

A Reconfigurable Butterfly Architecture for Fourier and Fermat Transforms

Ali Al Ghouwayel, Yves Louët and Jacques Palicot

Supélec-IETR, Avenue de la Boulaie BP 81127

35511 CESSON-SEVIGNE Cedex, France

Email: {ali.alghouwayel, yves.louet, jacques.palicot}@supelec.fr

Abstract—Reconfiguration is an essential part of Software Radio (SWR) technology. Thanks to this technique, systems are designed for change in operating mode with the aim to carry out several types of computations. In this SWR context, the Fast Fourier Transform (FFT) operator was defined as a common operator for many classical telecommunications operations [1]. In this paper we propose a new architecture for this operator that makes it a device intended to perform two different transforms. The first one is the Fast Fourier Transform (FFT) used for the classical operations in the complex field. The second one is the Fermat Number Transform (FNT) in the Galois Field (GF) for channel coding and decoding.

I. INTRODUCTION

SoftWare Radio, appeared in 1990 [2][3], is an every-growing technology that receives enormous recognition and generating widespread interest in the telecommunication industry. Over the last few years, analog radio systems have been replaced by digital radio systems for radio applications. In addition to this, programmable hardware modules are increasingly being used in digital radio systems at different functional levels. SWR technology aims to take advantages of these programmable hardware modules to build an open-architecture based on radio system softwares. SWR technology facilitates implementation of some of the functional modules in a radio system such as modulation/demodulation, coding/decoding in software. In this optics, we will present a new architecture for the FFT whose Butterfly is reconfigurable so as to perform two kinds of transforms over two different fields. The first one is the complex field (\mathbf{C}) on which the FFT carry out some functions as OFDM modulation, frequential Egalisation... . The second one is the GF where the FFT will be reconfigured as an FNT to carry out the coding and two main steps of the decoding process for Reed-Solomon (RS) codes. The paper is organized as follows: Section 2 describes the FNT and their applications. Section 3 investigates some

arithmetic operators to define a reconfigurable Butterfly. In section 4, we present the global structure of FNT. Finally, the conclusions are outlined in section V.

II. THE FNT FOR RS CODES OVER $\text{GF}(F_t)$

The Number Theoretic Transform (NTT) has been introduced as a generalization of the Discrete Fourier Transform (DFT) over residue class rings of integers in order to implement fast cyclic convolutions and correlations without round-off errors and with better efficiency than the FFT [4][5]. Interesting applications of the NTT lies in fast coding, decoding, long integer multiplication, cryptography, digital filtering, image processing and deconvolution. For the transform length equal to F_t , where $F_t = 2^{2^t} + 1$ is the Fermat number, the NTT is called the Fermat Number Transform (FNT) which presents some advantages. It is quite obvious, that FNT is suitable for VLSI implementations. The structure of the FNT is identical to that of the DFT for power of two lengths. Then the same algorithms can be used for the classical radix-2 FFT and the radix-2 FNT. The only one difference is the substitution of the complex multiplication in the Fourier transform by a modulo F_t real multiplication in the case of the FNT. The following gives the definitions of FFT and FNT.

In \mathbf{C} , the Discrete Fourier Transform of $v = (v_0, v_1, \dots, v_{N-1})$, a vector of real or complex numbers, is a vector $V = (V_0, V_1, \dots, V_{N-1})$, given by

$$V_k = \sum_{i=0}^{N-1} e^{-j \frac{2\pi i k}{N}} v_i \quad k = 0, \dots, N-1 \quad (1)$$

where $j = \sqrt{-1}$. The Fourier kernel $\exp(-j2\pi/N)$ is an N th root of unity in the field \mathbf{C} . In the finite field $\text{GF}(q)$, an element α of order N is an N th root of unity. Drawing on the analogy between $\exp(-j2\pi/N)$ and α , we have the following definitions:

Let $v = (v_0, v_1, \dots, v_{N-1})$ be a vector over $\text{GF}(q)$, and let α be an element of $\text{GF}(q)$ of order N . The vector v and its Discrete Fourier Transform are related by :

$$V_j = \sum_{i=0}^{N-1} \alpha^{ij} v_i \quad \Longleftrightarrow \quad v_i = \frac{1}{N} \sum_{j=0}^{N-1} \alpha^{-ij} V_j, \quad (2)$$

for $j = 0, \dots, N-1$, where N is interpreted as an integer of the field. Further details can be found in [6].

