

Research Article

High-Speed FPGA 10's Complement Adders-Subtractors

G. Bioul,^{1,2} M. Vazquez,² J. P. Deschamps,^{1,3} and G. Sutter⁴

¹ Faculty of System Engineering, FASTA University, Mar del Plata, Argentina

² Faculty of System Engineering, UNCPBA University, Tandil, Argentina

³ School of Engineering, Rovira I Virgili University, Tarragona, Spain

⁴ School of Engineering, Universidad Autónoma de Madrid, Madrid, Spain

Correspondence should be addressed to G. Sutter, gustavo.sutter@uam.es

Received 3 June 2009; Accepted 22 October 2009

Academic Editor: Elías Todorovich

Copyright © 2010 G. Bioul et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper first presents a study on the classical BCD adders from which a carry-chain type adder is redesigned to fit within the Xilinx FPGA's platforms. Some new concepts are presented to compute the P and G functions for carry-chain optimization purposes. Several alternative designs are presented. Then, attention is given to FPGA implementations of add/subtract algorithms for 10's complement BCD numbers. Carry-chain type circuits have been designed on 4-input LUTs (Virtex-4, Spartan-3) and 6-input LUTs (Virtex-5) Xilinx FPGA platforms. All designs are presented with the corresponding time performance and area consumption figures. Results have been compared to straight implementations of a decimal ripple-carry adder and an FPGA 2's complement binary adder-subtractor using the dedicated carry logic, both carried out on the same platform. Better time delays have been registered for decimal numbers within the same range of operands.

1. Introduction

In a number of computer arithmetic applications, decimal systems are preferred to the binary ones. The reasons come not only from the complexity of coding/decoding interfaces but mostly from the lack of precision and clarity in the results of the binary systems.

Decimal arithmetic plays a key role in data processing environments such as commercial, financial, and Internet-based applications [1–3]. Performances required by applications with intensive decimal arithmetic are not met by most of the conventional software-based decimal arithmetic libraries [1]. Hardware implementation embedded in recently commercialized general purpose processors [3, 4] is gaining importance.

Furthermore, *IEEE* has recently published a new standard 754-2008 [5] that supports the floating point representation for decimal numbers.

At the moment, Binary Coded Decimal (BCD) is used for decimal arithmetic algorithm implementations. Although other coding systems may be of interest, BCD seems to be the best choice until now. Issues of hardware realization of decimal arithmetic units appear to be widely open: potential

improvements are expected in what refers to algorithm concepts as well as to hardware design. This paper resumes some new concepts about carry-chain type algorithms for adding BCD numbers. Two key ideas have been introduced: (i) the Propagate P and generate G functions are computed from the input data instead of intermediate BCD sums, and (ii) the functions have been implemented in Xilinx Virtex-4 [6] and Virtex-5 *FPGA* platforms [7], taking advantage of the 6-input LUTs structure of Virtex-5 version.

Signed numbers addition is used as a primitive operation for computing most arithmetic functions, so that it deserves particular attention. It is well known that in classical algorithms the execution time of any program or circuit is proportional to the number N of digits of the operands. In order to minimize the computation time, several ideas have been proposed in the literature [8, 9]. Most of them consist in modifying the classical algorithm in such a way as to minimize the computation time of each carry; the time complexity may still be proportional to N , but the proportionality constant may be reduced. Moreover, it has to be pointed out that, within the same range, decimal addition involves shorter carry propagation process than for the straight binary code. It will be shown in the practical

implementations that adding BCD digits can not only save coding interfaces but moreover provides time delay reductions. Hardware consumption for BCD will be greater, if coding and decoding processes are not considered; as of today, the dramatic decreasing of hardware cost stimulates work on time saving.

In this paper, decimal carry-chain and ripple-carry adders have been implemented on Virtex-4 Xilinx FPGA platforms, for a number of operand sizes; comparative performances are presented for binary and BCD digit operands.

Additionally, three implementations of adders-subtractors have been implemented on FPGA Xilinx Virtex-5 platforms for a number of operand sizes; comparative performances are presented for binary and BCD digit operands, respectively. Adder-subtractor inputs are 10's complement signed BCD numbers; sign-change algorithm is used whenever subtraction is at hand.

2. Base-B Ripple-Carry Adders

Consider the base- B representations of two n -digit numbers:

$$\begin{aligned} x &= x_{n-1} \cdot B^{n-1} + x_{n-2} \cdot B^{n-2} + \dots + x_0 \cdot B^0, \\ y &= y_{n-1} \cdot B^{n-1} + y_{n-2} \cdot B^{n-2} + \dots + y_0 \cdot B^0. \end{aligned} \quad (1)$$

Algorithm 1 (pencil and paper) computes the $(n+1)$ -digit representation of the sum $z = x + y + c_{in}$ where c_{in} is an initial carry equal to 0 or 1.

Algorithm 1. Classic addition (ripple carry):

```

 $c(0) := c_{in};$ 
for  $i$  in  $0 \dots n-1$  loop
  if  $x(i) + y(i) + c(i) > B-1$  then  $c(i+1) := 1;$ 
  else  $c(i+1) := 0;$  end if;
   $z(i) := (x(i) + y(i) + c(i)) \bmod B;$ 
end loop;
 $z(n) := c(n);$ 

```

As $c(i+1)$ is a function of $c(i)$, the execution time of Algorithm 1 is proportional to n (Figure 1). In order to reduce the execution time of each iteration step, Algorithm 1 can be modified as shown in Section 3.

3. Base-B Carry-Chain Adders

First define two binary functions of two B -valued variables, namely, the *propagate* (P) and *generate* (G) functions:

$$\begin{aligned} P(i) &\equiv \begin{cases} P(x(i), y(i)) = 1 & \text{if } x(i) + y(i) = B-1, \\ P(x(i), y(i)) = 0 & \text{otherwise;} \end{cases} \\ G(i) &\equiv \begin{cases} G(x(i), y(i)) = 1 & \text{if } x(i) + y(i) > B-1, \\ G(x(i), y(i)) = 0 & \text{otherwise.} \end{cases} \end{aligned} \quad (2)$$

The next carry c_{i+1} can be calculated as follows:

if $P(i) = 1$ **then** $c(i+1) := c(i);$

else $c(i+1) := G(i);$ **end if;**

The corresponding modified Algorithm 2 is the following one.

Algorithm 2. Carry-chain addition

– computation of the generation and propagation conditions:

```

for  $i$  in  $0 \dots n-1$  loop
   $G(i) := G(x(i), y(i));$ 

```

$P(i) := P(x(i), y(i));$

end loop;

– carry computation:

$c(0) := c_{in};$

```

for  $i$  in  $0 \dots n-1$  loop

```

if $P(i) = 1$ **then** $c(i+1) := c(i);$

else $c(i+1) := G(i);$ **end if;**

(3)

end loop;

– sum computation

```

for  $i$  in  $0 \dots n-1$  loop

```

$z(i) := (x(i) + y(i) + c(i)) \bmod B;$

end loop;

$z(n) := c(n);$

Comments.

(1) Instruction sentence (3) is equivalent to the following Boolean equation:

$$c(i+1) = P(i) \cdot c(i) \vee \text{not } (P(i)) \cdot G(i). \quad (4)$$

Furthermore, if the preceding relation is used, then the definition of the generate function can be modified:

$$G(i) = 1 \quad \text{if } x(i) + y(i) > B-1,$$

$$G(i) = 0 \quad \text{if } x(i) + y(i) < B-1, \quad (5)$$

$$G(i) = 0 \text{ or } 1 \quad (\text{do not care}) \text{ otherwise.}$$

(2) Another Boolean equation equivalent to (4) is

$$c(i+1) = G(i) \vee P(i) \cdot c(i). \quad (6)$$

If the preceding relation is used, then the definition of the propagate function can be modified:

$$P(i) = \begin{cases} 1 & \text{if } x(i) + y(i) = B-1, \\ 0 & \text{if } x(i) + y(i) < B-1, \end{cases} \quad (7)$$

$$P(i) = 0 \text{ or } 1 \quad (\text{do not care}) \text{ otherwise.}$$

The structure of an n -digit adder with separate carry calculation is shown in Figure 2. It is based on Algorithm 2. The G - P (*Generate-Propagate*) cell calculates the *Generate* and *Propagate* functions (2).

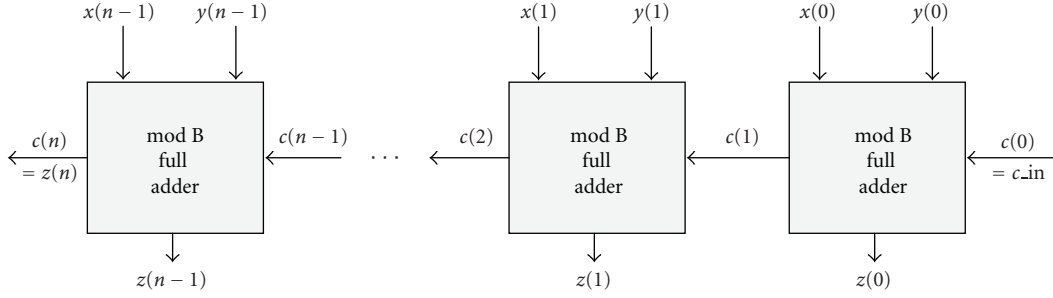


FIGURE 1: Ripple-Carry adder.

