter. Recently, novel one- and two-dimensional systolic structures are designed for computation of circular convolution using DA [4], where the structures involve significantly less area-delay complexity compared with the other existing DA-based structures for circular convolution. In [5] modified DA concept is used to obtain an area-time-power-efficient implementation of FIR filter in FPGA.

This work was partly supported by the Ministry of Science and Higher Education of Poland–research grant no. N N516 418538 for 2010-2012.

M. Rawski is with Institute of Telecommunications, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland (e-mail: rawski@tele.pw.edu.pl).

DA concept proves to be a powerful technique for implementing MAC unit as a multiplierless algorithm. The efficiency of implementations based on this concept and targeted FPGAs strongly depends on implementation of DA-LUT. These blocks have to be efficiently mapped onto FPGA's logic resources. With rapidly growing traditional FPGA industry, heterogeneous logic blocks are often used in the actual FPGA architectures such as Xillinx Virtex-5 and Altera Stratix III series. How to handle this kind of heterogeneous design network to generate LUTs with different input sizes in the mapping is a very important and practical problem. The existing CAD tools are not well suited to utilize all possibilities that modern heterogeneous programmable structures offer due to the lack of appropriate synthesis methods. Typically, after the logic synthesis stage, technology-dependent mapping methods are used to map design into available resources [6], [7]. However, such an approach is inefficient due to the fact that the quality of postsynthesis mapping is highly dependent on the quality of technology independent optimization step [8]. Recently, efforts have been made to develop methods based on functional decomposition that would allow for efficient utilization of heterogeneous structure of FPGA. The method presented in [9] is designed specifically to implement FIR filters using the concept of distributed arithmetic. In [10] advanced synthesis method based on functional decomposition was proposed that utilizes embedded memory block as large LUTs.

In this paper a modified distributed arithmetic concept has been proposed that is especially targeted at modern FPGA devices with heterogeneous structure. Presented results prove that application of this method allows to utilize heterogeneous resources of programmable structures very efficiently.

II. Preliminary Information

A. Architectures of Modern FPGAs

The technological advancements in Field Programmable Gate Arrays in the past decade have opened new paths for digital systems design engineers. The FPGA maintains the advantages of custom functionality like an ASIC while avoiding the high development costs and the inability to make design modifications after production. The FPGA also adds design flexibility and adaptability with optimal device utilization while conserving both board space and system power. An FPGA structure can be described as an array of LUT-based programmable logic elements (cells) interconnected by programmable connections. Each cell can implement a simple logic function (of a limited number of inputs) defined by a designer's CAD tool. A typical programmable device has a large number (64 to over 1 000 000) of such cells, that can be used to form complex digital circuits. The ability to manipulate the logic at the gate level means that the designer can construct a custom processor to efficiently implement the desired function. The technological advancements in microelectronics in the past decade have changed this picture by introducing embedded specialized blocks into structure of FPGA chip.

Modern FPGA devices have very complex structure. Today's FPGAs are entire programmable systems on a chip (SoC) which are able to cover an extremely wide range of applications. The Altera Stratix III and Xilinx Virtex-5 families of devices, both using a 65 nm manufacture process, can be used as examples of contemporary FPGAs. The basic architecture of FPGAs has not changed dramatically since their introduction in the 1980s. Early FPGAs used a logic cell consisting of a 4-input lookup table (LUT) and register. Present devices employ larger numbers of inputs (6-input for Virtex-5 and 7-input for Stratix III) and have other associated circuitry. Another enhancement extensively used in modern FPGAs are specialized embedded blocks, serving to improve delay, power and area if utilized by the application, but waste area and power if unused. Early embedded blocks included fast carry chains, memories, phase locked loops, delay locked loops, boundary scan testing and multipliers. More recently, multipliers have been replaced by digital signal processing (DSP) blocks (which add support for logical operations, shifting, addition, multiply-add, complex multiplication etc.), allowing designers to use methodology known from DSP programming. Some architectures even contain hardware CPU cores.