Our work is focused on the application of the FNT to the channel coding-decoding. Indeed, the application of the Discrete Fourier Transform in the complex field occurs throughout the subject of signal processing. The same transform technique can play an important role in the study and processing of $\text{GF}(q)$ valued signals, q a prime number, that is, of codewords. By using the Fermat transforms, the principles of coding theory can be described in a setting that is much closer to the methods of signal processing. In frequency-domain, cyclic codes can be defined as codes whose codewords have certain specified spectral components equal to zero [6]. In [7] we have presented the advantages of the application of such a transform to RS codes constructed over the $\text{GF}(F_t)$. In this paper we will describe in details the practical realization of the FFT operator defined in \mathbb{C} and which can be reconfigured to become the FNT operator with arithmetic carried out modulo Fermat numbers. This reconfiguration consists in reconfiguring each Butterfly of the FFT structure. In the next section we will present the Butterfly itself as a function which is constituted by several reconfigurable arithmetic operators.

III. RECONFIGURABLE BUTTERFLY

In the SWR concept, an new area of research called "Parametrization" has been defined [8][9]. This technique consists to identify common resources, i.e Common Operator (CO) or Common Function (CF) between all the standards involved in the reconfiguration and in the standards themselves. Then, the trick is to exploit the same resources to execute two or more applications. In this context, the main goal of this work is to exploit the resources already present in the FFT structure to get the FNT one. With this purpose, the arithmetic operators i.e multiplier, adder and subtracter realizing operations over \mathbb{C} should be redefined to realize a modulo(F_t) operations. Then the reconfiguration of the Butterfly (Figure 1) consists to reconfigure the aforementioned arithmetic

elements. Then, one need to define a modulo(F_t) multiplier, adder and subtracter.

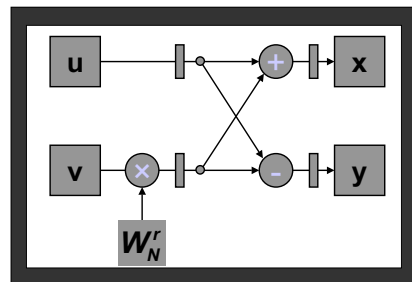


Fig. 1. The complex Butterfly

A. Modular Multiplication in $\text{GF}(F_t)$

The modulo $2^n + 1$ multiplication is widely used in the computation of convolutions and in Residue Number Systems (RNS) arithmetic. Several architectures of a modulo $(2^n + 1)$ multiplier based on Ma's algorithm [10] and on the modified Low-High algorithm [11] was described in [12][13]. Indeed, there are two categories of algorithms for the modulo $2^n + 1$ multiplication. The first one consists to perform the multiplication and after the correction [12]. The second one consists in the reduction of partial products [10][14]. In [12, Table 2] the author has compared the performances of the different architectures of Modular multiplier implemented in Virtex-II and Virtex-E. For Virtex II devices, the " $(n+1)*(n+1)$ " multiplier allows a significant gain in terms of slices and a reduced delay compared to the modified Low-High algorithm and Zimmermann's algorithm [14]. For Virtex-E devices, this " $(n+1)*(n+1)$ " multiplier offers the best compromises area-delay in case of non-pipelined architecture. On the other hand and for the pipelined Virtex-E devices, the operator based on Zimmermann's algorithm offers the best compromises area-delay. As previously mentioned, our works lie in the development of a modular multiplication by exploiting the already existing resources. Then, our proposed multiplier (Figure 2) is based on the architecture presented in [12, Figure 2-a] with some simplifications. In fact, for $n = 2^t$ the proposed multiplier works in $\mathbb{Z}_{2^n+1} = \{0, a \in \mathbb{Z}_{2^n+1} \mid \text{gcd}(a, 2^n + 1) = 1\}$ and the one of two operands of multiplication is the element α^i , $i = \{0, 1, \dots, \frac{F_t-1}{2} - 1\}$, $\alpha^{\frac{F_t-1}{2}} = -1$. Then the product that equal $F_t - 1$ and that requires $(n+1)$ -bit never occurs. From that simplifications come. In Figure 2, the white elements indicate the elements not used in the case of operation over GF . As it is noticed in this figure, there is a reconfiguration of the

connections inter-operators. The dotted lines connections represent the additional connections in this operating mode over GF.

