International Journal of Electronics
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tetn20

FPGA-Specific Decimal Sign-magnitude Addition and Subtraction

Martín Vázquez[ab] & Elías Todorovich[ab]

[a] Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Argentina

[b] Faculty of Engineering, FASTA University, Mar del Plata, Argentina

Accepted author version posted online: 04 Sep 2015.

To appear in the *International Journal of Electronics*
Vol. 00, No. 00, Month 20XX, 1–22

Publisher: Taylor & Francis
Journal: International Journal of Electronics DOI:
10.1080/00207217.2015.1089945

FPGA-Specific Decimal Sign-magnitude Addition and Subtraction

Martín Vázquez[ab*] , Elías Todorovich[ab]

[a] *Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Argentina*;
[b] *Faculty of Engineering, FASTA University, Mar del Plata, Argentina*

(*January 2015*)

The interest in sign-magnitude representation in decimal numbers lies in the IEEE 754-2008 standard, where the significand in floating-point numbers is coded as sign-magnitude. However, software implementations do not met performance constraints in some applications and more development is required in programmable logic, a key technology for hardware acceleration. Thus, in this work two strategies for sign-magnitude decimal adder/subtractors are studied and six new FPGA-specific circuits are derived from these strategies. The first strategy is based on ten's complement adder/subtractors and the second one is based on parallel computation of an unsigned adder and an unsigned subtractor. Four of these alternative circuits are useful for at least one area-time trade-off and specific operand size. For example, the fastest sign-magnitude adder/subtractor for operand sizes of 7 and 16 decimal digits is based on the second proposed strategy with delays of 3.43 and 4.33 ns, respectively; but the fastest circuit for 34-digit operands is one of the three specific implementations based on ten's complement adder/subtractors with a delay of 4.65 ns.

Keywords: Carry-chain; Decimal Arithmetic; IEEE 754-2008; Programmable Logic; Sign-magnitude

1. Introduction

Binary arithmetic is exposed to accuracy problems in commercial, financial, tax, scientific, and engineering applications (Aswal, Perumal, & Srinivasa Prasanna, 2012; Cowlishaw, 2003). In this way, results of arithmetic operations, currency conversion, rounding stages, etc., may not be accurate enough to satisfy legal requirements or may have an impact on bank balance sheets. As decimal arithmetic may solve these problems, IEEE has included decimal number specifications in the 754-2008 Standard for Floating Point Arithmetic (*754-2008 IEEE Standard for Floating-Point Arithmetic*, 2008). However, the performance required by applications with intensive decimal arithmetic may not be met by conventional software-based decimal arithmetic libraries (Cowlishaw, 2003). These libraries are one to two orders of magnitude slower than hardware implementations (Anderson, Tsen, Wang, Compton, & Schulte, 2009; Schulte, Lindberg, & Laxminarain, 2005). For this reason, decimal floating-point arithmetic logic units (ALU) are being implemented in some high-end processors. For example, decimal floating point has been introduced on the IBM System z9 processor, and an enhanced decimal floating-point unit has

*Corresponding author. Email: mvazquez@exa.unicen.edu.ar

been included in the IBM POWER6 and z10 processors (Schwarz, Kapernick, & Cowlishaw, 2009). Nevertheless, few decimal hardware cores have been designed by taking advantage of specific resources available in FPGAs (Field Programmable Gate Array). Programmable logic is one of the main technological options for hardware acceleration. In this way, decimal cores can be used in a high performance computing (HPC) context.

Intel reached different conclusions from those of IBM (Bhat, Crawford, Morin, & Shiv, 2007). They concluded that hardware implementations are only useful if applications spend a large percentage of their time in decimal floating-point computations. This becomes a strong argument in favor of using programmable logic, which can be applied when a certain threshold of decimal computation time is required.

Addition deserves particular attention because it is used as a primitive operation for computing most arithmetic functions. In classical addition algorithms the execution time is proportional to the number n of operand digits. One approach to reduce the computation time involves modifying the classical algorithms in such a way as to minimize the computation time of each digit; the time complexity is still proportional to n, but the proportionality constant can be reduced. This idea is reinforced in FPGA, where fast circuitry is available for carry-chain.

This paper focuses on signed decimal addition and subtraction, in particular, using sign-magnitude (SM) representation. Since the significand in floating-point numbers is coded as SM according to the IEEE 754-2008 standard, the study of circuits for efficient arithmetic operations in SM is relevant. Furthermore, to the best of the authors' knowledge, no previous work has studied the SM decimal addition and subtraction by taking advantage of the FPGA architecture.

The main contribution of this paper is the design and efficient implementation of circuits for adding and subtracting BCD (Binary-Coded Decimal) numbers represented as SM in FPGAs. There are three additional contributions. First, the best design techniques for BCD adders in FPGA are studied in detail. This is done by reviewing and comparing the most recent published results (Bioul, Vazquez, Deschamps, & Sutter, 2010), Vazquez and de Dinechin (2010). Second, a novel circuit for decimal subtraction of unsigned numbers is presented. This circuit takes advantage of a binary subtractor in an original way. Third, the best techniques for ten's complement BCD adders and subtractors in FPGA are studied in detail. This is done by reviewing and comparing the most recent published results (Bioul et al., 2010), Vazquez and de Dinechin (2010). The circuit proposed in Vazquez and de Dinechin (2010) is modified so that it can add and subtract ten's complement BCD numbers.

In what follows, lower case variables denote multiple digit words, lower case variables with subscripts denote digits, and lower case variables with subscripts and indices denote bits. Thus, a_i stands for digit i of operand a, and $a_i[j]$ corresponds to bit j in digit i. Upper case is reserved for Boolean functions of two one-digit variables in order to avoid confusion with BCD digits.

The rest of the paper is organized as follows: Section 2 is a review of the previous research. Sections 3 and 4 study unsigned BCD adders and subtractors, respectively. Section 5 explains ten's complement BCD adder/subtractors. Section 6 presents a study of SM BCD adder/subtractors. Section 7 includes the summary and conclusions of this work.

2. Previous Work

According to Erle (2009), three different approaches have been proposed in the literature for decimal addition: direct decimal addition, binary addition with correction, and binary addition with bias and correction. For example Schmookler and Weinberger (1971) and more recently Bayrakci and Akkas (2007); Juang, Peng, and Kuo (2012); Veeramachaneni, Kirthi Krishna, Avinash, P, and Srinivas (2007) present direct decimal adders based on carry-look ahead algorithms. In addition, Shirazi, Yun, and Zhang (1989); Svoboda (1969) and more recently Han, Chen, Wahid, and Ko (2011); Han and Ko (2013); Yehia, Fahmy, and Hassan (2010) among others, implement this type of adders by using signed-digit representations; this approach needs converters in order to deal with BCD operands and result. The advantage of the binary addition with bias and correction is that it uses optimized binary adders in such a way that the same circuit can handle both decimal and binary additions, for instance, Dorrigiv and Jaberipur (2014); Vazquez and Antelo (2009); Wang and Schulte (2007). Kenney and Schulte (2005) and Gorgin and Jaberipur (2009) explore binary adders with correction, while enabling representations from 1010 to 1111 (overloaded decimal representation) for intermediate results.

Biswas, Hasan, Hasan, Chowdhury, and Babu (2008); Thapliyal, Kotiyal, and Srinivas (2006); Zhou, Li, and Zhang (2013) implement reversible direct decimal adders. The purpose of this approach is to decrease the energy dissipation. In reversible logic circuits there is a one to one mapping of input and output vectors in such a way that input states can be reconstructed from the output states.

Another approach consists in converting BCD to binary, computing binary additions, and converting back to BCD. The drawback is the cost of the converters in terms of area and speed (Benedek, 1977; Iguchi, Sasao, & Matsuura, 2007; Nicoud, 1971).

As decimal fixed-point adders are required in decimal floating-point ALUs, several ideas have been developed in that context. For example, Vazquez and Antelo (2009); Wang, Schulte, Thompson, and Jairam (2009) present adders integrated into IEEE 754-2008 arithmetic cores.

Recently, some decimal adders have been designed for programmable logic devices. Gao, Al-Khalili, and Chabini (2012); Vazquez and de Dinechin (2010) present multi-operand decimal adders based on binary adders with pre- and post-correction stages for 6-input LUT Xilinx devices. A floating point adder/subtractor for BID (Binary Integer Decimal) operands is presented in Farmahini-Farahani, Tsen, and Compton (2009); Gonzalez-Navarro, Tsen, and Schulte (2013), whereas Baesler and Teufel (2009); James, Jacob, and Sasi (2009); Yixiong, Jun, Na, and Jun (2010) have developed decimal multipliers on FPGA using carry-save adders for the partial multi-operand additions and ripple-carry BCD adders for the final result. In Yixiong et al. (2010), FPGA implementations of decimal adders are optimized for 4-input LUTs by making use of carry-chain circuitry.