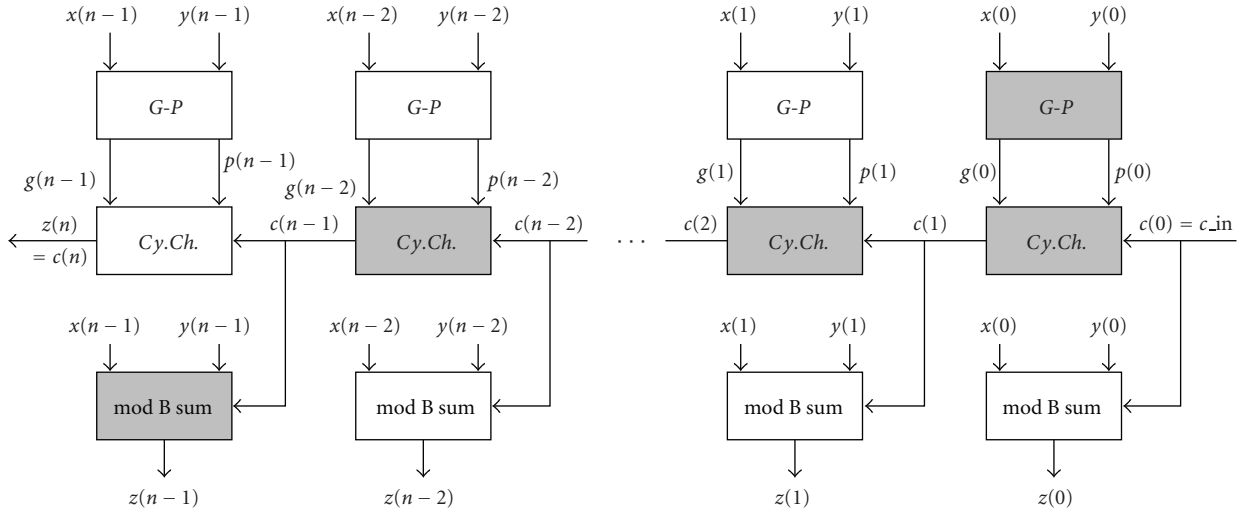


FIGURE 2: Carry-chain adder.

The *Cy.Ch* (*carry-chain*) cell computes the next carry, that is to say

$$c(i+1) = \begin{cases} c(i) & \text{if } P(i) = 1, \\ G(i) & \text{otherwise,} \end{cases} \quad (8)$$

so that $G(i)$ generates a carry, whatever happens upstream in the carry-chain, and $P(i)$ propagates the carry from level $i-1$. The *mod B sum* cell calculates

$$z(i) = (x(i) + y(i) + c(i)) \bmod B. \quad (9)$$

As regards the computation time T , the critical path is shaded in Figure 2. It has been assumed that $T_{\text{sum}} > T_{\text{Cy.Ch}}$.

Another interesting time is the delay $T_{\text{carry}}(n)$ from $c(0)$ to $c(n)$ assuming that all propagate and generate functions have already been calculated:

$$T_{\text{carry}}(n) = n \cdot T_{\text{mux2-1}}. \quad (10)$$

Comments. The carry-chain cells are binary circuits, whereas the *generate-propagate* and the *mod B sum* cells are B -ary ones.

Equation (4) can be implemented by a 2-to-1 binary multiplexer (Figure 3(a)) while (6) by a 2-gate circuit

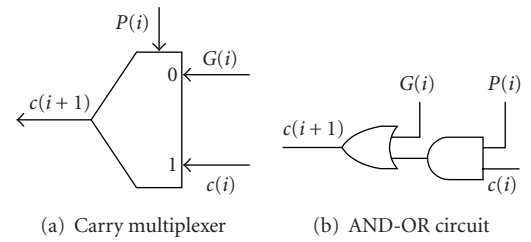


FIGURE 3: Carry-chain cells.

(Figure 3(b)). In the first case, the *per-digit-delay* of a carry-chain adder is equal to the delay $T_{\text{mux2-1}}$ of a 2-to-1 binary multiplexer, whatever the base B is.

If $B = 2$ and the carry-chain cell of Figure 3(a) is used, then $P(i) = x(i) \oplus y(i)$ and $G(i)$ can be chosen equal to, for example, $y(i)$. The corresponding cell for a n -bit binary adder is shown in Figure 4.

4. Base-10 Complement and Addition

4.1. Ten's Complement Numeration System. B 's complement representation general principles are available in the literature as, for example, computer arithmetic books such as

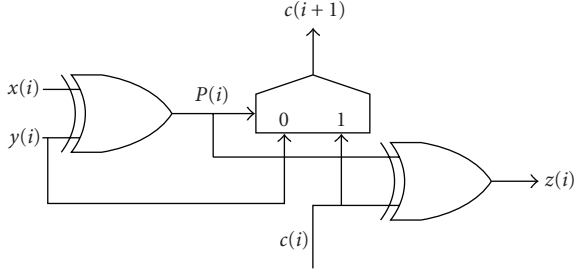


FIGURE 4: Binary adder cell.

[8, 9]. One restricts to 10's complement system to cope with the needs of this paper. A one-to-one function $R(x)$, associating a natural number to x , is defined as follows.

Every integer x belonging to the range

$$-\frac{10^n}{2} \leq x < \frac{10^n}{2} \quad (11)$$

is represented by $R(x) = x \bmod 10^n$, so that the integer represented in the form $x_{n-1} x_{n-2} \dots x_1 x_0$ is

$$x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0$$

$$\text{if } x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0 < \frac{10^n}{2}, \quad (12)$$

$$x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0 - 10^n$$

$$\text{if } x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0 \geq \frac{10^n}{2}.$$

The conditions (12) may be more simply expressed as

$$x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0 \quad \text{if } x_{n-1} < 5,$$

$$x_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0 - 10^n \quad \text{if } x_{n-1} \geq 5. \quad (13)$$

Another way to express a 10's complement number is

$$x'_{n-1} \cdot 10^{n-1} + x_{n-2} \cdot 10^{n-2} + \dots + x_0, \quad (14)$$

where $x'_{n-1} = x_{n-1} - 10$ if $x_{n-1} \geq 5$ and

$$x'_{n-1} = x_{n-1} \quad \text{if } x_{n-1} < 5, \quad (15)$$

while the sign definition rule is the following one: if x is negative, then $x_{n-1} \geq 5$; otherwise $x_{n-1} < 5$.

4.2. Ten's Complement Sign Change. Given an n -digit 10's complement integer x , the inverse $z = -x$ of x is an $(n+1)$ -digit 10's complement integer. Actually the only case that $-x$ cannot be represented with n digits is when $x = -10^n/2$, so $-x = 10^n/2$, that is to say $-x = 0.10^n + (5) \cdot 10^{n-1} + 0.10^{n-2} + \dots + 0.10^0$. The computation of the representation of $-x$ is based on the following property.

Assuming x to be represented as an n -digit 10's complement number $R(x)$, $-x$ may be readily computed as

$$-x = 10^{n+1} - R(x). \quad (16)$$

A straightforward inversion algorithm then consists in representing x with $n+1$ digits, complementing every digit to 9, then adding 1. Observe that sign extension is obtained by adding a digit 0 to the left of a positive number or 9 for a negative number, respectively.

5. Base-10 Adders

5.1. Base-10 Ripple-Carry Adders. For $B = 10$, the classic and naïve approach [8] of ripple-carry for a BCD decimal adder cell can be implemented as in Figure 5. Observe that the critical path involves the carry propagation through 7 binary adders plus a 4-bit Boolean circuit (checking if the sum s is greater than 9 or not).

5.2. Base-10 Carry-Chain Adders. If $B = 10$, the carry-chain circuit remains unchanged but the P and G functions as well as the modulo-10 sums are somewhat more complex. In base 2, the mod B sum cell appears to be a single XOR function, while the mod 10 sum cell is more complex as suggested by Figure 5.