A basic modulo $(2^n + 1)$ multiplication algorithm consists in computing $p=xy$, and dividing this product by $2^n + 1$:

$$xy \bmod (2^n + 1) = p \bmod (2^n + 1) = \sum_{i=0}^{2n-1} p_i 2^i$$

Since the division is a hard task, it will be interesting to use an algorithm to perform the modular reduction. We define c_L and c_H the lower and higher words respectively of the product p as follows:

$$c_L = \sum_{i=0}^{n-1} p_i 2^i \quad \text{and} \quad c_H = \sum_{i=0}^{n-1} p_{n+i} 2^i$$

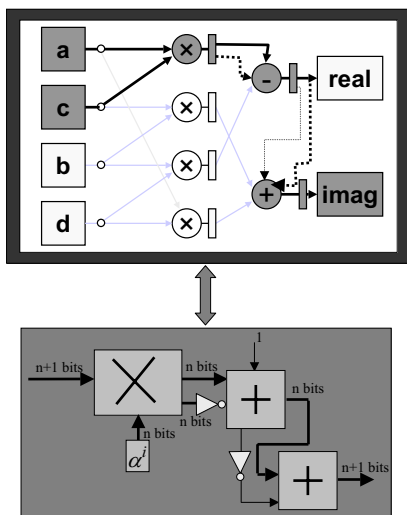


Fig. 2. The modulo $(2^n + 1)$ multiplier

The modulo $(2^n + 1)$ operator depicted in Figure 2 is carried out by :

$$xy \bmod (2^n + 1) =$$

$$\begin{cases} (c_L + \overline{c_H} + 2) \bmod 2^n & \text{if } c_L + \overline{c_H} + 1 < 2^n \\ (c_L + \overline{c_H} + 1) \bmod 2^n & \text{otherwise} \end{cases}$$

B. Modular Addition in $GF(F_t)$

Most of algorithms describing the addition modulo $(2^n + 1)$ are performed in the diminished-one number system, where a number x is represented by $x' = x - 1$ and the number 0 is not used or treated as a special case [13][14][15]. This implies:

$$\begin{aligned} (x' + y' + 1) \bmod (2^n + 1) &= \\ \begin{cases} x' + y' \bmod 2^n & \text{if } x' + y' \geq 2^n \\ (x' + y' + 1) \bmod 2^n & \text{if } x' + y' < 2^n \end{cases} \\ &= (x' + y' + \overline{c}_{out}) \bmod 2^n. \end{aligned}$$

Since the operators performing the complex addition processes the numbers in normal representation, the best way to perform a modular addition is to keep the same architecture to get the reconfigurability at lower costs. Let us now study the modulo $(2^n + 1)$ addition of two numbers in normal representations. In [13] the author described some algorithms that return the desired results increased by one. Nevertheless this property facilitates the design of the circuit. The modulo $(2^n + 1)$ addition is defined by:

$$\begin{aligned} (x + y + 1) \bmod (2^n + 1) &= \\ \begin{cases} 2^n & \text{if } x = 2^n \text{ and } y = 2^n \\ (x + y) \bmod 2^n + \overline{c}_{out} & \text{if } 0 \leq x + y < 2^{n+1} \end{cases} \end{aligned} \quad (3)$$

In [13] a direct implementation of equation (3) is presented. The circuit is shown in Figure 3-a. To improve the implementation, the author suggests an alternative architecture suppressing the multiplexer Figure 3-b. The modulo $(2^n + 1)$ addition is expressed as:

$$\begin{aligned} (x + y + 1) \bmod (2^n + 1) \\ = (x + y) \bmod 2^n + s_{n+1} 2^n + \overline{s_{n+1} \vee s_n} \end{aligned}$$

To perform an addition that returns directly the desired result, we propose an alternative adder shown in Figure 4. We define s^1, s^2 the sums at the first and second adders respectively with the $(n+2)$ -bit integer $s^1 = [s_{n+1}^1 s_n^1 \dots s_0^1] = x + y$. The modulo $(2^n + 1)$ addition can be expressed as:

$$\begin{aligned} (x + y) \bmod (2^n + 1) &= \\ \begin{cases} (x + y) \bmod 2^n & \text{if } 0 \leq x + y < 2^n \\ (x + y) \bmod 2^n + 2^n - 1 & \text{if } 2^n < x + y \leq 2^{n+1} \\ 2^n & \text{if } (x = 2^n \text{ and } y=0) \\ & \text{or } (x=0 \text{ and } y = 2^n) \end{cases} \end{aligned} \quad (4)$$

in other words:

$$(x + y) \bmod (2^n + 1) = \overline{s_n^2} s_2 + s_n^2 2^n$$

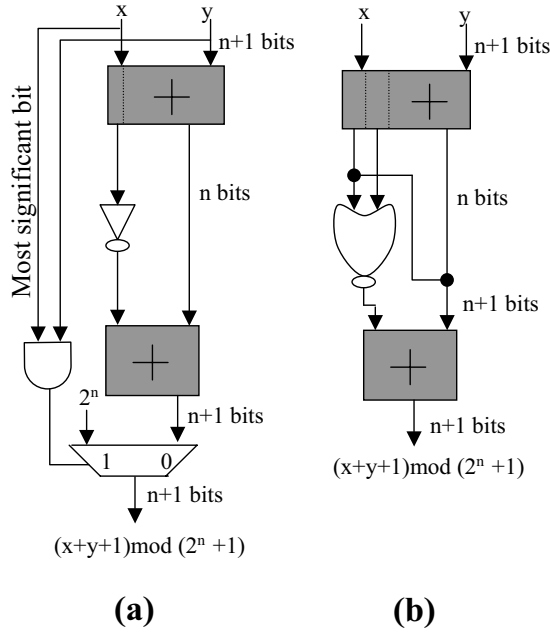


Fig. 3. The two architectures of the mod $(2^n + 1)$ Adder [13]

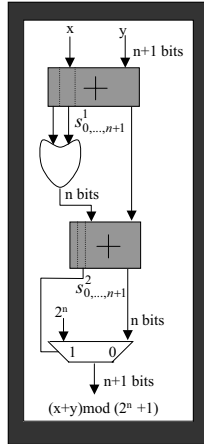


Fig. 4. The proposed mod $(2^n + 1)$ Adder

Now, let us demonstrate the correctness of equation (4). First of all, let us consider x and y two elements of $GF(F_t)$, $0 \leq x, y \leq 2^n$. Then,

$$0 \leq x + y \leq 2^{n+1}$$

We have to distinguish the four following cases to establish the correctness of our algorithm:

- For $x + y = 2^{n+1}$ (i.e. $x = y = 2^n$), we have $s^1 = 2^{n+1}$ (i.e. $s^1_{n+1} = 1$, $s^1_i = 0$ for $i = 0, \dots, n$). Consequently $s^2 = 0 + 2^n - 1$, $s^2_n = 0$, and our algorithm returns $2^n - 1$.

- For $x + y = 2^n$ (i.e. $x = 0$ and $y = 2^n$ or $x = 2^n$ and $y = 0$), we have:

$$s^1_n = 1, s^1_{n+1} = 0 \text{ and } (s^1_n \vee s^1_{n+1} = 1),$$

$$s^2 = 2^n + 2^n - 1 = 2^{n+1} - 1,$$

In this case $s^2_n = 1$ and the multiplexer selects 2^n as result. This is the only case where $s^2_n = 1$.

- For $2^n < x + y < 2^{n+1}$, we have:

$$s^1 = 0.2^{n+1} + s^1_n 2^n + \dots + s^1_0,$$

or

$$2^n \text{ mod } (2^n + 1) = (-1 + 2^n + 1) \text{ mod } (2^n + 1) \\ = (-1) \text{ mod } (2^n + 1),$$

The second adder of Figure 4 returns an addition mod 2^n , then $(-1) \text{ mod } 2^n = 2^n - 1$.

Consequently,

$$2^n + 2^n - 1 < s^2 = x + y + 2^n - 1 < 2^{n+1} + 2^n - 1,$$

$$2^{n+1} \leq s^2 < 3 * 2^n - 1,$$

what give $s^2_n = 0$. Then our algorithm return $(x + y + 2^n - 1) \text{ mod } 2^n$.

- Finally, for $0 \leq x + y < 2^n$, we have:

$$s^1_{n+1} = s^1_n = s^2_n = 0,$$

and $(x + y) \text{ mod } 2^{n+1} = x + y$.