In Bioul et al. (2010) a circuit that takes advantage of the fastest carry-chain circuitry and 6-input LUTs available in current FPGAs is introduced.

3. Unsigned Base-10 Adders

In Bioul et al. (2010) two algorithms were designed to implement efficient unsigned BCD adders in FPGAs with 6-input LUTs. Both approaches take advantage of

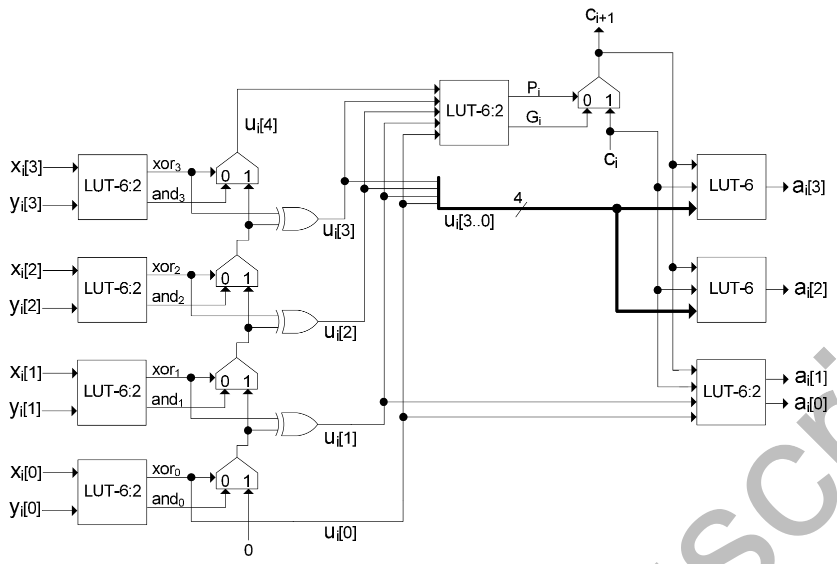

Figure 1. i-th stage implementation of the adder based on the computation of P and G from the binary addition (Add-I)

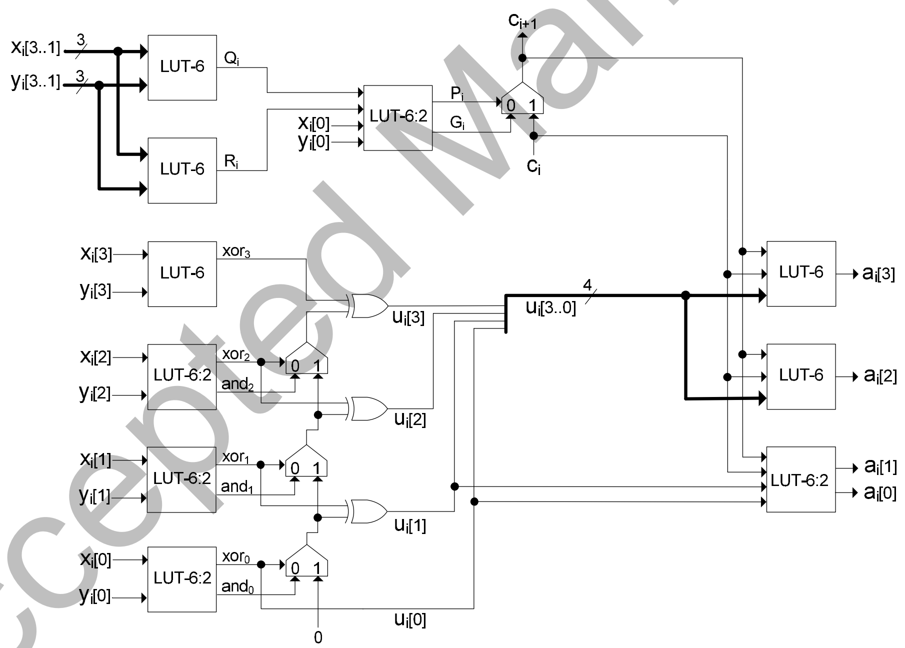

Figure 2. i-th stage implementation of the adder based on the computation of P and G from the input data (Add-II)

the fastest carry-chain circuitry, and are based on the Propagate P and Generate G functions, computed from the intermediate BCD sum (Add-I) and the input data (Add-II), respectively. Figs. 1 and 2 show the Add-I and Add-II i-th stage, respectively. Note that 6-input LUTs can be configured to implement two 5-input functions (LUT 6:2) (Xilinx Inc., 2012).

In both approaches, as the carry-out c_{i+1} is a function of c_i, the circuit delay is proportional to the operands width, n with the same slope. In both cases the

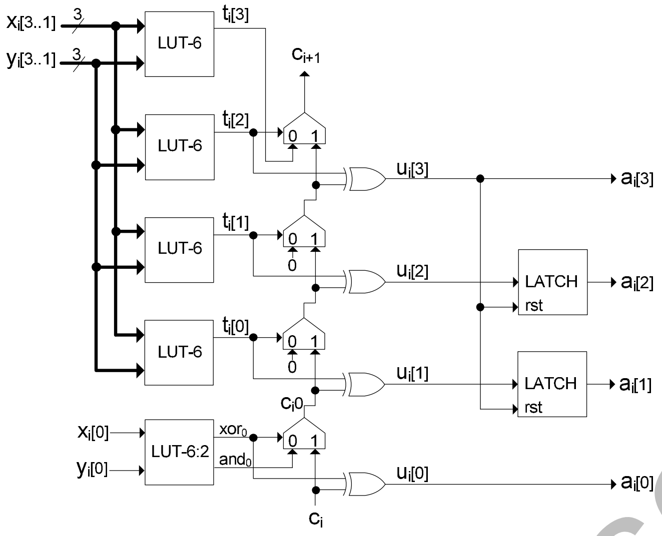

Figure 3. i-th stage implementation of the adder based on Vazquez and de Dinechin (2010)

linear parameter is the delay of the multiplexer embedded in the FPGA carry-chain circuitry, T_{muxcy}. The y-intercept for Add-I in (1), b_{Add-I}, includes the delay of the initial 4-bit addition, the LUT that computes P_i and G_i, the correction stage, and the routing time to connect three slices. On the other hand, b_{Add-II}, in (2), includes the delay of two LUT that computes P_i and G_i, the correction stage, and the routing time to connect three slices.

$$T_{Add-I} = n\, T_{muxcy} + b_{Add-I} \quad (1) \qquad T_{Add-II} = n\, T_{muxcy} + b_{Add-II} \quad (2)$$

Equations (3) and (4) calculate the area consumption in terms of 6-input LUTs. In short, Add-II is slightly faster than Add-I but consumes 25% more area.

$$A_{Ad-I} = 8\, n \quad (3) \qquad A_{Ad-II} = 10\, n \quad (4)$$

In Vazquez and de Dinechin (2010) a third approach, Add-III, is proposed to compute multi-operand decimal addition efficiently using FPGA resources such as the carry-chain and the 6-input LUT. The design in Vazquez and de Dinechin (2010) is oriented towards multiplication and it is based on binary addition with bias and correction. In the bias stage, six is added conditionally. Due to the complexity of the bias stage, the correction is simple and only uses a latch (see Fig. 3).

The circuit delay for a 2-operand adder proposed in Vazquez and de Dinechin (2010), Add-III, is proportional to the operands width, n. Its linear parameter is 4 T_{muxcy}. However, the y-intercept only includes the delay of one LUT, one latch, and the routing time to connect two slices. Equation (6) calculates the area consumption in terms of 6-input LUTs.

Table 1. Delays in ns for decimal adders in Virtex-7 -3

n	Add-I	Add-II	Add-III
7	2.93	2.48	2.11
16	3.10	2.59	2.56
34	3.33	2.82	3.46

Table 2. Area in terms of 6-input LUTs for decimal adders in Virtex-7 -3

n	Add-I	Add-II	Add-III
7	56	70	35
16	128	160	80
34	272	340	170

$$T_{Add-III} = 4\,n\,T_{muxcy} + b_{Add-III} \quad (5) \qquad\qquad A_{Add-III} = 5\,n \quad (6)$$

Add-I and Add-II are significantly bigger than Add-III, 60% and 100%, respectively. However, for a large enough n, these adders will be faster than Add-III. The question is if this n is less than or equal to 34, the largest width defined in the IEEE 754-2008 standard.