In base 2, the P and G cells are, respectively, synthesized by XOR and AND functions, while in base 10, P and G are now defined as follows:

$$P(i) \equiv \begin{cases} P(x(i), y(i)) = 1 & \text{if } x(i) + y(i) = 9, \\ P(x(i), y(i)) = 0 & \text{otherwise;} \end{cases} \quad (17)$$

$$G(i) \equiv \begin{cases} G(x(i), y(i)) = 1 & \text{if } x(i) + y(i) > 9, \\ G(x(i), y(i)) = 0 & \text{otherwise.} \end{cases}$$

A straightforward way to synthesize P and G is shown at Figure 6. Nevertheless, functions P and G may be directly computed from $x(i)$ and $y(i)$ inputs. The following formulas (18) are Boolean expressions of conditions (17),

$$P(i) = p_0 \cdot [k_1 \cdot (p_3 \cdot k_2 \vee k_3 \cdot g_2) \vee g_1 \cdot k_3 \cdot p_2]$$

$$G(i) = g_0 \cdot [p_3 \vee g_2 \vee p_2 \cdot g_1] \vee g_3 \vee p_3 \cdot p_2 \vee p_3 \cdot p_1 \quad (18)$$

$$\vee g_2 \cdot p_1 \vee g_2 \cdot g_1$$

where $p_j = x_j \oplus y_j$, $g_j = x_j \cdot y_j$, and $k_j = x'_j \cdot y'_j$ are the binary propagator, generator, and carry-kill for the j th components of the BCD digits $x(i)$ and $y(i)$.

The BCD carry-chain adder i th cell is shown at Figure 7. It is made of a first mod 16 adder stage, a carry-chain cell driven by the G - P functions, and an output adder stage performing a correction (adding 6) whenever the carry-out is one. Actually, a zero carry-out $c(i+1)$ identifies that the mod 16 sum does not exceed 9 if $c(i) = 0$, respectively, 8 if $c(i) = 1$; so no corrections are needed. Otherwise, the *add-6* correction applies.

The G - P functions may be computed according to Figure 6, using the outputs of the mod 16 stage, including the carry-out s_4 . With more hardware consumption, but saving time delays, formulas (18) may be used.

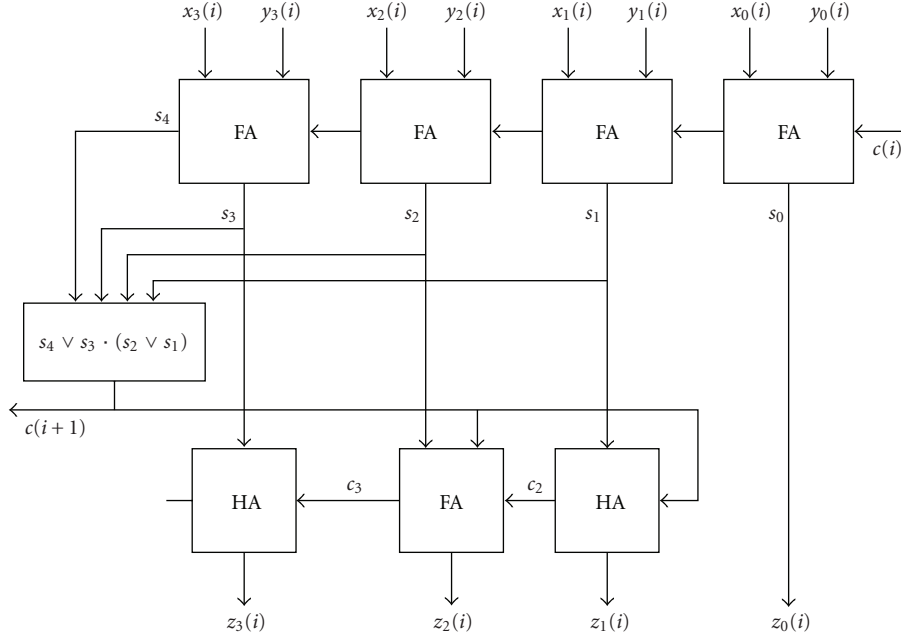


FIGURE 5: Ripple-Carry BCD adder cell.

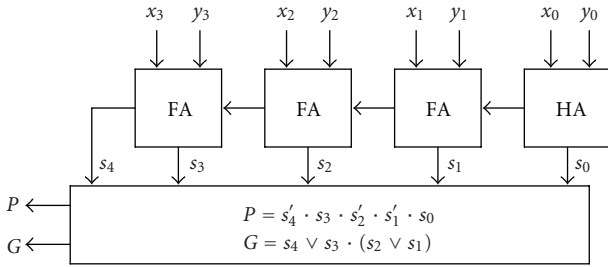
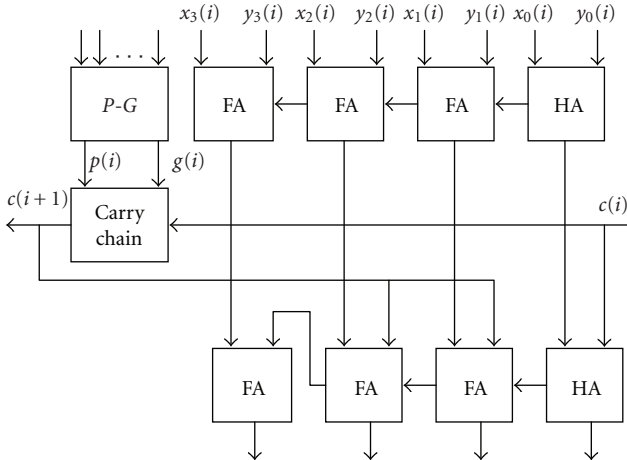


FIGURE 6: G-P cell for BCD adder.

FIGURE 7: Carry-chain BCD adder i th cell.

6. FPGA Implementations of the Base-10 Adders on 4-Input LUTs Xilinx Platforms

The base-10 adders of Figures 5 and 7 have been implemented on 4-LUTs (Look-Up Tables up to 4 inputs) Xilinx

devices. Virtex-4, Spartan 3, and the obsolete Virtex-2, Virtex and Spartan 2 are 4-input LUTs-based FPGA [6, 10]. In what follows the area is expressed in LUTs. In the Xilinx Virtex-4 technology a configurable logic block (CLB) involves 4 *slices* and a *slice* is made by two 4-LUTs and some additional logic. VHDL models are available at [11].

6.1. Base-10 Ripple-Carry Adder. The classic implementation of the ripple carry adder cell in *FPGA* implies a 4-bit adder, a 4-LUT to detect the carry condition, and a final 3-bit adder. The delay and area consumption of an N -digit ripple carry adder are

$$T_{N\text{-digit-B10-rc-adder}} = N \cdot [T_{\text{LUT}} + 4 \cdot T_{\text{mux-cy}} + T_{\text{XOR}} + T_{\text{con}} + T_{\text{LUT}} + T_{\text{con}} + T_{\text{LUT}} + 3 \cdot T_{\text{mux-cy}} + T_{\text{XOR}} + T_{\text{con}}], \quad (19)$$

$$C_{N\text{-digit-B10-rc-adder}} = 8 \cdot N \text{ LUTs.}$$

6.2. FPGA Implementation of the Base-10 Carry-Chain Adder. In order to make the best use of the resources, the design has been achieved using relative location techniques (RLOC) [12] with low-level component instantiations. This first architecture is called *GP_a*.

The adding stages are implemented as shown at Figures 8(a) and 8(b) while the carry-chain structure with the *G-P* functions has been implemented as shown at Figure 9 where *G* is computed according to Figure 6, while *P* is computed as

$$P = s_3 \cdot s_0 \cdot G' \quad (20)$$

equivalent to the expression of Figure 6. Figure 9 emphasizes that *G* depends on s_1, s_2, s_3 , and s_4 while *P* is computed from s_0, s_3 , and *G*.