Today's FPGA are composed of:

- Main logic resources in form of configurable logic elements.
- Embedded RAM blocks – dedicated memory blocks that can be used to implement in-chip data storage, FIFOs, etc.
- Dedicated DSP modules containing hardware MACs that are building blocks for digital filters and other digital signal processing algorithms.
- Dedicated hardware CPU – although including dedicated hardware CPU cores in FPGA chip can be considered now somewhat old-fashioned, some relatively new Virtex-5 FX FPGAs from Xilinx contain a hard PowerPC core.

While mapping a design into logic elements is done by logic synthesis procedures of CAD tools after technology-independent logic optimization, the mapping into coarse-grain structures such as multipliers, memories and DSP modules is much more appropriately done at the RTL synthesis level where these structures are more directly recognizable [11]. These specialized embedded blocks are available by describing them in HDL code using special HDL constructs. CAD tools identify these fragments of HDL code at high level synthesis stage and automatically instantiate library element corresponding to given specialized block. Usually, even slightly different HDL description prevents CAD from proper identification of a block.

As was already mentioned specialized embedded memory blocks (EMB) make it possible to implement data storage modules, such as shift registers or RAM blocks. In many cases, though, the designer does not need such elements in design or not all such resources are utilized. This chip area need not be wasted, however, if the unused memory arrays are used to implement logic. Configuring the arrays as ROM results in large lookup-tables that might very efficiently implement some logic circuits. The memories act as very large logic cell, where the number of inputs is equal to the number of address lines and the number of output is equal to the size of memory word. Since the size of address and memory word of single EMB can be configured in several different ways it can act as logic cell of different sizes.

The basic building block of logic in the Stratix III architecture is the adaptive logic module (ALM). Each ALM contains a look-up table (LUT)-based resources that can be divided between two combinational adaptive LUTs (ALUTs) and two registers (Fig. 1a). Combinational ALUTs may have up to eight inputs. An ALM can implement various combinations of two functions, any function of up to six inputs and certain seven-input functions. In addition to the adaptive LUT-based resources, each ALM contains two programmable registers, two dedicated full adders, a carry chain, a shared arithmetic

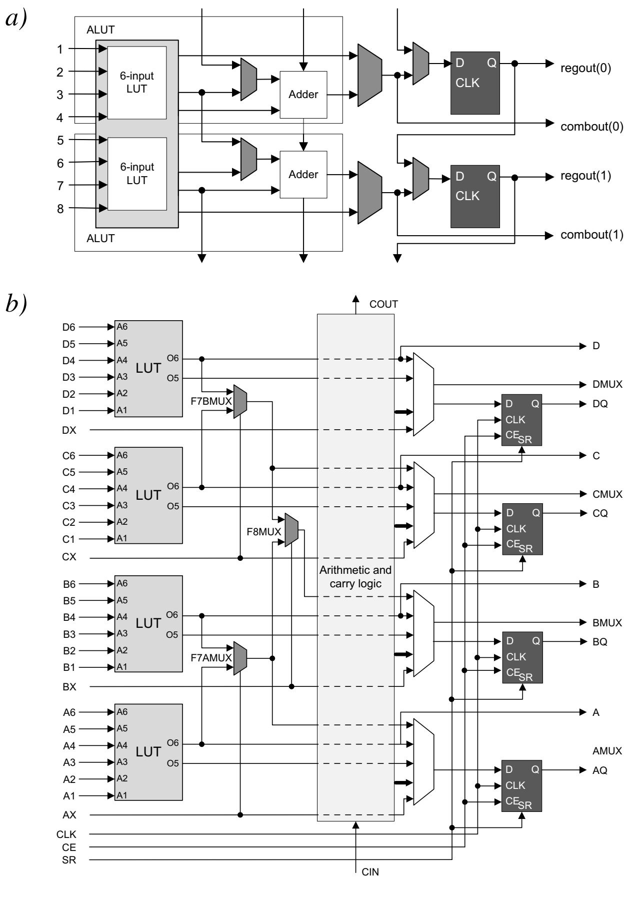

Fig. 1. Logic unit in FPGAs: a) ALM of Stratix III, b) slice of Virtex-5.