3.1. *Experimental Results*

The operand widths in all the tables in this work are those defined in the IEEE 754-2008 standard. For correct operation/rounding, floating point units consider one guard digit, one round digit, and one sticky bit. In some implementations such as the proposed in Wang and Schulte (2007), the fixed point adder is extended to $n+3$ digits to deal with rounding, but in others such as Vazquez and Antelo (2009), the adder is n-bit width and the rounding is computed by a separate module. Anyway, the proposed implementations are parameterizable in the operand width.

The three adders analyzed above were implemented in Xilinx Virtex-7 devices with speed grade 3. The synthesis was done using XST version 14.7 with default parameters.

Table 1 shows that Add-III is faster than Add-II only for $n = 7$. For $n = 16$, Add-III is still a good option because it has almost the same delay and is 50% smaller than Add-II, as shown in Table 2. For $n = 34$, Add-II is the best option in terms of speed; it is 23% better than Add-III but the former doubles the area of the latter. In Vazquez and de Dinechin (2010), Add-III results are even better because that work focuses on multi-operand addition, where redundant intermediate results are allowed and the correction is done at the final addition instead of at each partial addition. However, the Add-III approach, which is the basis for the designs that consume a smaller area, uses the slice flip-flops reconfigured as latches to implement the final correction step. To implement a registered output or pipeline this solution requires an extra level of slices. In the other circuits the FFs in the final level of slices are readily available to implement the output or pipeline register and therefore there is no area increase.

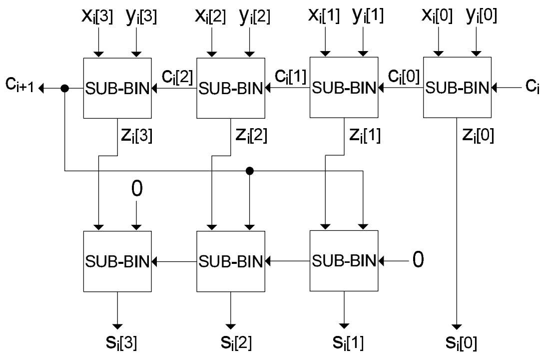

Figure 4. i-th stage of the unsigned decimal subtractor

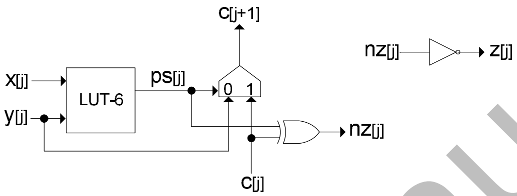

Figure 5. 1-bit subtractor. $z[j] = x[j] - y[j]$

4. Unsigned Base-10 Subtractors

In the subtraction the carry c_{i+1} is interpreted as a borrow of one unit from the next digit. When the minuend is bigger than the subtrahend, the most significant digit in the unsigned subtraction generates an overflow.

Let $s = x - y$ be the result of a BCD unsigned subtraction. The proposed implementation, Sub-U, has a binary subtraction stage followed by a correction. In the i-th digit, the binary subtraction $z_i = x_i - y_i$. If $x_i < y_i$, one unit is borrowed from the next decimal digit, i.e., 16 is added instead of 10, z_i is in [7, 15], and 6 is subtracted in the correction stage (See Fig. 4).

The implementation is based on an efficient design of a 1-bit subtractor $z[j] = x[j] - y[j]$, as shown in Fig. 5, where, $ps[j] = \overline{x[j] \oplus y[j]}$ is one when one borrowed binary unit must be propagated; the intermediate binary subtraction $z[j] = \overline{nz[j]}$.

If the binary subtractor is implemented without the inverter, $z_i = 15 - nz_i$. Then,

$$s_i = \begin{cases} z_i - 6 = 9 - nz_i & \text{if } c_{i+1} = 1, \\ z_i = \overline{nz_i} & \text{otherwise} \end{cases} \tag{7}$$

If $c_{i+1} = 1$, z_i is in [7, 15] and nz_i is in [0, 8]. The subtraction result, s_i, is computed as a function of nz_i and c_{i+1}:

$$\begin{aligned} s_i[0] &= \overline{nz_i[0]} \\ s_i[1] &= \overline{nz_i[1] \oplus c_{i+1}} \\ s_i[2] &= \overline{(nz_i[1] \oplus nz_i[2])} \cdot c_{i+1} \vee \overline{nz_i[2]} \cdot \overline{c_{i+1}} \\ s_i[3] &= \overline{nz_i[1]} \cdot \overline{nz_i[2]} \cdot \overline{nz_i[3]} \cdot c_{i+1} \vee \overline{nz_i[3]} \cdot \overline{c_{i+1}} \end{aligned} \tag{8}$$

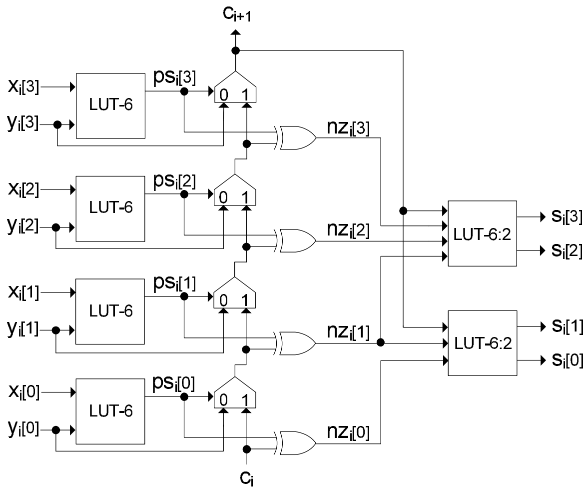

Figure 6. Implementation of the unsigned decimal subtractor i-th stage

Fig. 6 shows the i-th Sub-U digit. Note that $ps_i[j] = \overline{x_i[j] \oplus y_i[j]}$

The delay for an n-digit Sub-U subtractor is proportional to n and its linear parameter is 4 T_{muxcy}. The y-intercept includes the delay of two LUTs and the routing time to connect two slices. Equation (10) calculates the area consumption in terms of 6-input LUTs.

$$T_{Sub-U} = 4\, n\, T_{muxcy} + b_{Sub-U} \quad (9) \qquad\qquad A_{Sub-U} = 6\, n \quad (10)$$

The subtractor explained above only generates unsigned results. However, if results are represented as SM, there is no overflow. In this new circuit with SM results, Sub-SM, the subtraction $ss = x - y$ is done by a correction and a sign generation circuit, and is based on Algorithm 1.

Algorithm 1 SM subtraction

$s \leftarrow x - y$ {Unsigned subtraction (Sub-U)}
if $y > x$ **then**
  $ss \leftarrow 0 - s$ {Unsigned subtraction (Sub-U)}
else
  $ss \leftarrow s$
end if

Either $0 - s$ or $s - 0$ is computed in the Sub-U below in Fig. 7. The proposed binary subtractor can be modified with the multiplexers integrated in the logic. This is done by re-engineering the propagation, *pss*, generation, *gss*, and borrow functions between two consecutive binary subtractors. Both *pss* and *gss* are $4n$-bit vectors:

$$pss[4n-1..0] = (pss_{n-1}, pss_{n-2}, ..., pss_0) \qquad (11)$$

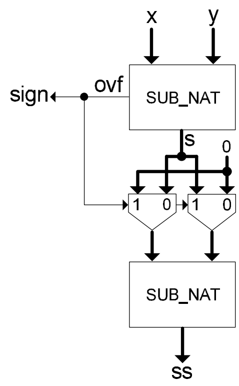

Figure 7. SM decimal subtractor

$$gss[4n-1..0] = (gss_{n-1}, gss_{n-2}, ..., gss_0) \tag{12}$$

where $pss_i = pss[4i+3..4i]$ and $gss_i = gss[4i+3..4i]$ with i in $[0..n-1]$.

A binary subtractor propagates the binary digit borrow when $s[j] = 0$. This happens in the case $0-0$, independently of the overflow in the Sub-U above. When $s[j] = 1$, the case $0-1$ generates a borrow, and the case $1-0$ kills the borrow. In this way, the propagation function in the Sub-U below is simply:

$$pss_i[j] = pss[4i+j] = \overline{s_i[j]}, \forall j \text{ in } [0..3] \tag{13}$$

On the other hand, a binary subtractor generates the binary digit borrow when the subtractor above produces overflow, i.e., $c_n = 1$, only if $s[j] = 1$. In the case that $c_n = 0$, no binary subtractor generates borrow. In this way, the generation function in the Sub-U below is simply:

$$gss_i[j] = gss[4i+j] = s_i[j] \cdot c_n, \forall j \text{ in } [0..3] \tag{14}$$

The borrow, cc_{i+1}, related to the i-th 4-bit binary subtractor, is computed using the carry-chain circuitry available in the FPGA as shown in Fig. 8.

