Reconfigurable Universal Adder

Humberto Calderón, Georgi Gaydadjiev and Stamatis Vassiliadis
Computer Engineering Laboratory
Electrical Engineering Dept.,EEMCS,TU Delft, The Netherlands
email: {H.Calderon,G.Gaydadjiev,S.Vassiliadis}@ewi.tudelft.nl
WWW home page: `http://ce.et.tudelft.nl`

Abstract

In this paper we present a novel adder/subtracter arithmetic unit that combines Binary and Binary Code Decimal (BCD) operations. The proposed unit uses effective addition/subtraction operations on unsigned, sign-magnitude, and various complement representations. Our design overcomes the limitations of previously reported approaches that produce some of the results in complement representation when operating on sign-magnitude numbers. The proposal can be implemented in ASIC as a run time configurable unit as well as in reconfigurable technology in form of a run-time reconfigurable engine. When reconfigurable technology is considered, a preliminary estimation indicates that 40 % of the hardware resources are shared by the different operations. This makes the proposed unit highly suitable for reconfigurable platforms with partial reconfiguration support. The proposed design together with some classical adder organizations were compared after synthesis targeting 4vfx60ff672-12 Xilinx Virtex 4 FPGA. Our design achieves a throughput of 82.6 MOPS with almost equivalent area-time product when compared to the other proposals.

1 Introduction

Various applications, e.g. commercial and financial electronic transactions [12], internet [19] and industrial control [22] require precise arithmetic for different data representation formats. Binary arithmetic is not capable of expressing exactly fractional numbers like, 0.2, and therefore, Binary Decimal Arithmetic emerges as a possible solution [9] to avoid binary approximations [13, 3]. When performing decimal operations on traditional binary based hardware, excessive delays are introduced due to the software emulations, conversions and corrections, typically 100 to 1000 times slower [8]. Therefore, flexible hardware solutions for both decimal and binary processing are considered in this paper. We assume universal units similar to those presented in [25] that are capable of performing various related operations sharing the same hardware. More specifically, the main contributions of this paper are:

- Twelve related operations were collapsed into a single Universal Adder (UA) hardware unit using Universal-Notation[i]. The proposed structure uses the *effective* addition/subtraction approach similar to [24] for both, binary and BCD notations.
- High hardware utilization: approximately 40% of the hardware resources implementing the 12 different operations are reused. This makes the unit a good candidate for implementation on partially reconfigurable platforms such as [26].

The remainder of this paper is organized as follows. Section 2 outlines the background and presents the related work. Section 3, exhibits with details the Reconfigurable Universal Adder design. Section 4 presents the experimental results and introduces the analysis in terms of used area and delay. Finally, Section 5 concludes the paper.

2 Background

Decimal digits are usually converted to binary representation as follows: the 8421 BCD code uses only the first ten combinations of a 4-bit binary code to represent each decimal digit. The remaining encodings (1010 to 1111) corresponding to decimal (10 to 15) are left unused when decimal computing is considered. Assuming addition of two decimal numbers $N1$ and $N2$ their SUM can exceed 1001_2 ($SUM > 9$). When such a situation occurs, correcting addition using 0110_2 (6) is required, so the complete operation becomes $SUM = N1 + N2 + 0110_2$. Please note that in case of subtraction N2 can be the complement representation of the original operand. In other words, decimal subtraction introduces additional processing. Recall

[i]In the context of this article, we assume operands and results to be in unsigned, sign-magnitude or complement (two's complement for binary and ten's complement for decimal) notations.

that a complement (CN) of an n-bit number N is computed by: $CN = r^n - N$; where r is the radix of the number ($r = 2$ for binary, and $r = 10$ for decimal). Thus, the ten's complement of an x-digit number N, is expressed by $CN_N = 10^x - N$. When we consider only a single digit D (as in the case with BCD), the computation is simplified to $CN_D = 10 - D$. Nevertheless, BCD encoding do not include a code for 10 and for this reason a nine's complement representation is used. In this case the computation becomes: $CN_D = 10 - D = 9 - D + 1$.

Some helpful techniques used to compute a nine's complement of a digit are [21]: **Pre-complement.** In this approach the subtrahend is one-complemented, a 1010_2 value is added for correction, and finally the generated carries are discarded. **Post-complement.** In this case, a binary 0110_2 value is added to the subtrahend operand, the result is one-complemented and the generated carries are used.