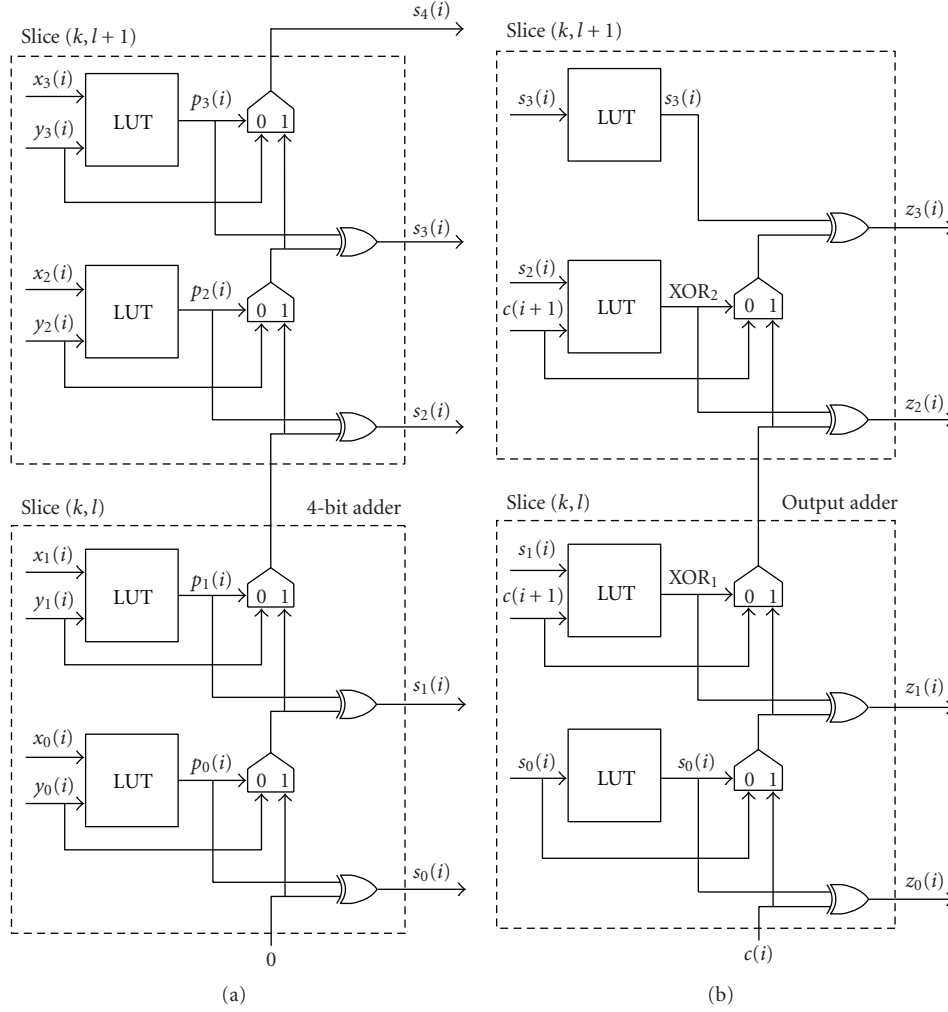


FIGURE 8: FPGA implementation of adders: (a) 4-bit adder stage and (b) output adder stage.

The time delay corresponding to the 4-bit adder stage (Figure 8(a)) and the output adder stage (Figure 8(b)) is given as

$$T_{4\text{-bit adder}} = T_{\text{LUT}} + 4 \cdot T_{\text{mux-cy}}, \quad (21)$$

$$T_{\text{output adder}} = T_{\text{LUT}} + 3 \cdot T_{\text{mux-cy}} + T_{\text{XOR}}. \quad (22)$$

Both adder stages of Figures 8(a) and 8(b) need the same hardware requirement; computed in slices, the area consumption is given as

$$C_{4\text{-bit adder}} = C_{\text{output adder}} = 4 \text{ LUTs}. \quad (23)$$

The complexity figures of the carry-chain circuit for a 4-digit unit, as shown at Figure 9, are given as

$$T_{\text{Cy-Ch-a}} = 2 \cdot T_{\text{LUT}} + T_{\text{con1}} + 4 \cdot T_{\text{mux-cy}}, \quad (24)$$

$$C_{\text{Cy-Ch-a}} = 8 \text{ LUTs}, \quad (25)$$

where T_{con1} stands for the average connection delay between two neighboring *slices* of the same *CLB*.

The overall circuit is represented in Figure 10. The overall time delay is computed from formulas (21), (22) and (24):

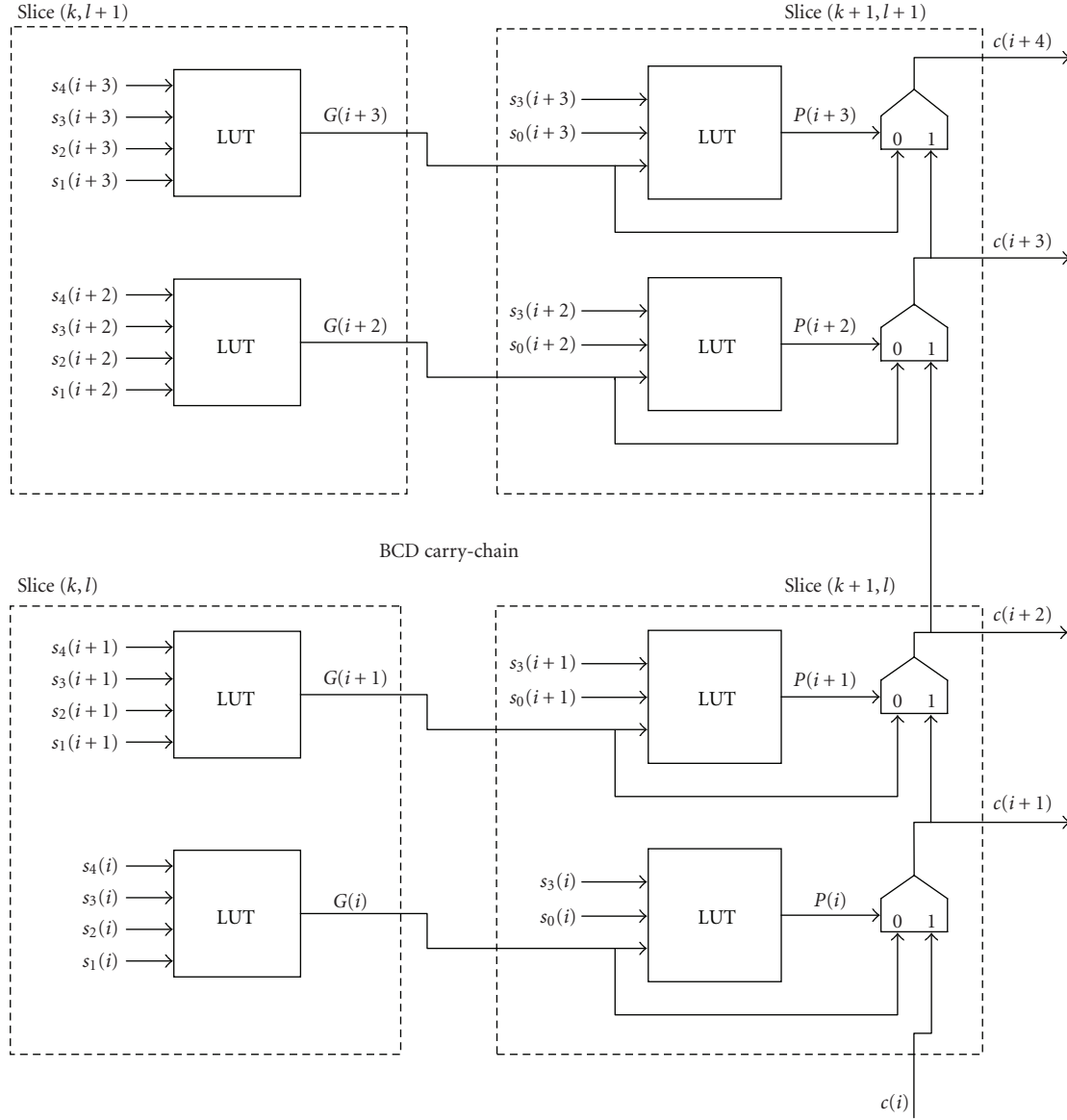
$$T_{N\text{-digit adder-a}} = 4 \cdot T_{\text{LUT}} + (N + 7) \cdot T_{\text{mux-cy}} + T_{\text{XOR}} + T_{\text{con1}} + 2 \cdot T_{\text{con2}}, \quad (26)$$

where T_{con2} stands for the average connection delays between two *slices* located in neighbor columns. T_{con2} has to be accounted twice to involve both the connection delay between the 4-bit adder and the carry-chain and the one between the carry chain and the output adder.

From (23) and (25), the area requirement may be computed as

$$C_{N\text{-digit adder-a}} = 10 \cdot N \text{ LUTs}. \quad (27)$$

6.3. Other Implementations of Base-10 Carry-Chain Adders. Functions P and G may be directly computed from $x(i)$ and

FIGURE 9: FPGA implementation of the carry-chain circuit (*GP_a* architecture).

$\gamma(i)$ inputs using the Boolean expression (18). Using 4-input LUTs (4-LUTs), a first implementation (Figure 11) computes

$$\begin{aligned}
 c &= (x_2 \oplus y_2) \cdot (x'_3 \cdot y'_3), \\
 b &= (x'_2 \cdot y'_2) \cdot (x_3 \oplus y_3) \vee (x_2 \cdot y_2) \cdot (x'_3 \cdot y'_3), \\
 a &= (x'_1 \cdot y'_1) \cdot b \vee (x_1 \cdot y_1) \cdot c, \quad \text{prop} = (x_0 \oplus y_0) \cdot a, \\
 h &= (x_0 \cdot y_0) \cdot (x_1 \cdot y_1), \quad g = (x_0 \cdot y_0) \vee x_1 \vee y_1, \\
 e &= (x_2 \cdot y_2) \cdot g \vee h \cdot (x_2 \oplus y_2), \\
 f &= (x_1 \oplus y_1) \vee (x_2 \oplus y_2), \\
 d &= (x_0 \cdot y_0) \vee f, \quad \text{gen} = (x_3 \cdot y_3) \vee (x_3 \oplus y_3) \cdot d \vee e.
 \end{aligned} \tag{28}$$

This architecture called *GP_b* is shown in Figure 11. The corresponding time and area of a carry-chain cell using this architecture is

$$\begin{aligned}
 T_{\text{Cy-Ch-b}} &= 4 \cdot T_{\text{LUT}} + 3 \cdot T_{\text{con1}} + T_{\text{mux-cy}}, \\
 C_{\text{Cy-Ch-b}} &= 10 \text{ LUTs}.
 \end{aligned} \tag{29}$$

The complete cell includes a 4-bit adder and a conditional 3-bit output adder adding 6 whenever necessary (similar to Figure 5). The overall time delay and area consumption using this carry-computation cell is:

$$\begin{aligned}
 T_{N\text{-digit adder-b}} &= 4 \cdot T_{\text{LUT}} + (N + 3) \cdot T_{\text{mux-cy}} + T_{\text{XOR}} \\
 &\quad + 3 \cdot T_{\text{con1}} + T_{\text{con2}},
 \end{aligned} \tag{30}$$

$$C_{N\text{-digit adder-b}} = 18 \cdot N \text{ LUTs}. \tag{31}$$

The results in area and speed are poor compared to the *GP_a* implementation (obtaining *G-P* from the results of the 4-bit adder).