TABLE I
CONFIGURATIONS OF ALMS AND EMBEDDED MEMORY BLOCKS IN STRATIX III

	ALM	MLABs	M9K Blocks	M144K Blocks
$2 \times [4 \times 1]$	(two independent 4-input LUTs)	6×8	13×1	14×8
$[5 \times 1]\&[3 \times 1]$	(5-input LUT and a 3-input LUT)	6×9	12×2	14×9
$[5 \times 1]\&[4 \times 1]$	(5-input LUT and a 4-input LUT with one inputs shared)	6×10	11×4	13×16
$[5 \times 1]\&[5 \times 1]$	(two 5-input LUTs with two inputs shared)	5×16	10×8	13×18
$[6 \times 1]$	(6-input LUT)	5×18	10×9	12×32
$[6 \times 1]\&[6 \times 1]$	(two 6-input LUTs with four inputs shared)	5×20	9×16	12×36
			9×18	11×64
			8×32	11×72
			8×36	

TABLE II
CONFIGURATIONS OF SLICES AND EMBEDDED MEMORY BLOCKS IN VIRTEX-5

	Slice	Block RAM 18K	Block RAM 36K
$4 \times [5 \times 2]$	(four independent 5-input, 2-output LUTs)	14×1	15×1
$4 \times [6 \times 1]$	(four independent 6-input LUTs)	13×2	14×2
$2 \times [7 \times 1]$	(two independent 7-input LUTs)	12×4	13×4
$[8 \times 1]$	(8-input LUT)	11×9	12×9
		10×18	11×18
		9×36	10×36
			9×72

chain, and a register chain. This dedicated resources allow efficiently implementing various arithmetic functions and shift registers. TriMatrix embedded memory blocks provide three different sizes of embedded SRAM: 640 bit (in ROM mode only) or 320 bit memory logic array blocks (MLABs), 9 Kbit M9K blocks, and 144 Kbit M144K blocks.

Table I presents configurations of logic elements and embedded memory blocks of Stratix III as LUTs of various sizes (number of inputs $\times$ number of outputs).

The elementary programmable logic blocks in Xilinx Virtex-5 FPGAs are called slices and are organized in Configurable Logic Blocks (CLBs). The CLBs are the main logic resources for implementing sequential as well as combinatorial circuits. Each CLB element is connected to a switch matrix for access to the general routing matrix. A CLB element contains a pair of slices. Each slice has four 6-input LUTs, embedded multiplexers, carry logic, and four registers (Fig. 1b). The function generators in Virtex-5 FPGAs are implemented as six-input LUTs. There are six independent inputs (A1 to A6) and two independent outputs (O5 and O6) for each of the four function generators in a slice (A, B, C, and D). The function generators can implement any arbitrarily defined six-input Boolean function. Each function generator can also implement two arbitrarily defined five-input Boolean functions, as long as these two functions share common inputs. Signals from the function generators can exit the slice (through A, B, C, D output for O6 or AMUX, BMUX, CMUX, DMUX output for O5), enter the XOR dedicated gate from an O6 output, enter the carry-logic chain from an O5 output, enter the select line of the carry-logic multiplexer from O6 output, feed the D input of the storage element, or go to F7AMUX/F7BMUX from O6 output.

In addition to the basic LUTs, slices contain three multiplexers (F7AMUX, F7BMUX, and F8MUX). These multiplexers are used to combine up to four function generators to provide any function of seven or eight inputs in a slice. F7AMUX and F7BMUX are used to generate seven input functions from LUTs A and B, or C and D, while F8MUX is used to combine all LUTs to generate eight input functions. Functions with more than eight inputs can be implemented using multiple slices. There are no direct connections between slices to form function generators greater than eight inputs within a CLB or between slices.

Virtex-5 CLBs also support distributed memory – each look-up table can be configured to operate as a 64-bit memory. Because of the structure of the Virtex-5 LUT, each LUT can be configured as a 64x1 or 32x2 RAM. However, when the slice is configured as RAM, it can no longer perform logic functions.