2.1 Related work

Many researchers addressed the problem of decimal arithmetic in the past. Early solutions proposed customized decimal adders, like Schmookler et al. [20] and Adiletta et al. [1]. Combined Binary and BCD adders were designed by Levine et al. [18] and Anderson [2], while true decimal sign-magnitude adder/substracter was presented by Grupe [14]. The latter used 3-binary-adders along with additional logic, achieving similar results as presented in this paper. An area efficient sign-magnitude adder was developed by Hwang [17]. In his approach two additional conversions are introduced before and after the binary addition. This is somehow similar to other proposals, e.g. [14]. The novelty in Hwang's proposal comes with the separation of the binary and the decimal results, using a multiplexor to select the correct output as is depicted in Figure 1.

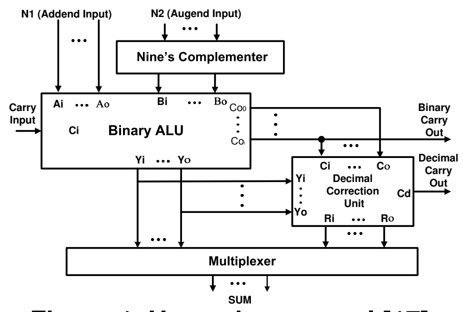


Figure 1. Hwang's proposal [17]

Flora [11] presents an adder that process concurrently two different results, one assuming the presence of an input carry and the other assuming no carry in available. This is following the principle of carry select adders [4], an approach actually used in several state of the art units. To cope with the area overhead, another compact design employing a single adder was introduced by Fischer et al. [10] depicted in Figure 2. This unit encodes ("edits") the incoming operands as well as the adder result to achieve the desired functionality.

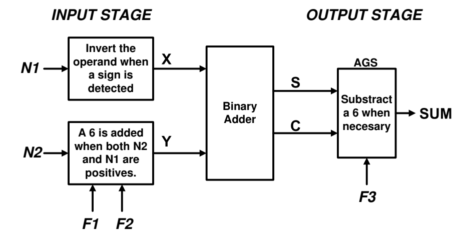

INPUT STAGE
OUTPUT STAGE
N1
Invert the operand when a sign is detected
X
N2
A 6 is added when both N2 and N1 are positives.
Y
F1
F2
Binary Adder
S
C
AGS
Substract a 6 when necesary
SUM
F3

Figure 2. Fischer's proposal [10]

During the last decade various binary and BCD adder/subtracter units for the IBM S/390, G4, G5 and G6 microprocessors [7] were developed. Following this line, the eServer z900 processor [23] includes a combined binary/decimal arithmetic unit working with binary-coded decimal numbers in sign-magnitude representation. The z900 unit was designed to operate in a single cycle, nevertheless a correction of the results is required in some of the cases [5]. For example, when an effective subtraction operation is performed, they need to calculate the complement value of the result, hence increasing the latency. The approach used to construct this decimal unit is based on the work presented in [15], which is shown in Figure 3. Recently, a similar approach was presented by Haller et al. [16]. In this work the carry chain is optimized resulting in a slight delay improvement with an increased area of the unit.

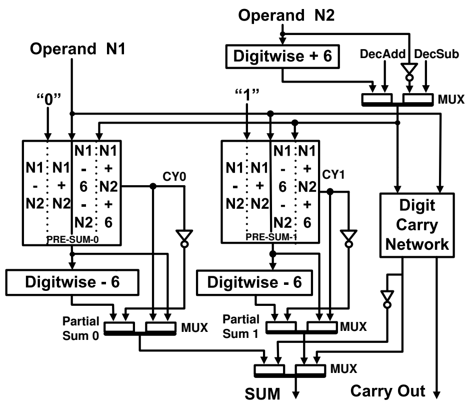


Figure 3. Haller's proposal (z900)

The following Table 1, summarizes the fundamental characteristics of the adders discussed above. The subset of the proposals aiming at signed arithmetic do not produce the correct result and need and additional complement operation in the cases when the subtrahend is greater than the minuend causing an additional cycle. As will be shown in the text to follow our approach does not have this disadvantage.