In case $c_n = 1$, $nzz = \overline{0-s}$, otherwise $nzz = \overline{s}$ is computed. In this way, the final result ss requires the correction in (15).

$$ss_i = \begin{cases} 9 - nzz_i & \text{if } cc_{i+1} = 1, \\ \overline{nzz_i} & \text{otherwise} \end{cases} \tag{15}$$

Delay (y-intercept in (16)) and area (slope in (17)) can be reduced if the correction stage of the first subtractor is merged with the binary subtraction stage of the second one, i.e., binary borrow propagate and generate functions in the second subtractor $pss_i[j]$ and $gss_i[j]$, respectively, can be computed as a function of nz_i instead of s_i.

Note that each pair of $pss_i[j]$ and $gss_i[j]$ functions can be implemented with one LUT6:2, as shown in Fig. 8. This figure shows the implementation of the i-th 4-bit

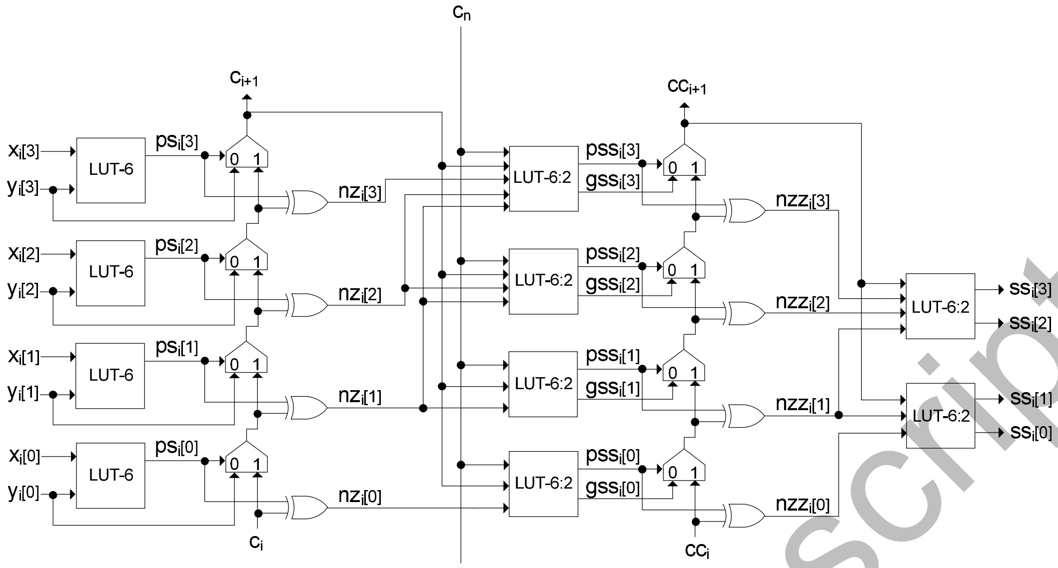

Figure 8. Implementation of the SM decimal subtractor i-th stage

Table 3. Delays in ns for decimal subtractors in Virtex-7 -3

n	Sub-U	Sub-SM
7	2.02	3.43
16	2.46	4.33
34	3.36	6.14

Table 4. Area in terms of 6-input LUTs for decimal subtractors in Virtex-7 -3

n	Sub-U	Sub-SM
7	42	70
16	96	160
34	204	340

binary subtractor where the result has SM representation.

The delay for an n-digit Sub-SM subtractor is proportional to n and its slope is $8\ T_{muxcy}$. The y-intercept includes the delay of three LUTs and the routing time to connect three slices. Experimental results are shown in Table 3. Equation (17) calculates the area consumption in terms of 6-input LUTs. Experimental results are shown in Table 4.

$$T_{Sub-SM} = 8\,n\,T_{muxcy} + b_{Sub-SM} \quad (16) \qquad\qquad A_{Sub-SM} = 10\,n \quad (17)$$

5. Decimal Ten's Complement Adder/Subtractors

Two algorithms were also designed in Bioul et al. (2010) to implement efficient ten's complement (C10) BCD adder/subtractors in FPGAs with 6-input LUTs. Both approaches are based on the unsigned BCD adders proposed in that work.

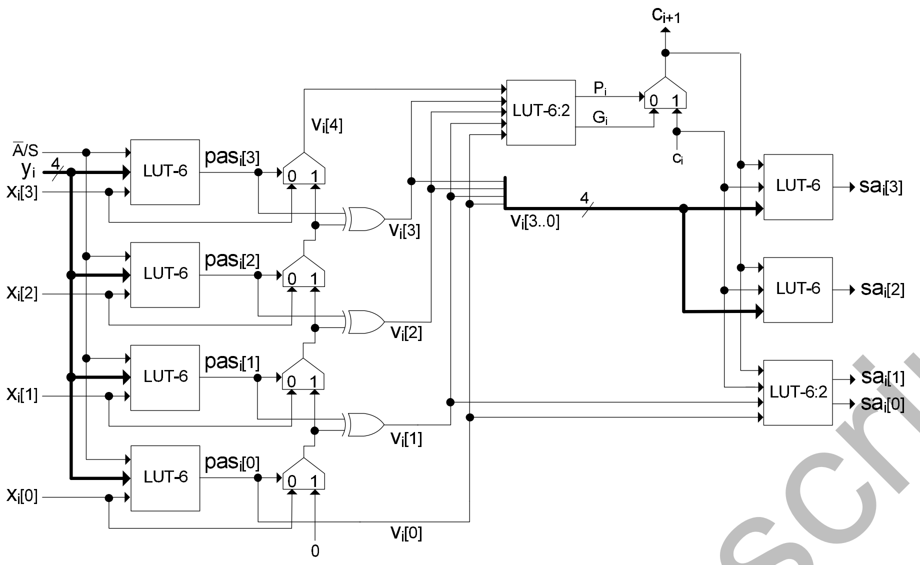

Figure 9. i-th stage implementation of the C10 adder/subtractor based on the computation of P and G from the binary addition (C10-I)

The subtraction, $x - y$ is computed as $x + (-y)$ where $-y$ is the ten's complement of y. $\overline{A}/S$ is the signal that selects the operation between x and y.

The first approach is based on Add-I, where the P and G functions are computed from the intermediate BCD sum (C10-I, see Fig. 9). The second approach is based on Add-II, where the P and G functions are computed from the input data (C10-II). Fig. 10 shows the circuit of the C10-II i-th instance.

The delay for an n-digit C10-I adder/subtractor is proportional to n and its slope is T_{muxcy}. The y-intercept includes the delay of the initial 4-bit adder, one LUT for P and G computation, an additional LUT for the correction, and the routing time to connect three slices. Equation (19) calculates the area consumption in terms of 6-input LUTs.

$$T_{C10-I} = n\, T_{muxcy} + b_{C10-I} \qquad (18) \qquad\qquad A_{C10-I} = 8\, n \qquad (19)$$

The delay for an n-digit C10-II adder/subtractor is proportional to n and its slope is T_{muxcy}. The y-intercept includes the delay of two LUTs for P and G computation, a dedicated multiplexer to combine the outputs of LUTs (MUXF7), an additional LUT for the correction, and the routing time to connect three slices. Equation (21) calculates the area consumption in terms of 6-input LUTs.

$$T_{C10-II} = n\, T_{muxcy} + b_{C10-II} \qquad (20) \qquad\qquad A_{C10-II} = 12\, n \qquad (21)$$

A third approach is introduced in this work based on the decimal adder proposed in Vazquez and de Dinechin (2010) (Add-III). As there is no room for the $\overline{A}/S$ input in Add-III LUTs, an additional stage is required implementing:

