Decimal Multiformat and Multioperand Online Addition

Carlos Garcia-Vega, Sonia Gonzalez-Navarro, Julio Villalba, Emilio L. Zapata Universidad de Málaga, Andalucía Tech, Departamento de Arquitectura de Computadores
Campus de Teatinos s/n, 29071 Málaga, España
E-mail: cgarcia@ac.uma.es, sonia@ac.uma.es, julio@ac.uma.es, zapata@uma.es

***Abstract*—This paper presents and analyzes two different strategies for designing multiformat online decimal adders (olDFA$_{\text{Mformat}}$). The first strategy uses a code conversion stage plus an online Decimal Full Adder (olDFA); the second one involves designing specific adders by modifying the architecture of the olDFA. Moreover, this paper presents the design of multioperand and multiformat adders using olDFAs and olDFA$_{\text{Mformat}}$s. We use synthesis results to verify the theoretical aspects of the designs and to analyze the robustness and lacks of the strategies. The guidelines presented in the paper are valuable to designers of online multiformat and multioperand-based solutions.**

I. Introduction

Computers represent, store, and manipulate numeric data in binary format. Many commercial applications, such as financial analysis, banking, fee calculation, and accounting operations are performed using binary arithmetic, which introduces a certain precision error when converting a decimal number to binary and vice versa [1], [2]. Therefore, decimal arithmetic is an alternative to counteract the loss of accuracy.

The recent demand for and growth in decimal arithmetic applications have been addressed by researchers from the point of view of both hardware and software. In terms of software, programming languages, such as COBOL, XML, Visual Basic, Java, and C provide support for decimal arithmetic. In the 1950s and 1960s, hardware solutions were represented by the early digital computers, such as ENIAC and UNIVAC [3]. A few years ago, the increasing demand for decimal technology led the IEEE society to issue the IEEE 754-2008 standard to deal with floating point decimal arithmetic. As a consequence, recent architectures, such as Power6, Power7, IBM z9, IBM z10, Fujitsu SparcX [4], [5], [6], [7], [8], decimal IP cores [9] and hardware designs [10], [11] have included decimal floating-point arithmetic.

On the other hand, online arithmetic defines algorithms for serial arithmetic operators that receive the input and generate the output starting from the most-significant digit (MSD first). The serial approach is advantageous because of the simplicity of the hardware required and the reduction in number and length of connections among modules. Moreover, MSD first can implement operations, such as division and square root, which are difficult to implement using least-significant digit first. This approach is advantageous even for multiplication and comparison because the relevant digits are produced first [12][13]. The drawback of this technique is the number of cycles required, which can be compensated by overlapping the execution of dependent operations. Given of all these characteristics, online arithmetic is used in a wide variety of applications in fields such as digital filtering [14], signal processing [15], [16], wireless communication systems [17], and neural networks [18].

Decimal parallel addition has efficient implementations [10], [19], [20], [21], [22]. However, to date, the literature on decimal online addition remains scarce, if not absent, except for the study [23]. In that paper, the authors present an online decimal adder, called olDFA, which operates with decimal digits represented with RBCD encoding [22].

Both multioperand and multiformat online addition for radix2 representation was analyzed in [24]. The corresponding radix-10 counterpart has not been addressed yet and is the main goal of this work. On one hand, decimal online multioperand addition was studied in [25]. On the other hand, to our knowledge, decimal multiformat online addition has never been addressed. Nevertheless, decimal format other than the classic BCD (with weight 8421 for the corresponding 4 bits) allows optimizing decimal algorithms. For example, for decimal CORDIC weights 5221 and 5421 are used in [26] and 5211 in [27]. For high performance decimal multipliers the codes 4221 and 5421 are used in [11] and in [28]. Thus, since the use of different formats can be useful for decimal algorithms, in this paper we deal with the multiformat case.

The main contribution of this paper is to propose techniques for designing customized architectures of decimal multiformat and multioperand online adders and to present their implementation. In this way, the theoretical aspects are corroborated by the actual implementation results. We base our designs on the use of the olDFA [23] and the adder trees presented in [25]. Nevertheless, the adaptation of the multiformat radix-2 online of [24] to the decimal case is not straightforward due to the complexity of the decimal codes, forcing us to use a completely different strategy to that proposed in [24].

The paper is organized as follows: background information on the olDFA is provided in section II; section III reviews several online decimal multioperand adders; section IV describes the strategies used to build online decimal multiformat adders, which are extended to build multioperand and multiformat adders in section V; section VI presents the synthesis results for some of the proposed architectures; and section VII

This research was supported by the Spanish grant CICYT TIN2013-42253-P

presents the conclusions.

II. Online decimal adders

This section describes the online decimal full adder (olDFA) and its pipelined version (olDFA$_p$), both presented in [23].

The most-frequent representation used in online arithmetic is signed-digit (SD), with both symmetric $\{-a, ..., a\}$ and asymmetric $\{b, ..., c\}$ digit sets. There are three popular representations in signed-digit: the Svoboda code [19], the positive/negative component representation [10], [20], [21], and the two's complement representation. The left to right mode of the online computation requires flexibility, which is achieved by the use of redundant representation. For the decimal case, consider the general case of having a 4-bit decimal code. Let $[-\alpha, ..., \beta]$ denote the different values corresponding to this code. The condition for this code to have sufficient redundancy to prevent a carry propagation is $11 \leq \alpha + \beta \leq 15$ [10]. The Redundant Binary Coded Decimal (RBCD) representation is composed by a 4-bit code that is described in [22]. This representation is signed-digit with symmetric digit set $\{-7, \ldots, 7\}$, and meets the previous condition. Each digit of the symmetric set is represented in two's complement using 4 bits and is shown in Table I.

Digit	RBCD	Digit	RBCD
0	0000		
1	0001	-1	1111
2	0010	-2	1110
3	0011	-3	1101
4	0100	-4	1100
5	0101	-5	1011
6	0110	-6	1010
7	0111	-7	1001

TABLE I
RBCD digits

The olDFA is obtained from the serialization of the parallel RBCD adder introduced in [10] and follows the concept of the radix–2 online full adder proposed in [29]. Fig. 1 depicts the top-level design of the olDFA. It has two RBCD digits as input operands, x_i, y_i, that are fed into a decomposition block. The outputs of this block, z_i and v_i, are stored in registers, whereas output t_{i+1} (current transfer bit) is added with the previously stored information in the registers, (z_{i+1}, v_{i+1}); this addition gives the value of the digit sum, s_{i+1}.

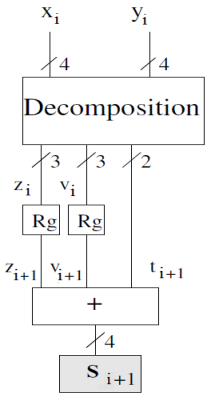


Fig. 1. olDFA adder.

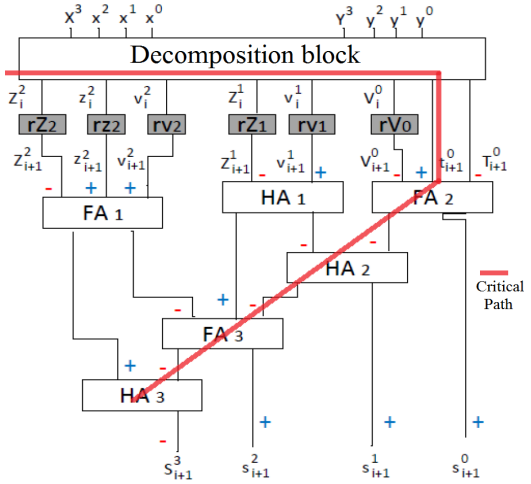


Fig. 2. olDFA structure and delay (bit level).

Fig. 2 shows, in bit-level detail, the structure of the olDFA and its critical path. The decomposition block performs the equations (1), which involves a rearrangement and partial addition of the bits of the RBCD operands. These operations are depicted in Fig. 3 in the upper part (rearrangement of the operand bits) and middle part (partial addition of the operand bits).

$$
\begin{aligned}
t^0_{i+1} &= \overline{X^3_i + Y^3_i} \\
T^0_{i+1} &= X^3_i \cdot Y^3_i \cdot (\overline{x^2_i} + \overline{x^1_i \cdot y^2_i}) + \\
&\quad (\overline{x^2_i + y^2_i}) \cdot (\overline{x^1_i \cdot (X^3_i + Y^3_i)}) \\
Z^2_i &= X^3_i \cdot Y^3_i \cdot (x^2_i \cdot x^1_i + \overline{y^2_i}) + \overline{X^3_i \cdot Y^3_i} \cdot \overline{x^1_i} \cdot y^2_i + \\
&\quad \overline{(X^3_i + Y^3_i) \oplus x^1_i \cdot \overline{y^2_i}} + (\overline{X^3_i} + \overline{Y^3_i}) \cdot x^2_i \cdot \overline{y^2_i} \\
z^2_i &= \overline{X^3_i \cdot Y^3_i} \cdot (\overline{x^2_i + y^2_i} \cdot \overline{x^1_i} + x^2_i \cdot y^2_i) + \\
&\quad X^3_i \cdot x^2_i \cdot (x^1_i \oplus Y^3_i) + \\
&\quad \overline{y^2_i} \cdot (\overline{X^3_i + Y^3_i} \cdot \overline{x^2_i} + x^2_i \cdot x^1_i \cdot Y^3_i) \\
Z^1_i &= (\overline{x^2_i + y^2_i} + x^1_i) \cdot (X^3_i \oplus Y^3_i) + \\
&\quad \overline{X^3_i} \cdot (\overline{x^2_i + y^2_i} \cdot x^1_i + \overline{x^1_i} \cdot \overline{Y^3_i} \cdot (y^2_i + x^2_i)) + \\
&\quad Y^3_i \cdot (x^2_i \cdot x^1_i \cdot y^2_i + X^3_i \cdot \overline{x^1_i}) \\
V^0_i &= x^0_i \oplus y^0_i \\
v^1_i &= y^1_i \oplus (x^0_i + y^0_i) \\
v^2_i &= y^1_i \cdot (x^0_i + y^0_i)
\end{aligned}
\tag{1}
$$

In Fig. 2, registers rZ_2, rZ_1, rz_2, rV_0, rv_1 and rv_2 store all the outputs of the decomposition block, except the transfer bits. These transfer bits, consisting of the pair (T^0_{i+1}, t^0_{i+1}), and the information previously stored in the registers are added using a network of full-adders (FAs) and half-adders (HAs). This addition is in the critical path of the olDFA, highlighted in Fig. 2. Therefore, the clock cycle (CC) of the olDFA is $CC_{olDFA} = T_{decom} + 2 \cdot T_{FA} + 2 \cdot T_{HA} + T_{reg}$, where T_{decom} is the delay of the decomposition block, T_{FA} is the delay of a FA, T_{HA} is the delay of a HA, and T_{reg} is the delay of a register load.

Due to the row of registers in Fig. 2 (grey blocks), the online

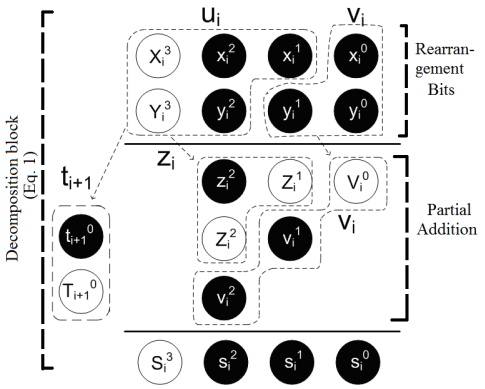


Fig. 3. Grouping scheme and sum of the bits of the RBCD operands [10].

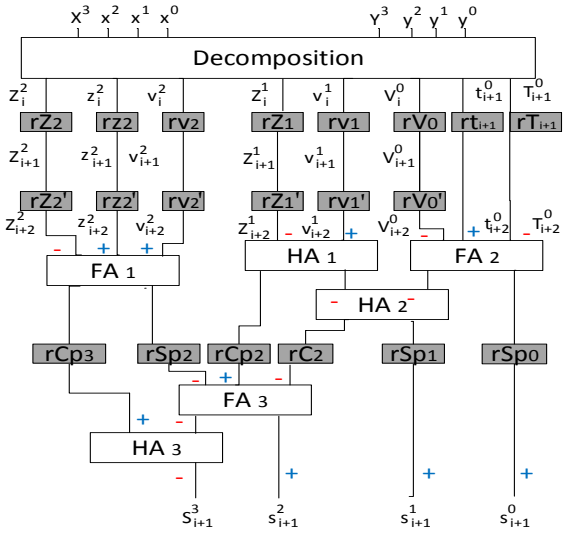


Fig. 4. olDFA$_p$ module: the pipelined version of the olDFA.

delay of the olDFA is $\delta_{olDFA} = 1$.

Therefore, the total execution time for adding two n-digit RBCD data is $T_{olDFA} = (1 + n) \cdot CC_{olDFA}$.

The pipelined version of the olDFA (called olDFA$_p$) is presented in Fig. 4. To obtain well-balanced stage, two levels of registers have been included because the delay of the decomposition block is close to that of a FA plus a HA ($T_{decom} \simeq T_{FA} + T_{HA}$).

The clock cycle of the olDFA$_p$ is $CC_{olDFA_p} = max\{T_{decom}, T_{FA} + T_{HA}\} + T_{reg}$.

Although CC_{olDFA_p} is less than CC_{olDFA}, the online delay of the olDFA$_p$ is $\delta_{olDFA_p} = 3$.

Therefore, the total execution time for adding two n-digit RBCD data using an olDFA$_p$ is $T_{olDFA_p} = (3 + n) \cdot CC_{olDFA_p}$.

In order to reduce the initiation interval between successive computations for data streaming, a hardware modification to both olDFA and olDFA$_p$ was introduced in [23]. Fig. 5 shows this modification in the olDFA. It consists of two AND-gates whose inputs are connected to the transfer bits (T^0_{i+1}, t^0_{i+1}) and a control signal referred as `Control`. This control signal is forced to be zero only when the most significant digit of the next two n-digit operands enter the olDFA (or olDFA$_p$). This modification makes it no necessary to insert separation cycles between the processing of two consecutive data streams; i.e. there is no penalty when it has to be processed a stream of data. It was proved in [23] that this modification neither increases area nor delay in both olDFA and olDFA$_p$.

The adder trees described in the next section were built using olDFA and olDFA$_p$ modules with this modification. In this way, the throughput of the adder trees is similar to that of the standard serial case.

III. Online Decimal Multioperand Addition

Several architectures for performing online decimal multioperand addition have been described in [25]. These multioperand adders are designed by following a method which minimizes hardware resources, and therefore, power consumption. This method was proposed in [24] and consists of building adder trees in which the maximum possible olDFA (or olDFA$_p$) modules are placed in each level. The resulting architectures have the same delay as other possible configurations, but with fewer external registers. Hence, this is the best configuration option in terms of saving hardware and power consumption. The design of the multiformat and multioperand adder trees presented in section IV follows this method as well.

A. olDFA-based and olDFA$_p$-based adder trees

Fig. 6 shows two examples of olDFA-based adder trees. With m being equal to the number of operands of the multioperand adder tree, and taking into account the level enumeration shown in the figure, the closed-form expressions of the following parameters can be derived:

- k, the total number of olDFAs in the adder tree:
  $k = m - 1$
- L, the total number of levels in the adder tree:
  $L = \lceil log_2 m \rceil$
- m_l, the number of operands at level l:
  $m_l = \lceil m_{l-1}/2 \rceil$ starting with $m_1 = m$

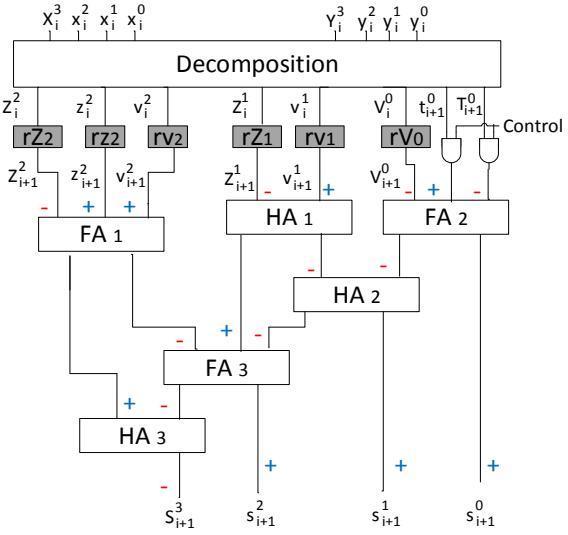


Fig. 5. olDFA with control of the input transfers.

- k_l, the number of olDFAs at level l:
  $k_l = \lfloor m_l/2 \rfloor$
- R_l, the number of external register at level l:
  $R_l = m_l \ mod \ 2$

These expressions can be useful in the process of designing an olDFA-based adder tree.

All the previous parameters depend on m and the expressions m_l, k_l, and R_l are only valid for the adder trees built according to the method mentioned at the beginning of this section. More details about the derivation of these parameters can be found in [25].

Another parameter derived in [25] is the clock cycle of an olDFA-based adder tree for m operands, CC_{olDFA_m}, which is expressed as:

$$CC_{olDFA_m} = \lceil log_2 m \rceil \cdot CC_{olDFA} \tag{2}$$

Note that the clock cycle CC_{olDFA_m} depends on the number of levels of the tree, and therefore, depends on m.

The online delay of a generic olDFA-based adder tree also depends on m since its expression is:

$$\delta_{olDFA_m} = L = \lceil log_2 m \rceil$$

and the total execution time for adding m n-digit RBCD operands is:

$$T_{olDFA_m} = (L + n) \cdot CC_{olDFA_m}$$

Due to the fact that the online delay of an olDFA$_p$ adder is equal to three, then the number of external registers in the olDFA$_p$-based adder trees is three times as much as that needed for the olDFA-based architectures. As a consequence, the parameter R_l (number of registers at level l) of an olDFA$_p$-based adder tree is $R_l = 3 \cdot (m_l \ mod \ 2)$. The remaining parameters k, L, m_l and k_l have the same expressions as those of olDFA-based adder trees.

The clock cycle expression of an olDFA$_p$-based adder tree for m operands is:

$$CC_{olDFA_{pm}} = T_{decom} + T_{FA} + T_{HA} + T_{reg} \tag{3}$$

that is, the clock cycle is equal to the last stage of an olDFA$_p$ plus the first stage of the next olDFA$_p$ in the tree. $CC_{olDFA_{pm}}$ is smaller than CC_{olDFA_m}, and in contrast to the latter, does not depend on m. The online delay expression of an olDFA$_p$-based adder tree is $\delta_{olDFA_{pm}} = 3 \cdot L$, and the total execution time for adding m n-digit RBCD operands is:

$$T_{olDFA_{pm}} = (3 \cdot L + n) \cdot CC_{olDFA_m}$$

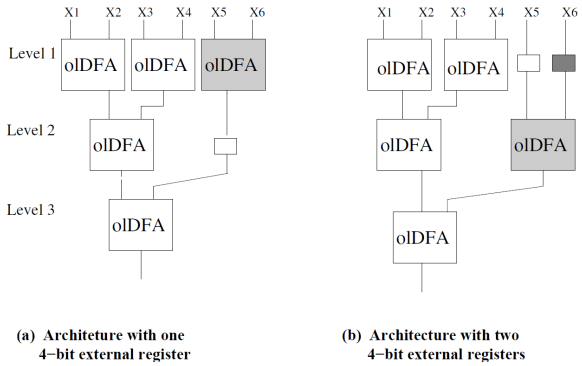


Fig. 6. Two olDFA-based architectures for 6 operands.

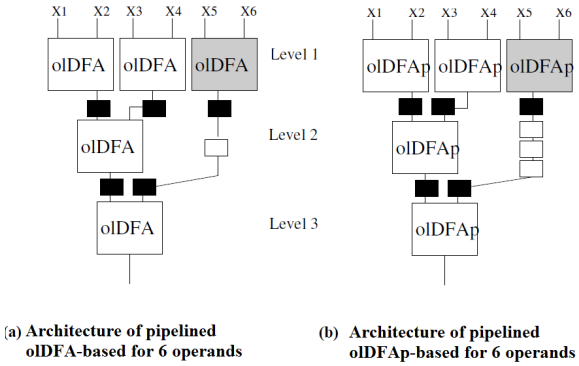


Fig. 7. Pipelined olDFA-based and olDFAp-based architectures for 6 operands.

B. Pipelining adder trees

Since the clock cycle of an olDFA$_p$-based adder tree is the sum of the first and last stages of two consecutive olDFA$_p$ modules, placing pipeline registers at every level of the tree gives the best performance in terms of delay and, at the same time, provides well-balanced stages.

Fig. 7(a) and Fig. 7(b) show pipelined versions of olDFA-based and olDFA$_p$-based adder trees for $m = 6$ operands, respectively. By placing the pipeline registers (colored in black) at each level of the trees, the clock cycle of a pipelined olDFA-based adder tree for m operands , $CC_{P-olDFA_m}$, matches the clock cycle of an olDFA. Thus, $CC_{P-olDFA_m} = CC_{olDFA}$.

In the same way, the clock cycle of a pipelined olDFA$_p$-based adder tree for m operands, $CC_{P-olDFA_{pm}}$, is the same as the clock cycle of an olDFA$_p$. Thus, $CC_{P-olDFA_{pm}} = CC_{olDFA_p}$.

If the clock cycle expressions of all architectures reviewed in this section are analyzed, it can be seen that the pipelined olDFA$_p$-based ones have the lowest clock cycle. A side effect of pipelining is that the online delay of the adder trees increases. The online delay of a pipelined olDFA-based adder tree is $\delta_{P-olDFA_m} = 2 \cdot L - 1$, whereas the online delay of a pipelined olDFA$_p$-based adder tree is: $\delta_{P-olDFA_{pm}} = 4 \cdot L - 1$.

The total execution time to operate m n-digit RBCD operands using a pipelined olDFA-based adder tree is:

$$T_{P-olDFA_m} = (2 \cdot L - 1 + n) \cdot CC_{olDFA}$$

and the total execution using a pipelined olDFA$_p$-based adder tree is:

$$T_{P-olDFA_{pm}} = (4 \cdot L - 1 + n) \cdot CC_{olDFA_p}$$

IV. Online decimal multiformat addition (olDFA$_{MFORMAT}$)

As shown in section I, the use of different formats can optimize decimal algorithms. This section describes the design

of online decimal full adders which support input operands encoded with different formats (multiformat) and provides the result in RBCD. As mentioned in section II, the supported decimal formats have to fulfill the condition $11 \leq \alpha+\beta \leq 15$ in order to have sufficient redundancy to prevent a carry propagation.

Apart from the RBCD, there are many other 4-bit codes that meet this condition. Nevertheless, we limit the study of redundant decimal codes to the cases in which the most significant bit has negative weight and the remaining bits have positive weight (similar to the RBCD). In this way, the delay and area of the corresponding adders are similar to the case of the pure RBCD adders (other codes involve a high hardware cost which makes them non-competitive). Table II shows the redundant decimal codes that are discussed in this paper. In the first column, we present the weight of each bit of the code as a function of its relative position. The second column provides the digit set of each code, and the third and fourth columns show some examples of each code. The first code of the table (with weight -8421) corresponds to RBCD. For the sake of clarity, from this point on we refer to $RBCD_{-7421}, RBCD_{-7321}$... as the RBCD code with weight -7421, -7321 ... , and keep the RBCD notation for the $RBCD_{-8421}$ code (i.e., $RBCD = RBCD_{-8421}$).

TABLE II
RBCD CODES FOR ONLINE DECIMAL MULTIFORMAT ADDITION

Code (weight)	Digit set	Example (digit 4)	Example (digit -4)
-8421*	{-7,..,7}	0100	1100
-7421	{-7,..,7}	0100	1011
-7321	{-7,..,6}	0101	1011—1100
-6421	{-6,..,7}	0100	1010
-6321	{-6,..,6}	0101	1010
-6221	{-6,..,5}	0110	1010—1100
-5421	{-5,..,7}	0100	1001
-5321	{-5,..,6}	0101	1001
-4421	{-4,..,7}	0100	1000

* Standard RBCD

Let olDFA$_{Mformat}$ denote a general two-operands online decimal multiformat adder. There are two ways to approach the design of multiformat adders: i) by using the existing olDFA adder with a pre-code conversion stage to RBCD; and ii) by designing tailored adders for the specific associated codes. These cases are addressed separately, and then, their advantages and disadvantages are discussed.

A. Multiformat by code conversion stage

The olDFA described in previous sections works with RBCD encoded operands. One way to design an online decimal multiformat adder is by using an olDFA that has undergone a previous conversion stage from a non-RBCD code shown in Table II to RBCD code. The delay and area of the conversion depend on the specific code conversion functions.

For example, the conversion functions from RBCD$_{-5421}$ code (X_i^3,x_i^2, x_i^1, x_i^0) to RBCD code ($X_i^{'3}$,$x_i^{'2}$, $x_i^{'1}$, $x_i^{'0}$) are:

$$\begin{aligned} X_i^{'3} &= X_i^3 \cdot (\overline{x_i^2} + \overline{x_i^1} \cdot \overline{x_i^0}) \\ x_i^{'2} &= \overline{X_i^3} \cdot x_i^2 + X_i^3 \cdot ((x_i^1 + x_i^0) \oplus x_i^2) \\ x_i^{'1} &= \overline{X_i^3} \cdot x_i^1 + X_i^3 \cdot (\overline{x_i^1 \oplus x_i^0}) \\ x_i^{'0} &= x_i^0 \oplus X_i^3 \end{aligned} \quad (4)$$

As an example of the above conversion expression, the number 1101_{-5421} is equal to 0000_{RBCD} since:

$$\begin{aligned} X_i^{'3} &= 1 \cdot (\overline{1} + \overline{0} \cdot \overline{1}) = 1 \cdot 0 = 0 \\ x_i^{'2} &= \overline{1} \cdot 1 + 1 \cdot ((0+1) \oplus 1) = 0 + 1 \cdot (0) = 0 \\ x_i^{'1} &= \overline{1} \cdot 0 + 1 \cdot (\overline{0 \oplus 1}) = 0 + 1 \cdot (\overline{1}) = 0 \\ x_i^{'0} &= 1 \oplus 1 = 0 \end{aligned}$$

Note that the critical path of these equations ($x_i^{'2}$) goes through four logic levels (a logic level corresponds to a gate of three inputs; this model was used in [10] and we follow it as a rough approximation. It is corroborated with the actual implementation results presented in section VI). Therefore, in order to add RBCD$_{-5421}$ and RBCD numbers, we need to convert RBCD$_{-5421}$ encoded operands to RBCD, and then add two RBCD operands by using an olDFA.

We have calculated the conversion functions of all the codes shown in Table II to RBCD (see Appendix I). Table III shows the number of logic levels required for those conversion functions.

TABLE III
DELAY FOR THE CONVERSION TO RBCD

Code	Logic Levels
RBCD	0
RBCD$_{-7421}$	4
RBCD$_{-7321}$	4
RBCD$_{-6421}$	3
RBCD$_{-6321}$	4
RBCD$_{-6221}$	2
RBCD$_{-5421}$	4
RBCD$_{-5321}$	3
RBCD$_{-4421}$	1

The architecture of a general olDFA$_{Mformat}$ by code conversion is shown in Fig. 8, where each 4-bit input (X,Y) has a signal (f1, f2) that represents the format of the incoming number. The area and delay of the `Code Conversion` module depend on the specific associated codes of the input operands, as well as, the number of different formats to be supported at each input.

As an example, the simplest olDFA$_{Mformat}$ by code conversion is that one that supports two formats for each input, one with RBCD and the other with RBCD$_{-4421}$ (see Fig. 9), whereas the most complex olDFA$_{Mformat}$ by code conversion is that one that covers all the codes of Table II (see Fig. 10). In any case, the online delay of an olDFA$_{Mformat}$ by code conversion is $\delta_{olDFA_{Mformat}} = 1$, which remains the same as that of the olDFA. However, the clock cycle of an olDFA$_{Mformat}$ is:

$$CC_{olDFA_{Mformat}} = CC_{olDFA} + \Delta \quad (5)$$

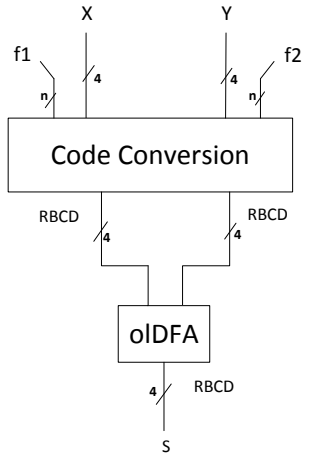


Fig. 8. olDFA$_{Mformat}$ by code conversion.

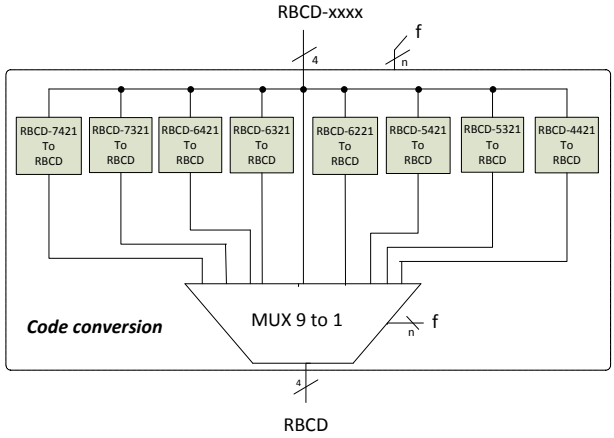


Fig. 10. The most complex case: Conversion stage covering all the codes of Table II.

where Δ varies and depends on the number of supported formats at the input of the olDFA$_{Mformat}$.

Returning to the above examples, if the olDFA$_{Mformat}$ supports the simplest case (RBCD and RBCD$_{4421}$), $\Delta = T_{mux2-1} + 1ll$, where T_{mux2-1} is the delay of a 2-1 multiplexor and ll the delay of one logic level (see Fig. 9 and Table III); if the olDFA$_{Mformat}$ supports the most complex case, then $\Delta = T_{mux9-1} + 4ll$, where T_{mux9-1} is the delay of a 9-1 multiplexor (see Fig. 10 and Table III).

B. Multiformat by specific design

This subsection shows the design of an olDFA$_{Mformat}$ that supports two different codes in both inputs of the adder without a code conversion module. The goal is to reduce the overhead due to the conversion stage of the previous proposal by designing tailored adders for each possible pair of codes of Table II. More than two codes per input is discussed in section IV-D. From this point on, we refer to these tailored adders as multiformat adders by specific design.

To design these tailored adders with RBCD output, the efficiency in the delay of the decimal adder proposed in [10] for RBCD operands has been taken into account. Thus, we decided to keep the basic decomposition scheme proposed in that paper (and depicted in Fig. 3 in this paper) in order to minimize the computation time for the new multiformat online decimal adders.

An example of this decomposition scheme for the multiformat case is shown in the upper part of Fig. 11, where the white circles denote bits with a negative weight (called negabits) and the black circles denote bits with a positive weight (called posibits); groups Z_i and V_i and t_{i+1} are calculated following equations (1) (shown in section II). Codes RBCD and RBCD$_{-5421}$ are considered in this example. We can see in Fig. 11 that the same binary data (1110 plus 1110) at input are interpreted in a different way as a function of the format of the operands. Note that three different results are obtained while maintaining the same decomposition scheme. Thus, three different functions are involved, and three different hardware submodules are required.

The olDFA$_{Mformat}$ proposed in this section is based on a modification of the internal architecture of the olDFA. Fig. 12 shows the block diagram of the olDFA and the new olDFA$_{Mformat}$ for comparison. It can be seen that there is a new module for format selection and a new decomposition module (`Mformat decomposition` module). Next, we present in detail the architecture of both the new `Mformat decomposition` module and the format selection module of Fig. 12(b).

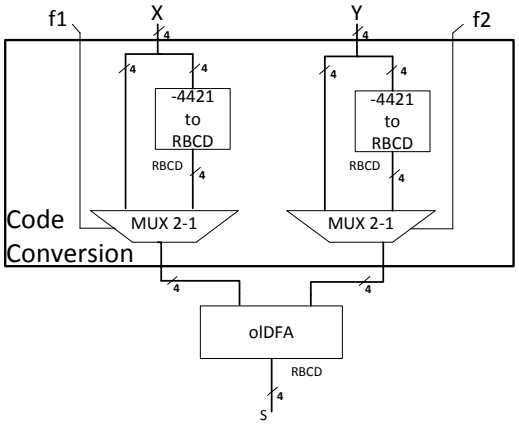


Fig. 9. The simplest case: olDFA$_{Mformat}$ supporting two codes (RBCD and RBCD$_{4421}$)

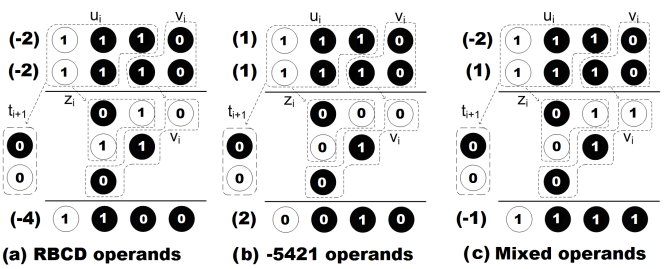


Fig. 11. Example of multiformat decomposition scheme (upper part of the figure)

1) Mformat Decomposition module: In an olDFA, two RBCD numbers are added and the decomposition module performs the equations (1).

In the olDFA$_{Mformat}$ the new decomposition module is more complex and the associated equations depend on the specific codes supported at the inputs. In fact, equations (1) are a particular case of having two RBCD inputs for the new olDFA$_{Mformat}$.

Let us assume an olDFA$_{Mformat}$ with inputs i1 and i2, which supports codes with format A and B in each input. Let f1 (f2) denote a bit to sign the format at the input i1 (i2). The four possible code combinations at the input are as follows:

```
i1 i2  <-- inputs
------
A  A
B  B   <-- Code combinations
A  B
B  A
```

The equations to deal with each code combination are different and require different hardware submodules for their implementation. Nevertheless, the equations for AB and BA are very similar and can be unified and integrated in a single hardware submodule (of similar complexity to that of the AA and BB combination). The general architecture of the `Mformat decomposition` module for online multiformat addition by specific design is shown in Fig. 13, where three different hardware submodules are devoted to dealing with the code combinations AA, BB, and AB&BA. Signal $f2$ in the submodule AB&BA is used to identify the actual code combination at the input (that is either AB or BA).

Taking into account the nine codes of Table II, there are 36 pairs of two different codes, and therefore, 36 different olDFA$_{Mformat}$ designs. We have designed and implemented some of these olDFA$_{Mformat}$. Although we have not designed all the 36 different possible adders, based on our study we can deduce that the area and delay are similar for all of them. In fact, in Appendix II, we analyze how to build a tailored olDFA$_{Mformat}$ that supports a RBCD encoded input plus any input encoded with any code of Table II.

In the following, as an example and without any loss of generality, we present a concrete design which involves RBCD and RBCD$_{-5421}$ codes (RBCD$_{-5421}$ is used in [11] for decimal multipliers to improve the area and latency for certain operations, as the decimal partial product reduction).

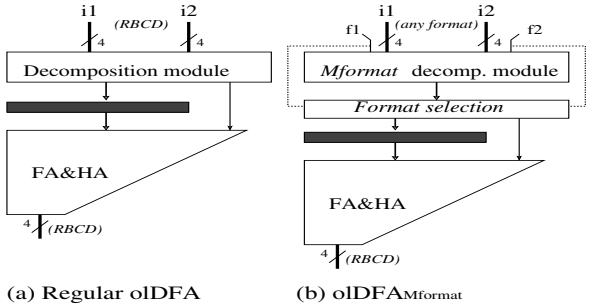


Fig. 12. Block diagram of the olDFA and the olDFA$_{Mformat}$ by specific design

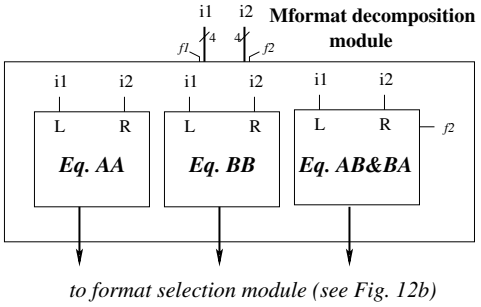


Fig. 13. Mformat Decomposition module for olDFA$_{Mformat}$ by specific design

Table IV shows the equivalence between the RBCD and the RBCD$_{-5421}$ codes.

The proposed olDFA$_{Mformat}$ can add two operands, both being RBCD encoded or both RBCD$_{-5421}$ or one being RBCD and the other RBCD$_{-5421}$. Let us study the three cases separately.

- **Both operands are RBCD encoded** (case of Fig. 11(a) and Eq. AA in Fig. 13). This is the case of the olDFA reviewed in section II, where equations (1) are implemented in the decomposition module. The critical path delay of these equations goes through four logic levels (linked to z_i^2).
- **Both operands are RBCD$_{-5421}$ encoded** (case of Fig. 11(b) and Eq. BB in Fig. 13). The equations implemented in the decomposition module are as follows:

$$
\begin{aligned}
t^0_{5421_{i+1}} &= (\overline{X_i^3 \cdot Y_i^3}) + (X_i^3 \oplus Y_i^3) \cdot x_i^2 \cdot y_i^2 \\
T^0_{5421_{i+1}} &= \overline{(X_i^3 \oplus Y_i^3)} \cdot (\overline{x_i^2} \cdot \overline{y_i^2}) \\
Z^2_{5421_i} &= \overline{X_i^3} \cdot \overline{Y_i^3} \cdot (\overline{\overline{x_i^2} \cdot x_i^1 \cdot \overline{y_i^2}}) \\
&\quad + X_i^3 \oplus Y_i^3 \cdot (x_i^2 \cdot \overline{x_i^1} + \overline{x_i^1} \cdot \overline{y_i^2}) \\
&\quad + X_i^3 \cdot Y_i^3 \cdot (x_i^2 \oplus y_i^2) \\
z^2_{5421_i} &= \overline{X_i^3} \cdot \overline{Y_i^3} \cdot (\overline{x_i^2 \oplus y_i^2}) \\
&\quad + X_i^3 \cdot Y_i^3 \cdot (\overline{x_i^2} \cdot x_i^1 \cdot \overline{y_i^2}) \\
&\quad + (X_i^3 \oplus Y_i^3) \cdot (x_i^2 \cdot \overline{y_i^2} + \overline{x_i^2} \cdot x_i^1 \cdot y_i^2) \\
Z^1_{5421_i} &= \overline{(X_i^3 \oplus Y_i^3)} \cdot (\overline{x_i^2} \cdot (x_i^1 \oplus y_i^2) + x_i^2 \cdot \overline{x_i^1}) \\
&\quad + (X_i^3 \oplus Y_i^3) \cdot (x_i^2 (x_i^1 \oplus y_i^2) + \overline{x_i^2} \cdot x_i^1) \\
V^0_{5421_i} &= (X_i^3 \oplus Y_i^3) \oplus (x_i^0 \oplus y_i^0) \\
v^1_{5421_i} &= \overline{(X_i^3 \oplus Y_i^3)} \cdot (y_i^1 \oplus (y_i^0 + x_i^0)) \\
&\quad + (X_i^3 \oplus Y_i^3) \cdot (y_i^1 \oplus (x_i^0 \cdot y_i^0)) \\
v^2_{5421_i} &= \overline{(X_i^3 \oplus Y_i^3)} \cdot (y_i^1 \cdot (y_i^0 + x_i^0)) \\
&\quad + y_i^1 \cdot x_i^0 \cdot y_i^0
\end{aligned}
\tag{6}
$$

The critical path goes through four logic levels (due to $Z^1_{5421_i}$). Note that the delay for two RBCD$_{-5421}$ inputs is the same as that of two RBCD inputs.

- **One operand is RBCD encoded and the other one is RBCD$_{-5421}$ encoded** (case of Fig. 11(c), Eq. AB&BA in Fig. 13). In this case, the decomposition is performed in two steps: i) computation in parallel of equations (6)

TABLE IV
RBCD VERSUS RBCD$_{-5421}$ CODES

	RBCD	RBCD$_{-5421}$
-7	1001	-
-6	1010	-
-5	1011	1000
-4	1100	1001
-3	1101	10101
-2	1110	1011
-1	1111	1100
0	0000	0000—1101
1	0001	0001—1110
2	0010	0010—1111
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110
7	0111	0111

and the following equations:

$$
\begin{aligned}
t^0_{neg_{i+1}} &= (X^3_i \cdot \overline{Y^3_i} \cdot f2 + \overline{X^3_i} \cdot Y^3_i \cdot \overline{f2}) \cdot x^2_i \cdot y^2_i \\
T^0_{neg_{i+1}} &= (X^3_i \cdot \overline{Y^3_i} \cdot f2 + \overline{X^3_i} \cdot Y^3_i \cdot \overline{f2}) \cdot \overline{x^2_i} \cdot \overline{x^1_i} \cdot \overline{y^2_i} \\
&\quad + X^3_i \cdot Y^3_i \cdot \overline{x^1_i} \cdot (x^2_i \oplus y^2_i) \\
Z^2_{neg_i} &= (X^3_i \cdot \overline{Y^3_i} \cdot f2 + \overline{X^3_i} \cdot Y^3_i \cdot \overline{f2}) \cdot (\overline{x^2_i} + x^2_i \cdot y^2_i) \\
&\quad + X^3_i \cdot Y^3_i \cdot \overline{x^1_i} \cdot (x^2_i + y^2_i) \\
z^2_{neg_i} &= X^3_i \cdot Y^3_i \cdot (\overline{y^2_i} \cdot (x^2_i \oplus x^1_i)) \\
&\quad + (X^3_i \cdot \overline{Y^3_i} \cdot f2 + \overline{X^3_i} \cdot Y^3_i \cdot \overline{f2}) \cdot (x^2_i \cdot x^1_i + \overline{x^1_i} \cdot \overline{y^2_i}) \\
Z^1_{neg_i} &= X^3_i \cdot Y^3_i \cdot \overline{(x^2_i \cdot \overline{x^1_i} \cdot \overline{y^2_i})} \\
&\quad + (X^3_i \cdot \overline{Y^3_i} \cdot f2 + \overline{X^3_i} \cdot Y^3_i \cdot \overline{f2}) \cdot \overline{x^1_i} \cdot \overline{(x^2_i \oplus y^2_i)} \\
V^0_{neg_i} &= X^3_i \cdot f2 + Y^3_i \cdot \overline{f2} \\
v^1_{neg_i} &= (X^3_i \cdot f2 + Y^3_i \cdot \overline{f2}) \cdot (y^0_i \oplus x^0_i) \\
v^2_{neg_i} &= (X^3_i \cdot f2 + Y^3_i \cdot \overline{f2}) \cdot y^1_i \cdot (y^0_i \oplus x^0_i)
\end{aligned}
\tag{7}
$$

and ii) XOR operation between them, that is,

$$
\begin{aligned}
t^0_{mix_{i+1}} &= t^0_{neg_{i+1}} \oplus t^0_{5421_{i+1}} \\
T^0_{mix_{i+1}} &= T^0_{neg_{i+1}} \oplus T^0_{5421_{i+1}} \\
Z^2_{mix_i} &= Z^2_{neg_i} \oplus Z^2_{5421_i} \\
z^2_{mix_i} &= z^2_{neg_i} \oplus z^2_{5421_i} \\
Z^1_{mix_i} &= Z^1_{neg_i} \oplus Z^1_{5421_i} \\
V^0_{mix_i} &= V^0_{neg_i} \oplus V^0_{5421_i} \\
v^1_{mix_i} &= v^1_{neg_i} \oplus v^1_{5421_i} \\
v^2_{mix_i} &= v^2_{neg_i} \oplus v^2_{5421_i}
\end{aligned}
\tag{8}
$$

Note that equations (7) include bit $f2$ which signs the code at the right input. Equations (6) and (7) are performed in the `Mformat decomposition` module. On the other hand, equations (8) are performed outside this module, in the format selection module, as explained in the next subsection.

2) Format selection module: Fig. 14 shows in detail the global architecture of the olDFA$_{Mformat}$ by specific design. Since the delay of equations (1), (6), and (7) is four logic levels and they work in parallel, the delay of the decomposition module of the olDFA$_{Mformat}$ is the same as that of the olDFA (which only implements equations (1)). Thus, from the point of view of computation time, the overhead for multiformat support compared to the regular olDFA is only due to the format selection modules of Fig. 14.

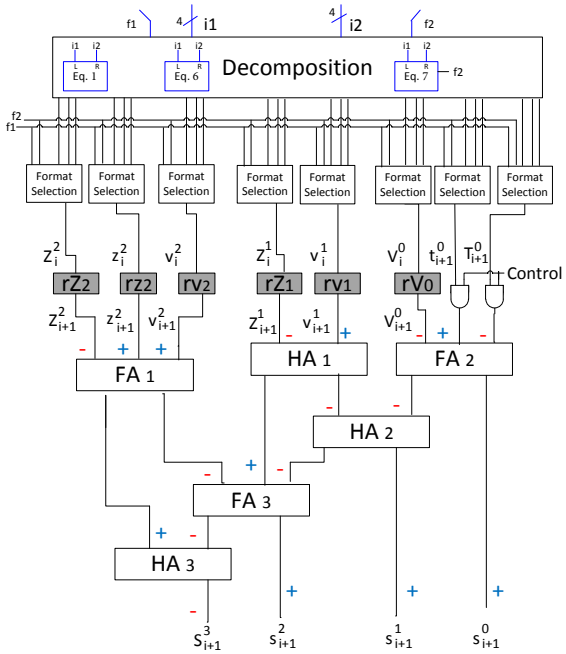


Fig. 14. Global architecture of the olDFA$_{\text{Mformat}}$ by specific design

Fig. 15 shows the internal structure of the format selection module for signal Z^1 (the same hardware is required for the remaining seven signals $t^0, T^0, Z^2, ...$, in Fig. 14). The logic of this module is in charge of executing equations (8). To switch between the four cases, we use two format signals $f1$ and $f2$, selecting the codes of the operands as shown in Fig. 15.

According to Fig. 15, the delay of the format selection module (and thus the time overhead for multiformat support) is that of a 2-1 multiplexor plus an XOR gate (note that the OR and XOR logic gates with inputs $f1, f2$ work in parallel

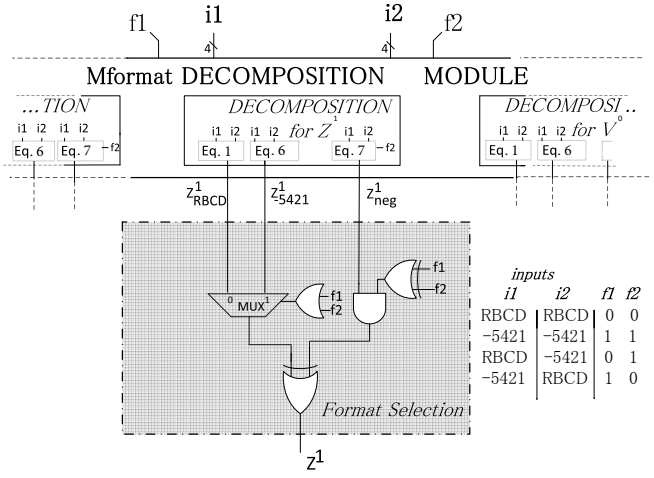


Fig. 15. Format Selection for Z^1 (similar HW for the remaining signals coming from the `Mformat decomposition` module)

with the `Mformat decomposition` module and, thus, they are not in the critical path of the format selection module).

C. Pipelined architectures for multiformat adders

1) Multiformat by code conversion stage: To pipeline the olDFA$_{Mformat}$ by code conversion with well-balanced stages, we have to take into account the complexity of the code conversion module. For the case of dealing with only two formats, since the delays of the conversion codes of Table III go from 1 to 4 logic levels and the decomposition module has a delay of 4 logic levels, the best option is to insert the corresponding registers just at the output of the code conversion stage and to replace the olDFA by an olDFA$_p$. The resulting architecture is presented in Fig. 16. This pipelined architecture has the same CC as that of the olDFA$_p$ and the online delay is $\delta_{olDFA_p}+1$ (due to the row of registers at the output of the `Code Conversion` module).

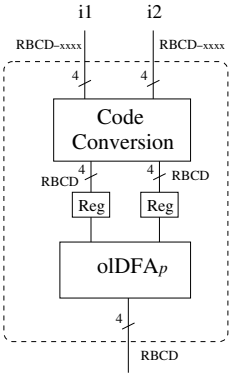


Fig. 16. Architecture of the Pipelined olDFA$_{Mformat}$ by code conversion

In the case of having a more complex code conversion module (thus, the olDFA$_{Mformat}$ supports more than two formats at the input), and to maintain a CC similar to that of olDFA$_p$, fine tuning may be performed by inserting a new level of pipeline registers just before the final multiplexor of the code conversion module (see Fig. 10).

2) Multiformat by specific design: To pipeline the olDFA$_{Mformat}$ by specific design, we have taken into account that the delay of the decomposition module is 4 logic levels and the delay of the format selection module is less than 4. Therefore, in order to maintain a CC similar to that of the olDFA$_p$, we insert pipeline registers between the decomposition module and the format selection modules, as shown in Fig. 17. In this case, the online delay of this pipelined architecture is δ_{olDFA_p}, i.e., it has the same online delay as an olDFA$_p$. Section VI presents and analyzes the experimental results of different pipelined architectures.

D. Code Conversion vs Specific Design

This section discusses the advantages and disadvantages of the proposed olDFA$_{Mformat}$ designs. The first case is that of olDFA$_{Mformat}$ supporting just two different formats. The overhead of olDFA$_{Mformat}$ by code conversion is due to the delays arising from the code conversion and a 2-1 multiplexor

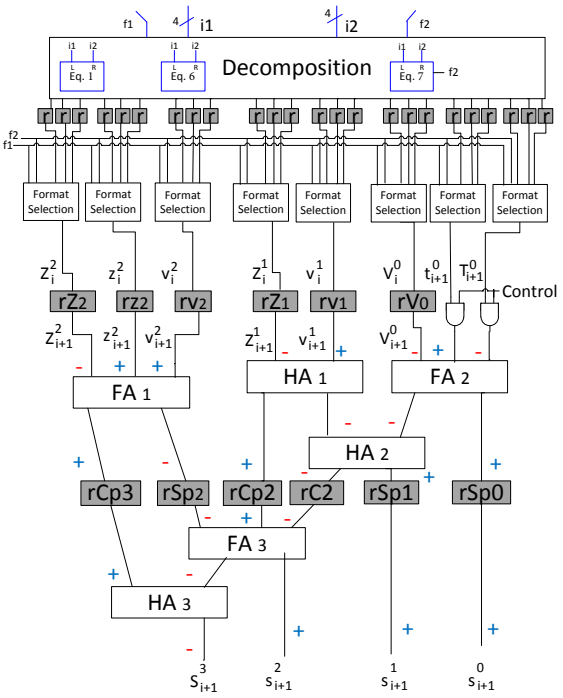


Fig. 17. Architecture of the pipelined olDFA$_{Mformat}$ by specific design

(to select one of the two formats at the input) shown in Fig. 9. On the other hand, the overhead of a olDFA$_{Mformat}$ by specific design is equal to the delay of a 2-1 multiplexor plus the delay of one XOR gate (see Fig. 15). Thus, from the point of view of time efficiency, the olDFA$_{Mformat}$ by specific design is the best, as expected. Nevertheless, the hardware cost of the first implementation is less than that of the second due to the complexity of the decomposition module of the olDFA$_{Mformat}$ by specific design.

For the case of an olDFA$_{Mformat}$ with three or more supported formats at each input, the corresponding decomposition module for a specific design becomes extremely complex, which makes an actual implementation impractical. Thus, for these cases, the olDFA$_{Mformat}$ by code conversion alternative is a reasonable choice since the code conversion module has a reduced hardware cost (see conversion functions in Appendix I), and the time penalty is due only to the final multiplexor (logarithmic increase). This is because all the conversion function elements work in parallel inside the code conversion module. In conclusion, for two operands the specific design approach is a good choice whereas for three or more operands the code conversion design seems more suitable.

V. Decimal Multioperand and Multiformat Addition

If an application needs to deal with multioperands with different formats (multiformat), it is possible to design a tree by following the design strategies of the online decimal multioperand adder presented in [25]. The first level of the tree has to be composed by online decimal full adders supporting multiformat operands (olDFA$_{Mformat}$), whereas the remaining levels are composed by regular olDFAs. This is due to

the fact that the output of the olDFA$_{Mformat}$ is an RBCD encoded number. Thus, the second and following levels of the tree deal with pure RBCD numbers, which can be managed by regular olDFAs. Fig. 18 shows a general tree architecture, where the inputs support different formats, whereas the second and successive levels of the tree support the RBCD format alone.

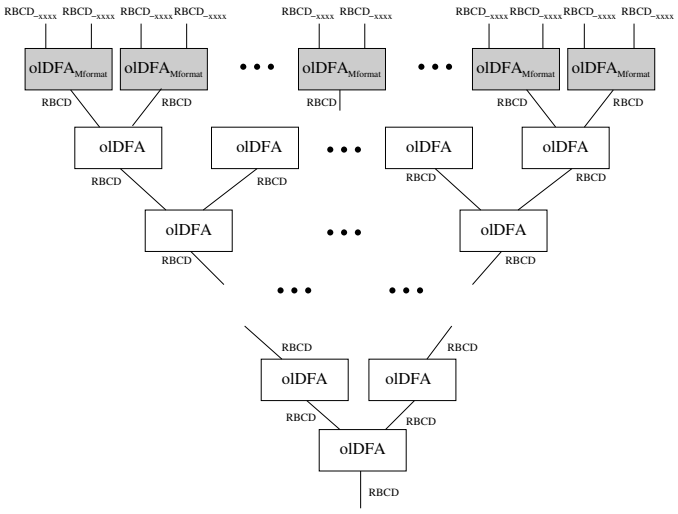


Fig. 18. Multiformat and multioperand tree architecture

The complexity of the different olDFA$_{Mformat}$ units of the first level of the tree depends on the format supported by each application. The analysis of the different configurations, time, and online delay presented in [25] are valid for the decimal multioperand and multiformat trees.

For a pipeline tree, all the alternatives proposed in [25] can be used (just replacing the olDFA$_{Mformat}$ and olDFA units of Fig. 18 by the corresponding pipelined versions). Section VI presents and analyzes in depth the experimental results of different pipelined architectures.

VI. Experimental Results

This section analyzes the performance of the architectures proposed in this study. These architectures were modeled in Verilog-HDL and the building blocks of the architectures (both $olDFA_{Mformat}$s by code conversion and by specific design) were verified using Mentor Graphics ModelSim SE-64 6.1e tool for all the 225 different input combinations (variations with repetition of 15 elements taken two at a time). We also synthesized the designs using Synopsys Design Compiler (DC) and the TSMC's tcbn65gplus 65 nm CMOS standard cell library. With this aim, we set up two different simulation scenarios by activating or deactivating the $dont\_touch$ attribute on the designs. In the first scenario (denoted as S1), we activated the $dont\_touch$ attribute to avoid modifying the structure of the olDFAs, olDFA$_p$s and $olDFA_{Mformat}$s in the trees. In this way, the structures are maintained and can be analyzed from a theoretical/analytical point of view. In the second simulation scenario (denoted as S2), we deactivated the $dont\_touch$ attribute to allow Synopsis DC to perform a balanced optimization between area and delay of the architectures.

A. $olDFA_{Mformat}$ Performance

Table V and Table VI show the area, delay, online delay (δ), initiation interval (I.I.), and throughput (THR.) of the olDFA$_{Mformat}$s described in section IV. More specifically, these tables show the results of the most representative olDFA$_{Mformat}$s: the simplest one by code conversion (that supports RBCD and RBCD$_{\text{-}4421}$ encoded operands, denoted as RBCD+RBCD$_{\text{-}4421}$ CodeConv), the most complex olDFA$_{Mformat}$ by code conversion (that supports the nine codes of Table III and denoted as 9 codes CodeConv), and two olDFA$_{Mformat}$s managing RBCD and RBCD$_{\text{-}5421}$ encoded operands, one designed following the code conversion scheme (denoted as RBCD+RBCD$_{\text{-}5421}$ CodeConv) and the other following the specific design scheme (denoted as RBCD+RBCD$_{\text{-}5421}$ SpecD). Note that CodeConv and SpecD stand for Code Conversion and Specific Design, respectively.

Table V shows the results of the designs under S1 and Table VI shows them under S2. We have shown the results under S1 and S2 of the olDFA and olDFA$_p$ adders for purposes of comparison in Table VII. Table V shows that all the results are consistent with the theory. That is, the 9 codes CodeConv olDFA$_{Mformat}$ is the one with the worst performance, whereas the RBCD+RBCD$_{\text{-}5421}$ SpecD is the best in terms of delay, throughput, and initiation interval. In fact, the overhead in the delay of the RBCD+RBCD$_{\text{-}5421}$ SpecD compared to the delay of the olDFA under S1 (see Table VII) is due to the Format Selection module (see Fig. 15). The area of the RBCD+RBCD$_{\text{-}5421}$ SpecD is higher than the area of the RBCD+RBCD$_{\text{-}5421}$ CodeConv, which is also consistent with the theory (due to the complex decomposition module). Regarding the pipelined version of the olDFA$_{Mformat}$s, the trend in their performance is similar to that of the non-pipelined versions. The results show that an optimization of 39% is achieved in the delay of the RBCD+RBCD$_{\text{-}5421}$ SpecD at the cost of an 8% increment in the area respect to the non-pipelined version. Note that the delay of the pipelined versions of the olDFA$_{Mformat}$s presented is practically the same (except for the 9 codes CodeConv design) and that this matches the delay of the olDFA$_p$, which again is consistent with the theory (the critical path of the pipelined olDFA$_{Mformat}$s is in the decomposition module of their olDFA$_p$s).

The results obtained under S2 are shown in Table VI. As mentioned, the goal of this scenario is to balance area and delay in the designs. Under this scenario, the RBCD+RBCD$_{\text{-}4421}$ CodeConv achieves the best results in terms of area and delay for both the non-pipelined and pipelined versions. In fact, the delays of both pipelined and non-pipelined RBCD+RBCD$_{\text{-}4421}$ CodeConv are practically the same as the delay of the olDFA and olDFA$_p$ under S2 (see Table VII).

Although there is no any other proposed decimal multiformat and multioperand online adder in the literature, for comparison purposes, we synthesized the RBCD parallel adder presented in [10]. Specifically, we synthesized two RBCD parallel adders: a 16-digit RBCD adder and a 32-digit RBCD adder (these figures correspond to the significand digits of a decimal64 and decimal128 DFP number, respectively). The delay and area of a 16-digit RBCD parallel adder are 0.365

TABLE V
RESULTS OF OLDFA$_{Mformat}$ IN THE FIRST SIMULATION SCENARIO (S1)

olDFA$_{Mformat}$	Non pipelined					Pipelined				
	delay ns	area $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s	delay ns	area $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s
RBCD+RBCD$_{-4421}$ CodeConv	0.387	530	1	6.58	161	0.218	649	4	4.36	286
9 codes CodeConv	0.493	1999	1	8.38	126	0.236	2073	4	4.72	264
RBCD+RBCD$_{-5421}$ CodeConv	0.412	747	1	7.00	151	0.218	797	4	4.36	286
RBCD+RBCD$_{-5421}$ SpecD	0.342	1046	1	5.81	182	0.21	1124	3	3.99	297

I.I.=Initiation Interval, THR.=Throughput, δ=online delay, CodeConv= Code Conversion, SpecD= Specific Design

TABLE VI
RESULTS OF OLDFA$_{Mformat}$ IN THE SECOND SIMULATION SCENARIO (S2)

olDFA$_{Mformat}$	Non pipelined					Pipelined				
	delay ns	area $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s	delay ns	area $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s
RBCD+RBCD$_{-4421}$ CodeConv	0.309	522	1	5.25	202	0.247	520	4	4.94	253
9 codes CodeConv	0.43	1366	1	7.31	145	0.273	1117	4	5.46	228
RBCD+RBCD$_{-5421}$ CodeConv	0.34	610	1	5.78	183	0.264	510	4	5.28	236
RBCD+RBCD$_{-5421}$ SpecD	0.341	896	1	5.79	183	0.271	990	3	5.15	230

I.I.=Initiation Interval, THR.=Throughput, δ=online delay, CodeConv= Code Conversion, SpecD= Specific Design

TABLE VII
RESULTS OF OLDFA AND OLDFA$_p$ IN BOTH SCENARIOS OF SIMULATION (S1 AND S2)

Scenarios of Simulation	olDFA (Non pipelined)					olDFA$_p$ (Pipelined)				
	delay ns	area $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s	delay ns	area $um2$	δ	I.I.(decimal64) ns	THR. (decimal64) mill. operat. per s
Scenario 1	0.297	479	1	5.05	210	0.218	549	3	4.14	286
Scenario 2	0.28	486	1	4.76	223	0.252	439	3	4.78	248

I.I.=Initiation Interval, THR.=Throughput, δ=online delay, CodeConv= Code Conversion, SpecD= Specific Design

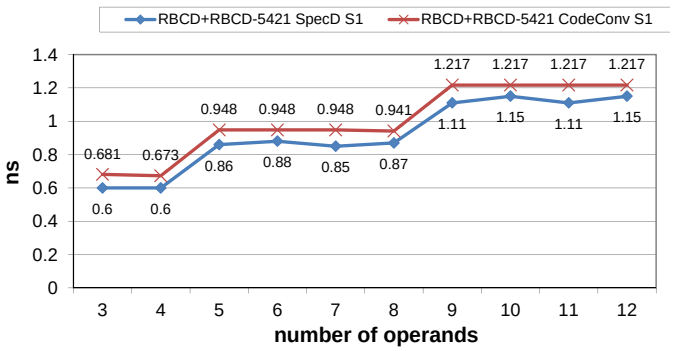


Fig. 19. Delay of RBCD+RBCD$_{-5421}$-based adder trees by SpecD and CodeConv under simulation scenario 1 (NON-PIPELINED VERSION)

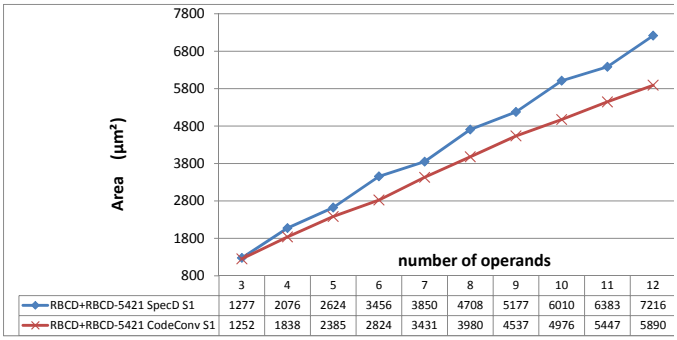


Fig. 20. Area of RBCD+RBCD$_{-5421}$-based adder trees by SpecD and CodeConv under simulation scenario 1 (NON-PIPELINED VERSION)

ns and 5671 μm^2, whereas the delay and area of a 32-digit RBCD parallel adder are 0.42 ns and 9348 μm^2. Comparing the results with those obtained for a RBCD+RBCD$_{-5421}$ by SpecD (see Table VI), we conclude that our online approach needs 80% (90%) less area than the counterpart parallel adder for 16 digits (32 digits), with a similar time to obtain the MSD. As conclusion, these figures are consistent with those expected for an online system: similar delay (for the MSD) and an important hardware cost reduction.

B. Performance of Multiformat and Multioperand Adder Trees

As in the previous section, the results shown below were obtained by running S1 and S2.

Recall that the olDFA$_{Mformat}$s (SpecD or CodeConv) are placed in the first level of the adder trees (see Fig. 18) and the remaining levels are composed of olDFAs (or olDFA$_p$s in the pipelined trees). Fig. 19 and Fig. 20 depict the delay and area of the non-pipelined multiformat and multioperand adder trees by running S1. As expected, the delay of the non-pipelined trees with olDFA$_{Mformat}$s by SpecD is less (around 4%-10%) than that of the delay of the trees with olDFA$_{Mformat}$s by CodeConv. Similarly, the results obtained regarding the area are the expected ones, i.e., the area of the trees with olDFA$_{Mformat}$s by SpecD is greater (around 15%-23%) than the area of the trees with olDFA$_{Mformat}$s by CodeConv. This is due to the number of equations implemented in the decomposition module in the olDFA$_{Mformat}$s by SpecD.

Fig. 21 and Fig. 22 show the results obtained by running

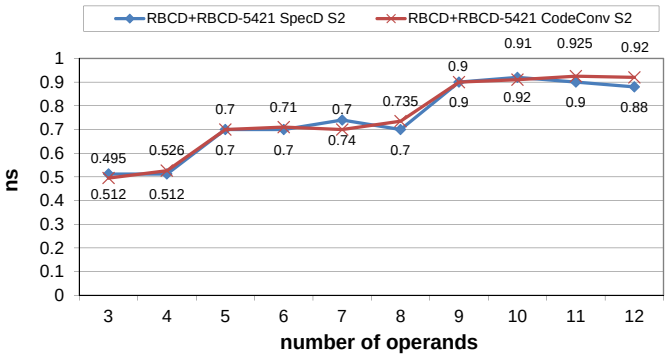
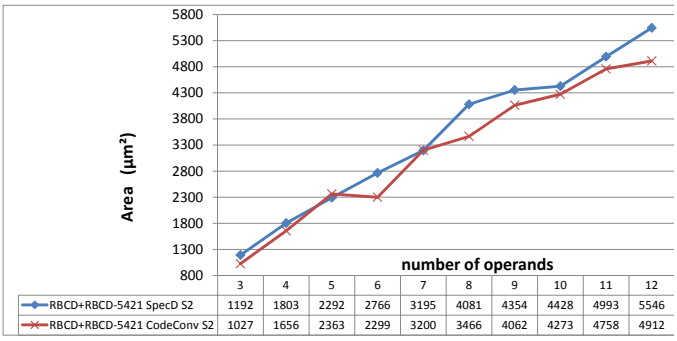


Fig. 21. Delay of RBCD+RBCD$_{-5421}$-based adder trees by SpecD and CodeConv under simulation scenario 2 (NON-PIPELINED VERSION)



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD S2	1192	1803	2292	2766	3195	4081	4354	4428	4993	5546
RBCD+RBCD-5421 CodeConv S2	1027	1656	2363	2299	3200	3466	4062	4273	4758	4912

Fig. 22. Area of RBCD+RBCD$_{-5421}$-based adder trees by SpecD and CodeConv under simulation scenario 2 (NON-PIPELINED VERSION)

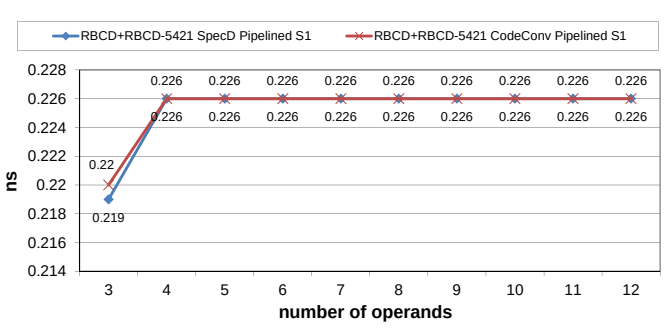
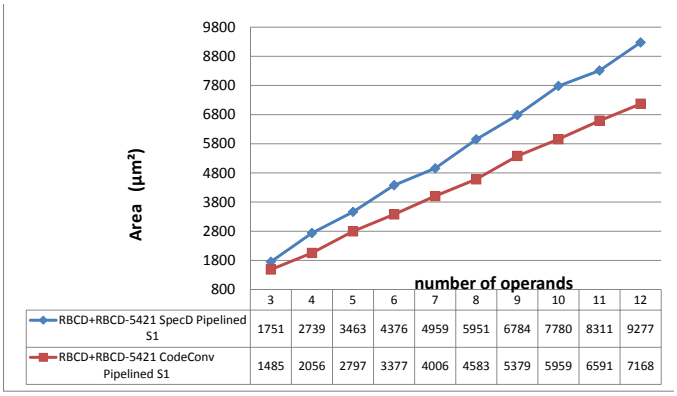


Fig. 23. Delay of RBCD+RBCD$_{-5421}$-based pipelined adder trees by SpecD and CodeConv under simulation scenario 1 (PIPELINED VERSION)



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD Pipelined S1	1751	2739	3463	4376	4959	5951	6784	7780	8311	9277
RBCD+RBCD-5421 CodeConv Pipelined S1	1485	2056	2797	3377	4006	4583	5379	5959	6591	7168

Fig. 24. Area of RBCD+RBCD$_{-5421}$-based pipelined adder trees by SpecD and CodeConv under simulation scenario 1 (PIPELINED VERSION)

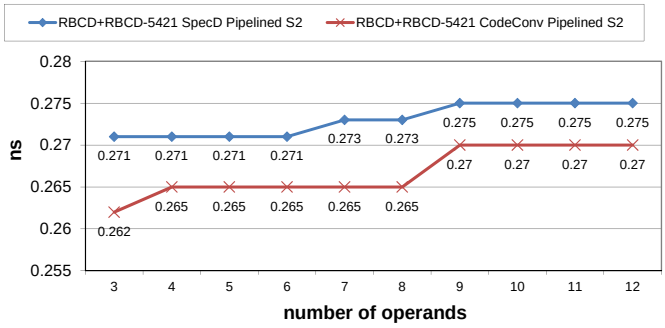


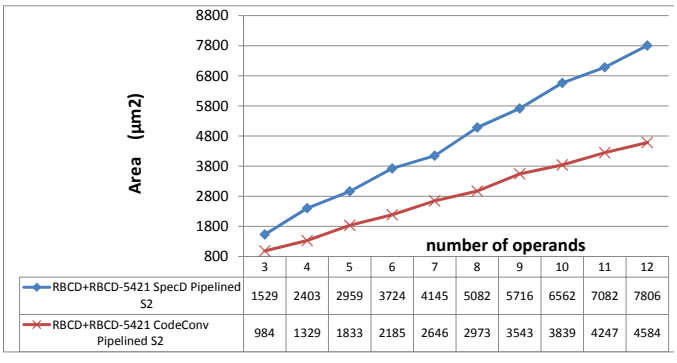
Fig. 25. Delay of RBCD+RBCD$_{-5421}$-based pipelined adder trees by SpecD and CodeConv under simulation scenario 2 (PIPELINED VERSION)

S2 for the non-pipelined architectures. The delay and area are practically the same for both the SpecD and CodeConv schemes. In this case, the tool not only obtains a balance between area and delay, but also reduces the delay and area compared to the data obtained under S1 (see subsection VI-C).

Fig. 23, Fig. 24, Fig. 25, and Fig. 26 show the delay and area of the pipelined adder trees by setting S1 and S2, respectively. Under S1, the delay of the pipelined trees is equal for both SpecD and CodeConv schemes (Fig. 23). The reason for this is that the critical path is located in the same place: the decomposition module of the olDFA$_p$s placed from the second level of the trees onwards.

In fact, the delay is practically the same as that of the olDFA$_p$ under S1 (see Table VII). Note out that as the online delay of the olDFA$_{Mformat}$s by SpecD ($\delta = 3$) is one less than that of the olDFA$_{Mformat}$s by CodeConv ($\delta = 4$), it is obvious that the performance in terms of throughput is better in the case of the pipelined adder trees by SpecD. The area of the pipelined RBCD+RBCD$_{-5421}$ adder trees by SpecD is around 17%-33% greater than that of the pipelined trees by CodeConv (Fig. 24) due to the extra area of the olDFA$_{Mformat}$s decomposition modules.

Under S2, the difference in delay in both schemes is around 2%-3% (see Fig. 25), whereas the difference in area is around



	3	4	5	6	7	8	9	10	11	12
RBCD+RBCD-5421 SpecD Pipelined S2	1529	2403	2959	3724	4145	5082	5716	6562	7082	7806
RBCD+RBCD-5421 CodeConv Pipelined S2	984	1329	1833	2185	2646	2973	3543	3839	4247	4584

Fig. 26. Area of RBCD+RBCD$_{-5421}$-based pipelined adder trees by SpecD and CodeConv under simulation scenario 2 (PIPELINED VERSION)

55%-80% (see Fig. 26).

C. Comparison between Scenario 1 and Scenario 2 (S1 and S2)

As mentioned, under S1, the tool maintains the structures and the results can be analyzed from a theorical/analytical point of view. On the other hand, under S2, the tool balances area and delay in the architectures. This subsection analyzes the increase/decrease rate (in %) of the delay and area obtained under scenario S2 compared to the results obtained under S1. The operation performed for each parameter is (S1-S2)/S1, which means that a positive value corresponds to a decrease and a negative value corresponds to an increase.

A comparison of the results obtained for olDFA$_{Mformat}$ under both scenarios, depicted on Fig. 27 and Fig. 28, shows that the delay and area obtained under S2 have decreased for non-pipelined architectures. The delay is practically the same for RBCD+RBCD$_{-5421}$ by SpecD, whereas the area has decreased by around 14%. However, for pipelined architectures the delay under S2 has increased and the area has decreased. In particular, the delay of pipelined RBCD+RBCD$_{-5421}$ by CodeConv has increased by 21% and the area has decreased by 36%. On the other hand, the delay of pipelined RBCD+RBCD$_{-5421}$ by SpecD has increased by 29% and the area has decreased by 11%.

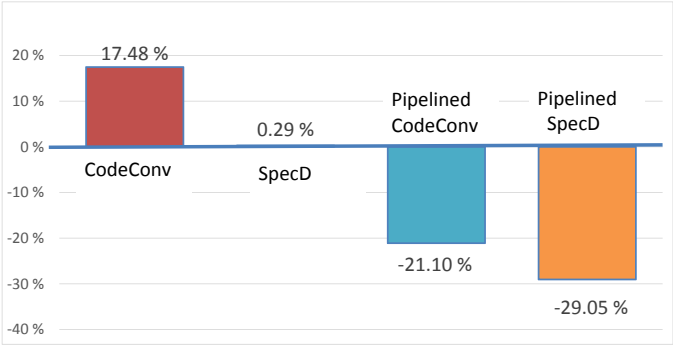


Fig. 27. Increase (decrease) rate for the delay under scenarios S1 and S2 of olDFA$_{Mformat}$s

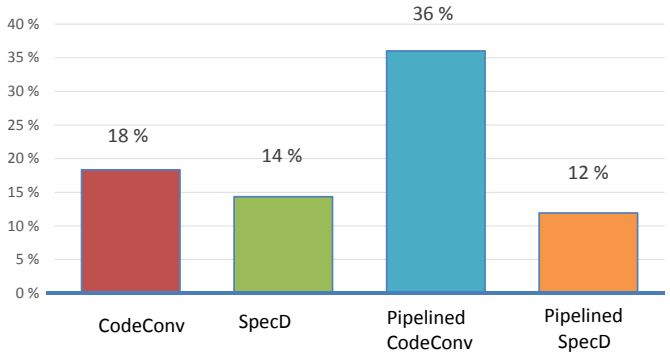


Fig. 28. Increase (decrease) rate for the area under scenarios S1 and S2 of olDFA$_{Mformat}$s

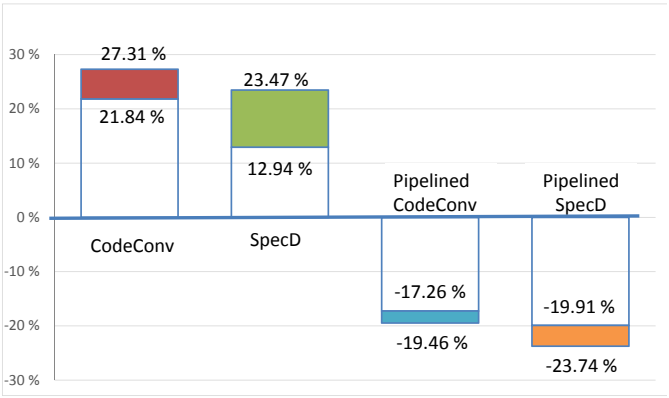


Fig. 29. Range of the increase (decrease) rate for the delay under scenarios S1 and S2 of trees based on olDFA$_{Mformat}$s

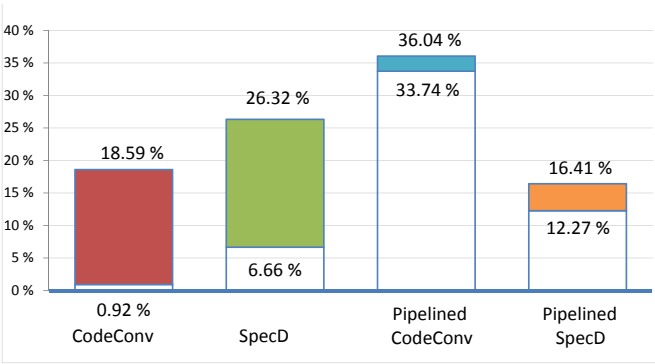


Fig. 30. Range of the increase (decrease) rate for the area under scenarios S1 and S2 of trees based on olDFA$_{Mformat}$s

For non-pipelined trees (see Fig. 29 and Fig. 30), the reduction of the delay obtained under S2 of the olDFA$_{Mformat}$ by SpecD trees (around 13%-23%) is less than the reduction of the delay of the olDFA$_{Mformat}$s by CodeConv trees (around 21%-27%). This is due to the fact that the olDFA$_{Mformat}$ by SpecD has more area than the area by CodeConv. Thus, the balance obtained by the tool is more effective regarding the area of the trees by SpecD (6%-26%) than the area of the trees by CodeConv (1%-18%).

A comparison of the results obtained for pipelined trees under S2 compared to S1 (see Fig. 29 and Fig. 30) shows that the area of the pipelined trees by CodeConv and by SpecD decreases by 33%-36% and 12%-16%, respectively. However, the delay of pipelined trees by CodeConv and by SpecD increases by 17%-19% and 19%-23%, respectively. Thus, the balance obtained under S2 is more effective regarding area and delay for pipelined trees by CodeConv.

Therefore, for non-pipelined architectures, the best scenario is S2 due to the fact that the tool decreases the delay and area compared to S1. However, for pipelined architectures the best scenario for delay is S1 and the best scenario for area is S2.

VII. Summary and Conclusions

It is known that the use of specific representation formats allows optimization of decimal algorithms and tree architectures minimize the computation time when many operands have

to be operated together. In this paper we present a method for designing decimal multiformat and multioperand online adders. First, the issue of the decimal multiformat online adder (olDFA$_{Mformat}$) is addressed in which two different design strategies are presented: the first is based on using an olDFA with a pre-code conversion stage; and the second one is a specific design based on modifying the internal architecture of the olDFA. A comparison of the strategies shows that the first is best when more than two codes are involved at each input of the adder, whereas if only two codes are possible at the inputs the second strategy results in decreased delay.

The decimal multiformat online adder is extended to the multioperand case and a general architecture of the corresponding tree is presented. In these trees, the first level is composed by olDFA$_{Mformat}$ units whereas the remaining levels are built with standard olDFA. Pipelined and non-pipelined architectures are analyzed for all the proposed multiformat adders.

Finally, a detailed study of the main proposed architecture implementations under two different simulation scenarios is presented. Under the first scenario S1, the theoretical result is corroborated by the experimental result. Under the second scenario S2, we let the synthesis tool optimize the design to obtain a balanced optimization between area and delay. On the basis of these experimental results, it can be concluded that for non-pipelined architectures S2 is the best scenario, whereas for pipelined architectures S1 is the best scenario.

In summary, we have presented a guideline for designing decimal multiformat and multioperand online adders. This guideline may help designers to decide what option is best for their application.

REFERENCES

[1] M. Cowlishaw, "Decimal floating-point: algorism for computers," in *Computer Arithmetic, 2003. ARITH 2003. 16th IEEE Symposium on*, 2003, pp. 104–111.

[2] A. Aswal, M. Perumal, and G. Srinivasa Prasanna, "On basic financial decimal operations on binary machines," *Computers, IEEE Transactions on*, vol. 61, no. 8, pp. 1084–1096, 2012.

[3] H. H. Goldstine and A. Goldstine, "The electronic numerical integrator and computer (eniac)," *Annals of the History of Computing, IEEE*, vol. 18, no. 1, pp. 10–16, 1996.

[4] L. Eisen, J. W. Ward, H.-W. Tast, N. Mading, J. Leenstra, S. M. Mueller, C. Jacobi, J. Preiss, E. M. Schwarz, and S. R. Carlough, "IBM POWER6 accelerators: VMX and DFU," *IBM Journal of Research and Development*, vol. 51, no. 6, pp. 1 –21, nov. 2007.

[5] R. Kalla *et al.*, "Power7: IBM's next-generation server," *Processor. IEEE Micro 30*, p. 715, 2010.

[6] A. Y. Duale, M. H. Decker, H.-G. Zipperer, M. Aharoni, and T. J. Bohizic, "Decimal floating-point in z9: An implementation and testing perspective," *IBM Journal of Research and Development*, vol. 51, no. 1/2, 2007.

[7] E. M. Schwarz, J. S. Kapernick, and M. F. Cowlishaw, "Decimal floating-point support on the IBM system z10 processor," *IBM Journal of Research and Development*, vol. 53, no. 1, pp. 4:1 –4:10, january 2009.

[8] T. Yoshida, T. Maruyama, Y. Akizuki, R. Kan, N. Kiyota, K. Ikenishi, S. Itou, T. Watahiki, and H. Okano, "Sparc64 X: Fujitsu's new-generation 16-core processor for unix servers," *Micro, IEEE*, vol. 33, no. 6, pp. 16–24, Nov 2013.

[9] SilMinds. (2015, April) DFP Unit (DFPU). [Online]. Available: http://www.silminds.com/ip-products/dfp-unit

[10] S. Gorgin and G. Jaberipur, "Fully redundant decimal arithmetic," in *Computer Arithmetic, 2009. ARITH 2009. 19th IEEE Symposium on*, June 2009, pp. 145–152.

[11] A. Vazquez, E. Antelo, and P. Montuschi, "A new family of high-performance parallel decimal multipliers," in *Computer Arithmetic, 2007. ARITH 2007. 18th IEEE Symposium on*, June 2007, pp. 195–204.

[12] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2004.

[13] J. Olivares, J. Hormigo, J. Villalba, and I. Benavides, "Minimum sum of absolute differences implementation in a single fpga device," in *Field Programmable Logic and Application, FPL 2004, LNCS*. Springer, 2004, pp. 986–990.

[14] S. Singh, S. hyun Pan, and M. Ercegovac, "Accelerating the photon mapping algorithm and its hardware implementation," in *Application-Specific Systems, Architectures and Processors (ASAP 2011), Proceedings of the 22nd IEEE International Conference on*, sept. 2011, pp. 149 –157.

[15] M. Ercegovac and T. Lang, "On-line arithmetic for DSP applications," in *Circuits and Systems, 1989., Proceedings of the 32nd Midwest Symposium on*, aug 1989, pp. 365 –368 vol.1.

[16] W. Natter and B. Nowrouzian, "Digit-serial online arithmetic for high-speed digital signal processing applications," in *Signals, Systems and Computers (ASILOMAR), 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, vol. 1, nov. 2001, pp. 171 –176 vol.1.

[17] S. Rajagopal and J. Cavallaro, "Truncated online arithmetic with applications to communication systems," *Computers, IEEE Transactions on*, vol. 55, no. 10, pp. 1240 –12 529, oct. 2006.

[18] B. Girau and A. Tisserand, "On-line arithmetic-based reprogrammable hardware implementation of multilayer perceptron back-propagation," in *Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on*, feb 1996, pp. 168 –175.

[19] A. Svoboda, "Decimal adder with signed digit arithmetic," *Computers, IEEE Transactions on*, vol. C-18, no. 3, pp. 212–215, 1969.

[20] J. Moskal, E. Oruklu, and J. Saniie, "Design and synthesis of a carry-free signed-digit decimal adder," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, may 2007, pp. 1089 –1092.

[21] H. Nikmehr, B. Phillips, and C. C. Lim, "A decimal carry-free adder," in *in Proceedings of the SPIE Symposium on Smart Materials, Nano-, and Micro-Smart Systems*, vol. 5649, February 28 2005, pp. 786–797.

[22] B. Shirazi, D. Yun, and C. Zhang, "RBCD: redundant binary coded decimal adder," *Computers and Digital Techniques, IEE Proceedings E*, vol. 136, no. 2, pp. 156 – 160, Mar. 1989.

[23] C. Garcia, S. Gonzalez-Navarro, J. Villalba, and E. Zapata, "On-line decimal adder with RBCD representation," in *Application-Specific Systems, Architectures and Processors (ASAP 2012), Proceedings of the 23rd IEEE International Conference on*, July 2012, pp. 53–60.

[24] J. Moreno, T. Lang, and J. Hormigo, "Radix-2 multioperand and multiformat streaming online addition," *Computers, IEEE Transactions on*, vol. 61, no. 6, pp. 790–803, June 2012.

[25] C. Garcia-Vega, S. Gonzalez-Navarro, J. Villalba, and E. Zapata, "Decimal online multioperand addition," in *Signals, Systems and Computers (ASILOMAR), 2012. Conference Record of the Forty Sixth Asilomar Conference on*, 2012, pp. 350–354.

[26] A. Vazquez, J. Villalba-Moreno, E. Antelo, and E. Zapata, "Redundant floating-point decimal cordic algorithm," *Computers, IEEE Transactions on*, vol. 61, no. 11, pp. 1551–1562, Nov 2012.

[27] A. Vazquez, J. Villalba, and E. Antelo, "Computation of decimal transcendental functions using the cordic algorithm," in *Computer Arithmetic, 2009. ARITH 2009. 19th IEEE Symposium on*, June 2009, pp. 179–186.

[28] A. Vazquez, E. Antelo, and P. Montuschi, "Improved design of high-performance parallel decimal multipliers," *Computers, IEEE Transactions on*, vol. 59, no. 5, pp. 679–693, May 2010.

[29] J. Villalba, J. Hormigo, and E. Zapata, "Improving the throughput of on-line addition for data streams," in *Application-Specific Systems, Architectures and Processors (ASAP 2007), Proceedings of the 18th IEEE International Conference on*, July 2007, pp. 272–277.