Table 1. Adders - data representation

Adder	BCD		Binary	
	Addition	Subtraction	Addition	Subtraction
Schmookler [20]	U	-	-	-
Addileta [1]	U	-	-	-
Levine [18]	U	-	U	-
Anderson [2]	U	-	U	-
Grupe [14]	S	S	S	S
Hwang [17]	S	ten's	S	two's
Flora [11]	S	ten's	S	two's
Fischer [10]	S	ten's	S	two's
z900 [5]	S	ten's	S	two's
Haller2006 [16]	S	ten's	S	two's

U: unsigned; S: sign-magnitude. Two's and ten's: for complement representation. The last two represent the inconsistency to drive the correct result for ($N2 > N1$).

3 Reconfigurable Universal Adder

The main goal of our work was to create an adder capable of carrying out decimal operations using effective addition/subtraction scheme, overcoming the limitations of the adders presented in the related work section. To achieve this we used the S/370 sign-magnitude binary adder presented in [24] as a base. We extend its functionality to perform decimal addition/subtraction operations along with the original binary ones. Next, we briefly introduce the original binary adder and focus on our extensions. In the next subsection we will discuss the specific decimal functionalities of our design.

Let us assume $N1$ and $N2$ being two n-bit sign-magnitude numbers, such that $N1 = [N1_{n-1}N1_{n-2}...N1_0]$ and $N2 = [N2_{n-1}N2_{n-2}...N2_0]$, with $N1_{n-1}$ and $N2_{n-1}$ used as sign bits of binary or BCD representations. Assuming absolute values, equation (1) is used to determine the exact operation, e.g. effective addition either subtraction. The logic one value of the signal Effective Addition ($EAdd$) indicates an effective addition operation, logic zero indicates that an effective subtraction is carried out.

$$\begin{aligned} EAdd &= (\overline{N1_S} \cdot \overline{N2_S} \cdot \overline{Add}) | (\overline{N1_S} \cdot N2_S \cdot Add) | \\ &\quad (N1_S \cdot \overline{N2_S} \cdot Add) | (N1_S \cdot N2_S \cdot \overline{Add}) \qquad (1) \\ &= (N1_S \oplus N2_S \oplus \overline{Add}) \end{aligned}$$

where the input $Add = 0$ invokes addition operation, otherwise subtraction is performed. Please note that we use "|" to represent a logical *OR* and "**+**" for addition. Also observe the use of modified sign bits in (1). This modification proposed here is to cope with the cases of unsigned and complement numbers as will be explained later. The two modified sign bits are computed as follows:

$$N1_S = N1_{n-1} \cdot Type \qquad (2)$$

$$N2_S = N2_{n-1} \cdot Type \qquad (3)$$

where $Type = 0$ signal, masks both sign bits $N1_{n-1}$ and $N2_{n-1}$ in the case of unsigned and complement representations. The result of this will be that $EAdd$ is equal to the $\overline{Add}$ signal making sure the final effective operation to be realized by the sign-magnitude adder is always correct. In addition, we should use $N1_{n-1}$ and $N2_{n-1}$ values in the result (SUM) computation. How those values participate into the above operation for all possible cases is stated in Table 2.

Table 2. Adder setting up considerations

Rep.	$EAdd$	SUM	$Type$
S.Mag.	$N1_{n-1}$,$N2_{n-1}$	$N1_{n-1} = N2_{n-1} = 0$	1
Uns.	$N1_{n-1} = N2_{n-1} = 0$	$N1_{n-1} = N2_{n-1} = 0$	0
Comp.	$N1_{n-1} = N2_{n-1} = 0$	$N1_{n-1}$,$N2_{n-1}$	0

The binary sign-magnitude addition is performed using the absolute values of the input addends as is stated by equation (4):

$$\sum = |N1| + |N2|^* \qquad (4)$$

where $|N2|^*$ is equal to $|N2|$ for effective addition and $|N2|^*$ equals to $\overline{|N2|}$ in case of effective substraction[ii]. In order to generate a correct sign-magnitude result, an additional correction step $\oplus\triangle$ is used. The final magnitude result becomes:

$$|SUM| = \sum_k \oplus \triangle \quad \forall \quad 0 < k < n-2 \qquad (5)$$

The $\triangle$ is computed as follows:

$$\triangle = \overline{C_o} \cdot \overline{EAdd} \qquad (6)$$

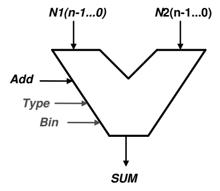
where the carry output (Co) is equal to:

$$\begin{aligned} Co &= G_0^{n-2}[|N1|, |N2|^*] \\ &= G_0 | (P_0 \cdot G_1 | ... (P_0 \cdot P_1 ... \cdot P_{i-1} \cdot G_i \qquad (7) \\ &\quad | ... | (P_0 \cdot P_1 ... \cdot P_{i-3} \cdot G_{n-2})) \end{aligned}$$

where G_i and P_i are the *generate* and *propagate* signals of the single bit adders. Finally, the sign bit of the result is updated as shown in equation 8 (see [24] for details):

$$SUM_{n-1} = [N1_{n-1} \oplus (\overline{Co} \cdot \overline{EAdd})] \cdot \overline{(SUM \equiv 0)} \qquad (8)$$