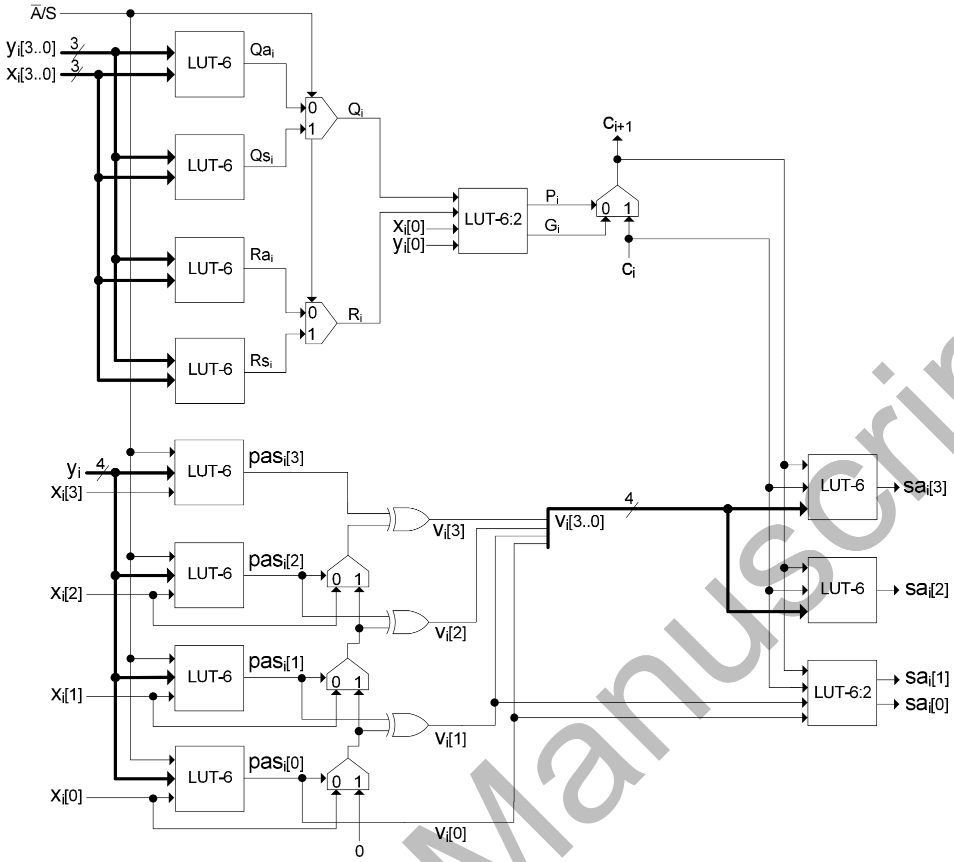

Figure 10. i-th stage implementation of the C10 adder/subtractor based on the computation of P and G from the input data (C10-II)

$$q_i = \begin{cases} C9(y_i) & \text{if } \overline{A}/S = 1, \\ y_i & \text{otherwise} \end{cases} \tag{22}$$

where $C9(y_i)$ is computed by

$$\begin{aligned} C9_i[0] &= \overline{y_i[0]} \\ C9_i[1] &= y_i[1] \\ C9_i[2] &= \overline{y_i[2] \oplus y_i[1]} \\ C9_i[3] &= \overline{y_i[3]} \cdot \overline{y_i[2]} \cdot \overline{y_i[1]} \end{aligned} \tag{23}$$

The i-th instance of this C10-III adder/subtractor is shown in Fig. 11. The delay for an n-digit C10-III adder/subtractor is proportional to n and its slope is 4 T_{muxcy}. The y-intercept includes the delay of two LUTs, a latch, and the routing time to connect three slices. Equation (25) calculates the area consumption in terms of 6-input LUTs.

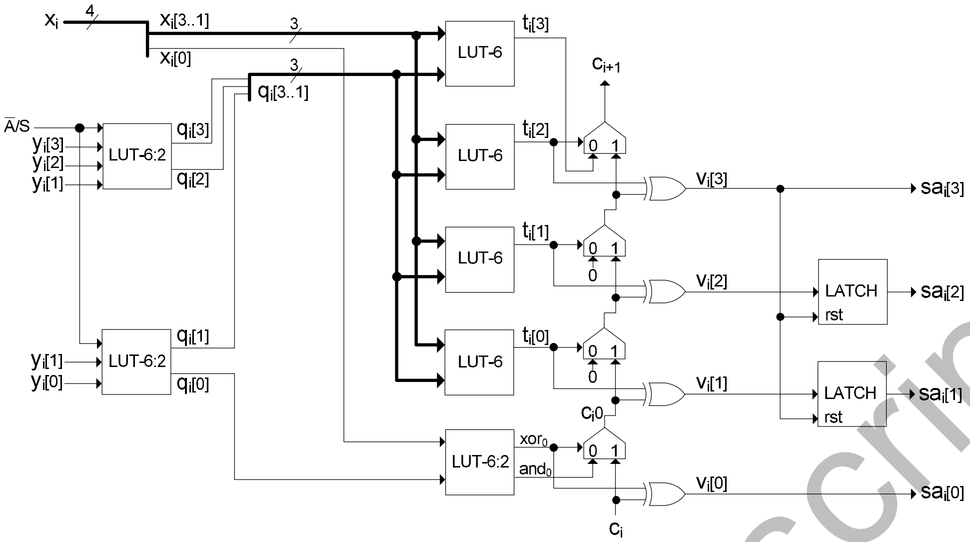

Figure 11. i-th stage implementation of the C10 adder/subtractor based on Vazquez and de Dinechin (2010) (C10-III)

Table 5. Delays in ns for ten's complement decimal adder/subtractors in Virtex-7 -3

n	C10-I	C10-II	C10-III
7	3.09	2.67	2.77
16	3.21	2.78	3.26
34	3.44	3.01	4.16

Table 6. Area in terms of 6-input LUTs for ten's complement decimal adder/subtractors in Virtex-7 -3

n	C10-I	C10-II	C10-III
7	56	84	49
16	128	192	112
34	272	408	238

$$T_{C10-III} = 4\,n\,T_{muxcy} + b_{C10-III} \quad (24)$$

$$A_{C10-II} = 7\,n \quad (25)$$

5.1. *Experimental Results*

As shown in Table 6, C10-III is the best option in terms of area. C10-I has a penalty of 14% and C10-II 71% with respect to C10-III. C10-I is relatively better than Add-I in this context because both circuits require the same area while C10-II and C10-III need $2n$ additional LUTs over Add-II and Add-III, respectively.

As shown in Table 5, although C10-II is the worst circuit in terms of area, it is the best in terms of speed. The y-intercept for C10-III is more relevant in this context than for the decimal adders because its logic depth is increased by one LUT plus the corresponding routing delay. In this way, C10-II is up to 38% faster than C10-III (n=34), instead of 23% in the case of Add-II with respect to Add-III.

For $n = 34$, C10-I is interesting for some trade-offs. It is in all cases 50% smaller

Table 7. Effective operation between SM operands and result sign

sx	sy	op	ope	sr
+	+	+	$\|X\|+\|Y\|$	+
+	+	-	$\|X\|-\|Y\|$	**if** $\|X\| \geq \|Y\|$ **then** $sr=0$ **else** $sr=1$
+	-	+	$\|X\|-\|Y\|$	**if** $\|X\| \geq \|Y\|$ **then** $sr=0$ **else** $sr=1$
+	-	-	$\|X\|+\|Y\|$	+
-	+	+	$\|X\|-\|Y\|$	**if** $\|X\| \geq \|Y\|$ **then** $sr=1$ **else** $sr=0$
-	+	-	$\|X\|+\|Y\|$	-
-	-	+	$\|X\|+\|Y\|$	-
-	-	-	$\|X\|-\|Y\|$	**if** $\|X\| \geq \|Y\|$ **then** $sr=1$ **else** $sr=0$

than C10-II; and 21% faster than C10-III for $n = 34$.

6. Decimal SM Adder/Subtractors

Let xx and yy be the SM representations of the integer numbers X and Y, respectively. The operation, op, between xx and yy is 1 for subtraction and 0 for addition, but the effective operation, ope, is computed as explained in Table 7 and (26) and (27) considering the sign of the operands, where sx, sy, and sr are the sign of the operands xx and yy, and result, respectively.

$$ope = sx \oplus sy \oplus ope \tag{26}$$

$$sr = \overline{ope} \cdot sx \vee ope \cdot sx \cdot (|X| \leq |Y|) \vee \overline{sx} \cdot (|X| < |Y|) \tag{27}$$

Two main strategies are developed in this section. The first one is based on the ten's complement adder/subtractors presented in Section 5. The second one is based on the natural adders presented in Section 3 and the natural subtractor with SM results introduced in Section 4.