As known, the arithmetic subtracter is usually based on the arithmetic adder structure. For the modulo $(2^n + 1)$ subtracter, we propose an operator shown in Figure 5. The subtraction modulo $(2^n + 1)$ can be expressed as follows:

$$(x - y) \text{ mod } (2^n + 1) =$$

$$\begin{cases} 2^n & \text{if } (x = 2^n \text{ and } y = 0) \\ (x + \bar{y} + 1 + s_n) \text{ mod } 2^n & \text{otherwise} \end{cases} \quad (5)$$

A proof of the correctness of this algorithm is provided in Annex A. Once the different elements of the Butterfly are defined, one can implement them to obtain the reconfigurable Butterfly. Figure 6 depicts the resulting hardware operator. The switch from an operating mode to another requires a change of the Fourier kernel and the reconfiguration of connection inter-operators. Assuming

that the Butterfly is configured to operate over \mathbf{C} and one wants to perform a calculation over $GF(F_t)$. To do this, the Butterfly should download the primitive element α^i , activate the different logic gate (AND,OR and the multiplexers) and reconfigure the connection inter-operators as shown in Figure 6. In the next section, the global architecture of the FNT is presented.

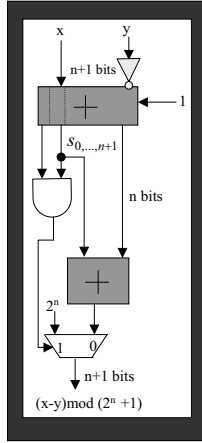


Fig. 5. The proposed mod $(2^n + 1)$ Subtractor

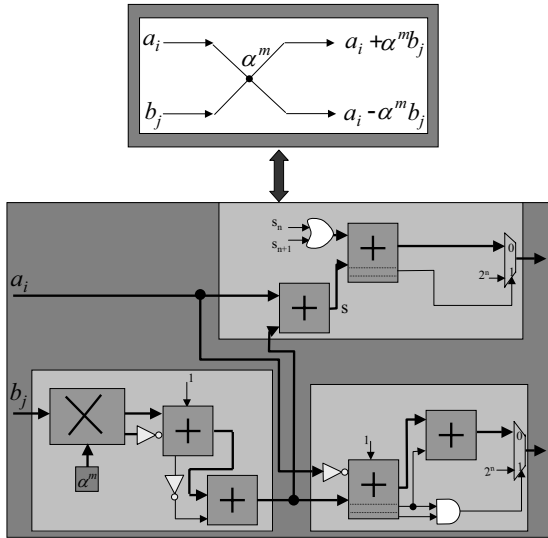


Fig. 6. The architecture of the Butterfly over $GF(F_t)$

IV. THE FNT ARCHITECTURE

In the previous sections, we have presented the reconfiguration at a rather low level. The Butterfly constitutes a high parameterized function level. The fact to have this parameterized function allows to design a reconfigurable

operator whose Butterfly forms the highest level operator. Figure 7 depicts the global reconfigurable operator. Over \mathbf{C} it is called FFT and over $GF(F_t)$ is called FNT. This architecture has been validated by software. A simple test of calculation of FFT and IFFT, showed the validity of this structure.

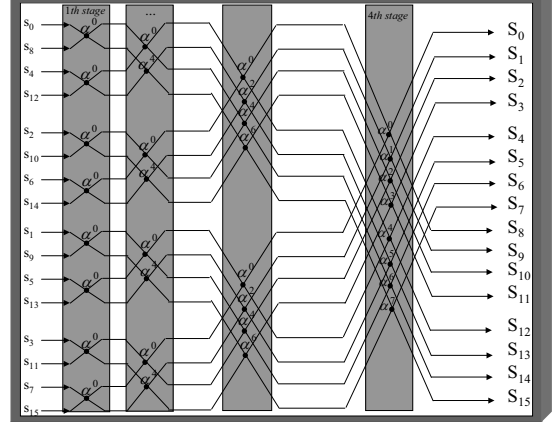


Fig. 7. The architecture of the FNT operator

V. CONCLUSIONS

A new arithmetic modular operators has been defined to build a reconfigurable Butterfly. Once the reconfigurable Butterfly has been identified, the FFT common operator is obtained. This operator is dedicated to be a reconfigurable operator that operates over \mathbf{C} to perform the Fourier transform (i.e. OFDM modulation,...) and over $GF(F_t)$ to perform RS channel coding and two main steps of the decoding process in computing the Fermat transform. As a current work, the objective is to implementing this operator with FPGAs to test its reconfigurability and the performances in term of slices and delay.

A PROOF OF THE NEW MODULO $(2^n + 1)$ SUBTRACTOR ALGORITHM

Let us demonstrate that the algorithm defined by Equation (5) carries out $(x - y) \bmod (2^n + 1)$ when $0 \leq x, y \leq 2^n$. First of all, let us note that x, y are $(n+1)$ -bit length with $0 \leq x + y \leq 2^{n+1}$.

We have: $2^n - 1 \leq \bar{y} \leq 2^{n+1} - 1$, that gives,

$$2^n - 1 + 1 \leq x - y = x + \bar{y} + 1 \leq 2^n + 2^{n+1} - 1 + 1$$

$$2^n \leq x + \bar{y} + 1 \leq 3 * 2^n$$

We have to distinguish the three following cases to establish the correctness of our algorithm:

• if $x \geq y \implies x + \bar{y} + 1 \geq 2^{n+1}$, $s_{n+1} = 1$, $s_n = 0$
and the algorithm returns $x + \bar{y} + 1$

• if $x \leq y \implies x + \bar{y} + 1 < 2^{n+1}$, $s_{n+1} = 0$, $s_n = 1$
and the algorithm returns $x + \bar{y} + 1 + 1$

• if ($x = 2^n$ and $y = 0$) $\implies s_{n+1} = s_n = 1$
and the algorithm returns 2^n .

REFERENCES

- [1] J. Palicot, C. Roland, *FFT: a basic Function for a Reconfigurable Receiver*, ICT'2003, Feb. 2003, Papeete, Tahiti.
- [2] J. Mitola, *The software Radio Architecture*, IEEE Communications Magazine, May 95, pp. 26-38.
- [3] W. Tuttlebee, *Evolution of radio systems into the 21st century*, Proc.IEEEInt. Conf. on 'Radio receivers an associated systems',1995.
- [4] M.A. Sonderstrand et al., *Reisdue Number System Arithmetic: Modern Applications in Digital Signal Processing*, New York: IEEE Press, 1986.
- [5] M.A. Bayoumi,G.A. Julien, and W.C. Miller, *A Look Up Table VLSI Design Methodology for RNS Structures Used in DSP Applications*, IEEE Trans. Circuits and systems, vol.34, pp. 604-616, June 1987.
- [6] Richard E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University press, 2001.
- [7] A. Al Ghouwayel, Y. Louët and J. Palicot, *A Reconfigurable Architecture for the FFT Operator in a Software Rdio Context*, IEEE ISCAS'2006,Greece, May 21-24, 2006.
- [8] W. Tuttlebee, *software defined radio: Enabling technologies*, Wiley, 2003.
- [9] J. Palicot, D. Giri, C Moy, *A Theoretical Approach of Parameterization Design for SDR Systems*, Workshop on Software Defined Radio : theory, design and applications, ESSIRC 2005 - Grenoble -France.
- [10] Y. Ma, *A Simplified Architecture for Modulo $(2^n + 1)$ Multiplication*, IEEE Transactions on Computers, 47(3) :333-337, 1998.
- [11] X. Lai, *On the Design and Security of Block Ciphers*, ETH Series in Information Processing. Hartung-Gorre Verlag Konstanz, 1992.
- [12] Jean-Luc Beuchat, *Modular Multiplication for FPGA Implementation of the IDEA Block Cipher*, Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03).
- [13] Jean-Luc Beuchat, *Some Modular Adders and Multipliers for Field Programmable Gate Arrays*, Proceedings of the 17th International Parallel and Distributed Processing Symposium. IEEE Computer Society,2003.
- [14] R. Zimmerman, *Efficient VLSI Implementation of Modulo $(2^n \pm 1)$ Addition and Multiplication*, In Proceedings of the 14th IEEE Symposium of Computer Arithmetic, pages 158-167.
- [15] Haridimos T. Vergos, Costas Efstathiou, and Dimitris Nikolos, *Diminished-One Modulo $2^n + 1$ Adder Design*, IEEE Transactions on Computers, Vol. 51, No. 12, Dec. 2002.