All of the different cases of the original adder are presented in the table shown in Figure 4.



N1(n-1)	N2(n-1)	*Add*	*EAdd*	Result Operation
0	0	0	1	+ (|N1| + |N2|)
0	0	1	0	+ (|N1| - |N2|)
0	1	0	0	+ (|N1| + |N2|)
0	1	1	1	+ (|N1| - |N2|)
1	0	0	0	- (|N1| + |N2|)
1	0	1	1	- (|N1| - |N2|)
1	1	0	1	- (|N1| + |N2|)
1	1	1	0	- (|N1| - |N2|)

Figure 4. Sign Magnitude Adder

[ii]EAdd signal is used to control the complement operation of N2.

For decimal operation we reuse the $EAdd$ and Add signals and introduce the two additional control signals (*Bin* and *Type*). $Bin = 1$ causes a binary operation, otherwise a BCD operation is performed. The Type signal has the functionality as explained earlier.

3.1 Decimal Arithmetic Additions

In this section we describe in more details the specific additions to the original binary adder needed for decimal addition/subtraction operations. As already indicated, previous proposals use ± 6 correction terms for the input operands and the result of the addition. Furthermore, when the subtrahend is greater than the minuend the result of the addition is in its complement representation and an extra cycle for result correction is required [10, 15, 16]. In our proposal the Co signal stated in equation (7) is used to detect the case when $N2 > N1$ and decimal subtraction operations is performed. Its value in combination with the Decimal Carry signals determine the specific correction of the result, e.g. the additional ten's complement operation. A simple example of such a case is presented in Figure 5, which depicts all operations involved in a subtraction operation (when $N2 > N1$).

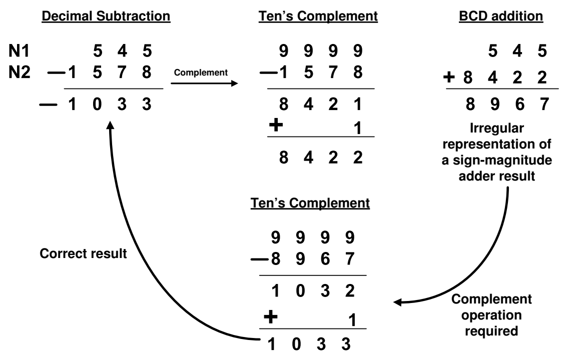


Figure 5. Decimal subtraction: N2 > N1

In the proposed adder depicted in Figure 6, the nine's complement computation for the subtrahend is performed using a "DigitWise-6" (*DW*) hardwired logic as used in many previous designs [15]. The DW value $ND = N - 6_{10}$ is obtained with the following equations that modify each bit of the BCD nibble as follows:

$$ND_{(3)} = \overline{N2_{(3)}} \cdot \overline{N2_{(2)}} | \overline{N2_{(3)}} \cdot \overline{N2_{(1)}} | N2_{(3)} \cdot N2_{(2)} \cdot N2_{(1)}$$
$$ND_{(2)} = N2_{(2)} \oplus N2_{(1)}$$
$$ND_{(1)} = \overline{N2_{(1)}}$$
$$ND_{(0)} = N2_{(0)}$$

The digit carry logic (DC) [5] signals for decimal operations are obtained as follows:

$$DC = X \mid Y \cdot CI \qquad (10)$$

where

$$X = G_3 \mid P_3 \cdot P_2 \mid P_3 \cdot P_1 \mid G_2 \cdot P_1$$
$$Y = P_3 \mid G_3 \mid P_1 \cdot G_1$$
$$CI = G_1 \mid P_1 \cdot C_{in}$$

All necessary correction cases for decimal data arithmetic, proposed here are summarized in Table 3. Our slightly more complex coder enables effective addition-subtraction characteristics for our universal adder. Please note that in the post-complement operation case, two necessary correction operations are required, both using 0110_2 value (one on the subtrahend value and one on the correction of addition result). In our proposal, in contrast to all previous adders, a single addition with 1100_2 value is used, achieving with this a real effective addition/subtraction capability.