6.1. *Based on Ten's Complement Adder/subtractors*

A direct approach to develop n-digit decimal SM adder/subtractors is using C10 adder/subtractors regardless of the overflow. When the effective operation between the absolute values is subtraction, the C10 adder/subtractor computes:

$$(|X| + (10^n - |Y|)) \bmod 10^n \tag{28}$$

where two cases are identified: i) $|X| - |Y| \geq 0$, and ii) $|X| - |Y| < 0$. In case i), operation in (28) generates carry, and as
$(|X| + (10^n - |Y|)) \bmod 10^n = |X| - |Y|$,
this is the final result. On the other hand, case ii) does not generate carry because the result is less than 10^n, and as
$(|X| + (10^n - |Y|)) \bmod 10^n = 10^n + |X| - |Y|$,

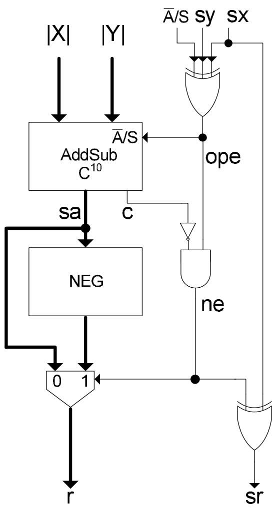

Figure 12. Decimal SM adder/subtractor based on C10 adder/subtractors

the result is ten's complemented, i.e., negative and the final result is its ten's complement:

$10^n - (|X| + (10^n - |Y|)) \bmod 10^n = |X| - |Y|$,

Equation (29) is based on (26) and (27) to compute the result sign, sr, where c is the carry of the C10 adder/subtractor.

$$sr = \overline{ope} \cdot sx \vee ope \cdot sx \cdot c \vee ope \cdot \overline{sx} \cdot \overline{c} = (\overline{ope} \vee c) \cdot sx \vee \overline{sx} \cdot ope \cdot \overline{c} = (ope \cdot \overline{c}) \oplus sx \quad (29)$$

Fig. 12 shows the decimal SM adder/subtractor based on C10 adder/subtractors. The Add_Sub_C10 module is implemented by the circuits in Section 5, i.e., C10-I, C10-II, and C10-III, called SM-C10-I, SM-C10-II, and SM-C10-III, respectively. Module Neg computes the ten's complement. The result of the C10 adder/subtractor is complemented when $ne = \overline{c} \cdot ope$.

Module Neg and the multiplexer can be implemented efficiently by the addition of the nine's complement (C9) of the adder/subtractor output, sa, plus one:

$$r = \begin{cases} C9(sa) + 1 & \text{if } ne = 1, \\ sa & \text{otherwise} \end{cases} \quad (30)$$

Where the input carry of the decimal adder is ne. The propagation function, Pn_i, in this adder is one when $C9(sa_i)$ is nine:

$$Pn_i = \overline{sa_i[3]} \cdot \overline{sa_i[2]} \cdot \overline{sa_i[1]} \cdot \overline{sa_i[0]} \quad (31)$$

The carry, nc_{i+1}, is computed as:

$$nc_{i+1} = Pn_i \cdot nc_i \quad (32)$$

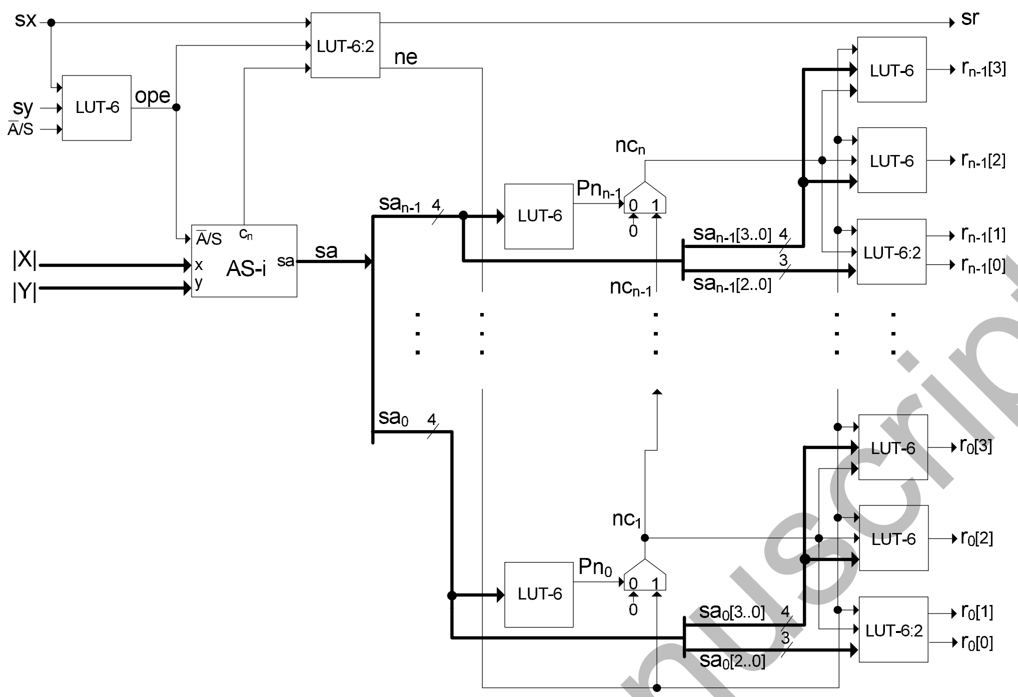

Figure 13. Implementation of the n-digit SM adder/subtractor based on C10 adder/subtractors (SM-C10-I,-II,-III)

where $nc_0 = ne$. Note that when $nc_{i+1} = 1$, then $nc_j = 1$ with j in $[0,\ i]$. This means that when $ne = 0$, all carries are zero.

Each result digit, r_i is:

$$r_i = \begin{cases} sa_i & \text{if } ne = 0, \\ C9(sa_i) + nc_{i+1} & \text{otherwise} \end{cases} \tag{33}$$

Fig. 13 shows the proposed implementation for the SM adder/subtractor based on C10 adder/subtractors with n-digit operands.

The delay for an n-digit SM-C10-I, SM-C10-II, and SM-C10-III adder/subtractor is proportional to n and its slope is 8 T_{muxcy} in the first two cases and 12 T_{muxcy} in the third case ((34), (36), and (38)). The y-intercept in these circuits includes the delay of b_{C10-i}, three LUTs, and the routing time to connect three additional slices. Equations (35), (37), and (39) calculate the area consumption in terms of 6-input LUTs. Each circuit needs the area of the corresponding C10 adder/subtractor plus 3 n LUTs to implement the Neg module and 2 LUTs for *ope*, *ne*, and *sr* computation.

$$T_{SM-C10-I} = 8\ n\ T_{muxcy} + b_{SM-C10-I} \tag{34}$$

$$A_{SM-C10-I} = 12\ n + 2 \tag{35}$$

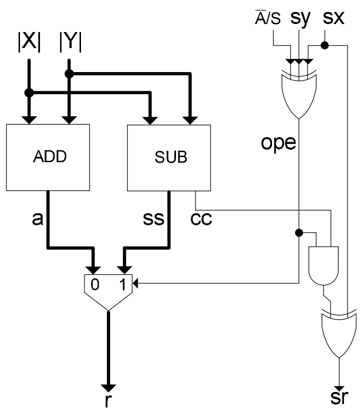

Figure 14. Decimal SM adder/subtractor based on parallel adder and subtractor

$$T_{SM-C10-II} = 8\,n\,T_{muxcy}+b_{SM-C10-II} \quad (36) \qquad A_{SM-C10-II} = 16\,n+2 \quad (37)$$

$$T_{SM-C10-III} = 12\,n\,T_{muxcy}+b_{SM-C10-III} \quad (38) \qquad A_{SM-C10-III} = 11\,n+2 \quad (39)$$

6.2. *Based on Unsigned Decimal Adder and Subtractor*

The second approach proposed in this work to develop n-digit decimal SM adder/subtractors is to compute the absolute-value addition and subtraction in parallel. Then the right result is selected by the effective operation signal, *ope* (see Fig. 14). The addition can be implemented by any of the three circuits in Section 3 and the subtraction is implemented by the circuit with results in SM, proposed in Section 4. When the subtraction result is negative, $cc = 1$. When the effective operation is a subtraction, $ope = 1$.