The block RAM in Virtex-5 FPGAs stores up to 36K bits of data and can be configured as either two independent 18 Kb RAMs, or one 36 Kb RAM. Each 36 Kb block RAM can be configured as a 64K x 1 (when cascaded with an adjacent 36 Kb block RAM), 32K x 1, 16K x 2, 8K x 4, 4K x 9, 2K x 18, or 1K x 36 memory. Each 18 Kb block RAM can be configured as a 16K x 1, 8K x 2 , 4K x 4, 2K x 9, or 1K x 18 memory. Each 36K block RAM can be set to simple dual-port mode, doubling data width of the block RAM to 72 bits. The 18K block RAM can also be set to simple dual-port mode, doubling data width to 36 bits. Simple dual-port mode is defined as having one read-only port and one write-only port with independent clocks.

Table II presents configurations of logic elements and embedded memory blocks of Virtex-5 as LUTs of various sizes (number of inputs $\times$ number of outputs).

Such architecture of modern programmable FPGAs greatly extends the space of possible solution during the process of mapping the design into FPGA structure. Unfortunately this heterogeneous structure of available logic resources also greatly increases the complexity of mapping algorithms. The existing CAD tools are not well suited to utilize all possibilities that such modern programmable structures offer due to the lack of appropriate logic synthesis methods.

B. Distributed Arithmetic

The distributed arithmetic is a method of computing the sum of products:

$$y = \sum_{n=0}^{N-1} c[n] \times x[n]. \tag{1}$$

In many applications, a general purpose multiplication is not required. This is the case of filter implementation, if filter coefficients are constant in time. The partial product term $x[n] \times c[n]$ becomes multiplication with a constant. Then taking into account the fact that the input variable x is a binary number:

$$x[n] = \sum_{b=0}^{B-1} x_b[n] \times 2^b, \qquad \text{where } x_b[n] \in [0,1] \tag{2}$$

the whole convolution sum can be described as shown in (3).

$$y[n] = \sum_{b=0}^{B-1} 2^b \times \sum_{n=0}^{N-1} x_b[n] \times c[n] = \sum_{b=0}^{B-1} 2^b \times f(x_b) \tag{3}$$

Since $c[n]$ are constant the second sum in (3) can be implemented as a mapping $f(x_b)$, where $x_b = (x_b[0], x_b[1], \ldots, x_b[N-1])$. The efficiency of implementations based on this concept strongly depends on implementation of the function $f(x_b)$. The preferred implementation method is to realize the mapping $f(x_b)$ as the combinational module with N inputs. The schematic representation of such implementation is shown in Fig. 2, where the mapping f is presented as a lookup table (DA-LUT) that includes all the possible linear combinations of the coefficients and the bits of the incoming data samples [1].

If the number of coefficients N is large, the size of DA-LUT may become too big to efficiently implement it in available resources. In such case simple modification of DA concept can be used. Suppose the N coefficients have been grouped in L sets each containing K coefficients ($N = L \times K$). Then computing the sum of products can be described as (4).

$$\begin{aligned} y[n] = \sum_{n=0}^{L\times K-1} c[n] \times x[n] &= \sum_{l=0}^{L-1} \sum_{n=0}^{K-1} c[Kl+n] \times x[Kl+n] = \\ &= \sum_{b=0}^{B-1} 2^b \times \sum_{l=0}^{L-1} \sum_{n=0}^{K-1} c[Kl+n] \times x_b[Kl+n] = \\ &= \sum_{b=0}^{B-1} 2^b \times \sum_{l=0}^{L-1} f_l(x_b^l) \end{aligned} \tag{4}$$

Function $f(x_b)$ has been decomposed into L functions $f_l(x_b^l)$, where $x_b^l = (x_b[K \times l], x_b[K \times l + 1], \ldots, x_b[K \times l + N - 1])$. The sum is partitioned into L independent DA-LUTs with K inputs each. This allows implementing DA architecture as shown in Fig. 3. At the cost of additional adders the size of DA-LUTs can be significantly reduced.

III. Modified Distributed Arithmetic

DA concept proves to be a powerful technique for implementing MAC unit as a multiplierless algorithm through the use of combinational DA-LUT to store the precomputed values of $f(x_b)$ (3). The efficiency of implementations based on this concept strongly depends on implementation of DA-LUT representing the function $f(x_b)$. These blocks have to be efficiently mapped onto FPGA's logic resources. Since heterogeneous logic blocks are often used in the modern FPGA architectures (Xillinx Virtex-5 and Altera Stratix III series) construction of efficient mapping algorithm is very challenging task.

Modern heterogeneous structures are composed of programmable logic elements and embedded memory blocks. However all these resources can be referred to as LUTs of various sizes (Tab. I and Tab. II). It can be proposed a modification of distributed arithmetic concept that would allow decomposing DA-LUT into LUTs of sizes available in given FPGA architecture.

Let assume that available logic resources of specific FPGA are composed of LUT-like blocks grouped in M groups according to their sizes $Lin_i \times Lout_i$. Lin_i is the number

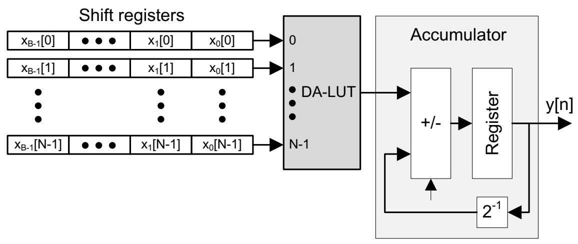
Shift registers
x_{B-1}[0] ••• x_1[0] x_0[0]
x_{B-1}[1] ••• x_1[1] x_0[1]
x_{B-1}[N-1] ••• x_1[N-1] x_0[N-1]
0
1
N-1
DA-LUT
Accumulator
+/-
Register
2^{-1}
y[n]

Fig. 2. DA architecture with lookup table (LUT).

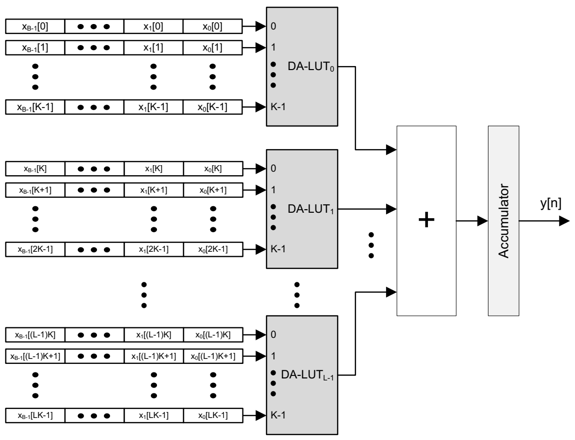
x_{B-1}[0] ••• x_1[0] x_0[0]
x_{B-1}[1] ••• x_1[1] x_0[1]
x_{B-1}[K-1] ••• x_1[K-1] x_0[K-1]
DA-LUT_0
x_{B-1}[K] ••• x_1[K] x_0[K]
x_{B-1}[K+1] ••• x_1[K+1] x_0[K+1]
x_{B-1}[2K-1] ••• x_1[2K-1] x_0[2K-1]
DA-LUT_1
x_{B-1}[(L-1)K] ••• x_1[(L-1)K] x_0[(L-1)K]
x_{B-1}[(L-1)K+1] ••• x_1[(L-1)K+1] x_0[(L-1)K+1]
x_{B-1}[LK-1] ••• x_1[LK-1] x_0[LK-1]
DA-LUT_{L-1}
0
1
K-1
+
Accumulator
y[n]

Fig. 3. DA architecture with partitioned lookup table.

of inputs and $Lout_i$ is the number of outputs of LUT blocks belonging to group i.

To implement DA-LUT block using LUTs of specific size a modification of (4) can be used. Suppose the N coefficients have been grouped in L sets each containing K_i coefficients, where

$$\sum_{l=0}^{L-1} K_l = N. \tag{5}$$

Moreover K_l can only be equal to one of available sizes Lin_i. As can be noticed groups may have various sizes, differently that in case (4). Let denote the number of i-th coefficient form j-th set as n_i^j . Then computing the sum of products can be described as (6).

$$y[n] = \sum_{l=0}^{L-1} \sum_{k=0}^{K_l-1} c[n_k^l] \times x[n_k^l] = \tag{6}$$

$$= \sum_{b=0}^{B-1} 2^b \times \sum_{l=0}^{L-1} \sum_{k=0}^{K_l-1} c[n_k^l] \times x_b[n_k^l] = \sum_{b=0}^{B-1} 2^b \times \sum_{l=0}^{L-1} f_l(x_b^l)$$

Function $f(x_b)$ has been decomposed into L functions $f_l(x_b^l)$. The sum is partitioned into L independent DA-LUTs. This allows implementing DA architecture similarly to this shown in Fig. 3. The number of inputs of l-th block (DA-LUT$_l$) is equal to K_l, while the number of outputs is equal to $(\lceil \log_2 K_l \rceil + q)$, where q is the number of bits used to represent coefficients $c[n]$.

Corollary 1. To implement l-th block (DA-LUT$_l$) of modified distributed arithmetic described by (6) it is needed at most $\lceil (\lceil \log_2 K_l \rceil + q)/Lout_i \rceil$ LUTs from group i for which Lin_i is equal to K_l.

Example 1.

Let us assume that two types of LUTs are available: 6×8 and 3×1. Let assume that sum of products (1) for $N = 9$ has to be computed for size of coefficients $q = 8$. To implement such sum of products using concept (6) we can group coefficients c into two sets $S_0 = \{$c[0], c[1], c[2], c[3], c[4], c[5]$\}$ and $S_1 = \{$c[6], c[7], c[8]$\}$. Then the sum of product can be represented as (7).

$$y[n] = \sum_{l=0}^{1} \sum_{k=0}^{K_l-1} c[n_k^l] \times x[n_k^l] =$$

$$= \sum_{k=0}^{5} c[n_k^0] \times x[n_k^0] + \sum_{k=0}^{2} c[n_k^1] \times x[n_k^1] =$$

$$= \sum_{b=0}^{B-1} 2^b \times \Big(\sum_{k=0}^{5} c[n_k^0] \times x_b[n_k^0] + \sum_{k=0}^{2} c[n_k^1] \times x_b[n_k^1] \Big) =$$

$$= \sum_{b=0}^{B-1} 2^b \times (f_0(x_b^0) + f_1(x_b^1)) \tag{7}$$

According to (7) the sum of product can be implemented in structure presented on Fig. 3 with two DA-LUT blocks such that the size of DA-LUT$_0$ is 6 inputs ($K_0 = 6$) and 11 outputs ($\lceil \log_2 K_0 \rceil + q = 11$), and the size of DA-LUT$_1$ is 3 inputs ($K_1 = 3$) and 10 outputs ($\lceil \log_2 K_1 \rceil + q = 10$). Implementation of DA-LUT$_0$ requires 2 logic elements of size 6×8 and DA-LUT$_1$ requires 10 logic elements of size 3×1.

Modification (6) allows to adjust the number of inputs of DA-LUTs of structure presented on Fig. 3 to the size of available logic resources. Then the mapping of each DA-LUT into logic elements is straightforward (Corollary 1). However in some cases not all logic elements used for mapping are utilized in 100%. It happens, when the number of outputs of given DA-LUT is not the multiple of number of outputs of logic element used for mapping. In Example 1 DA-LUT described by $f_0(x_b^0)$ has 6 inputs and 11 outputs, thus 2 logic elements of size 6×8 are required to implement it. One will implement 8 outputs of $f_0(x_b^0)$, what will utilize its capacity in 100%. However the other will implement only 3 outputs, what means, that only 37.5 % of its capacity is used and the rest is wasted.

This issue can be addressed by application of another modification of DA concept. Let represent all coefficients of sum of product from (3) as follows:

$$c[n] = 2^d \times c_A[n] + c_B[n], \tag{8}$$

where $c_B[n]$ represents d least significant bits of coefficient $c[n]$ and $c_A[n]$ represents $q - d$ most significant bits of coefficient $c[n]$. Then (3) can be expressed in following way:

$$y[n] = \sum_{b=0}^{B-1} 2^b \times \sum_{k=0}^{N-1} x_b[k] \times (2^d \times c_A[k] + c_B[k]) =$$

$$= \sum_{b=0}^{B-1} 2^b \times (2^d \times \sum_{k=0}^{N-1} x_b[k] \times c_A[k] + \sum_{k=0}^{N-1} x_b[k] \times c_B[k] =$$

$$= \sum_{b=0}^{B-1} 2^b \times (2^d \times f_A(x_b) + f_B(x_b)). \tag{9}$$

Function $f(x_b)$ has been decomposed into two functions $f_A(x_b)$ and $f_B(x_b)$. The sum is partitioned into two independent DA-LUTs, one with N inputs and $\lceil \log_2 N \rceil + q - d$ outputs and the second with N inputs and $\lceil \log_2 N \rceil + d$ outputs. This allows implementing DA architecture as shown in Fig. 4. At the cost of additional adders the number of outputs of DA-LUTs can be reduced.

Application of this concept allows to adjust the number of outputs of DA-LUTs. Application of both described techniques

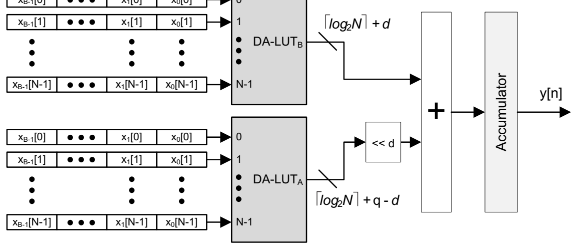

Fig. 4. DA architecture with decomposed lookup table.

makes it possible to map DA-LUT into heterogeneous architectures of modern FPGAs very efficiently.

Example 2.

Let us express coefficients from Example 1 as in (8), where $d = 5$. Then function $f_0(x_b^0)$ can be expressed using (9) as follows:

$$f_0(x_b^0) = \sum_{k=0}^{5} c[n_k^0] \times x_b[n_k^0] =$$

$$= 2^5 \times \sum_{k=0}^{5} c_A[n_k^0] \times x_b[n_k^0] + \sum_{k=0}^{5} c_B[n_k^0] \times x_b[n_k^0] =$$

$$= (2^d \times f_{0A}(x_b^0) + f_{0B}(x_b^0)). \tag{10}$$

Function $f_0(x_b^0)$ has been decomposed into two functions $f_{0A}(x_b^0)$ and $f_{0B}(x_b^0)$. The sum is partitioned into two independent DA-LUTs, one with 6 inputs and $\lceil \log_2 N \rceil + q - d = 6$ outputs and the second with 6 inputs and $\lceil \log_2 N \rceil + d = 8$ outputs. The second DA-LUT directly fits 6×8 logic element. However, first one still requires one 6×8 logic element. To implement $f_{0A}(x_b^0)$ we can use concept described by (6) to decompose it into two DA-LUTs with 3 inputs and then implement them using 3×1 logic elements.

Recursive application of concepts described by (6) and (9) allows to decompose initial DA-LUT into blocks that have required number of inputs and outputs. This makes it possible to adjust the size of partial DA-LUTs to size of available logic elements. Then the mapping of resultant structure into FPGA resources is straightforward.

IV. Application Results

This section demonstrates the application of introduced modified distributed arithmetic concept for implementation of DA-LUT in heterogeneous programmable structures.

There are presented results of implementation of DA-LUT for a low pass FIR filter with cutoff frequency 1.5 MHz. This is 15-tap filter with coefficients c = {23, -53, -108, -79, 55, 257, 438, 511, 438, 257, 55, -79, -108, -53, 23}. The filter accepts 8-bits signed input samples and coefficients are 10-bits numbers in fixpoint notation. As a target FPGA a EP3SL50F484C2 device from Stratix III family was used. All designs have been modeled in VHDL and synthesized with Quartus II Version 9.1 Web Edition.

Table III presents results of DA-LUT implementation in selected FPGA structure. The number of ALMs and M9K Blocks describe the utilization of resources, while f_{MAX} characterizes the performance.

The method described in this paper allows the designer to implement DA-LUT in many different ways, using different types of available resources. The rows *DA_Modified_1*$\div$4 present results of implementation for several different structures obtained by applying concepts described by (6) and (9). Modified DA structures have been modelled in VHDL and implemented using Quartus.

The row *DA_Non-modified* presents the result obtained by describing DA-LUT as truth table in VHDL and implementing it using CAD system.

TABLE III
Implementation Results of DA-LUT in Stratix III

Method	Resources		f_{MAX}
	ALMs	M9K Blocks	[MHz]
DA_Non-modified	616	0	241.95
DA_Modified_1	38	0	486.38
DA_Modified_2	19	1	290.87
DA_Modified_3	10	2	337.61
DA_Modified_4	9	3	335.01

All implementation using presented modified DA concept are characterized by much lower resource utilisation and higher performance (f_{MAX}) than implementation using classic DA approach.

V. Conclusion

Distributed Arithmetic concept is a very important technique used in designing digital signal processing modules for FPGA architectures. The quality of implementation of DSP algorithms based on DA strongly depends on efficiency of DA-LUT mapping. Heterogeneous structure of modern FPGAs greatly increases the complexity of this process. Modified DA concept presented in this paper allows for very efficient implementation of DA-LUT blocks in heterogeneous programmable structures. The method introduced here may have great impact on performance of DSP modules based on DA and targeted at modern FPGA architectures.

References

[1] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 2nd ed. Berlin: Springer-Verlag, 2004.

[2] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York: Wiley, 1999.

[3] M. A. M. Eshtawie and M. Othman, "On-line DA-LUT architecture for high-speed high-order digital FIR filters," in *Proceedings of the IEEE International Conference on Communication Systems*, Singapore, November 2006, p. 5.

[4] P. K. Meher, "Hardware-efficient systolization of DA-based calculation of finite digital convolution of finite digital convolution," *IEEE Transactions on Circuit and Systems II: Express Briefs*, vol. 53, no. 8, pp. 707–711, 2006.

[5] J. Xie, J. Heand, and G.Tan, "FPGA realization of FIR filters for high-speed and medium-speed by using modified distributed arithmetic architectures," *Microelectronics Journal*, vol. 41, no. 6, pp. 365–370, 2010.

[6] J. Cong and K. Yan, "Synthesis for FPGas with embedded memory blocks," *FPGA. New York*, pp. 75–82, 2000.

[7] S. Krishnamoorthy and R. Tessier, "Technology mapping algorithms for hybrid FPGAs containing lookup tables and plas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 5, pp. 545–559, 2003.

[8] M. Rawski, T. Łuba, Z. Jachna, and P. Tomaszewicz, "The influence of functional decomposition on modern digital design process," *Design of Embedded Control Systems*, pp. 193–203, 2005.

[9] T. Sasao, Y. Iguchi, and T. Suzuki, "On LUT cascade realizations of fir filters," in *Proceedings of Eighth Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, C. Wolinski, Ed., Porto, 2005, pp. 467–475.

[10] M. Rawski, P. Tomaszewicz, H. Selvaraj, and T. Łuba, "Efficient implementation of digital filtres with use of advanced synthesis methods targeted FPGA architectures," in *Proceedings of Eighth Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, C. Wolinski, Ed., Porto, 2005, pp. 460–466.

[11] P. Jamieson and J. Rose, "A verilog RTL synthesis tool for heterogeneous FPGAs," in *International Conference on Field Programmable Logic and Applications*, 2005, pp. 305–310.