Table 3. Decimal Correction Terms

	DC		$\overline{DC}$	
Operation	$N1 > N2$	$N2 \geq N1$	$N1 > N2$	$N2 \geq N1$
Addition	0110_2	0110_2	0000_2	0000_2
Subtraction	0110_2	1100_2	0000_2	0110_2

Please note that when a binary operation is performed $Bin = 1$ the decimal correction term is not needed. A value of 0000_2 is used for any binary operation. The universal adder is set up with the aforementioned logic, a one-dimensional (3/2)counter array, a Carry-propagate Adder and a set of XOR gates. The final organization is depicted in figure 6.

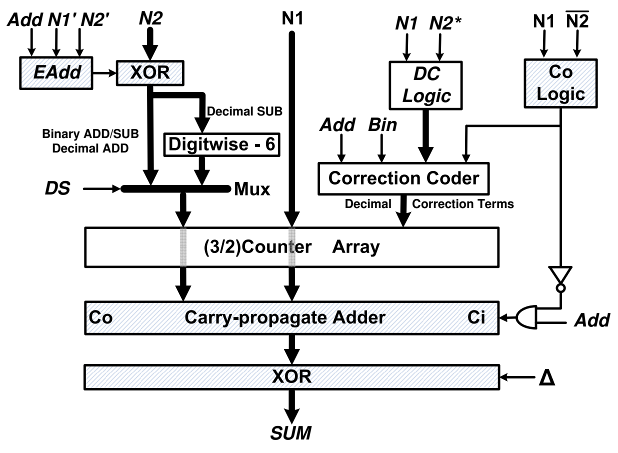


Figure 6. Universal adder micro-architecture

Note that the input $N2^*$ for computing the digit carry logic is equal to $\overline{N2} + 10$ when processing decimal subtraction otherwise is equal to $N2$. The multiplexor signal control for decimal subtraction or any addition (*DS*) is computed by:

$$DS = \overline{Add} \cdot \overline{Bin} \qquad (12)$$

The final complement operation is controlled by equation (13) which is a modified version of equation (6):

$$\triangle = (\overline{C_o} \cdot \overline{EAdd}) | (C_o \cdot \overline{Add} \cdot \overline{Bin}) \quad (13)$$

The rationale behind the design of the universal adder is based on the following observations and considerations:

- $EAdd$ signal controls one's complement operation of the subtrahend in binary and decimal operations.
- The $\triangle$ logic (see equation (13)) corrects the final result in binary and decimal operations.
- We need a (3/2)counter row to selectively add the necessary correction terms (correction coder) as indicated by Table 3.
- The hardware reuse is fundamental when a partial reconfigurable hardware units are designed for reconfigurable processor scenarios.

When our design is mapped on partially reconfigurable hardware platforms, the following modules (shown in dark in Figure 6) can be reused in both modes: the Carry-propagate Adder, the $EAdd$ logic, the two XOR logic blocks (at the input and output of the unit) and the Co logic. These "permanent" blocks can be configured at the beginning when the processor is set up. In order to provide the BCD functionality the remaining (shown as not colored in Figure 6) modules can be configured on demand. Such dynamic partial reconfiguration will diminish the penalty due to reconfiguration latencies. A preliminary estimation [6] indicates that the "permanent" blocks account for approximately 40 % of the total hardware resources implementing the operations when targeting Xilinx Virtex4 devices.

Finally, the fundamental decimal addition and subtraction operations of our proposal can be summarized with the following equations[iii] :

Decimal Addition

$$Sum = N1 + N2 + 0110_2 \; \forall \; DC = 1$$
$$Sum = N1 + N2 + 0000_2 \; \forall \; DC = 0$$

Decimal Subtraction

$$Sum = N1 + DW(\overline{N2}) + 0110_2 \; \forall \; DC = 1 \wedge Co = 0$$
$$Sum = N1 + DW(\overline{N2}) + 0000_2 \; \forall \; DC = 0 \wedge Co = 0$$
$$Sum = N1 + DW(\overline{N2}) + 1100_2 \; \forall \; DC = 1 \wedge Co = 1$$
$$Sum = N1 + DW(\overline{N2}) + 0110_2 \; \forall \; DC = 0 \wedge Co = 1$$

[iii]Please note that the decimal Sum value presented in those equations requires the one complemented operation as was stated in (5). Where the Δ control signal (13) is used to obtain the final SUM result

4 Experimental Results Analysis

The proposed Universal Adder was implemented using VHDL, synthesized, functionally tested, and evaluated using the ISE 8.1i Xilinx design tools [27] targeting 4vfx60ff672-12 VIRTEX 4 FPGA device. Furthermore, the designs proposed by Fischer [10], Hwang [17], Busaba [5] and Haller [16] were also implemented and synthesized using the same methodology. Table 4, summarizes the latency and hardware utilization results for all of the considered designs.

Table 4. Latency&Area - Adder comparison

	Latency (ns)			Resources used	
32-bit wide units	Logic	Wire	Total	Slices	LUTs
Our proposal	7.3	4.7	12.1	256	495
Fischer [10]	5.5	4.8	10.3	123	233
Hwang [17]	5.9	4.6	10.5	82	158
Busaba [5]	5.3	5.6	10.9	242	466
Haller [16]	5.5	4.5	10.0	305	584

Please note that all of the units proposed before require an additional cycle to produce the correct result in cases when $N2 > N1$ for the BCD subtraction operations (see Figure 5). To deal with this problem additional hardware unit or software fix is required. In the Fischer proposal the output stage can possible be modified for effective addition/subtraction, however, the custom made "editing logic", is expected to become very complex. In addition, none of the above proposals presented how they will detect the $N2 > N1$ (BCD) situation. Our design does not require such additional effort or resources with some increase in latency and area.

In terms of latency the "best" design is the one proposed by Haller. Compared to it, our proposal is 21% slower, but uses 15% to 16% less hardware resources (LUTs and Slices). When considering area, Hwang's proposal ranks best, but has the deficiency as described above. In general, our design is between 11% and 21% slower than the others. Assuming a design with a latency of 10ns, e.g. Haller, (worse case for our design) and a detection of $N2 > N1$ (BCD) "for free", the average execution time is going to be similar to ours (12ns) when 20% of the operations require the compensation cycle as introduced earlier.

When absolute performance is considered, our design achieves 82.6MOPS (1/12ns). Please note that this will be significantly improved when a pipelined version of the design is considered. The latency of the Carry-propagate Adder is expected to de determine the operating frequency in such a case.

5 Conclusions

This paper provides the details for a novel adder/subtracter arithmetic unit that combines Binary and Binary Code Decimal (BCD) operations in a single structure. The unit is able to perform effective addition-subtraction operations on unsigned, sign-magnitude, and various complement representations. Our proposal can be implemented in ASIC as a run time configurable unit as well as in reconfigurable technology as a run-time reconfigurable engine. Under the assumption that significant part of the hardware in the proposed structure is shared by the different operations reconfigurable platforms with partial reconfiguration become an interesting target. The proposed unit and several widely used adder organizations were synthesized for 4vfx60ff672-12 Xilinx Virtex 4 FPGA for comparison. Our design achieves a throughput of 82.6 MOPS with similar area x time (AxT) product when compared to the other proposals, using only 2% of the available resources of the targeted FPGA.

References

[1] M. J. Adiletta and V. C. Lamere. BCD Adder Circuit. *Digital Equipment Corporation, US patent 4805131*, pages 1 – 18, Jul 1989.

[2] J. L. Anderson. Binary or BCD Adder with Precorrected Result. *Motorola, Inc., US patent 4172288*, pages 1 – 8, Oct 1979.

[3] C. Bashe, W. Buchholz, and N. Rochester. The IBM type 702: An electronic Data Processing Machine for Business. *Journal of the Association for Computing Machinery*, pages 149–169, Oct 1954.

[4] O. Bedrij. Carry-select adder. *IRE Transctions on Computers*, pages 340 – 346, Jun 1962.

[5] F. Y. Busaba, C. A. Krygowsky, W. H. Li, E. M. Schwarz, and S. R. Carlough. The IBM z900 Decimal Arithmetic Unit. *Conference Record of the 35th Asilomar conference on Signals, Systems and Computers.*, pages 1353 – 1339, Sep 2001.