Result sign computation is based on (26) and (27) considering cc.

$$sr = \overline{ope}\cdot sx \vee ope\cdot sx\cdot \overline{cc} \vee ope\cdot \overline{sx}\cdot cc = (\overline{ope}\vee\overline{cc})\cdot sx \vee ope\cdot cc\cdot \overline{sx} = (ope\cdot cc)\oplus sx \quad (40)$$

In order to optimize area and time, a minor change is necessary for Sub-SM presented in Section 4. Here, the correction stage in Sub-SM is merged with the multiplexer that selects the right result according to the effective operation. Thus, the correction stage of the i-th 1-digit SM adder/subtractor computes:

$$r_i = \begin{cases} a_i & \text{if } ope = 0, \\ 9 - nzz_i & \text{if } cc_{i+1} = 1, \\ \overline{nzz_i} & \text{otherwise} \end{cases} \quad (41)$$

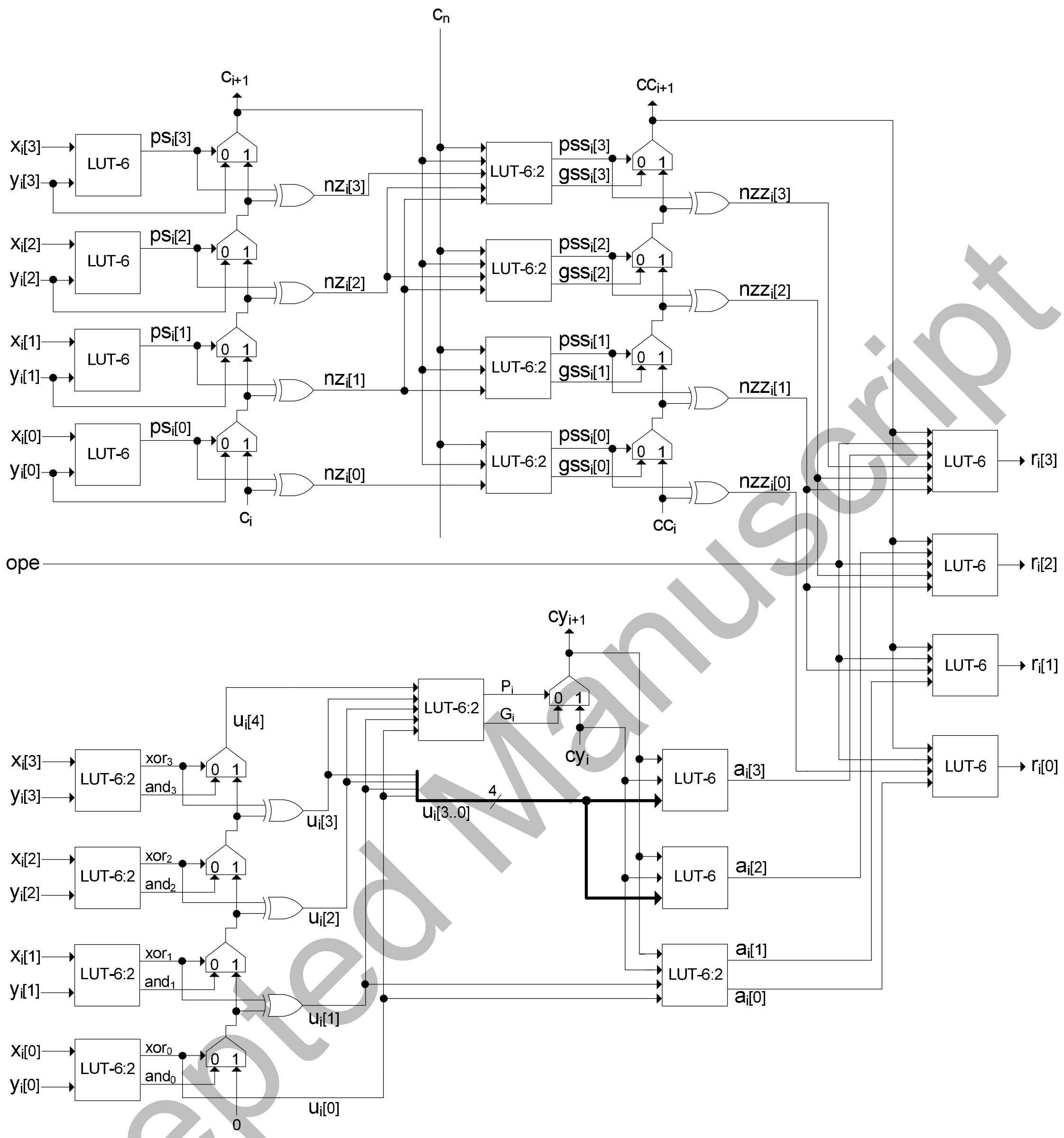

Figure 15. Implementation of the i-th SM adder/subtractor based on unsigned adder and subtractor (SM-U-I). Note that x is $|X|$ and y is $|Y|$

$$
\begin{aligned}
r_i[0] &= a_i[0] \cdot \overline{ope} \vee \overline{nzz_i[0]} \cdot ope \\
r_i[1] &= a_i[1] \cdot \overline{ope} \vee \overline{nzz_i[1] \oplus c_{i+1}} \cdot ope \\
r_i[2] &= a_i[2] \cdot \overline{ope} \vee ((\overline{nzz_i[2] \oplus nzz_i[1]}) \cdot c_{i+1} \vee \overline{nzz_i[2]} \cdot \overline{c_{i+1}}) \cdot ope \\
r_i[3] &= a_i[3] \cdot \overline{ope} \vee (\overline{nzz_i[3]} \cdot \overline{nzz_i[2]} \cdot \overline{nzz_i[1]} \cdot c_{i+1} \vee \overline{nzz_i[3]} \cdot \overline{c_{i+1}}) \cdot ope
\end{aligned}
\tag{42}
$$

In Fig. 15 the circuit is based on the Add-I i-th instance. For the sake of brevity, the circuits based on Add-II and Add-III are not shown. The SM adder/subtractors proposed in this section then are SM-U-I, SM-U-II, and SM-U-III respectively. $c_n = 1$ when there is overflow in the subtractor above (See Fig. 6).

Table 8. Delays in ns for SM decimal adder/subtractors in Virtex-7 -3

n	SM-C10-I	SM-C10-II	SM-C10-III	SM-U-I	SM-U-II	SM-U-III
7	4.88	4.28	4.37	3.44	3.43	3.43
16	5.00	4.40	4.86	4.33	4.33	4.33
34	5.23	4.65	5.77	6.14	6.14	6.14

Table 9. Area in terms of 6-input LUTs for SM decimal adder/subtractors in Virtex-7 -3

n	SM-C10-I	SM-C10-II	SM-C10-III	SM-U-I	SM-U-II	SM-U-III
7	86	114	79	142	156	121
16	194	258	178	322	354	274
34	410	546	376	682	750	580

The delay for these three n-digit adder/subtractors is dominated by the subtractor; therefore, there is only one expression for all of them (43). $b_{SM-U} = b_{Sub-U}$ (see (16)).

$$T_{SM-U} = 8\,n\,T_{muxcy} + b_{SM-U} \tag{43}$$

Equations (44), (45), and (46) calculate the area consumption in terms of 6-input LUTs. Each circuit needs the area of the corresponding unsigned adder plus the subtractor and the multiplexer. This means $2\,n$ additional LUTs to implement the correction and the multiplexer, and 2 LUTs for *ope* and *sr* computation.

$$A_{SM-U-I} = 20\,n + 2 \tag{44}$$

$$A_{SM-U-II} = 22\,n + 2 \tag{45}$$

$$A_{SM-U-III} = 17\,n + 2 \tag{46}$$

6.3. *Experimental Results*

As shown in Table 9, the strategy based on C10 adder/subtractors is better than that based on unsigned adder and subtractor in terms of area. Moreover, SM-C10-III is the best circuit in terms of area. For example, SM-U-III, which is the smallest circuit based on unsigned decimal adder and subtractors, is 54% bigger than SM-C10-III.

On the other hand, as shown in Table 8, for $n = 7$ and $n = 16$, the circuits based on unsigned adder and subtractor are faster than those based on C10 adder/subtractors. However, for $n = 34$ results show otherwise, SM-C10-II being the fastest option, 32% faster than the circuits based on unsigned adder and subtractor.

Note that SM-U-I and SM-U-II, based on the unsigned adders Add-I and Add-II respectively, can be discarded because SM-U-III is better in terms of area and has the same delay.

7. Conclusion

~~In this work several alternative FPGA-specific circuits were proposed for signed and unsigned decimal addition, subtraction, and addition/subtraction. However the~~

~~focus is on SM decimal adder/subtractors, since the significand in floating-point numbers is coded as SM according to the IEEE 754-2008 standard.~~

Two strategies for sign-magnitude decimal adder/subtractors were proposed and six new FPGA-specific circuits are derived from these strategies. The first strategy is based on ten's complement adder/subtractors and the second one is based on parallel computation of an unsigned adder and an unsigned subtractor. Four of the alternative circuits are useful for at least one area-time trade-off and specific operand size. For example, the fastest sign-magnitude adder/subtractor for operand sizes of 7 and 16 decimal digits is based on the second strategy, with delays of 3.43 and 4.33 ns, respectively; but the fastest circuit for 34-digit operands is one of the three specific implementations, where propagation and generation functions are computed from inputs, based on ten's complement adder/subtractors with a delay of 4.65 ns. On the other hand, the smallest circuit is another specific implementation based on ten's complement adder/subtractors, where the decimal addition has an efficient binary adder and pre- and post-correction stages.