Another alternative is based on the use of dedicated multiplexers. Xilinx Spartan 3, Virtex-2, and Virtex-4 devices have Look-Up Table multiplexers (muxf5, muxf6, muxf7, muxf8) in order to construct functions of 5, 6, 7, and 8 variables without using the general purpose routing fabric.

Using this feature the circuit of Figure 12 (*GP_c*) can be implemented using the following relations:

$$\begin{aligned}
 b &= y'_3 \cdot y'_2 \cdot y'_1; & d &= x'_2 \cdot x'_1; \\
 c &= y_3 \cdot d \vee y'_3 \cdot e; & a &= x_3 \cdot b \vee x'_3 \cdot c; \\
 p1 &= (x'_0 \vee y'_0) \cdot a, \\
 e &= (x_2 \cdot x_1 \cdot y'_2 \cdot y_1) \vee (x'_2 \cdot x_1 \cdot y_2 \cdot y_1) \vee (x_2 \cdot x'_1 \cdot y_2 \cdot y'_1), \\
 k &= (x_2 \cdot y_2) \vee (x_1 \cdot y_1) \cdot (x_2 \vee y_2); & h &= j = 1; \\
 i &= y_3 \cdot j \vee y'_3 \cdot k; & g1 &= x_3 \cdot h \vee x'_3 \cdot i; \\
 p2 &= (x_0 \vee y_0).
 \end{aligned} \tag{32}$$

The corresponding time and area of a carry-chain cell *GP_c* is

$$\begin{aligned}
 T_{Cy-Ch-c} &= T_{6-LUT} + T_{LUT} + T_{con1} + T_{mux-cy}, \\
 C_{Cy-Ch-c} &= 8 \text{ LUTs},
 \end{aligned} \tag{33}$$

where T_{6-LUT} stands for the delay from an LUT input to a muxf6 output. The complete cell also includes 4-bit adder and a conditional 3-bit adder. The overall delay-area for *GP_c* cell is

$$\begin{aligned}
 T_{N-digit \text{ adder-c}} &= T_{6-LUT} + T_{LUT} + (2 \cdot N + 3) \cdot T_{mux-cy} \\
 &+ T_{XOR} + T_{con1} + T_{con2},
 \end{aligned} \tag{34}$$

$$C_{N-digit \text{ adder-c}} = 16 \cdot N \text{ LUTs.} \tag{35}$$

7. FPGA Implementations of Base-10 Adders and Adders-Subtractors on 6-Input LUTs Xilinx Platforms

7.1. Base-10 BCD Carry-Chain Adder. In a first version, *Ad-I*, the adding stage and correction stage are implemented as shown at Figures 8(a) and 8(b), respectively, while the carry-chain structure with the *G-P* functions is computed according to Figure 6.

Xilinx Virtex-5 6-input/2-output LUT is built as two 5-input functions while the sixth input controls a 2-1 multiplexer allowing to implement either two 5-input functions or a single 6-input one; so *G* and *P* functions fit in a single LUT as shown at Figure 13.

In a second version, *Ad-II*, the carry-chain is speeded up thanks to a direct computation of the *G-P*, namely, using inputs $x(i)$ and $y(i)$, instead of the intermediate sum bits s_k .

For this purpose one could use formulas (18); nevertheless, in order to minimize time and hardware consumption the implementation of $P(i)$ and $G(i)$ is revisited as follows. Remembering that $P(i) = 1$ whenever the arithmetic sum $x(i) + y(i) = 9$, one defines a 6-input function $pp(i)$ set to be 1 whenever the arithmetic sum of the first 3 bits of $x(i)$ and $y(i)$ is 4. Then $P(i)$ may be computed as

$$P(i) = (x_0(i) \oplus y_0(i)) \cdot pp(i). \tag{36}$$

On the other hand, $gg(i)$ is defined as a 6-input function set to be 1 whenever the arithmetic sum of the first 3 bits of $x(i)$ and $y(i)$ is 5 or more. So, remembering that $G(i) = 1$ whenever the arithmetic sum $x(i) + y(i) > 9$, $G(i)$ may be computed as

$$G(i) = gg(i) \vee (pp(i) \cdot x_0(i) \cdot y_0(i)). \tag{37}$$

As Xilinx Virtex-5 LUTs may compute 6-variable functions, then $gg(i)$ and $pp(i)$ may be synthesized using 2 LUTs in parallel while $G(i)$ and $P(i)$ are computed through an additional single LUT as shown at Figure 14.

7.2. 10's Complement BCD Carry-Chain Adder-Subtractor. To compute $X + Y$ similar algorithm as in Section 7.1 is used. In order to compute $X - Y$, 10's complement subtraction algorithm actually adds $(-Y)$ to X .

7.2.1. 10's Complement (AS-I). 10's complement sign change algorithm may be implemented through a digitwise 9's complement stage followed by an add-1 operation. It can be shown that the 9's complement binary components z_3, z_2, z_1 , and z_0 of a given BCD digit y_3, y_2, y_1 , and y_0 are expressed as

$$z_3 = y'_3 \cdot y'_2 \cdot y'_1; \quad z_2 = y_2 \oplus y_1; \quad z_1 = y_1; \quad z_0 = y'_0. \tag{38}$$

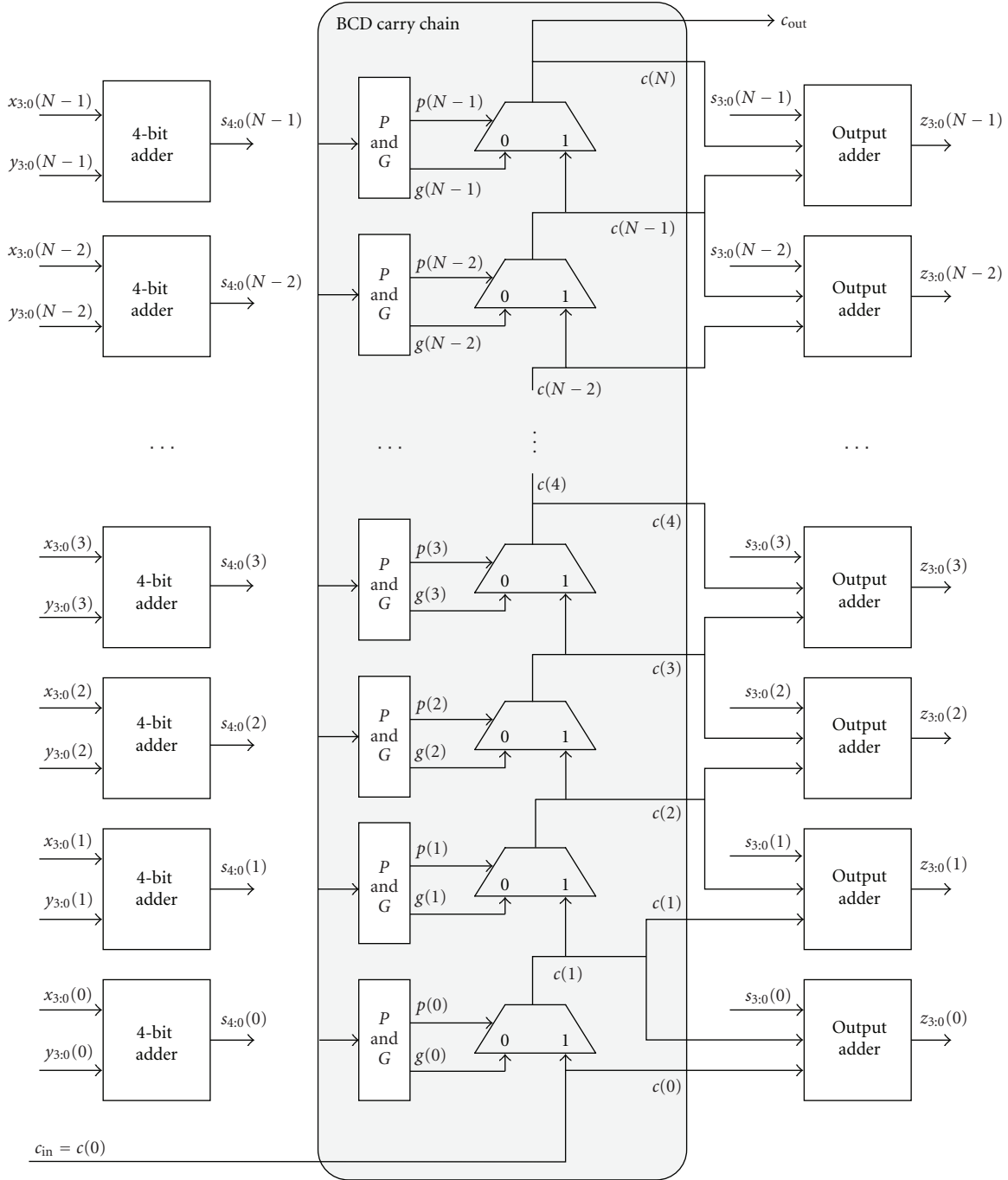
To compute $X - Y$, 10's complement subtraction algorithm actually adds $(-Y)$ to X . So for a first implementation, AS-I, Figure 15 presents a 9's complement implementation using 6-input/2-output LUTs, available in the Virtex-5 Xilinx technology. A'/S is the add/subtract control signal; if $A'/S = 1$ (subtract), formulas in (38) apply; otherwise $A'/S = 0$ and $z_j(i) = y_j(i)$ for all i, j .

The AS-I circuit is similar to the *Ad-I* (Figures 8 and 13) using, instead of input Y , the input Z as produced by the circuit of Figure 15.

7.2.2. Improving the Adder Stage. To avoid the delay produced by the 9's complement step, this operation may be carried out within the first binary adder stage, as depicted in Figure 16, where $p(i)$ and $g(i)$ are computed as

$$p_0(i) = x_0(i) \oplus y_0(i) \oplus (A'/S),$$

$$p_1(i) = x_1(i) \oplus y_1(i),$$

FIGURE 10: FPGA implementation of an N -digit BCD Adder.

$$\begin{aligned}
 p_2(i) &= x_2(i) \oplus y_2(i) \oplus y_1(i) \cdot (A'/S), \\
 p_3(i) &= x_3(i) \oplus (y_3(i)' \cdot y_2(i)' \cdot y_1(i)') \cdot (A'/S) \\
 &\quad \oplus y_3(i) \cdot (A'/S)', \\
 g_k(i) &= x_k(i), \quad \forall k.
 \end{aligned}$$

(39)

7.2.3. *Carry-Chain Stage Computing G and P Directly from the Input Data (AS-II).* As far as addition is concerned, the P and G functions may be implemented according to formulas (36) and (37). The idea of the AS-II is computing the corresponding functions in the subtract mode and then multiplexing according to the add/subtract control signal A'/S . For this reason, assuming that the operation at hand is $X + (\pm Y)$, one defines on one hand $ppa(i)$ and $gga(i)$

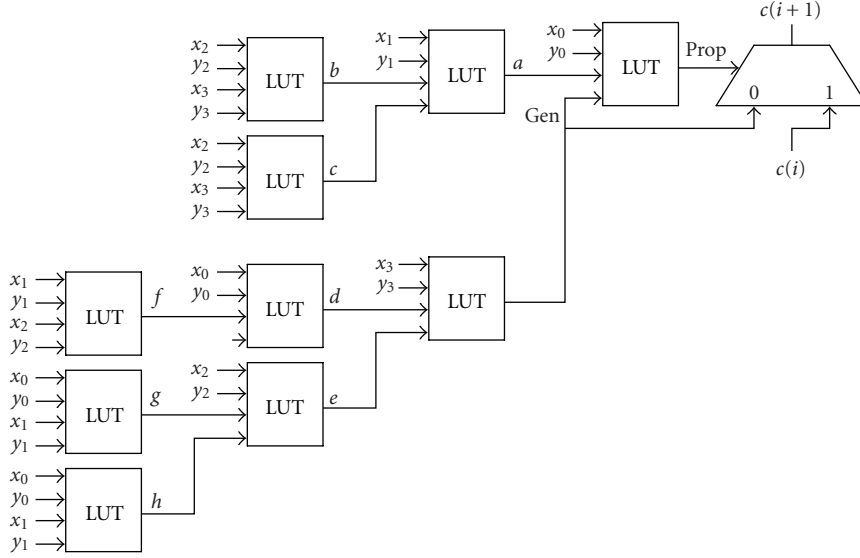


FIGURE 11: Carry generation (GP_b architecture).

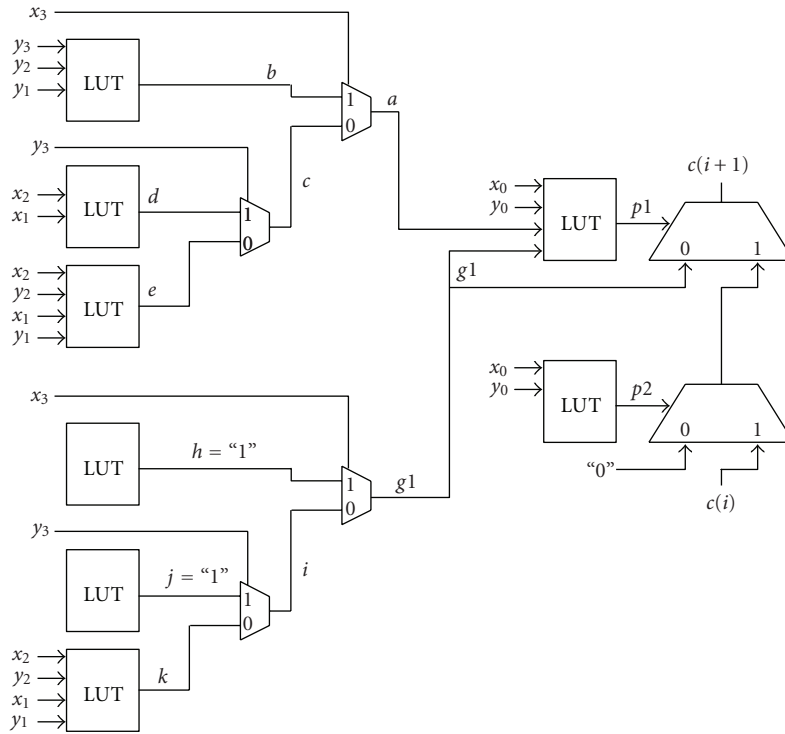


FIGURE 12: Carry generation (GP_c architecture).

according to Section 7.1, that is, using the straight values of Y 's BCD components.

On the other hand, $pps(i)$ and $ggs(i)$ are defined according to the same Section 7.1 but using $z_k(i)$ as computed by the 9's complement circuit shown at Figure 15. As $z_k(i)$ are expressed from the $y_k(i)$ (38), both $pps(i)$ and $ggs(i)$ may be computed directly from $x_k(i)$ and $y_k(i)$ as shown

in Figure 17. Nevertheless, for subtraction, the computation of $z_0(k) = y'_0(k)$ is carried out at the output LUT level. So formulas (36) and (37) are then expressed as

$$\begin{aligned} P(i) &= (x_0(i) \oplus y_0(i) \oplus (A'/S)) \cdot pp(i), \\ G(i) &= gg(i) \vee (pp(i) \cdot x_0(i) \cdot (y_0(i) \oplus (A'/S))). \end{aligned} \quad (40)$$

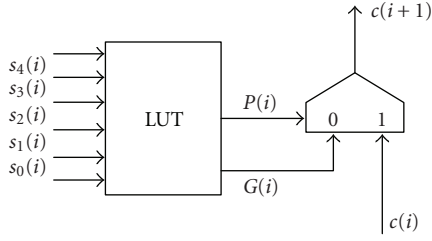


FIGURE 13: FPGA carry-chain circuit for Ad-I.

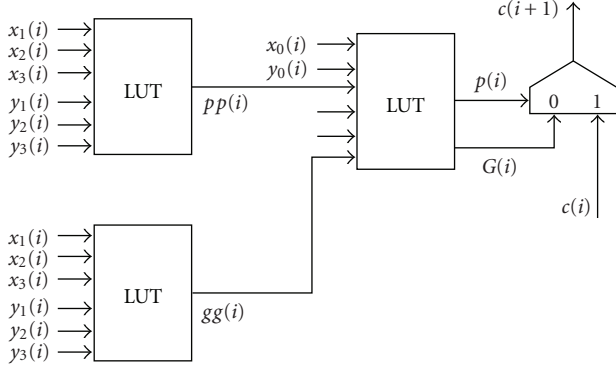


FIGURE 14: FPGA carry-chain circuit for Ad-II.

8. Experimental Results

8.1. Xilinx Virtex-4 Adder Implementations. The base-10 adders have been implemented on Xilinx Virtex-4 FPGA family speed grade-11 [6]. The Synthesis and implementation have been carried out on XST (Xilinx Synthesis Technology) [13] and Xilinx ISE (Integrated System environment) version 10.1 [14].

Performances of different N -digit BCD adders have been compared to those of an M -bit binary carry chain adder (implemented by XST [13] using Xilinx fast carry logic) covering the same range, that is, as

$$M = \lceil N \cdot \log_2(10) \rceil \cong 3.322 \times N. \quad (41)$$

The time and hardware complexities of an M -bit ripple-carry adder implemented on the same 4-LUT based Xilinx FPGA are given by

$$\begin{aligned} T_{M\text{-bit adder}} &= T_{\text{LUT}} + M \cdot T_{\text{mux-cy}}, \\ &= T_{\text{LUT}} + 3.322 \times N \times T_{\text{mux-cy}}. \end{aligned} \quad (42)$$

$$C_{M\text{-bit adder}} = M \text{ LUTs} = 3.322 \times N \text{ LUTs} \quad (43)$$

Formulas (26), (30), (34), and (42) show that, asymptotically, $T_{N\text{-digit adder}}$ should be somewhat inferior to $T_{M\text{-bit adder}}$. Nevertheless, as shown by the experimental results, the additive values appearing in (26), (30), and (34) are not negligible for reasonable values of N ; so the saving in time will mainly appear for applications where BCD-to-binary coding and decoding operations play a significant role in the overall delay.

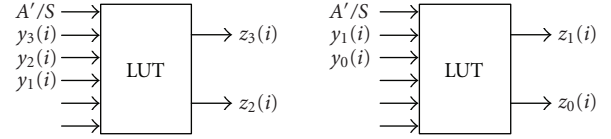


FIGURE 15: FPGA 9's complement circuit for AS-I.

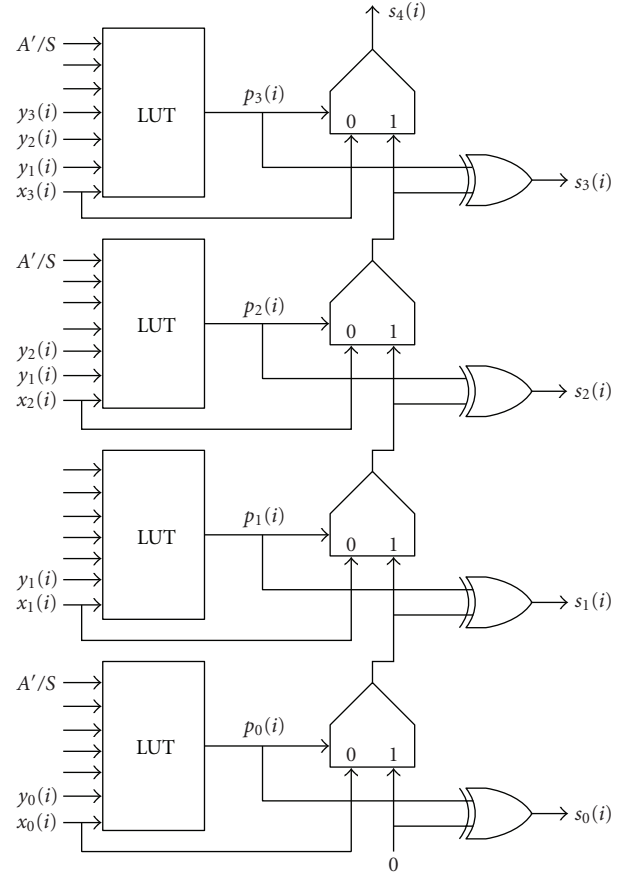


FIGURE 16: FPGA implementation of the adder stage for a 10's complement BCD adder-subtractor.

Post place-and-route time delays and area consumptions are quoted in Tables 1 and 2, respectively, where N stands for the number of BCD digits while M stands for the number of bits required to cover the decimal N -digit range. The results presented in the table are as follows:

- (i) *Ripple*: Naïve implementation of base-10 ripple-carry (Section 5.1, Figures 1 and 5),
- (ii) *PG-a*: base-10 carry chain using an adder to produce P - G values (Section 5.2, Figures 2, 6, 8, 9, and 10),
- (iii) *PG-b*: base-10 carry chain computing directly P - G values (Section 6.3, Figures 2 and 11),
- (iv) *PG-c*: base-10 carry chain computing directly P - G values using muxf5 and muxf6 (Section 6.3, Figures 2 and 12),
- (v) *M-bit Binary*: base-2 carry chain adder covering the same range as an N -digit adder.

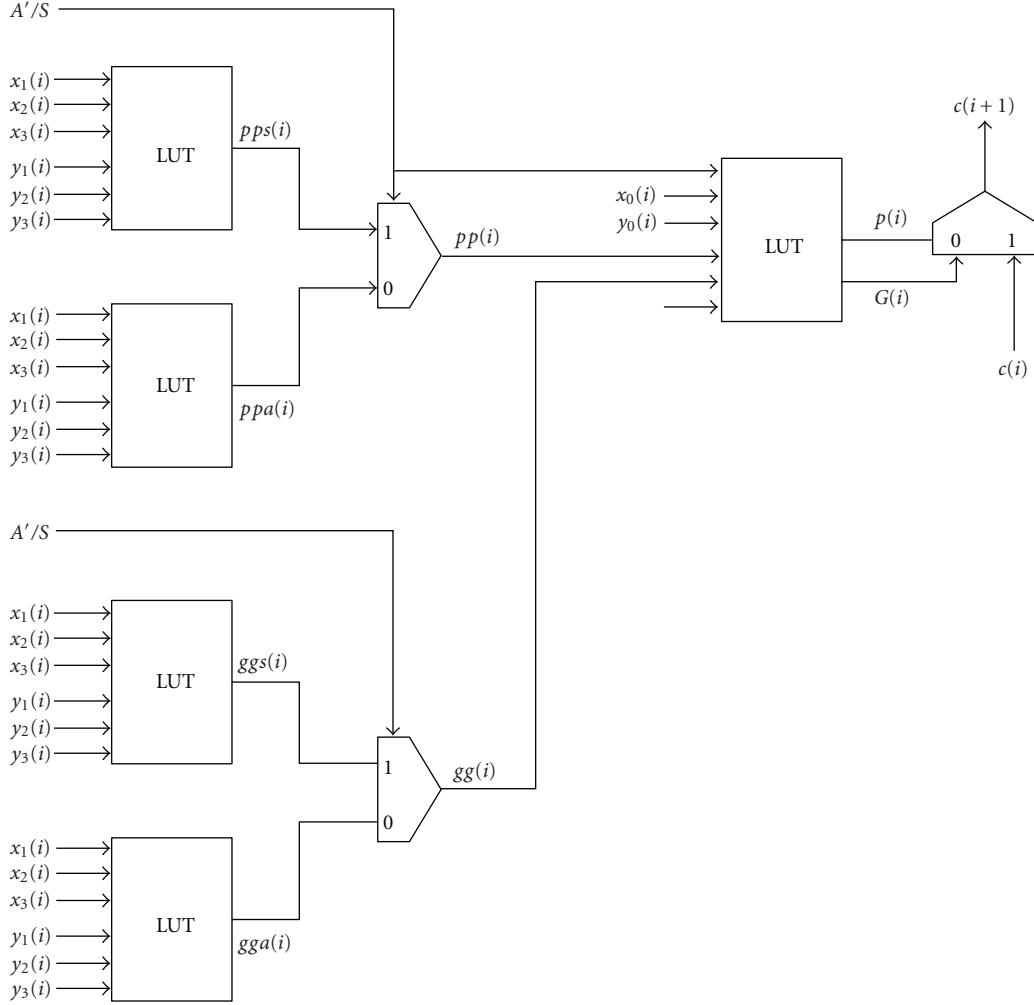


FIGURE 17: FPGA implementation of the carry-chain stage AS-II for BCD adder-subtractor.

Figure 18 shows the delays for the compared adders. Observe that, for the technology at hand, Table 1 and Figure 18 suggest that for $N > 48$ the carry-chain decimal implementation of adders is faster than the binary one for the equivalent range. Furthermore for small numbers of digits to add ($N < 40$) the PG_c architecture is faster than other decimal implementations.

8.2. Virtex-5 Adder-Subtractor Implementations. The adder-subtractor circuits have been implemented on Xilinx Virtex-5 family with speed grade-2 [7]. The synthesis and implementation have been carried out on XST (Xilinx Synthesis Technology) [13] and Xilinx ISE (Integrated System environment) version 10.1 [14]. The critical parts were designed using low-level components instantiation (lut6.2, muxcy, xorcy, etc.) in order to obtain the desired behavior. Performances of different N -digit BCD adders have been compared to those of an M -bit binary carry chain adder (implemented by XST) covering the same range, that is, such that $M = \lceil N \cdot \log_2(10) \rceil \approx 3.322 N$.

TABLE 1: Time delays (nsec) for different adders in Virtex-4 -11.

| N | N -digit BCD adder | | | | M | M -bit Binary |
|-----|----------------------|---------|---------|---------|-----|-----------------|
| | <i>Ripple</i> | PG_a | PG_b | PG_c | | |
| 8 | 14 | 5.8 | 6.7 | 4.5 | 27 | 2.7 |
| 12 | 20 | 6.0 | 6.8 | 4.8 | 40 | 3.2 |
| 16 | 27 | 6.1 | 7.0 | 5.1 | 54 | 3.7 |
| 24 | 40 | 6.4 | 7.3 | 5.7 | 80 | 4.7 |
| 32 | 53 | 6.7 | 7.6 | 6.3 | 107 | 5.7 |
| 40 | 66 | 7.0 | 7.9 | 6.9 | 133 | 6.6 |
| 48 | 79 | 7.3 | 8.2 | 7.5 | 160 | 7.6 |
| 64 | 105 | 7.9 | 8.7 | 8.7 | 213 | 9.6 |
| 96 | — | 9.1 | 9.9 | 11.0 | 319 | 13.5 |
| 128 | — | 10.3 | 11.1 | 13.4 | 426 | 17.5 |

Table 3 exhibits the postplacement and routing delays in ns for the decimal adder implementations Ad -I and Ad -II of Section 7.1; Table 4 exhibits the delays in ns for

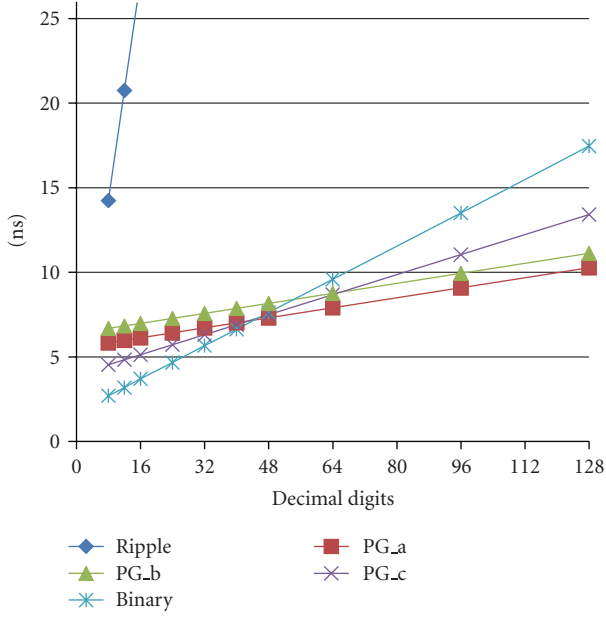
FIGURE 18: Delay in *ns* for different adders in Virtex-4.

TABLE 2: Area (in LUTs) for different adder's size in Virtex-4.

| Adder Circuit | # LUT's |
|---------------|-------------------|
| Ripple | $8 \times N$ |
| PG.a | $10 \times N$ |
| PG.b | $18 \times N$ |
| PG.c | $16 \times N$ |
| Binary | $[3.32 \times N]$ |

TABLE 3: Delays in ns for decimal and binary adders in Virtex-5 -2.

| <i>N</i> (BCD digits) | <i>Ad</i> -I | <i>Ad</i> -II | <i>M</i> (bits) | Binary Adder |
|-----------------------|--------------|---------------|-----------------|--------------|
| 8 | 3.8 | 3.6 | 27 | 2.1 |
| 16 | 4.4 | 4.0 | 54 | 2.6 |
| 32 | 5.0 | 4.5 | 107 | 3.8 |
| 48 | 5.2 | 5.1 | 160 | 5.1 |
| 64 | 5.6 | 5.2 | 213 | 6.6 |
| 96 | 6.1 | 5.9 | 319 | 8.7 |

the decimal adder-subtractor implementations AS-I and AS-II of Section 7.2. Table 5 lists the consumed areas expressed in terms of 6-input look-up tables (6-input LUTs). The estimated area presented in Table 5 was empirically confirmed.

Comments. Observe that, for large operands, the decimal operations are faster than the binary ones.

The overall area with respect to binary computation is not negligible. In Virtex-4 the area increases, with respect to an equal range binary adder, in a factor between 2.4 and 5.4. In the 6-input LUT family Virtex-5 an adder-subtractor is between 3.0 and 3.9 times bigger.

TABLE 4: Delays in ns for decimal and binary adder-subtractor in Virtex-5 -2.

| <i>N</i> (BCD digits) | AS-I | AS-II | <i>M</i> (bits) | Binary Add-Sub |
|-----------------------|------|-------|-----------------|----------------|
| 8 | 3.8 | 3.8 | 27 | 2.1 |
| 16 | 4.1 | 4.0 | 54 | 2.6 |
| 32 | 4.7 | 5.0 | 107 | 3.8 |
| 48 | 5.3 | 5.2 | 160 | 5.2 |
| 64 | 5.7 | 5.5 | 213 | 6.6 |
| 96 | 6.3 | 6.2 | 319 | 8.8 |

TABLE 5: Area in 6-input LUTs for different adders and adders-subtractors.

| Circuit | # LUTs |
|-------------------------|-------------------|
| Adder <i>Ad</i> -I | $8 \times N$ |
| Adder <i>Ad</i> -II | $10 \times N$ |
| Binary Adder | $[3.32 \times N]$ |
| Adder-Subtractor AS-I | $10 \times N$ |
| Adder-Subtractor AS-II | $13 \times N$ |
| Binary Adder-Subtractor | $[3.32 \times N]$ |

9. Conclusions

The present interest in BCD arithmetic systems stimulates further researches at both the algorithmic and design levels. Considering that the hardware costs are everyday more affordable, full hardware BCD units are now very attractive, with moreover a growing potential in the near future.

This paper has developed some implementations of BCD adders and subtractors in *FPGA* platforms. Experimental results emphasize time performances with reasonable costs in terms of area. Matched with the binary system, the decimal implementations are faster as operand sizes are growing (break even around 50 digits).

One of the key points about delays comes from the fact that the carry-propagation computation remains binary; then a faster carry-chain circuit can be designed because, for the same operand range, the number of digits (therefore of carries to propagate) is lower in decimal than in binary. In the carry-chain structures studied in this paper, the propagate *P* and generate *G* functions are more complex and therefore more time and area consuming than in the binary ones; therefore, the speed improvements only appear for large enough operands. The breakeven point is obviously technology dependent; so it could be expected to occur for a smaller number of digits in the near future.

The area overhead with respect to binary computation is not negligible; it is around five times in Virtex-4 and nearly four times in Virtex-5. That is mainly due to the more complex definition of the *carry propagate* and *carry generate* functions and to the final mod 10 reduction. The decreasing costs of technology make hardware consumption less central.

For BCD addition, the performance considerations on Xilinx Virtex-5 platform are similar to those of 4-input LUTs-based Virtex-4 technology. That is, the addition time of BCD digits remains faster than the binary counterpart in the same conditions.

Finally, the BCD adder/subtractor, with a relatively small penalty in area, presents time performances quite similar to those of a straight BCD adder.

Acknowledgments

This work is supported by the Universities FASTA, Mar del Plata, Argentina, UNCPBA Tandil, Argentina, UAM, Madrid, Spain, and URV, Tarragona, Spain; it has been partially granted by the CICYT of Spain under contract TEC2007-68074-C02-02/MIC.

References

- [1] M. F. Cowlshaw, "Decimal floating-point: algorism for computers," in *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pp. 104–111, June 2003.
- [2] G. Jaberipur and A. Kaivani, "Binary-coded decimal digit multipliers," *IET Computers and Digital Techniques*, vol. 1, no. 4, pp. 377–381, 2007.
- [3] M. Cowlshaw, "General Decimal Arithmetic," <http://speleotrove.com/decimal/>.
- [4] F. Y. Busaba, C. A. Krygowski, W. H. Li, E. M. Schwarz, and S. R. Carlough, "The IBM Z900 decimal arithmetic unit," in *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, vol. 2, pp. 1335–1339, November 2001.
- [5] *IEEE Standard for Floating-Point Arithmetic (IEEE 754)*, IEEE, 2008.
- [6] Xilinx Inc., "Virtex-4 User Guide," April 2007, <http://www.xilinx.com/>.
- [7] Xilinx Inc., "Virtex-5 User Guide," 2008, <http://www.xilinx.com/>.
- [8] J.-P. Deschamps, G. Bioul, and G. Sutter, *Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems*, John Wiley & Sons, New York, NY, USA, 2006.
- [9] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, Oxford, UK, 2000.
- [10] Xilinx Inc., Xilinx, <http://www.xilinx.com/>.
- [11] "Decimal Arithmetic in FPGA," <http://arithmetic-circuits.org/decimal/>.
- [12] Xilinx Inc., *Constraints Guide—ISE9.2i*, chapter 2, Relative Location (RLOC), 2008.
- [13] Xilinx Inc., "XST User Guide-10.1i," 2008, <http://www.xilinx.com/>.
- [14] Xilinx Inc., "ISE 10.1 Documentation," 2008, <http://www.xilinx.com/>.