[6] H. Calderón, G. Gaydadjiev, and S. Vassiliadis. FPGA based implementation of Reconfigurable Universal Adders. *Technical Report CE-TR-2007-01*, pages 1 – 21, Jan 2007.

[7] M. A. Check and T. J. Slegel. Custom S/390 G5 and G6 Microprocessors. *IBM J. Res. & Dev. 43,No. 5/6*, pages 671 – 680, Sep 1999.

[8] M. F. Cowlishaw. Decimal Floating-Point: Algorism for Computers. *Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH'03)*, pages 1 – 8, Aug 2003.

[9] M. F. Cowlishaw. General Decimal Arithmetic. *IBM Corporation, Available at: http://www2.hursley.ibm.com/decimal/*, 2004.

[10] H. Fischer and W. Rohsaint. Circuit Arrangement for Adding or Subtracting Operands Coded in BCD-Code or Binary-Code. *Siemens Aktiengesellschaft, US patent 5146423*, pages 1 – 9, Sep 1992.

[11] L. P. Flora. Fast BCD/Binary Adder. *Unisys Corp., US patent 5007010*, pages 1 – 16, Apr 1991.

[12] E. C. D. Genera. The Introduction of the Euro and the Rounding of Currency Amounts. *European Commission Directorate General II Economic and Financial Affairs*.

[13] H. Goldstine and A.Goldstine. The Electronic Numerical Integrator and Computer (ENIAC). *IEEE annals of the History of Computing*.

[14] U. Grupe. Decimal Adder. *Vereinigte Flugtechnische Werke-Fokker gmbH, US patent 3935438*, pages 1 – 11, Jan 1976.

[15] W. Haller, U. Krauch, and H. Wetter. Combined Binary/Decimal Adder Unit. *International Business Machines Corporation, US patent 5928319*, pages 1 – 9, Jul 1999.

[16] W. Haller, W. H. Li, M. R. Kelly, and H. Wetter. Highly Parallel Structure for Fast Cycle Binary and Decimal Adder Unit. *International Business Machines Corporation, US patent 2006/0031289*, pages 1 – 8, Feb 2006.

[17] I. S. Hwang. High-Speed Binary and Decimal Arithmetic Logic Unit. *American Telephone and Telegraph Company, AT&T Bell Laboratories, US patent 4866656*, pages 1 – 11, Sep 1989.

[18] S. R. Levine, S. Singh, and A. Weinberger. Integrated Binary-BCD Look-Ahead Adder. *International Business Machines Corporation, US patent 4118786*, pages 1 – 13, Oct 1978.

[19] S. Microsystems. BigDecimal (Java 2 Platform SE v1.4.0" http//java.sun/com/products. *Sun Microsystems Inc.*

[20] M.S.Schmookler and A. Weinderger. Decimal Adder for Directly Implementing BCD Addition Utilizing Logic Circuitry. *International Business Machines Corporation, US patent 3629565*, pages 1 – 19, Dic 1971.

[21] R. M. M. Oberman. Digital Circuits for Binary Arithmetic. *The Macmillan Press LTD. ISBN: 0-333-25535-6*, 1979.

[22] M. Palmer. A Comparison of 8-bit microcontrollers. *Application note 520, Microchip Technology Inc.*, pages 1 – 11.

[23] E. Schwarz, M. Check, C. L. Shum, T. Koehler, S. Swaney, J. D. MacDougall, and C. A. Krygowski. The Microarchitecture of the IBM eServer z900 Processor. *IBM J. Res. & Dev. 46, No 4/5*, pages 381 – 395, Jul 2002.

[24] S. Vassiliadis, D. Lemon, and M. Putrino. s/370 Sign-Magnitude Floating-Point Adder. *IEEE Journal of Solid-State Circuits*, pages 1062 – 1070, Aug 1989.

[25] S. Vassiliadis, E. Schwarz, and M. Putrino. Quasi-universal VLSI multiplier with signed digit arithmetic. *Proceedings of the 1987 IEEE,Southern Tier Technical Conference*, pages 1–10, Apr. 1987.

[26] S. Vassiliadis, S. Wong, G. Gaydadjiev, K. Bertels, G. Kuzmanov, and E. Panainte. The MOLEN Polymorphic Processor. *IEEE Transactions on Computers*, pages 1363 – 1375, Nov 2004.

[27] Xilinx. The XILINX Software manuals, XILINX 8.1i. *http://www.xilinx.com/support/sw_manuals/xilinx8/index.htm*, 2006.