All the designs derived from Vazquez and de Dinechin (2010) are smaller than those based on propagation and generation functions, but the relation is opposite in terms of speed. Moreover, although solutions based on P and G functions computed from the input data are the biggest, they are the fastest. The solutions based on P and G functions computed from intermediate BCD sums consume less LUTs than those that compute P and G from input data and more LUTs than the solutions derived from Vazquez and de Dinechin (2010). Additionally, the solutions based on P and G functions computed from intermediate BCD sums are faster than those derived from Vazquez and de Dinechin (2010) for operands of size 34.

Acknowledgments

This work was supported in part by the Agencia Nacional de Promoción Científica y Tecnológica, Argentina, through Project PICT 2009-0041.

References

754-2008 IEEE standard for floating-point arithmetic. (2008, June).

Anderson, M., Tsen, S., Wang, L.-K., Compton, K., & Schulte, M. (2009, Oct). Performance analysis of decimal floating-point libraries and its impact on decimal hardware and software solutions. In *Computer design, 2009. iccd 2009. ieee international conference on* (p. 465-471). doi:

Aswal, A., Perumal, M., & Srinivasa Prasanna, G. (2012, Aug). On basic financial decimal operations on binary machines. *Computers, IEEE Transactions on*, *61*(8), 1084-1096. doi:

Baesler, M., & Teufel, T. (2009, dec). FPGA implementation of a decimal floating-point accurate scalar product unit with a parallel fixed-point multiplier. In *International conference on reconfigurable computing and fpgas* (p. 6 -11). doi:

Bayrakci, A., & Akkas, A. (2007, july). Reduced delay BCD adder. In *Ieee international conf. on application -specific systems, architectures and processors* (p. 266 -271). doi:

Benedek, M. (1977, july). Developing large binary to BCD conversion structures. *IEEE Transactions on Computers*, *C-26*(7), 688 -700. doi:

Bhat, M., Crawford, J., Morin, R., & Shiv, K. (2007, April). Performance characterization of decimal arithmetic in commercial java workloads. In *Performance analysis of systems* *software, 2007. ispass 2007. ieee international symposium on* (p. 54-61). doi:

Bioul, G., Vazquez, M., Deschamps, J. P., & Sutter, G. (2010). High-speed FPGA 10's complement adders-subtractors. *International Journal of Reconfigurable Computing*, *2010*. doi:

Biswas, A., Hasan, M., Hasan, M., Chowdhury, A., & Babu, H. (2008). A novel approach to design BCD adder and carry skip BCD adder. In *VLSI design, 2008. VLSID 2008. 21st international conference on* (p. 566-571). doi:

Cowlishaw, M. F. (2003, June). Decimal floating-point: algorism for computers. In *16th ieee symposium on computer arithmetic* (p. 104-111).

Dorrigiv, M., & Jaberipur, G. (2014). Low area/power decimal addition with carry-select correction and carry-select sum-digits. *Integration, the {VLSI} Journal*, *47*(4), 443 - 451. doi:

Erle, M. A. (2009). *Algorithms and hardware designs for decimal multiplication* (Unpublished doctoral dissertation). Dep. Electrical and Computer Eng, Lehigh University.

Farmahini-Farahani, A., Tsen, C., & Compton, K. (2009, dec). FPGA implementation of a 64-bit BID-based decimal floating-point adder/subtractor. In *International conference on field-programmable technology, fpt* (p. 518 -521). doi:

Gao, S., Al-Khalili, D., & Chabini, N. (2012). An improved BCD adder using 6-LUT FPGAs. In *New circuits and systems conference (newcas), 2012 ieee 10th international* (p. 13-16). doi:

Gonzalez-Navarro, S., Tsen, C., & Schulte, M. (2013). Binary integer decimal-based floating-point multiplication. *Computers, IEEE Transactions on*, *62*(7), 1460-1466. doi:

Gorgin, S., & Jaberipur, G. (2009). A fully redundant decimal adder and its application in parallel decimal multipliers. *Microelectronics Journal*, *40*(10), 1471 - 1481. doi:

Han, L., Chen, D., Wahid, K., & Ko, S.-B. (2011). Nonspeculative decimal signed digit adder. In *Circuits and systems (iscas), 2011 ieee international symposium on* (p. 1053-1056). doi:

Han, L., & Ko, S.-B. (2013). High-speed parallel decimal multiplication with redundant internal encodings. *Computers, IEEE Transactions on*, *62*(5), 956-968. doi:

Iguchi, Y., Sasao, T., & Matsuura, M. (2007, may). On designs of radix converters using arithmetic decompositions–binary to decimal converters–. In *Ieee international symposium on multiple-valued logic* (p. 32 -32). Oslo, Norway. doi:

James, R., Jacob, K., & Sasi, S. (2009, april). Performance analysis of double digit decimal multiplier on various FPGA logic families. In *5th southern conference on programmable logic, spl* (p. 165 -170). doi:

Juang, T.-B., Peng, H.-H., & Kuo, H.-L. (2012). Parallel and digit-serial implementations of area-efficient 3-operand decimal adders. In *Soc design conference (isocc), 2012 international* (p. 239-242). doi:

Kenney, R., & Schulte, M. (2005, aug). High-speed multioperand decimal adders. *IEEE Transactions on Computers*, *54*(8), 953 - 963. doi:

Nicoud, J.-D. (1971, dec). Iterative arrays ror radix conversion. *IEEE Transactions on Computers*, *C-20*(12), 1479 - 1489. doi:

Schmookler, M., & Weinberger, A. (1971, aug). High speed decimal addition. *IEEE Transactions on Computers*, *C-20*(8), 862 - 866. doi:

Schulte, M. J., Lindberg, N., & Laxminarain, A. (2005). Performance evaluation of decimal floating-point arithmetic. In *6th ibm austin center for advanced studies conference.*

Schwarz, E. M., Kapernick, J. S., & Cowlishaw, M. F. (2009, jan.). Decimal floating-point support on the IBM system z10 processor. *IBM Journal of Research and Development*, *53*(1), 4:1-4:10. doi:

Shirazi, B., Yun, D., & Zhang, C. (1989, mar). RBCD: redundant binary coded decimal adder. *Computers and Digital Techniques, IEE Proceedings E*, *136*(2), 156 - 160.

Svoboda, A. (1969, march). Decimal adder with signed digit arithmetic. *IEEE Transactions on Computers*, *C-18*(3), 212 - 215. doi:

Thapliyal, H., Kotiyal, S., & Srinivas, M. (2006). Novel BCD adders and their reversible logic implementation for ieee 754r format. In *VLSI design, 2006. held jointly with 5th international conference on embedded systems and design., 19th international conference on* (p. 6 pp.-). doi:

Vazquez, A., & Antelo, E. (2009, june). A high-performance significand BCD adder with IEEE 754-2008 decimal rounding. In *19th ieee symposium on computer arithmetic* (p. 135 -144). doi:

Vazquez, A., & de Dinechin, F. (2010, October). *Multi-operand Decimal Adder Trees for FPGAs* (Research Report No. RR-7420). INRIA. Retrieved from `http://hal.inria.fr/inria-00526327`

Veeramachaneni, S., Kirthi Krishna, M., Avinash, L., P, S. R., & Srinivas, M. (2007, march). Novel, high-speed 16-digit BCD adders conforming to IEEE 754r format. In *Ieee computer society annual symposium on vlsi* (p. 343 -350). doi:

Wang, L.-K., & Schulte, M. (2007, june). Decimal floating-point adder and multifunction unit with injection-based rounding. In *18th ieee symposium on computer arithmetic* (p. 56 -68). doi:

Wang, L.-K., Schulte, M., Thompson, J., & Jairam, N. (2009, march). Hardware designs for decimal floating-point addition and related operations. *IEEE Transactions on Computers*, *58*(3), 322 -335. doi:

Xilinx Inc. (2012). *Virtex-5 user guide.* http://www.xilinx.com. Retrieved from `http://www.xilinx.com`

Yehia, K., Fahmy, H., & Hassan, M. (2010). A redundant decimal floating-point adder. In *Signals, systems and computers (asilomar), 2010 conference record of the forty fourth asilomar conference on* (p. 1144-1147). doi:

Yixiong, G., Jun, D., Na, L., & Jun, Y. (2010, jan). A research and design of decimal floating multiplier based on FPGA. In *Third international conference on knowledge discovery and data mining* (p. 314 -319). doi:

Zhou, R.-G., Li, Y.-C., & Zhang, M.-Q. (2013). Novel designs for fault tolerant reversible binary coded decimal adders. *International Journal of Electronics*, *0*(0), 1-21. doi: