TESI DI DOTTORATO

UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"

DIPARTIMENTO DI INGEGNERIA ELETTRONICA
E DELLE TELECOMUNICAZIONI

DOTTORATO DI RICERCA IN
INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

DIGITAL BUILDING BLOCKS FOR TELECOMMUNICATION SYSTEMS

NICOLA PETRA

Il coordinatore del Corso di Dottorato
Ch.mo Prof. Giovanni POGGI

Il Tutore
Ch.mo Prof. A.G.M. STROLLO

Anno Accademico 2005-2006

To Elisa

Acknowledgments

I would like to give my most sincere thanks to my tutor, professor Antonio Strollo. His experience, knowledge, scientific methodology have been a constant example for me. His enthusiasm, support, and willingness in sharing his invaluable experience have been the stimulus at the basis of my activity as a Ph.D. student.

I want to thank the professors of the Electronics Group at the University of Napoli, for their priceless lessons.

I also want to thank the professor Alan Willson. The Wednesday talks gave me insights, stimulus and ideas I will always remember.

A great thank goes to the friends of the department of Electronics and Telecommunications Engineering, Davide, Ettore, Enzo, Fabrizio, Francesco, Dino, Luigi, Ilaria, for their support and friendship. Most of my passion for electronics comes from the opportunity to work with such wonderful people.

Many other people deserve my thanks for they, in many ways, helped me in the years of my Ph.D. study. A small, possibly incomplete list is: Luca, Paola, Giandomenico, Ciccio, Ferdinando, Teresa, Alessia, Alessandro, Ana, Ricki, Guichang, RuGang, Kate.

The final, most important thank, goes to my family. They are my strength, my pride, my joy.

Chapter 1
Introduction

1.1 Emerging Trends in Telecommunications Systems

In the recent years there is a growing trend in the communication technologies to shift from analogue toward digital techniques. The use of digital techniques, in fact, overcomes many analogue hardware limitations (like high sensitivity to process and temperature variations, difficult portability as the VLSI technology scales down etc.). Moreover, the programmability offered by digital techniques provides flexibility that is especially important in the context of rapidly evolving communication standards.
Owing to advances in CMOS circuit performances, digital techniques are today able of handling Intermediate Frequency (IF) or even low Radio Frequency (RF) tasks.
Several companies are working in the direction to implement the largest part of their transceivers with a digital approach, using analogue design techniques only for RF front-ends.
Even though the technology is ready to face the challenge of the implementation of digital transceiver, the design techniques still lack in many aspects. For this reason in the last years several technical papers have been published on the issues related to the efficient implementation of digital building blocks for telecommunication systems.
The field of digital electronics for telecommunications is very wide. Nevertheless some specific characteristic are highly desirable in almost all the digital circuits used in the modern transceiver.
The first characteristic is the speed. In order to operate at IF, digital circuits for telecommunications must be able to operate with clocks running at frequencies of hundreds of megahertzs.
The second important characteristic is the power dissipation. As a matter of fact, due to the high diffusion of wireless transceivers (used

in notebooks, PDAs, cell phones, etc.) the power dissipated by every building block of the transceiver must be as low as possible.
In order to match these two requirements, the design of a high-performance building block for telecommunication systems must go through subsequent optimization stages.
At the highest level, the analytical description of the functionality of the circuit must be optimized in order to reduce the hardware architecture needed to implement the computation. This is possibly the main difference between the building blocks proposed in *Literature*. An efficient definition of the analytical computation in a digital domain can indeed highly reduce the power dissipation of the circuit and increase its operating speed.
Gate and transistor level design techniques can be developed to further improve the performance of the circuits.
As a final note, since the emerging possibilities of digital VLSI circuits are related to the advance in technology, experimental verification of the performance of a digital building block for telecommunication systems is not an option. It is hence important to verify the performance of a digital circuit for telecommunication through experimental analysis of fabricated prototypes.

1.2 Research Topics

On the basis of the emerging trends in telecommunications, my research activity is based on the design and the optimization of digital building blocks for telecommunication systems. The field of telecommunication circuits is very wide. My Ph.D. research activity has been focused on two main topics.
The first topic is the efficient design of digital circuits for signal processing at intermediate and base band. In this category I have worked to the development of new architectures for Direct Digital Frequency Synthesizers (DDFSs), Direct Digital Frequency Synthesizers/Mixers (DDFSMs), Cartesian to Polar coordinates converters and Interpolators for digital modems. These are the main blocks for the processing of the signals elaborated by the analogue front-end. They are used to generate sinusoids waveforms, to make up/down conversion, to change the sample frequency in the digital

domain and to implement a large family of computations used in digital modulation schemes.
The second topic is the development of new architectures for Reed-Solomon Decoders. Even though the scheme of the Reed-Solomon decoding procedure has been developed more than forty years ago, efficient implementation of the decoding circuit and its sub-circuits is still a topic of main concern.

1.3 Thesis Outline

This thesis is organized as follows.
The chapter two synthesizes my research activity focused on the development of efficient architectures for signal processing. The chapter two is divided in four sections focused on the implementation of DDFSs, DDFSMs, Cartesian to Polar converter and Digital Interpolator respectively.
The chapter three synthesizes my research activity on the implementation of Reed-Solomon decoders. The first section of the chapter describes a new architecture for Galois fields multipliers, whereas the second section describes a new architecture for Reed-Solomon decoder.
In the chapter four the results obtained in the development of high-performance Flip-Flop and Truncated multipliers is discussed. These two circuits are not used only in the design of building blocks for telecommunications, but are of main relevance in the design of digital VLSI circuits. Nevertheless, the results achieved in the development of the two mentioned circuits had a high impact on the development of the other circuits.

1.4 PUBLICATIONS

- A.G.M. Strollo, N. Petra, D. De Caro, “Dual-tree error compensation for high performance fixed-width multipliers”, *IEEE Transactions on Circuits and System II*, vol.52, no.8, pp.501-507, Aug.2005.

- A.G.M. Strollo, D. De Caro, E. Napoli, N. Petra, “A Novel High-Speed Sense Amplifier based Flip-flop”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vo.13, no.11, pp.1266-1274, Nov.2005.
- A.G.M. Strollo, N. Petra, D. De Caro, E. Napoli, “Fixed-width Multipliers with Dual-tree Error Compensation”, 16th *European Conference on Circuits Theory and Design* (ECCTD 2003), Krakow (Poland), Sept.1-4 2003.
- A.G.M. Strollo, D. De Caro, N. Petra, E. Napoli, “True Random Number Generator based on LFSR sampling”, 16th *European Conference on Circuits Theory and Design* (ECCTD2003), Krakow (Poland), Sept.1-4 2003.
- A.G.M. Strollo, N. Petra, D. De Caro, E. Napoli, “VLSI Design of a (255,239) Reed-Solomon Decoder”, 16th *European Conference on Circuits Theory and Design* (ECCTD 2003), Krakow (Poland), Sept.1-4 2003.
- A.G.M. Strollo, D. De Caro, E. Napoli, N. Petra, “Direct Digital Frequency Synthesis with Dual-slope Approach”, 29th *European Solid-State Circuits Conference* (ESSCIRS 2003), Estoril (Portugal), Sept. 16-18 2003.
- A.G.M. Strollo, D. De Caro, E. Napoli, N. Petra, “High-speed Direct Digital Frequency Synthesizers in 0.25-um CMOS”, Proc. of *IEEE Custom Integrated Circuits Conference* (CICC2004), Orlando (USA), Oct.3-6 2004.
- A.G.M. Strollo, N. Petra, D. De Caro, E. Napoli, “An Area-Efficient High-Speed Reed-Solomon Decoder in 0.25um CMOS”, 30th *European Solid-State Circuits Conference* (ESSCIRC 2004), Leuven (Belgium), Sept. 20-24 2004.
- D. De Caro, E. Napoli, N. Petra, A.G.M. Strollo, “A High-Speed Sense-Amplifier based Flip-flop”, 30th *European Conference on Circuits Theory and Design* (ECCTD 2005), Cork (Ireland), 2005.
- D. De Caro, N. Petra, A.G.M. Strollo, “ A 630MHz Direct Digital Frequency Synthesizer with 90dBc SFDR in 0.25um CMOS”, Proc. of *IEEE International Solid-State Circuit Conference* 2006, San Francisco, USA, pp.256-257, Feb.2006.
- D. De Caro, N. Petra, A.G.M. Strollo, “A 380MHz, 150mW Direct Digital Synthesizer/Mixer in 0.25um CMOS”, Proc. of

IEEE International Solid-State Circuit Conference 2006, San Francisco, USA, pp.258-259, Feb.2006.

- D. De Caro, N. Petra, A.G.M. Strollo, “A 380MHz Direct Digital Synthesizer/Mixer with Hybrid CORDIC Architecture in 0.25um CMOS”, *IEEE Journal of Solid State Circuits*, Volume 42, Issue 1, January 2007.
- A.G.M. Strollo, D. De Caro, N. Petra, “A 630MHz, 76mW, Direct Digital Frequency Synthesizer Using Enhanced ROM Compression Technique”, *Journal of Solid State Circuits*, Volume 42, Issue 2, February 2007.

Chapter 2
Signal Processing

2.1 Direct Digital Frequency Synthesis

Direct Digital Frequency Synthesizers (DDFSs) compute single-phase or quadrature sinusoids with excellent frequency resolution, good spectral purity, very fast frequency switching and phase continuity on switching [1]-[4]. Typical applications include modern communication systems (including spread-spectrum and frequency hopping systems) and measurement instrumentations.
As shown in Fig. 2.1, a quadrature DDFS is basically composed by the series of a phase accumulator and a sine/cosine generator. Analog outputs, when needed, are obtained by using DACs followed by low-pass reconstruction filters.
The phase accumulator is an overflowing *N*-bit accumulator that produces a digital sweep with a slope imposed by the value *FCW* of the frequency control word. The frequency f_{out} of generated signals is proportional to *FCW* and is given by

$$f_{out} = \frac{FCW}{2^N} f_{clk}; \qquad 0 \leq FCW \leq 2^{N-1} \qquad (2.1)$$

where f_{clk} is the clock frequency.
The most critical block in a DDFS is the sine/cosine generator. In the simplest implementation, the output of the accumulator addresses a read only memory (ROM). The ROM implements a big lookup table storing *R*-bit digitized sine and cosine waveforms. To reduce the ROM size, the phase value passed to the sine-cosine generator is normally truncated to *P*-bits. Phase truncation introduces spurious

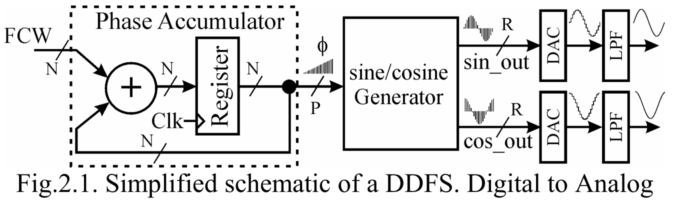


Fig.2.1. Simplified schematic of a DDFS. Digital to Analog Converter and Low-Pass Filter are included when analog output are needed.

noise in the DDFS outputs, and the P value is chosen [5]-[7] according to the required Spurious Free Dynamic Range (SFDR). The lookup table size is also typically reduced by storing sine and cosine values only for angles in $[0, \pi/4)$. Output values for the full range $[0, 2\pi)$ of input phase are generated by exploiting the quarter-wave symmetry of trigonometric functions and trigonometric identities. Using both phase truncation and quarter wave symmetry, the total ROM size is: $(1/4)R \cdot 2^P$ bits. This value is usually prohibitive for high-speed and low-power implementations. For instance, a DDFS with 90dBc SFDR uses R=13 and P=16, with a total ROM size larger than 2.1×10^5 bit.
For this reason, several alternative approaches for the implementation of the sine/cosine generator have been proposed. A comprehensive review has been recently published in [8]. Roughly speaking, proposed algorithms can be subdivided in three categories.
Angle rotation techniques (including CORDIC algorithm and its modifications) [9]-[14], basically, start from a vector in the complex plane for which sine and cosine values are known, and proceed with coordinate rotations until an angle sufficiently close to the desired angle is reached. These techniques use very small look-up memories, but require complex arithmetic circuitry. Some of the angle rotation techniques [12]-[14], give the possibility to realize a Direct Digital Synthesizer/Mixer (DDFSM). A DDFSM rotates an input vector in the complex plane by an angle linearly increasing with time and is, therefore, able to modulate both the output frequency and amplitude. A DDFSM reduces to a DDFS when the input vector is kept constant. However, if only frequency modulation is required, a DDFSM circuit, also when realized in a very effective way [14], results in much larger silicon area and power dissipation when compared to optimized DDFS circuits. Optimized angle rotation techniques for DDFS implementation are proposed in [9]-[11]. The solutions of [9], [10], being based on the CORDIC algorithm, require a large number of cascade rotation stages, and therefore result inherently slow in comparison to other approaches. A state of the art angle rotation technique optimized for DDFS implementation is presented in [11]. In this approach the circuit latency is reduced by employing only two multiplier-based rotation stages.
In polynomial and piecewise polynomial interpolation architectures [15]-[19] a small ROM is used to store polynomial coefficients while

additional arithmetic hardware (multipliers, squarers etc.) is required to implement the polynomial approximation. In some approaches the ROM is eliminated altogether, by using high-order polynomial approximations with hardwired coefficients [16]. However, piecewise linear approximation with optimized coefficients seems a better approach when high speed is required [15], [19]. In fact in this approach the required ROMs can be effectively implemented as random logic, while two simple multiplier-accumulators are required to compute the sine and cosine outputs.

Angular decomposition ROM compression techniques use approximations in which the lookup table storing sine values is subdivided in two smaller parts (a “coarse” ROM and a “fine” ROM). The outputs of the coarse and fine ROMs are added together to yield the final sine/cosine values. One of the most effective and more popular algorithms was developed by Nicholas [20]. The DDFS recently presented in [21] uses an improved approach, in which the total ROM size is further reduced by decomposing both coarse and fine ROMs as the sum of an “error” ROM and a “quantization” ROM.

In this section ([22], [35]) we will introduce the developed DDFS architecture based on the recently proposed Multipartite Table Method [23]. This method has been found ideally suited for high-performance synthesizers, requiring both very small lookup tables and simple arithmetic circuitry. The algorithm generalizes the Nicholas technique by decomposing the lookup table in $K \geq 2$ small ROMs, whose outputs are added together to obtain the final result. The content of the ROMs is calculated by optimizing the SFDR. The DDFSs designed with the proposed technique require only small lookup tables and simple multi-operand adders. The circuit developed reaches 90dBc SFDR while using a total ROM size less than 1400 bits. The small ROMs are effectively implemented as random logic, while pipelined tree-based multi-operand adders are employed. An operating frequency of 630MHz was obtained by using a standard-cell design in a 2.5V, 0.25μm CMOS. In order to reduce the power dissipation, we employed a power-driven synthesis and included two flip-flop topologies (with different power and delay performances) in the standard cell library. In this way, a total power dissipation of 76mW at 630MHz was achieved. By reducing the power supply at 1.8V, a maximum operating frequency of 430MHz was measured, with a total power dissipation as low as 24.9mW.

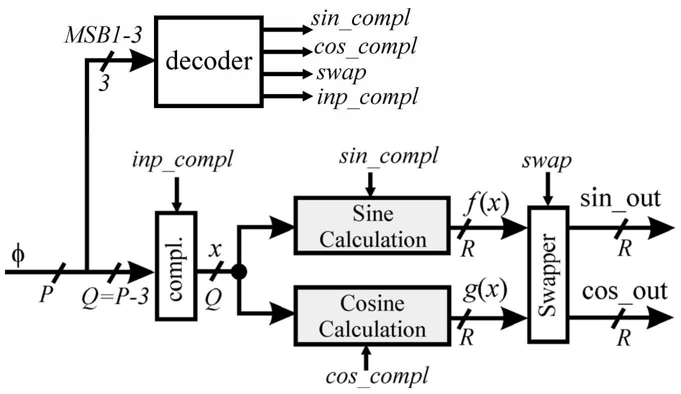


Fig.2.2. Sine/cosine generator architecture.

2.1.1 ROM Compression Algorithm

The architecture of the sine/cosine generator block is shown in Fig.2.2. The P-bits input signal ϕ represents the input phase $[0,2\pi)$. The signal x (obtained from the $Q=P-3$ less significant bits of ϕ) represents an angle in $[0,\pi/4)$, scaled to a binary fraction in $[0,1)$. The two blocks "sine calculation" and "cosine calculation" in Fig. 2.2 are the heart of the DDFS, and compute:

$$f(x) = Z\sin\left(\frac{\pi}{4}x\right);\quad g(x) = Z\cos\left(\frac{\pi}{4}x\right) \qquad (2.2)$$

where Z is the maximum amplitude of the generated signals, given by: $2^{R-1}-1$. The three most significant bits of ϕ determine the octant in which the input phase lies and are input of a decoding logic. The output of the decoding logic are four signals that control the complementing of x, $f(x)$, $g(x)$ and the swapping between $f(x)$ and $g(x)$, needed to properly reconstruct sine and cosine waveforms [1].

In our circuit the sine and cosine calculation blocks of Fig.2 have been implemented with a Multipartite Table approximation. In order to introduce the Multipartite Table Method (MTM), which is described in detail in [23], let us focus on the sine calculation block (which approximates $f(x)$ function) in Fig.2.2. In MTM the Q-bit input signal x is decomposed in $K+1$ non overlapping sub-words: x_0, x_1, x_K of lengths q_0, q_1... q_K respectively. Hence the value of the input operand is: $x = x_0 + x_1 + \ldots + x_K$ and the length is: $Q = q_0 + q_1 + \ldots + q_K$. A piecewise linear approximation of $f(x)$ can be written as:

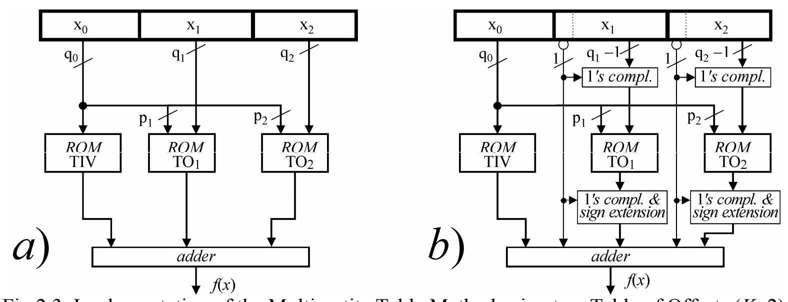


Fig.2.3. Implementation of the Multipartite Table Method using two Table of Offsets (*K*=2).
a) Straightforward implementation of the MTM for *K*=2.
b) Reduced complexity implementation obtained exploiting symmetry of the TOs values.

$$\begin{aligned} f(x) &= f(x_0 + x_1 + ... + x_K) \\ &\simeq A(x_0) + B(x_0)\cdot(x_1 + ... + x_K) \\ &= A(x_0) + B(x_0)\cdot x_1 + ... + B(x_0)\cdot x_K \end{aligned} \qquad (2.3)$$

where the interval [0,1) of *x* has been divided in 2^{q_0} subintervals, the quantity x_0 represents the starting point of each subinterval and x 1+...+x_K is the offset in each interval between *x* and x_0.
The term $B(x_0)\cdot x_1$ is then approximated as $B_1(\xi_1)\cdot x_1$, where ξ_1 is a sub-word of x_0 including its $p_1 \le q_0$ most-significant bits. Likewise the term $B(x_0)\cdot x_2$ is approximated as $B_2(\xi_2)\cdot x_2$, where ξ_2 is a sub-word of x_0 including its $p_2 \le p_1$ most-significant bits. Similar approximation can be done for the terms $B(x_0)\cdot x_i$ (*i*=3...*K*). The equation (2.3) becomes:

$$f(x) \simeq A(x_0) + B_1(\xi_1)\cdot x_1 + B_2(\xi_2)\cdot x_2 + ... + B_K(\xi_K)\cdot x_K \qquad (2.4)$$

In this equation, the term $A(x_0)$ is realized with a ROM (named Table of Initial Values, TIV) with 2^{q_0} entries. Similarly, the terms $B_i(\xi_i)\cdot x_i$ (*i*=1,…,*K*) are implemented with *K* ROMs (Table of Offsets, TO*i*) with $2^{p_i+q_i}$ entries each (see Fig.2.3*a*). In [23] it is shown that the Tables of Offsets can be made symmetric. In this way the Tables of Offsets can be reduced in size by a factor of two, at the expense of a few XORs, as shown in Fig.2.3*b*.
The values to be stored in the ROMs are obtained by using the min-max approach (see [23]). For completeness, the formulas to calculate ROMs content and size are reported in the Appendix A, by particularizing the approach of [23] to DDFS application.
For a given value of *K*, the optimal decomposition $\mathcal{D}_f = \{q_0, q_1, p_1, ..., q_K, p_K\}$ is defined in [23] as the one that allows to

minimize the total memory size, while fulfilling the accuracy requirement known as *faithful rounding* (the returned result is one of the two fixed-point numbers closest to the exact value of $f(x)$).
For a quadrature DDFS two (possibly different) decompositions should be obtained for the two functions $f(x)$ and $g(x)$ in (2.2). We indicate as $\mathcal{D}_T = \{\mathcal{D}_f, \mathcal{D}_g\}$ the total DDFS configuration. Moreover, faithful rounding is not required in DDFS, where the error metric is the SFDR. As shown in [15] locally increasing the approximation error may, in some cases, result in improved spurious performances. Moreover, relaxing the specification on the local approximation error, a substantial memory saving can be achieved with respect to a faithful rounding approach (see Tab.2.2 in the following).
Therefore we developed a novel algorithm to find the optimal decomposition $\mathcal{D}_T = \{\mathcal{D}_f, \mathcal{D}_g\}$ which minimizes the ROM size while achieving a specified SFDR. Our algorithm, shown in Fig.2.4, considers only the case in which the same value of K is used in both $\mathcal{D}_f$ and $\mathcal{D}_g$. Moreover it assumes that, for a given decomposition, the lookup tables' content is evaluated in order to minimize the maximum approximation error (see appendix A). The algorithm includes three main steps. A time-consuming search is performed in the final step, while the first two steps of the algorithm are aimed to reduce the search space. Two parameters are introduced in the first two steps. The *maxROM* parameter limits the maximum ROM size of each considered decomposition. Since the algorithm is aimed to find the solution with the minimal ROM size, the parameter *maxROM* does not affect the final calculated solution and is included only to reduce the search space. The second parameter is the maximum approximation error ε_{max} of each considered decomposition. This parameter allows not only to reduce the search space, but also to find solutions which match a constraint on both the SFDR and the maximum approximation error.
In the first step of the algorithm, for the prescribed K value, all the possible decompositions for the input x are enumerated. Each decomposition which is acceptable to implement $f(x)$ is saved. A decomposition $\mathcal{D}_f$ is considered acceptable if the approximation error is smaller than ε_{max} and the size of the ROM for the computation of $f(x)$ is smaller than $0.75 \cdot maxROM$. Similarly, each decomposition $\mathcal{D}_g$

```
set Nsin=0; Ncos=0; Nt=0;
enumerate all the N possible input decomposition $\mathcal{D}(i)$;
for h in 1 to N
  if $\mathcal{D}(h)$ is acceptable for f(x) then
  (the decomposition is considered acceptable if the approximation error
  is smaller than $\varepsilon_{max}$ and the ROM size is smaller than 0.75 $\times$ maxROM)
      set: Nsin = Nsin +1; $\mathcal{D}_f(Nsin)=\mathcal{D}(h)$;
  end if
  if $\mathcal{D}(h)$ is acceptable for g(x) then
      set: Ncos = Ncos +1; $\mathcal{D}_g(Ncos)=\mathcal{D}(h)$;
  end if
end for
for i in 1 to Nsin and for j in 1 to Ncos
  if {$\mathcal{D}_f$(i), $\mathcal{D}_g$(j)} is acceptable for DDFS implementation then
  (the set of the two decompositions is acceptable if the total ROM size is
  smaller than maxROM and the values of q0, q1, q2... are the same in
  $\mathcal{D}_f(i)$ and $\mathcal{D}_g(j)$)
      set: Nt = Nt +1; $\mathcal{D}_T(Nt)$ = {$\mathcal{D}_f(i)$, $\mathcal{D}_g(j)$};
  end if
end for
sort $\mathcal{D}_T$ in ascending order of the total ROM size;
for i in 1 to Nt
  Calculate SFDR assuming $\mathcal{D}_T$ as DDFS configuration;
  if ( SFDR > target_SFDR – $\Delta$) then
      Perform "amplitude optimization";
      if (final_SFDR >= target_SFDR) then
          Optimal solution found; exit program
      end if
  end if
end for
```

Fig.2.4. Algorithm to determine the optimal MTM decomposition, for a given number of table of offsets. Input parameters are: *target_SFDR* (the required SFDR), ε_{max} (the maximum approximation error) and *maxROM* (the maximum ROM size).

acceptable to implement $g(x)$ is also saved[1]. In spite of the initially large search space, the computation time for this first step is reduced since the approximation error and the ROM size can be calculated according to [23], with simple formulas (see appendix A).

In the second step of the algorithm, all the possible DDFS configuration that can be realized by using the saved $f(x)$ and $g(x)$ decompositions are enumerated. Also in this case, each acceptable DDFS configuration is saved. A DDFS configuration is considered acceptable if the total ROM size (the sum of the size of the ROMs needed to implement both $f(x)$ and $g(x)$) is smaller than *maxROM* and, moreover, the values of q_0, q_1, q_K... are the same for $f(x)$ and $g(x)$. This second condition is imposed to simplify the DDFS hardware implementation, as will be detailed in the section 2.1.2.

[1] The value 0.75·*maxROM* has been obtained heuristically. In all our implementations, in fact, the cosine ROM size is always smaller than 0.75·*maxROM*. The same holds true for the sine ROM size.

K	$\mathcal{D}_f$	sine ROM	$\mathcal{D}_g$	cosine ROM	total ROM
1	$q_0=7$; $q_1=6$, $p_1=1$ $g=0$	1792	$q_0=7$; $q_1=6$, $p_1=3$ $g=0$	2432	4224
2	$q_0=5$; $q_1=3$, $p_1=3$ $q_2=5$, $p_2=2$; $g=0$	768	$q_0=5$; $q_1=3$, $p_1=5$ $q_2=5$, $p_2=1$; $g=0$	1056	1824
3	$q_0=5$; $q_1=2$, $p_1=3$ $q_2=3$, $p_2=3$ $q_3=3$, $p_3=1$; $g=0$	616	$q_0=5$; $q_1=2$, $p_1=5$ $q_2=3$, $p_2=2$ $q_3=3$, $p_3=1$; $g=0$	728	1344

Tab.2.1. Optimal decompositions for sine and cosine calculation blocks for a DDFS with 90 dBc SFDR, with R=13 and P=16;
g is the number of guard bits.

A final search is performed in the third step of the algorithm. For each saved DDFS configuration, starting from the one with the lowest total memory size and proceeding in ascending ROM size order, the values to be stored in the tables are calculated following [23] (see appendix A) and the SFDR is calculated by numerical simulation. The first solution found that meets the target SFDR is the optimal one, with the minimum ROM size. As can be seen in Fig.2.4, if the SFDR is near to the target value, amplitude optimization is carried-out to improve spectral purity. Amplitude optimization [20] consists in scaling the values to be stored in the lookup tables before rounding, to provide improved performance in the presence of amplitude quantization. To that purpose, a search is performed by varying the amplitude Z in (2.2) . For each trial Z value, the content of the lookup tables is calculated and the resulting SFDR is computed. The best amplitude value is selected as the one yielding the largest SFDR. The use of a very small step size (less than one LSB) during the search in Z guarantees a negligible reduction of the output signals amplitude. The value of Δ in Fig.2.4 has been chosen equal to 3dB, which is a SFDR gain which can be reasonably obtained with the amplitude optimization.
The Tab.2.1 shows the optimal DDFS configurations obtained by using one, two or three Tables of Offsets, for a 90dBc SFDR, with R=13 and P=16. Please note that the MSB of the Table of Initial Values of the cosine function is constant, and has not been considered in determining the ROM size values reported in Tab.2.1. As can be seen, even for K=1 the total DDFS ROM size is only a small fraction of the memory size (2.1×10^5 bits) that would be required by using an uncompressed lookup table. The ROM size decreases significantly using K=2, while the additional improvement obtained with K=3 is less evident.

From the above considerations it follows that increasing the number *K* of Tables of Offsets allows reducing the total memory size. However, any additional table requires the introduction of additional adder inputs and additional complementers, with a trade-off in terms of hardware complexity. Moreover, using more tables increases the approximation and rounding errors (see Appendix A), requiring the introduction of guard bits that may partly overcome the advantages in terms of memory size. By synthesizing many circuits we have found that using *K*=3 gives a good trade-off between ROM and arithmetic circuit complexity.
The Tab.2.1 also highlights that the optimal solution, given by the algorithm of Fig.2.4, provides two different decompositions for the sine and the cosine functions, with the sine ROM sensibly smaller than the cosine ROM. Actually the error introduced by the MTM approximation depends on the amplitude of the second derivative of the approximated function. Therefore, by using a different decomposition for *f*(*x*) and *g*(*x*), it is possible to obtain a similar approximation error for the two functions, while reducing the total ROM size.
The Fig.2.5 shows the approximation error for input angles in the first quadrant, for the 90dBc DDFS with *K*=3. As can be seen, the maximum absolute error is about 2.5 LSB. In spite of the fairly large error in the time domain, the Fig. 2.6 shows that the DDFS is able to reach the required SFDR. This characteristic is common in SFDR

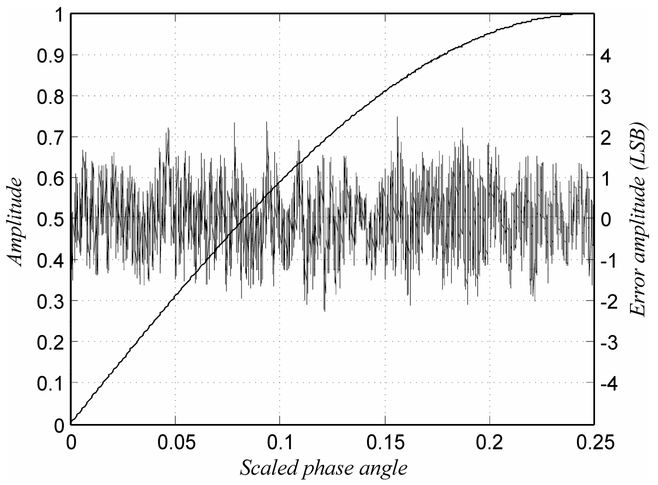


Fig.2.5. Approximated sine wave and amplitude error, for the 90dBc DDFS with *K*=3.

optimized DDFS (see [15], [18]).

In this paper, the sine/cosine tables are optimized in the sense of the SFDR metric. The original MTM [23] is based on the faithful rounding requirement. A comparison between the two approaches is presented in Tab.2.2. This comparison is done for R=13, P=16, considering a SFDR constraint (for the proposed approach) of 92.4dBc. This is the maximum possible SFDR for a DDFS with P=16 phase bits.

From Tab.2.2 it can be observed that the proposed approach results in substantial memory saving with respect to the faithful rounded MTM of [23]. The memory saving is close to 40% for K=1, and reaches about 50% for K=3. This improvement is due to the relaxed specification on maximum approximation error.

It is worth to highlight that the SFDR is a long-term averaged metric, while the maximum absolute error is a short-term characteristic. In same applications, especially when the digital output is directly used, it is important to have both long-term and short-term good error characteristics. Our algorithm allows designing DDFS circuits with excellent error characteristics and reduced memory size also in this case. This is possible owing to the presence of the parameter ε_{max} (see the algorithm of Fig.2.4) which allows designing DDFS circuits that meet a constraint on both SFDR and maximum absolute error.

The Tab.2.3 shows a comparison between the ROM size and the

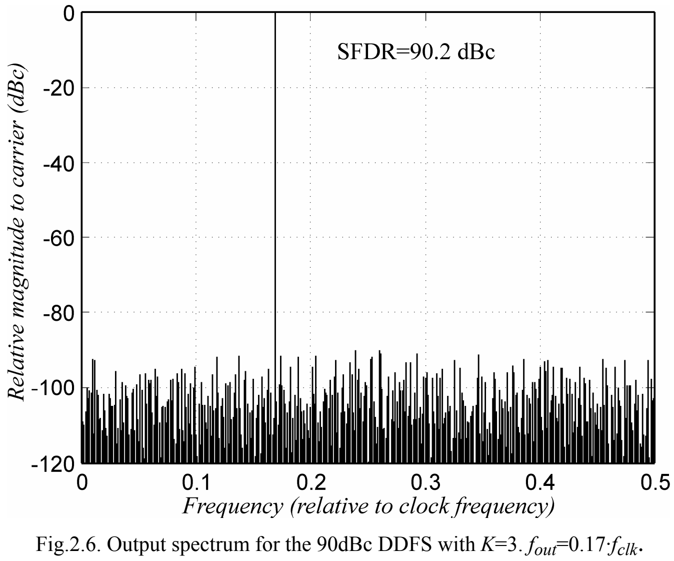


Fig.2.6. Output spectrum for the 90dBc DDFS with K=3. f_{out}=0.17·f_{clk}.

Approach	K	$\mathcal{D}_f$	$\mathcal{D}_g$	ROM size (bits)			SFDR (dBc)	max erro (LSB)
				sine	cosine	total		
This paper	1	q_0=7; q_1=6, p_1=2, g=0	q_0=7; q_1=6, p_1=3, g=0	2048	2432	4480	92.4	1.71
	2	q_0=6; q_1=2, p_1=3 q_2=5, p_2=2; g=0	q_0=6; q_1=2, p_1=4 q_2=5, p_2=2; g=0	1040	1024	2064	92.4	2.02
	3	q_0=5; q_1=2, p_1=4 q_2=2, p_2=2 q_3=4, p_3=0; g=1	q_0=5; q_1=2, p_1=5 q_2=2, p_2=3 q_3=4, p_3=1; g=0	704	752	1456	92.4	2.35
Faithful rounded [23]	1	q_0=8; q_1=5, p_1=2 g=1	q_0=8; q_1=5, p_1=3 g=1	3584	3840	7424	92.4	0.92
	2	q_0=6; q_1=3, p_1=4 q_2=4, p_2=2; g=2	q_0=6; q_1=3, p_1=5 q_2=4, p_2=3; g=2	1472	1856	3328	92.4	0.99
	3	q_0=6; q_1=2, p_1=5 q_2=2, p_2=2 q_3=3, p_3=1; g=2	q_0=6; q_1=2; p_1=5 q_2=3, p_2=3 q_3=2, p_3=1; g=2	1408	1412	2820	92.4	0.99

Tab.2.2. Comparison between DDFS circuits designed by using the proposed algorithm and the faithful rounded approach of [23].
In all circuits R=13 and P=16; g is the number of guard bits.

arithmetic circuits employed in several published DDFS architectures with SFDR values ranging from 80 to 100 dBc. As it can be seen, the optimized MTM approach compares favorably with CORDIC-based architectures that, while requiring less memory, are plagued by the need of many arithmetic circuits. A similar consideration applies to the ROM-less third-order polynomial approximation technique proposed in [16]. Piecewise-linear approximation techniques (represented by [15], [18] in Tab.2.3) need slightly less memory with respect to the proposed technique, but require multipliers (or equivalent circuitry) that easily becomes a speed bottleneck when high clock frequency is required.
The comparison of our DDFS with the ROM compression approaches highlights that the DDFS of [24] requires a 38% smaller ROM size. However the circuit of [24] achieves an 85dBc SFDR and needs two multipliers. The DDFS of [25] reaches a 6 dB larger SFDR in comparison to our circuit, while requiring a larger ROM size and, again, the need of two multipliers. A comparison with the recently proposed Yang technique [21] shows that the proposed approach requires reduced memory size, while using higher phase and amplitude resolutions.

Reference	Approx. technique	Phase Resolution, P	Amplitude Resolution, R	SFDR (dBc)	ROM (bits)	Output	Arithmetic Circuits
Proposed [35]	Multipartite Table with SFDR optimization, K=3	16	13	90.2	1344	quadrature	2 multi-operand adders
Madisetti [9]	Modified CORDIC	22	16	100	572	quadrature	$\pi/4$ multiplier, butterfly stages
Song [11]	Interpolation-based angle rotation	18	16	100	270	quadrature	6 multipliers, 6 adders
De Caro [16]	Third-order polynomial approximation	14	12	80	0	quadrature	Partial products gen. and sum.
Langlois [15]	Piecewise linear interpolation	18	14	96.2	1152	quadrature	2 multi-operand adders and multiplexers
De Caro [18]	Piecewise linear interpolation	14	12	83.6	896	quadrature	2 merged multiply-add units
Nicholas [20]	ROM compression: coarse-fine decomposition	15	12	90.3	3072	single phase	multi-operand adder (3 operands)
Curticapean [24]	ROM compression: sine addition formulas	14	12	85	832	quadrature	2 multipliers, 2 adders
Curticapean [25]	ROM compression: sine addition formulas	16	16	96	2304	quadrature	2 multipliers, 2 adders
Yang [21]	ROM compression: coarse-fine decomposition	14	12	~ 80	2176	single phase	multi-operand adder (6 operands)

Tab.2.3. Comparison between ROM size and arithmetic circuits in recently proposed DDFSs.

2.1.2 DDFS Architecture

Some optimizations can be carried out when implementing MTM-based DDFSs. As can be seen from the figures 2.2 and 2.3*b*, some of the address bits of the Tables of Offsets are computed through the cascade of two 1's complementers. These two complementers can actually be replaced by a single modified 1's complementer. Moreover, since our search algorithm imposes the same q_0, q_1, ..., q_K values for the $\mathcal{D}_f$ and $\mathcal{D}_g$ decompositions, an unique modified 1's complementer can be shared for the two blocks, with additional hardware saving. In the following we will focus on the implementation of the 90dBc DDFS, with K=3. The modified 1's complementer for this design, and the detailed architecture of the sine calculation block, are shown in Fig.2.7.

In the Multipartite Table approach the content of the tables of offsets (TOs) has to be conditionally added or subtracted from the value stored in the Table of Initial Values (TIV). The architecture of Fig.2.7 implements this operation by using the sign-extension prevention (SEP) technique [26], in order to reduce the word length of the terms to be summed in the multi-operand adder. In the SEP technique the conditionally complemented offset values (stored in TOs tables) are extended by using a single bit, and suitable sign extension prevention constant (SEPC) is summed to the final result. The SEPC is known beforehand, therefore the SEP technique is implemented by using a modified Table of Initial Values, storing the sum of the "original" Table of Initial Values plus the SEPC.

The SEP technique is also exploited in Fig.2.7 to conditionally 2's complement the output of the sine generator. This is usually performed by firstly performing a 1's complement and then adding 1 LSB:

$$-f = \overline{f} + 1 \tag{2.5}$$

However, it is also possible to compute the 2's complement by firstly subtracting 1 LSB and then performing a 1's complement:

$$-f = \overline{f - 1} \tag{2.6}$$

This is the approach employed in Fig.2.7. Subtracting 1 LSB from f is achieved by applying the SEP technique. Accordingly, the complement of the *sin_compl* signal is included in the final addition and the relevant SEPC is added to the modified Table of Initial Values. For this reason the output word length of the modified Table

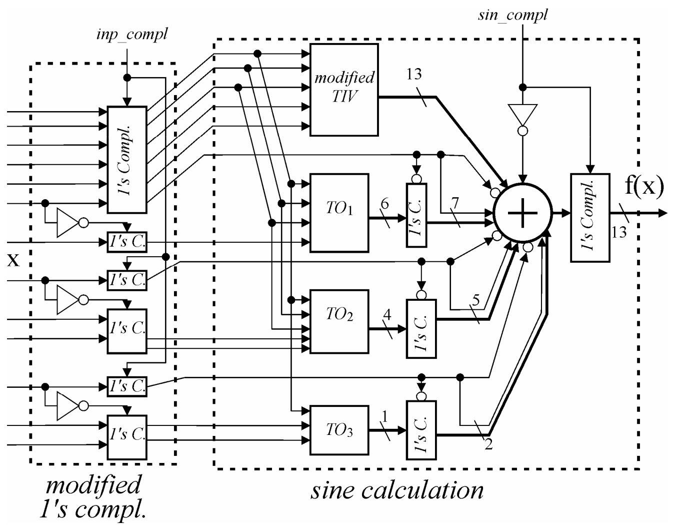


Fig.2.7. Schematic of the modified 1's complementer and of the optimized sine generator using sign-extension prevention technique.

of Initial Values is one bit larger than the original Table of Initial Values. This results in a slightly larger memory requirement with respect to the values reported in Tab.2.1 (for instance, the total sine ROM of the circuit in Fig.2.7 increases from 616 to 648 bit). This disadvantage is more than compensated by the simplification in the final multi-operand adder.

2.1.3 Power-Driven Flip-Flop Selection

The developed DDFS architecture can easily be pipelined to reach a high clock frequency. When a fine-grain pipeline is employed, the propagation delay of the few logic levels between the pipeline stages is comparable with the amount of clock cycle time taken by the flip-flops. Therefore using a high performance flip-flop is of outmost importance to increase the maximum clock frequency or reduce the number of pipelining flip-flops. When ultimate speed performances are required, using a flip-flop topology with reduced D-Q delay is mandatory and the power dissipation of the large number of pipelining flip-flops represents the main portion of the overall power dissipation. Real designs often include many flip-flops that are not on the critical path. This timing slack can be exploited by using slower, more energy-efficient, flip-flops on the non-critical paths, improving the overall power dissipation [30],[31].

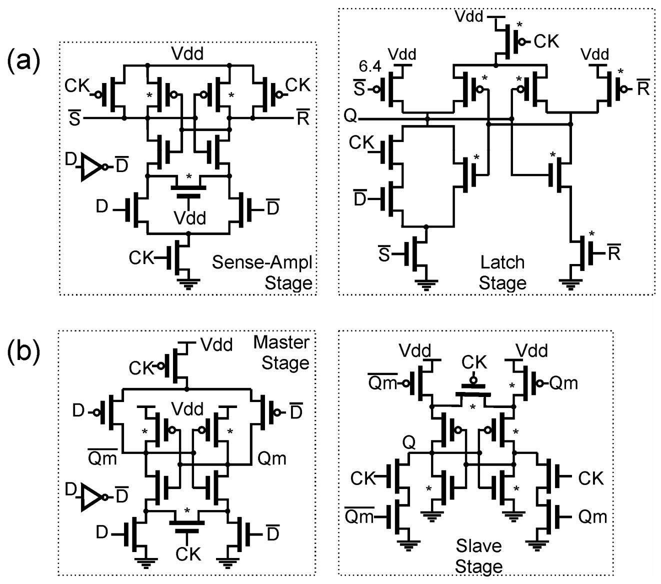


Fig.2.8. Schematic of the flip-flops used for the DDFS implementation.
(a) Imporved Sense-Amplifier based Flip-flop (SAFF);
(b) Static Ratio Insensitive Flip-flop (SRIS)
The asterisk (*) indicates the minimum sized devices.

A number of novel flip-flop topologies have been recently proposed and figure of merit like Power-Delay (or Energy-Delay) product are often used to determine the "best" topology [29]. When a single flip-flop topology is employed, the transistors sizing can be exploited to meet the timing requirements while optimizing the power dissipation.
Unfortunately this approach is not optimal for several reasons. First of all, an aggressing transistor downsizing worsens the Power-Delay product [29]. Moreover cells with a very low transistors sizing show a large dependence of the D-Q delay on the output capacitance. This makes timing closure very hard, especially in a standard cell approach, since the exact output parasitics values are known only after the detailed routing. Finally, there is no flip-flop topology with minimal Power–Delay product for any value of the input switching activity.
To overcome the limitation of the single flip-flop topology approach, in the design of our IC, we selected two different flip-flop topologies with different speed and power dissipation characteristics. The first flip-flop topology, shown in Fig.2.8a, is the improved Sense-Amplifier based flip-flop (SAFF) introduced in [32], [33] and optimized for a single-ended output. This circuit is one of the fastest

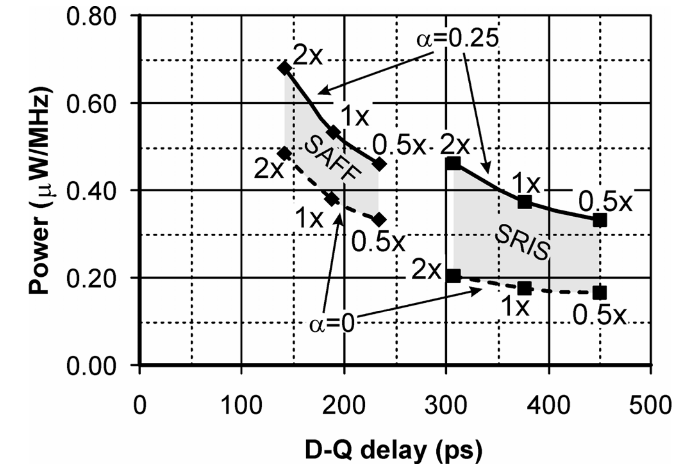


Fig.2.9. Flip-flop Power and D-Q Delay considering different sizing. SAFF is shown in Fig.8a, while SRIS is shown in Fig.8b.

sequential elements available today thanks to the setup time close to zero, given by the Sense-Amplifier stage, and the reduced clock to output delay provided by the Latch stage. Another advantage of this topology is the self-timed sampling window closing mechanism which gives a reduced hold time and an intrinsic robustness against sizing, process, voltage, and temperature variations. In the flip-flop of Fig.2.8a, the $\overline{S}$ and $\overline{R}$ exhibits a switching activity close to that of the clock signal, providing a large power consumption, especially in the case of a *D* input with a low switching activity.

The second flip-flop topology, shown in Fig.2.8b, is the Static Ratio Insensitive (SRIS) flip-flop of Yuan and Svensson [34]. This circuit presents very low power dissipation because of its non-precharged nature and the reduced clock load. The speed is the drawback of this topology because of the large setup time (due to the master/slave configuration), and the two delay stages of the slave latch.

The Fig.2.9 compares the speed and the power dissipation of the two flip-flops considering different circuit sizing (2x, 1x, 0.5x) for a 0.25μm technology. In this figure the switching activity α is defined as the ratio of the average *D* frequency and the clock frequency. For α=0.25 the SAFFs provide a Power-Delay product lower than the SRIS topologies. On the other hand, for α=0 the best Power-Delay product is obtained with the SRIS topology.

We have implemented the proposed 90dBc DDFS with *K*=3 by using a standard cell approach, with a 0.25μm, 2.5V technology. The DDFS

Employed Flip-flops	Area (10^3 μm^2)	clock freq. (MHz)	P_D (μW/MHz)
SRIS 2x	79.6	516	147
SAFF 2x	78.9	600	216
SAFF 2x/1x	65.5	600	176
SAFF 2x/1x/05x	59.5	600	157
SAFF 2x/1x/05x + SRIS 2x/1x/05x	63.2	600	130

Tab.2.4. Simulation results for the proposed 90dBc DDFS by employing different flip-flop topologies.

includes a 32-bit accumulator, composed by four pipelined stages, each one of 8 bits. The circuit has been synthesized for a target clock frequency of 600MHz.
The Tab.2.4 shows the performances achieved by synthesizing the proposed DDFS, with different flip-flops sizing and topologies. The required clock frequency is not achieved by using only SRIS flip-flops. By using only SAFFs, the target clock frequency is reached and a reduction of 27% in the power dissipation is observed when several flip-flop sizing are used. By using the two flip-flop topologies together a further 17% power reduction can be obtained without any speed penalty.

2.1.4 VLSI Implementations Results

The 90dBc DDFS circuit has been fabricated on a test chip, see Fig.2.10. The circuit includes a built-in self-test structure (BIST Logic) to make easier the measurements of DDFS maximum clock frequency and power dissipation.
The Tab.2.5 reports the experimental performances of the circuit. The DDFS exhibits a maximum operating frequency of 630MHz, with a

Technology	0.25μm 2.5V 5M 1P	
Die Area	0.063 mm^2	
Frequency Control Word	32 bit	
Output Resolution	13 bit	
SFDR	90 dBc	
Maximum Clock Frequency	630 MHz 430 MHz	@ 2.5V @ 1.8V
Frequency Resolution	0.15 Hz	@ 630 MHz
Power Dissipation	76.2 mW 24.9 mW	@ 2.5V - 630 MHz @ 1.8V - 430 MHz

Tab.2.5. Experimental performances of the proposed IC.

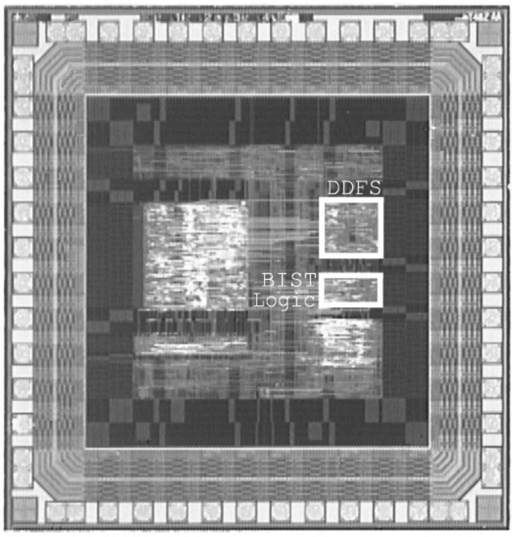


Fig.2.10. Test chip micrograph.

total power dissipation of 76mW at 630MHz. By reducing the power supply at 1.8V, a maximum operating frequency of 430MHz was measured, with a total power dissipation of only 24.9mW.
The Fig.2.11 shows the simulated power breakdown of the proposed DDFS. It is interesting to observe that the flip-flop power dissipation is close to the 60% of the total power. The power dissipation of the accumulator is not negligible, while the sine and cosine computation logic requires about the 56% of the total power. The power contribution of the lookup tables is very low (9%).
The data shown in Tab.2.6 compare the performances of the developed DDFS with the ones of some recently published state of the art circuits, using CMOS technology, SFDR and clock frequency similar to the ones considered in this paper. The developed DDFS uses only a fraction of the area of previous circuits. Comparing with our

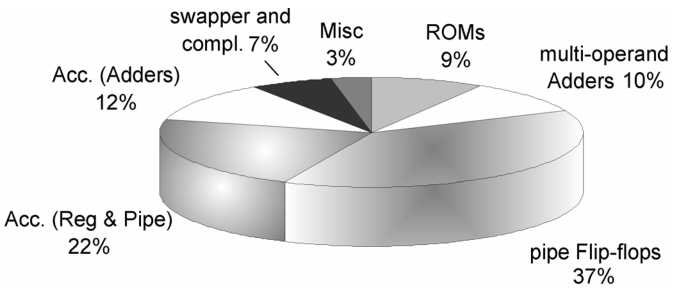


Fig.2.11. Power dissipation Breakdown.

Circuit	Technique	Accum. (bit)	SFDR (dBc)	Pipeline levels	Output	Process (μm)	Area (mm^2)	fclk max (MHz)	Power (μW/MHz)
This paper	Multipartite table with SFDR opt.	32	90	5	13 bits quadrature	0.25	0.063	630	121
Strollo [19] *JSSC 2005*	Dual-slope Piecewise linear (128 segments)	24	80	6	12 bits quadrature	0.25	0.090	600	120
Langlois [27] *CICC 2003*	Piecewise linear (32 segments)	32	84	4	12 bits quadrature	0.18	0.090	150	600
Song [11] *TVLSI 2004*	Interpolation based angle rotation	32	100	9	16 bits quadrature	0.35	1.400	150	2333
Song [28] *JSSC 2004*	Piecewise linear (128 segments)	30 $\Sigma\Delta$	110	-	14 bits single	0.25	0.120	250	400
Yang [21] *JSSC 2004*	ROM Compression: quad-line approximation	32	~ 60	26	9 bits single	0.35	0.440	820	153
De Caro [14] *JSSC 2007*	Hydrid-CORDIC DDFSM circuit	32	90	5	13 bits quadrature	0.25	0.220	385	400

Tab.2.6. Comparison with recently proposed DDFS IC realizations.

previous implementation of [19], this novel IC allows maintaining about the same maximum clock frequency and power dissipation, while providing increased SFDR and a larger accumulator. The DDFS of [21] dissipates 25% more power and is faster than our IC. However, the output resolution and the SFDR of [21] are lower than our design. Moreover two parallel sine generators are used in [21] to increase the maximum clock frequency, with a large area penalty. The proposed IC exhibits a reduction of the power dissipation by a factor larger than 3 and a substantial increase of the maximum clock frequency, with respect to the solutions proposed in [27]-[28].
The last row in Tab.2.6 corresponds to the DDFSM circuit that we have proposed in [14]. The Tab.2.6 highlights that a state of the art DDFSM provides significantly lower performances with respect to optimized DDFS circuits. This is the price to be paid to have both frequency and amplitude output modulations.

2.1.5 Conclusions

The implementation of a high performance DDFS IC based on a Multipartite Table Method has been described. The circuit has been optimized at the architectural and transistor levels. At the architectural level, for the first time, the Multipartite Table approach has been employed to implement a DDFS. At the transistor level two different flip-flop topologies with different power and delay characteristics are used to optimize the circuit performances. The implemented DDFS exhibits large clock frequency and reduced power dissipation.

2.2 Digital Mixer

The Direct Digital Frequency Synthesizer/Mixer (DDFSM) is in ubiquitous use for many communication subsystems such as tuners, derotators, up and down frequency converters etc.. In addition, the quadrature mixer is the front-end of various modulation/demodulation schemes, such us BPSK (Binary Phase Shift Keying), QPSK (Quadrature Phase Shift Keying) and QAM (Quadrature Amplitude modulation).

The inputs of a DDFSM are two signals X_{in} and Y_{in}, and a frequency control word f_{CW}. The outputs of the system are computed according to the following equations:

$$\begin{cases} X_{out}(n) = X_{in}(n)\cdot\cos\left[\theta(n)\right] - Y_{in}(n)\cdot\sin\left[\theta(n)\right] \\ Y_{out}(n) = X_{in}(n)\cdot\sin\left[\theta(n)\right] + Y_{in}(n)\cdot\cos\left[\theta(n)\right] \end{cases} \tag{2.7}$$

where:

$$\theta(n) = \omega\cdot n = 2\pi\cdot f_{CW}\cdot n \tag{2.8}$$

The equations (2.7),(2.8) correspond to a complex multiplication between an input vector in the complex plane, with coordinates $[X_{in}(n), Y_{in}(n)]$, and a unitary modulus vector $e^{j\theta}$: $X_{out} + jY_{out} = (X_{in} + jY_{in})\,\mathrm{e}^{j\theta}$.

A first implementation for the DDFSM includes two distinct functional units [55], see Fig.2.12a. The first one is a Direct Digital Frequency Synthesizer (DDFS) [56], [57] that generates the sequences $\sin(2\pi\cdot f_{CW}\cdot n)$ and $\cos(2\pi\cdot f_{CW}\cdot n)$. The second one is a complex multiplier, which uses four real multipliers, one adder and one subtractor to generate the outputs $X_{out}(n)$ and $Y_{out}(n)$ according to (2.7) . This implementation is generally non-optimal [58], [62]. The DDFS is in fact a cumbersome circuit itself. Moreover, the complex multiplier does not exploit the property that the modulus of one of the inputs ($e^{j\theta}$) is unitary.

A second possible implementation [59], [60] employs a simple overflowing accumulator that generates the angle θ and a rotator using the CORDIC algorithm [61] to implement the equations (2.7), see Fig.2.12b. Unfortunately, the CORDIC algorithm in its standard

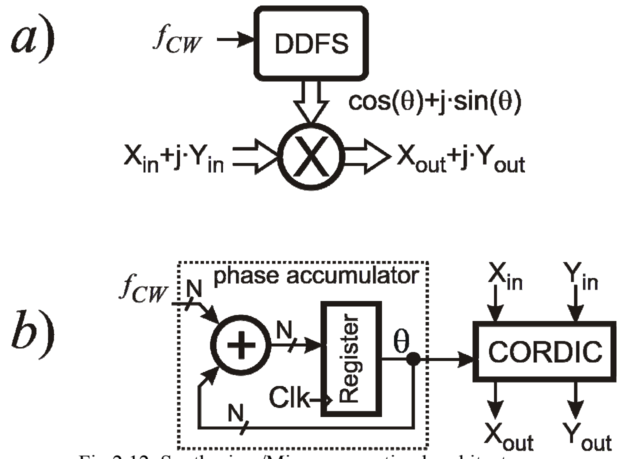


Fig.2.12. Synthesizer/Mixer non-optimal architectures:
a) DDFS based architecture
b) CORDIC based architecture

implementation is inherently slow, using many cascade computation stages.

The recent approaches [62]-[65] overcome the limitations of the simple architecture of Fig.2.12 by implementing the Synthesizer/Mixer as the cascade of two stages: a “coarse angle rotation” followed by a “fine rotation stage”. In [62]-[64] both the coarse rotation and the fine rotation employ a multiplier based architecture, while the approach of [65] uses a CORDIC architecture for the coarse rotation and a multiplier based fine rotation. The IC implementations [62], [64] are very effective, with a high speed operation and reduced hardware complexity. Until now, no IC implementation exists of the mixed approach of [65].

This section ([66], [89]) introduces a novel combined approach, named Hybrid CORDIC, to realize a Synthesizer/Mixer. This approach splits the rotation required in the Synthesizer/Mixer circuit in three rotations. A first rotation is performed by employing a CORDIC datapath in which the rotation directions are computed in parallel, by employing a lookup table. The second rotation is also CORDIC based, with rotations directions computed in parallel analytically. The final (third rotation) is multiplier based.

The parallel evaluation of the rotations directions allows an efficient use of the Carry Save arithmetic in the CORDIC datapath of the first two rotation blocks, without requiring additional Carry Propagate adders (as in [73],[74]) or the introduction of additional CORDIC sub-rotations (as in [75]). The final multiplier based rotation allows

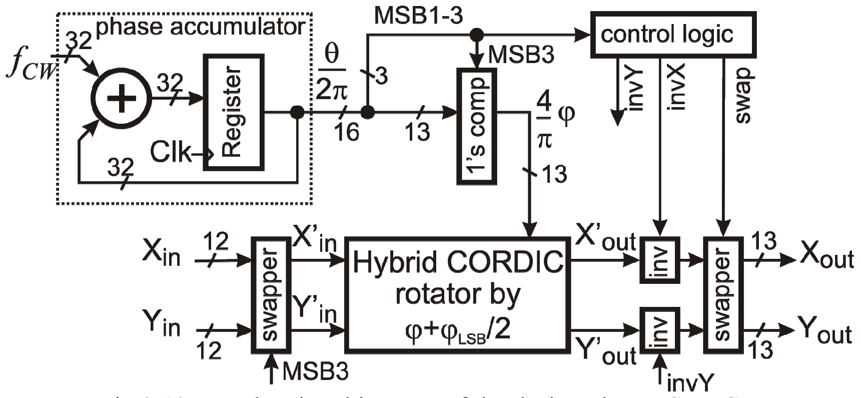


Fig.2.13. Top-level architecture of the designed DDFSM IC. φ_{LSB} is given by $(\pi/4)\cdot 2^{-13}$.

reducing the overall number of pipelining levels and the circuit complexity as well.

At the transistor level, a novel approach, which combines full-CMOS and Double-Pass-Transistor logic (DPL) [84] design styles, is presented to implement the CORDIC datapath.

2.2.1 Synthesizer/Mixer Basic Architecture

The top-level architecture of the designed DDFSM IC is shown in Fig.2.13. The circuit is sized in order to exhibit a 90dBc Spurious Free Dynamic Range (SFDR). The input word-length is 12 bit while output word-length is 13 bit. The 32 bit phase accumulator generates the rotation angle $\theta\in[0,2\pi]$, represented with a binary fractional value in [0,1]. The rotation angle θ is truncated to 16 bit, introducing output spurs that are below the 90dBc SFDR constraint. The hearth of the circuit is the Hybrid CORDIC rotator block. This block is able to perform a rotation by an angle $\varphi\in[0,\pi/4]$ represented with a binary fractional value in [0,1]:

$$\frac{4}{\pi}\varphi = f_1\cdot 2^{-1} + \ldots + f_{13}\cdot 2^{-13} \qquad (2.9)$$

The less significant bit of φ has a weight that will be indicated in the following as $\varphi LSB=(\pi/4)\cdot 2^{-13}$.

The other minor subsystems in Fig.2.13 (1's complementer, swappers and 2's complementers controlled by control logic) employ the sine and cosine functions symmetries [62], [64] to perform the complete rotation in the full $[0,2\pi]$ interval. It is worth to highlight that the

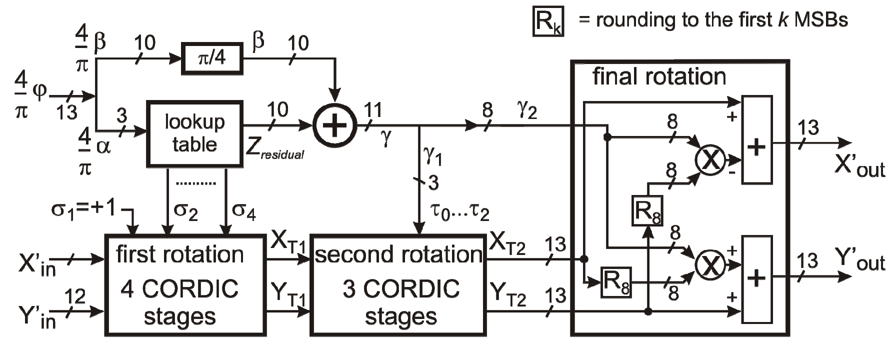


Fig.2.14. Hybrid CORDIC rotator architecture.

introduction of a φLSB/2 phase shift in the rotator block, allows to completely eliminate [56], [57], [70] the error due to the employ of a simple 1's complementer to evaluate the angle φ.

2.2.2 Hybrid cordic rotator algorithm

The architecture of the Hybrid Cordic rotator is shown in the Fig.2.14. The circuit rotates its input vector $[X'_{in}, Y'_{in}]$ by the angle $\varphi+\varphi_{LSB}/2$. The rotation is performed in three steps. The first two steps are performed with a CORDIC datapath, while the final step is realized by using two multipliers.

2.2.3 First rotation

In the first step, the angle φ is divided in two sub-words: $\varphi = \alpha + \beta$, where:

$$\alpha = \left(f_1 \cdot 2^{-1} + \ldots + f_3 \cdot 2^{-3} + 2^{-4}\right) \cdot \frac{\pi}{4} \qquad (2.10)$$

$$\beta = \left(-\overline{f_4} \cdot 2^{-4} + \ldots + f_{13} \cdot 2^{-13}\right) \cdot \frac{\pi}{4} \qquad (2.11)$$

and $\overline{f_4}$ is the complement of f_4.

The goal of the first stage is to perform a rotation by an angle close to $\alpha+\varphi_{LSB}/2$. To that purpose, the *first rotation* block uses the CORDIC algorithm, described by the following equations:

$$\begin{cases} X_{i+1} = X_i - \sigma_i \cdot 2^{-i} \cdot Y_i \\ Y_{i+1} = Y_i + \sigma_i \cdot 2^{-i} \cdot X_i & i = 1, ..., 4 \quad (2.12) \\ Z_{i+1} = Z_i - \sigma_i \cdot \tan^{-1} 2^{-i} \end{cases}$$

where σ_i is equal to sign(Z_i). The algorithm starts with: X_1=X'_{in}, Y_1=Y'_{in} and: Z_1=α+φ_{LSB}/2. To simplify hardware implementation, only 4 CORDIC sub-rotations are performed in (2.12), resulting in a rotation by an angle $\alpha' \neq \alpha + \varphi_{LSB}/2$. From the CORDIC algorithm properties, it can be easily shown that the absolute value of the residual angle $Z_{residual}$=α+φ_{LSB}/2-α' is upper bounded by 2^{-4}. Therefore, by choosing four rotations in the first stage, we have about the same maximum absolute value for both $Z_{residual}$ and β (see (2.11)).
The direction σ_1 of the first rotation in (2.12) is fixed (σ_1=+1) since Z_1>0. The directions of the remaining rotations (σ_2,…,σ_4) depend only on α. These directions, therefore, can be precomputed, by using (2.12) , and stored in the lookup table shown in the Fig.2.14. The lookup table is very small, having 3 address bits (f_1, f_2, f_3). The residual angle $Z_{residual}$, similarly to σ_i values, depends only on f_1, f_2, f_3. Also $Z_{residual}$, therefore, can be stored in the lookup table.
Finally, let us note that the four CORDIC sub-rotations (2.12) amplify the modulus of the input vector by a factor:

$$\rho_1 = \prod_{i=1}^{4} \sqrt{1 + 2^{-2i}} \simeq 1.16 \qquad (2.13)$$

The amplification is inconsequential in many applications [58], [59], [60], [65] and is not compensated in the proposed approach.

2.2.4 Second rotation

In order to complete their operation, the second and third stages of the Hybrid CORDIC architecture rotate the vector [X_{T1}, Y_{T1}] (the output of the first stage) by an angle:

$$\gamma = Z_{residual} + \beta \qquad (2.14)$$

The angle γ is computed by using the π/4 multiplier and the adder shown in the Fig.2.14. The π/4 multiplier is needed to calculate β from its scaled representation, see (2.11). Since, as we have observed before, the absolute values of β and Z_{residual} are both lower than 2^{-4}, the absolute value of γ is lower than 2^{-3}. By representing γ with 11 bits, we have:

$$\gamma = 2^{-3}\left(-g_0 + g_1 2^{-1} + ... + g_{10} \cdot 2^{-10}\right) \quad (2.15)$$

A phase quantization error in the range $[-2^{-14}, 2^{-14})$ is introduced in (2.15). This results in a maximum error at the X'_{out}, Y'_{out} outputs of the DDFSM equal to $\varepsilon_q = 1.16 \cdot 2^{-14}$. This value is much lower than the weight of the less significant bit at the outputs of the DDFSM (2^{-11}).

The angle γ is then split as the sum of two sub-angles: $\gamma = \gamma_1 + \gamma_2$, where:

$$\gamma_1 = 2^{-3}\left(-g_0 + g_1 \cdot 2^{-1} + g_2 \cdot 2^{-2} + 2^{-3}\right) \qquad (2.16)$$

$$\gamma_2 = 2^{-3}\left(-\overline{g_3} \cdot 2^{-3} + g_4 \cdot 2^{-4} ... + g_{10} \cdot 2^{-10}\right) \qquad (2.17)$$

The *second rotation* block is aimed to perform the rotation by the angle γ_1, whereas the rotation by the angle γ_2 is assigned to the *final rotation* block.

In the *second rotation* we employ a CORDIC algorithm without the Z_i computation. The rotation directions τ_i are obtained directly by the bits of γ_1 as follows:

$$\tau_0 = 2\overline{g_0} - 1, \qquad \tau_i = 2g_i - 1 \quad \text{for: } i = 1,2 \qquad (2.18)$$

The corresponding CORDIC equations are:

$$\begin{cases} X'_{i+1} = X'_i - \tau_i \cdot 2^{-(i+4)} \cdot Y'_i \\ Y'_{i+1} = Y'_i + \tau_i \cdot 2^{-(i+4)} \cdot X'_i \end{cases} \quad i = 0,1,2 \qquad (2.19)$$

where $[X'_0, Y'_0]$ is the output $[X_{T1}, Y_{T1}]$ of the first rotation stage, see Fig.2.14.

The actual rotation angle obtained with (2.19) is not exactly the required angle γ_1 but is instead an angle $\gamma'_1 \approx \gamma_1$, given by:

$$\gamma'_1 = \tau_0 \cdot \tan^{-1}\left(2^{-4}\right) + ... + \tau_2 \cdot \tan^{-1}\left(2^{-6}\right) \qquad (2.20)$$

From (2.16),(2.18) the angle γ_1 can be written as:

$$\gamma_1 = \tau_0 \cdot 2^{-4} + ... + \tau_2 \cdot 2^{-6} \qquad (2.21)$$

As a consequence the *second rotation* block introduces a phase error, φ_{err}:

$$\begin{aligned}\varphi_{err} &= \gamma'_1 - \gamma_1 \\ &= \tau_0 \cdot \left(\tan^{-1}(2^{-4}) - 2^{-4}\right) + \ldots + \tau_2 \cdot \left(\tan^{-1}(2^{-6}) - 2^{-6}\right)\end{aligned} \tag{2.22}$$

With simple manipulations, it is possible to show that φ_{err} is upper bounded by:

$$\left|\varphi_{\mathrm{err}}\right| < 0.77 \cdot 2^{-13} \tag{2.23}$$

The phase error of the *second rotation* introduces an error ε_0 on each component of the DDFSM output. From (2.23), ε_0 is much lower than the weight of the output LSB (2^{-11}).

Like the first rotation block, also the CORDIC rotations (2.19) amplify the modulus of the input vector, by a factor:

$$\rho_2 = \prod_{i=4}^{6} \sqrt{1 + 2^{-2i}} \simeq 1.003 \tag{2.24}$$

Therefore the total amplification factor ρ is:

$$\rho = \rho_1 \cdot \rho_2 \simeq 1.16 \tag{2.25}$$

2.2.5 Final (third) rotation

The *final rotation* block in the Fig.2.14 implements the rotation by γ_2. The operation to be performed by this block can be written as:

$$\begin{cases} X'_{out} = X_{T2} \cdot \cos\gamma_2 - Y_{T2} \cdot \sin\gamma_2 \\ Y'_{out} = X_{T2} \cdot \sin\gamma_2 + Y_{T2} \cdot \cos\gamma_2 \end{cases} \tag{2.26}$$

This final rotation could also be computed by using the CORDIC algorithm. However, as observed in [71], [72], when the rotation angle is small a complex multiplier is able to reduce the latency and improve the performances.

In our case, the absolute value of γ_2 is lower than 2^{-6}. Therefore we can approximate sine and cosine functions in (2.26) as:

$$\sin\gamma_2 \simeq \gamma_2 \quad , \quad \cos\gamma_2 \simeq 1 \tag{2.27}$$

In this way, the final rotation is realized without the need of lookup tables to store sine and cosine values.

The approximation (2.27) introduces an error ε_1 on the DDFSM outputs *X'*$_{out}$ and *Y'*$_{out}$. It can be easily shown that this error component is upper bounded by: $\left|\varepsilon_1\right| < 1.16 \cdot 2^{-13}$.

TABLE 2.7

PERFORMANCES OF THE PROPOSED ARCHITECTURE

parameter	value	
SFDR	93.3 dBc	DDS mode: (-1,0) input
Maximum error	1.01 LSB	
SNR	73.9 dB	
SFDR	90.8 dBc	SSB mode: sine input
Maximum error	1.80 LSB	
SNR	70.3 dB	

As shown in the Fig.2.14, we have introduced two rounders in the final rotation stage, to reduce the wordlength of multipliers input. The two rounders introduce an additional error ε_2 at the output. We have: $|\varepsilon_2| \le |\gamma_2| \cdot 2^{-7} \le 2^{-13}$.

An analytical derivation of the joined effect of all algorithmic and quantization errors is not easy. We performed bit-level simulations, by operating the DDFSM in two modes. In DDS mode X_{in}=−1 and Y_{in}=0 so that the circuit generates two quadrature sine wave outputs. In SSB mode a sinusoidal input is applied to the DDFSM that operates as a digital up converter with image rejection. The Tab.2.7 summarizes the performances of the developed architecture.

2.2.6 Comparison With State Of The Art Approaches

The main advantage of the proposed Hybrid CORDIC architecture is the parallel computation of the rotations directions σ_i and τ_i. This computation is performed with a small look-up table, a multiplier by constant and an adder. Therefore simple and effective Carry Save [85] implementation for the datapaths can be used, avoiding the speed penalties due to carry propagation [59].

Previously proposed Carry Save CORDIC architectures require a Z datapath, and also additional carry propagate adders to determine rotations directions [73], [74]. Other techniques do not include carry propagate adders, but require the introduction of extra rotations [75].

The first two CORDIC rotation blocks in our architecture resemble the algorithms proposed in [76]. However, in the partitioned Hybrid CORDIC algorithm of [76] the partitioning and the handling of the rotation angle would require a huge look-up table for its implementation. On the other hand, the mixed Hybrid CORDIC algorithm, also proposed in [76], does not partition the rotation angle. Therefore its implementation requires in the first stage either a full Z datapath or a look-up table addressed by all the bits of the rotation angle.

The solution of [65] uses two rotation stages. The first one is a CORDIC rotator, while the second one is multiplier based (as originally proposed in [71]). The CORDIC rotator of [65] uses a number of stages comparable to the overall stages used in the first and second block of our architecture. The use in [65] of a single CORDIC rotator, however, requires a lookup table much larger than the one used in our architecture.

The recently proposed DDFSM implementations [62]-[64] use an architecture composed by two multiplier-based rotation stages. These architectures require a total of 8 small-width multipliers. The experimental results shown in the following demonstrate that the Hybrid CORDIC architecture is more effective, especially in terms of power and area occupation.

2.2.7 Hybrid Cordic Implementation

The most critical subsystem in the Hybrid CORDIC architecture of Fig.2.14 are the CORDIC stages. In fact the lookup table is very small and can be effectively be synthesized as a random logic. The $\pi/4$ multiplier requires only the sum of few partial products that can easily be merged with the adder needed to compute γ in a single summation tree.

The final rotation of the Hybrid CORDIC architecture of Fig.2.14 uses multiply-accumulate circuits also realized with a single summation tree. The sign-extension prevention technique [77] has been used to realize the subtraction needed to compute X'_{out}.

2.2.8 Implementation of the Cordic Stages

An innovative architecture has been devised to implement the first and second CORDIC rotation stages. The basic equation to implement the CORDIC stages is:

$$O = X - \sigma \cdot 2^{-i} \cdot Y \qquad (2.28)$$

where σ is the direction of the CORDIC sub-rotation, while i represents the order of the sub-rotation. The equation (2.28)

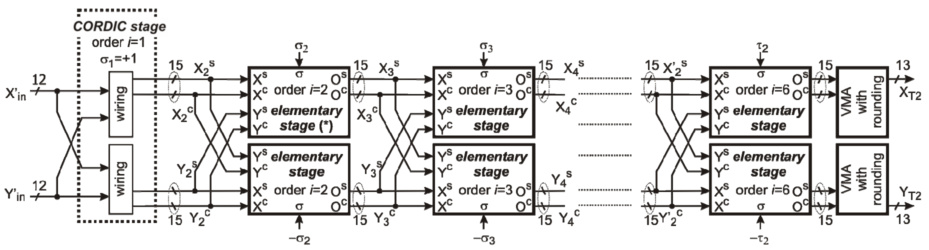


Fig. 2.15. Detailed implementation of the *first* and *second rotation* blocks with Carry -Save arithmetic.
The datapath is built by one wiring block and six CORDIC sub-rotations driven by the directions $\sigma_2 ... \sigma_4, \tau_0 ... \tau_2$.

implements the X computation in (2.12),(2.19). The Y computation can be easily obtained by swapping *X* and *Y* in (2.28) and changing the sign of σ.
Since, in our architecture, the CORDIC rotations directions are efficiently evaluated in parallel, the implementation was performed by using Carry Save arithmetic. Rewriting the (2.28) in Carry Save [85] form, we obtain the main equation to be implemented:

$$O^s + O^c = X^s + X^c - \sigma \cdot 2^{-i} \cdot Y^s - \sigma \cdot 2^{-i} \cdot Y^c \qquad (2.29)$$

The Fig.2.15 shows the detailed Carry Save datapath of the seven CORDIC stages needed in the architecture of Fig.2.14.
The X'_{in}, Y'_{in} inputs of the circuit of Fig.2.15, are in two's complement representation. The first two blocks in Fig.2.15 implement the first CORDIC sub-rotation with a fixed direction (σ_1=+1). These blocks are also in charge of the conversion from two's complement to Carry Save representation and therefore can be realized by simple wiring and complementations, without additional logic.
The six remaining CORDIC sub-rotations are implemented by using the elementary stages in Fig.2.15. Each elementary stage implements the equation (2.29). The wordlength of the X-Y signals inside the datapath of Fig.2.15 is increased by 2 LSBs (in order to reduce the overall error introduced by the CORDIC elaboration) and by 1 MSB (to avoid overflow).
The two final vector merging adders (VMA), in Fig.2.15, convert the result to two's complement representation. Rounding is also performed in the VMAs to provide the final X_{T2}, Y_{T2} signals with a wordlength of 13 bits.

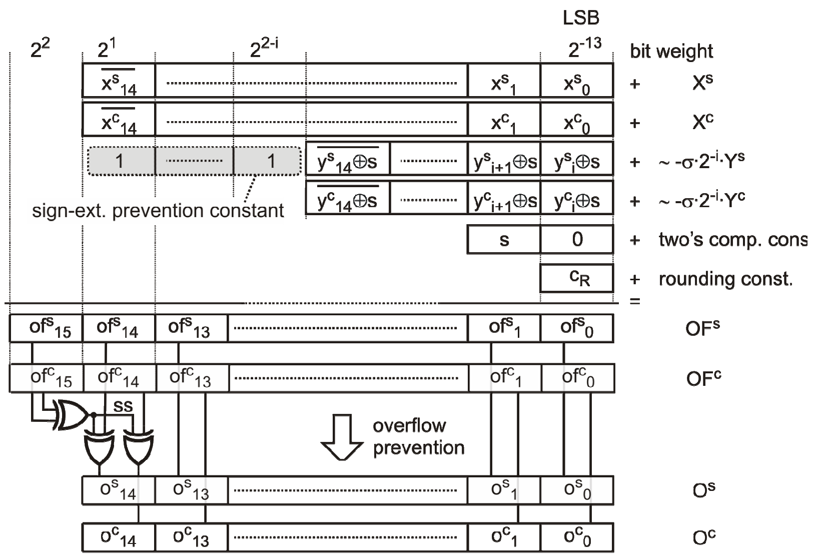


Fig.2.16. Optimized bit-level implementation in Carry Save arithmetic of
the *elementary stage* (eq. (2.29));
i is the order of the *elementary stage*.

The Fig.2.16 shows the terms to be added to implement (2.29) at the bit level. In this Figure, *s* is the binary value associated to σ (*s*=0 if σ=+1 and *s*=1 for σ=-1). The Fig.2.16 highlights the use of both the sign-extension prevention of [77] and the overflow prevention of [73]. Both techniques allow reducing the circuit complexity with respect to simpler Carry Save approaches [60].
In order to implement the two subtractions of equation (2.29) the bits of Y^s and Y^c are XORed with *s*. Moreover a two's complement constant (the bit equal to *s* in the column of weight 2·LSB) is also added.
The rounding constant c_R has been computed in order to minimize the rounding error. For all elementary stages, but the one marked with a star in Fig.2.15, the rounding error is minimized when: c_R=+1 if *s*=0 and c_R=-1 if *s*=1. Therefore the sum of the two's complement constant and c_R is equal to 1·LSB. For the elementary stage indicated with a star in Fig.2.15, the optimal rounding constant is zero.

2.2.9 Elementary stage implementation

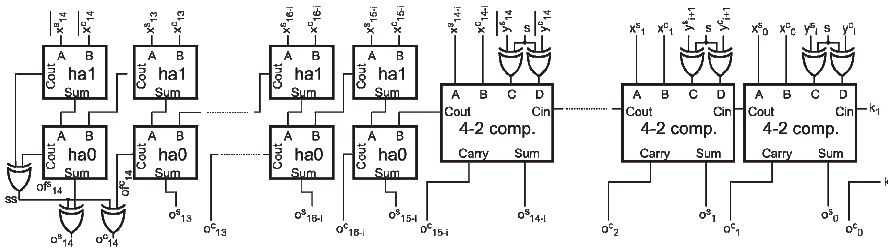


Fig.2.17. Implementation of the *i*-th order *elementary stage* by using 4-2 compressors and half-adders. For the *elementary stage* marked with a star in Fig.2.15, $k_1=s$ and $k_2=s$. For the other *elementary stages* $k_1=0$ and $k_2=1$.

The Fig.2.17 shows that the terms of Fig.2.16 can be summed with a single row of 4-2 compressors [78]. Besides these blocks, the circuit requires half adders (*ha0* and *ha1* for the compression of the MSBs) and XOR gates (for conditional complementing). The *ha0* circuits are the “traditional” half adders which compute $A+B$. The *ha1* circuits, instead, compute $A+B+1$. These blocks allow the summation of the sign-extension prevention constant (see Fig.2.16) without requiring additional hardware.
The *ha0* circuit is well known [88]. An effective implementation in CMOS logic is shown in Fig.2.18a. The *ha1* circuit is described by the following equations:

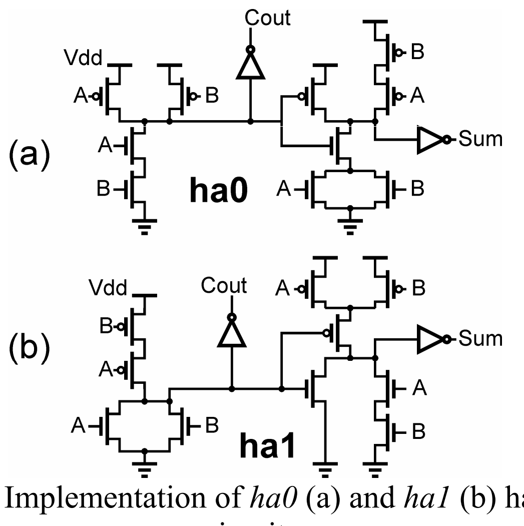


Fig.2.18. Implementation of *ha0* (a) and *ha1* (b) half-adder circuits.

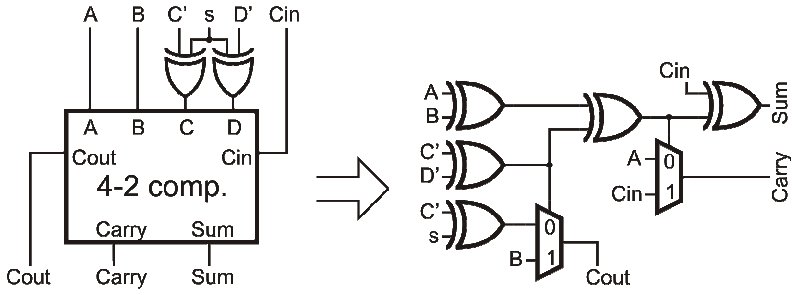


Fig.2.19. Optimal implementation of the first timing critical block in Fig.2.17.
(a) logical function; (b) detailed implementation with simple gates

$$Sum = \overline{A \oplus B} \quad ; \quad Cout = A + B \qquad (2.30)$$

and is implemented as shown in Fig.7b.

It is interesting to observe, in Fig.2.17, that the employ of the sign-extension prevention allows the use a couple of half adder circuits in place of a single 4-2 compressor, to compute the most significant bits. The most efficient realizations of the 4-2 compressor [79]-[82] requires about 60 MOS, while the couple of half adder circuits, realized as shown in Fig.2.18 require only a total of 28 transistors. The sign-extension prevention technique is, therefore, able to provide a very low circuit complexity. The number of 4-2 compressors decreases with the increase of the order *i* of the stage and, in our approach, this results in a substantial gain in area.

The timing performances of the *elementary stage* shown in Fig.2.17 are limited by two critical paths.

The first timing critical circuit, shown in Fig.2.19a, is composed by a 4-2 compressor with two inputs conditionally complemented. The best available implementations of the 4-2 compressor [81], [82] provide a delay of 3 XOR gates, and include a total of 4 XOR gates plus two multiplexers. Therefore, a straightforward implementation of the circuit of Fig.2.19a requires a maximum delay of four XOR gates.

An optimized implementation of this first timing critical circuit can be obtained by embedding the two XOR gates driven by *s* in the 4-2 compressor. This is not straightforward, since (due to redundancy of the Carry Save arithmetic) different Boolean equations sets exist which provide the same arithmetic function of a 4-2 compressor. We have found that an optimal solution can be obtained starting from the Boolean equations set of the 4-2 compressor introduced by Ghosh *et*

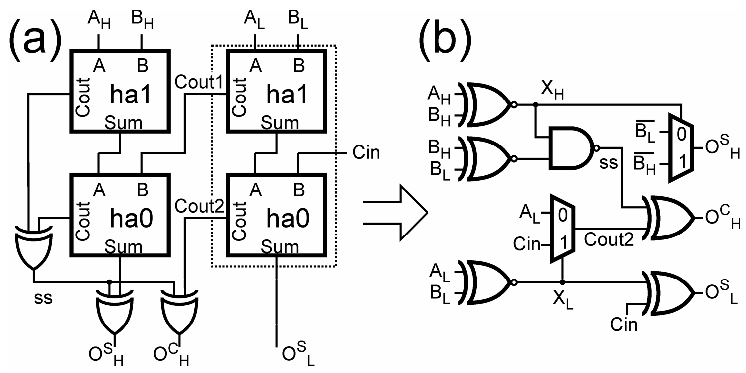


Fig.2.20. Optimal implementation of the second timing critical block in Fig.2.17.
(a) logical function; (b) detailed implementation with simple gates

al. [83], and embedding the XOR gates in the circuit, as shown in the following equations:

$$Cout = (C' \oplus D') \cdot B + \left(\overline{C' \oplus D'}\right) \cdot (C' \oplus s)$$

$$Carry = (A \oplus B \oplus C' \oplus D') \cdot Cin + \left(\overline{A \oplus B \oplus C' \oplus D'}\right) \cdot A \qquad (2.31)$$

$$Sum = A \oplus B \oplus C' \oplus D' \oplus Cin$$

The Fig.2.19b shows the gate level implementation of (2.31). Our circuit exhibits only 3 XOR gates on the critical path, highlighting an evident advantage in terms of speed with respect to the implementation of Fig.2.19a. Moreover, the circuit of Fig.2.19b, requiring a total of 5 XOR gates plus 2 multiplexers, results in one less XOR gate with respect to the implementation of Fig.2.19a using the state of the art 4-2 compressor of Hsiao *et al.* [82].
Let us now consider the second timing critical circuit of Fig.2.20a, corresponding to the overflow prevention logic, on the left-hand side of Fig.2.17. A straightforward implementation of the circuit would present four gates on the critical path (by assuming the delay of a half adder comparable to the delay of an XOR gate). An optimized implementation, with a delay of three XOR gates can be obtained by exploiting the redundancy of Carry Save arithmetic. In fact, the two half adders surrounded by the dashed line in Fig.2.20a are described by the following Boolean equations:

$$Cout1 = A_L + B_L; \quad Cout2 = X_L \cdot Cin; \quad O^S{}_L = X_L \oplus Cin \qquad (2.32)$$

where $X_L = \overline{A_L \oplus B_L}$. By exploiting the redundancy of the Carry Save arithmetic, we can rewrite the Boolean equations of this block, preserving its arithmetic function, as follows:

$$Cout1 = B_L;\quad Cout2 = X_L \cdot Cin + \overline{X_L} \cdot A_L\ ;\quad O^S{}_L = X_L \oplus Cin \tag{2.33}$$

Proceeding in a similar way for the second column of half adders in Fig.2.20*a*, with some manipulations, we obtain for the whole circuit of Fig.2.20a the following equivalent equations:

$$\begin{aligned} O^S{}_L &= X_L \oplus Cin \\ O^S{}_H &= X_H \cdot \overline{B_H} + \overline{X_H} \cdot \overline{B_L} \\ O^C{}_H &= \left[X_L \cdot Cin + \overline{X_L} \cdot A_L \right] \oplus ss \end{aligned} \tag{2.34}$$

where $X_H = \overline{A_H \oplus B_H}$; $ss = \overline{X_H \cdot \left(B_L \oplus B_H \right)}$.

The resulting circuit is shown in Fig.2.20*b*, where the critical path from all inputs to all outputs is composed by 3 gates (2 XOR and 1 multiplexer or 2 XOR and 1 NAND gate). The delay from *Cin* to output is 2 gates (1 XOR and 1 multiplexer). Since the *Cin* input arrives with a delay of 1 gate (see Fig.2.17 and Fig.2.18b), this path results again in a total delay of 3 gates.

2.2.10 Mixed CMOD-DPL Implementation

In order to simplify IC design, the DDFSM has been implemented by using a standard cell approach, with automatic place and routing. To optimize performances special purpose cells were designed to implement the timing critical circuits of Fig.2.19b and Fig.2.20b. These circuits, being composed mainly by XOR gates and multiplexers, are well suited for a pass transistor logic implementation. Having high speed operation as our main target, we employed the Double-Pass-Transistor (DPL) logic style [84], as shown in Fig.2.21.

DPL is a double-rail logic. In the developed cells, each input is converted from single to dual rail by using a couple of inverters. In this way passgate inputs, that are not suited for the timing models used by the timing analysis tools, are also avoided. The inverters 1-5 in the circuit of Fig.2.21*b* and the inverters 1-6 in Fig.2.21*c* increase the

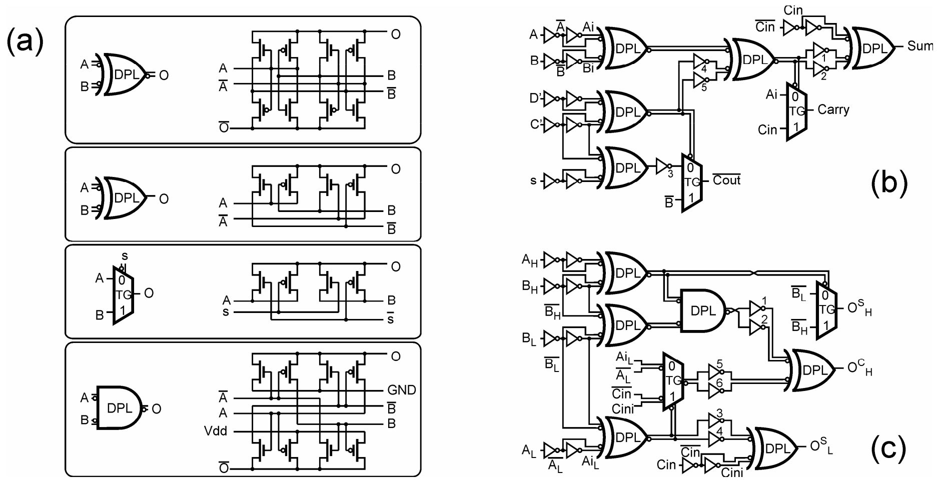


Fig.2.21. Transistor level implementation of the special cells of Fig.2.19b and Fig.2.20b;
(a) Basic gates implementation; (b) DPL implementation of the circuit of Fig.2.19b;
(c) DPL implementation of the circuit of Fig.2.20b.

circuit speed by limiting the maximum number of series transistors. Moreover, the inverters 1-2 in Fig.2.21*b* make the propagation delay of the *Carry* output independent from the capacitive load on the *Sum* output. A similar consideration applies to the inverters 1-2 and 3-4 in Fig.2.21*c*. In this way the developed DPL circuits are fully compatible with the other full-CMOS standard cells of the library.

It is worth noting that not all gates have to be dual rail. The gates which drive the outputs, both in Fig.2.21*b* and *c*, can be single rail. Also the XOR gate which drives the single rail multiplexer that calculates *Cout* in Fig.2.21*b* can be implemented in a single rail style.

The number of transistors, the propagation delay and the power dissipation obtained by employing the proposed DPL-CMOS design style are reported in the Tab.2.8. For comparison, the same table reports also the performances achievable by using a standard cell library with only full-CMOS logic, without special cells. All stages have been designed for a 0.25μm, 2.5V technology. The analysis of the Tab.2.8 reveals that proposed design style allows about a 35% reduction of the propagation delay by providing about the same number of transistors and power dissipation of the full-CMOS realization.

TABLE 2.8
SIMULATED PERFORMANCES OF DIFFERENT CORDIC STAGES CONFIGURATIONS BY EMPLOYING PROPOSED DPL-CMOS AND FULL-CMOS STYLES

	i	proposed mixed **DPL-CMOS** style # MOS	delay (ns)	Power (μW/MHz)	**full-CMOS** style # MOS	delay (ns)	Power (μW/MHz)
elementary stage	3	1074	0.67	10.7	1104	1.03	10.3
elementary stage	4	1028	0.67	10.2	1056	1.03	9.9
elementary stage	5	982	0.67	9.7	1008	1.03	9.3

2.2.11 Experimental Results

The DDFSM with the optimized Carry Save CORDIC architecture and the mixed CMOS-DPL design style has been fabricated on a test chip (see Fig.2.22) in a 2.5V, 0.25μm CMOS technology. The DDFSM has been synthesized from a VHDL description, and has been automatically placed and routed by using a commercial tool. The DDFSM accepts a 32 bit frequency control word, resulting in a frequency resolution of about 0.088Hz. The Tab.2.9 summarizes the main characteristics of the circuit. The Fig.2.23 reports the experimental digital output spectrum of the DDFSM when operated in DDS mode (X_{in}=-1, Y_{in}=0), showing an SFDR larger than 93dBc.

TABLE 2.9

EXPERIMENTAL PERFORMANCES OF THE SYNTHESIZER/MIXER

Technology	CMOS 0.25μm 1P5M
Supply Voltage	2.5 V
Phase Accumulator	32 bits
Phase Input	30 bits
Amplitude Inputs	12 bits
Outputs	13 bits
Worst Case Spur	-93.3 dBc (DDS mode) -90.8 dBc (SSB mode)
Tuning Latency	5 clock cycles
Maxmum Clock Frequency	385 MHz
Frequency Resolution	0.09 Hz
Core Area	0.22 mm^2
Power Dissipation	0.40 mW/MHz

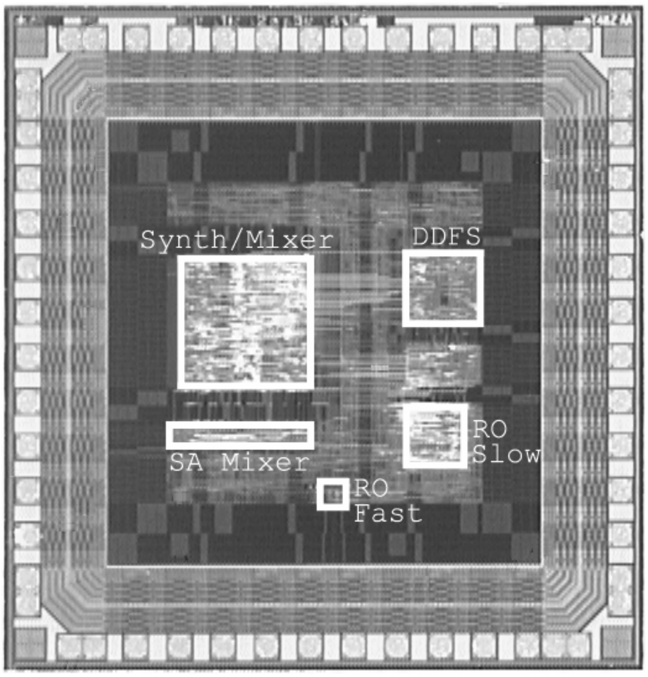


Fig.2.22. Test chip realized in CMOS 0.25μm technology. The chip includes our optimized Synthesizer/Mixer ("Synth/Mixer") a DDFS, two ring-oscillators ("RO1" and "RO2") and the built-in seft-test logic ("SA Mixer") to easy circuit testing.

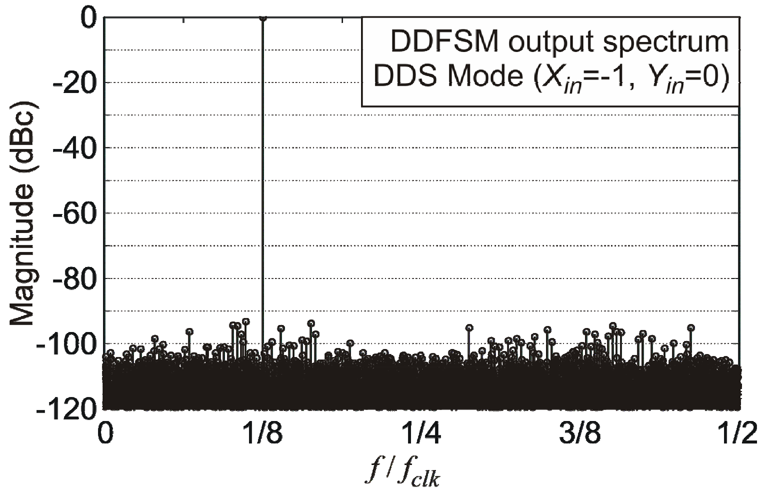


Fig.2.23. Output spectrum of the DDFSM in DDS mode (Xin=-1, Yin=0). f_{clk}=380MHz

Besides the DDFSM the chip includes a built in self test structure ("SA Mixer") and two programmable ring oscillators ("RO1" and "RO2") to make easier the measurement of DDFSM maximum clock frequency and power dissipation. Also included in the chip it is a DDFS which can provide [X_{in}, Y_{in}] inputs to the Synthesizer/Mixer to test the single and double sideband modulation functionality of the circuit.

The Tab.2.9 also reports the experimental performances of the developed DDFSM. The circuit exhibits very low power dissipation with a maximum clock frequency slightly lower than 400MHz.

The experimental performances of the proposed circuit are compared in the Tab.2.10 with the performances of the best DDFSMs available in Literature based on two stage multiplier architecture and implemented with the same 0.25μm technology. It can be observed

TABLE 2.10

COMPARISON WITH RECENTLY PROPOSED DESIGNS

Circuit	Technique	Accum. (bits)	SFDR (dBc)	Input (bits)	Output (bits)	Process (μm)	Area (mm^2)	Data Rate (MHz)	P_D (mW/MHz)
This paper	Hybrid-CORDIC	32	90	12	13	0.25	0.22	385	0.40
Torosyan [62] ***JSSC 2003***	Two stages mult-based	32	90	12	13	0.25	0.36	300	1.33
Song [64] ***JSSC 2004***	Two stages mult-based	32	100	14	15	0.25	0.51	330	1.39

that the developed architecture allows a more than three-fold reduction of power dissipation, with also a substantial reduction in the silicon area with respect to [62]. The circuit in [64], while able to reach a SFDR of 100dBc, requires about a 2.32 times larger area with respect to our implementation. The Tab.2.10 shows, moreover, that our circuit is able to work correctly up to 385MHz, whereas the best result obtained in Literature was of 330MHz.

2.3 Rectangular to Polar

2.3.1 Introduction

A processor for the conversion of rectangular to polar coordinates accepts as inputs a complex number, represented with its real and imaginary components *x*, *y*, and produces as outputs the modulus ρ and the phase ϕ of the input:

$$\begin{aligned}\rho &= \sqrt{x^2 + y^2}\\ \phi &= \arctan\left(y / x\right)\end{aligned} \qquad (2.35)$$

The relation between rectangular and polar coordinates is pictorially represented in Fig.2.24.

Cartesian to polar coordinate conversion is required in many digital communication applications, including: AM and FM demodulation for digital radios [36]-[38], automatic gain control and carrier tracking in digital implementation of Costas loop [39], modem synchronizers [40], quadrature amplitude modulation [41] and so on. Polar

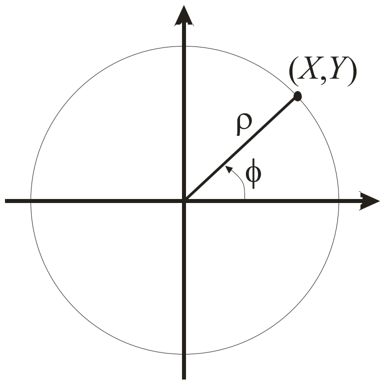


Fig. 2.24 Cartesian to Polar coordinate conversion.

modulation, that offer the capability of achieving high linearity and high efficiency simultaneously in wireless transmitter, in its digital implementation also require Cartesian to polar coordinate conversion [42]-[43]. In addition, Cartesian to polar conversion is required in imaging systems [44]-[45] and other DSP algorithms [46]. In many applications the essential component of the algorithm is the extraction of the phase, and the computation of the modulus can be carried-out with limited accuracy or is not required altogether.

The simplest and most popular approach to perform Cartesian to polar coordinate conversion uses the CORDIC algorithm in the so-called vectoring mode [47], [48]. The CORDIC algorithm is very simple to implement, requiring only shift and add operation. Unfortunately the algorithm is also inherently slow, requiring approximately N-stages for N-bit precision. Moreover, the internal word-lengths must be larger than the output precision to reduce truncation/rounding errors of CORDIC processors. Finally, efficient carry-save arithmetic can hardly be employed in vectoring mode, and slow carry-propagate adders are required instead. In [49] an improved hybrid vectoring CORDIC implementation is proposed to reduce the number of iteration. This approach, however, improves latency only, with minor advantages in terms of power and area, due to the requirements of look-up tables and multipliers for magnitude scaling.

When the phase is the only parameter to be computed, a look-up table approach can be considered [37]. In this case, if N is the word-length of x and y, the required ROM size is in the order of 2^{2N}. Therefore this technique can be considered only for small word-length applications. An approach less memory-intensive uses a divider to compute y/x (possibly based on a reciprocal table and a multiplier), followed by a look-up table to compute ϕ.

A more efficient approach has been recently proposed in [50]-[52]. The phase extraction is split in two stages. The first one computes a coarse estimate of the phase, ϕ_1. The second stage computes the residual angle, ϕ_2, and the final value of the phase: $\phi = \phi_1 + \phi_2$. The magnitude is calculated, with a limited precision, by using the coarse estimate ϕ_1. The approach of [50]-[52] requires only very small look-up tables, with a total ROM size on the order of $2^{(N+1)/3}$. The additional hardware is limited to six small-width multipliers and a few adders.

In this section we present a new architecture to realize the conversion of rectangular to polar coordinates. The input vector (x,y) is firstly

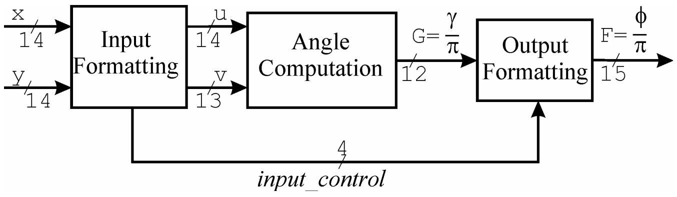
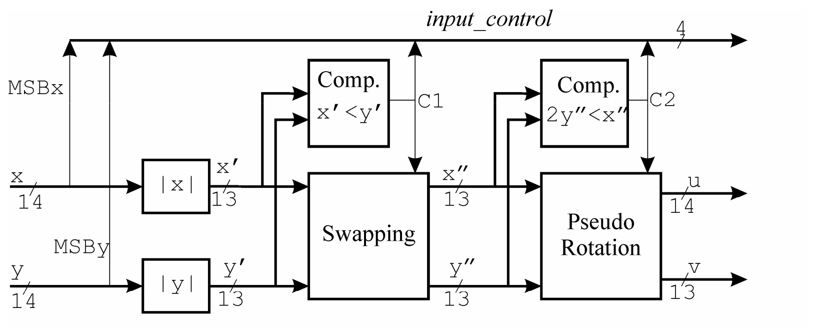


Fig.2.25 Top level of the processor for phase calculation.

rotated, generating another vector (u,v) whose phase is in the range [arctan(1/3), π/4]. This initial rotation is multiplier-less and requires only swapping and additions/subtractions. The phase of the vector (u,v) is then computed by using a table-based approach and a logarithmic number system. The required phase ϕ is easily reconstructed by using a simple output formatting stage. The look-up tables use the recently proposed Multipartite Table Method [53]. This method is ideally suited for VLSI implementation, being characterized by a very good ROM-compression factor and requiring only a multi-operand adder as additional arithmetic processing. Overall, the proposed technique for phase calculation does not require any multiplication, but only a few small tables and a few multi-operand additions. If needed, the modulus ρ is computed by including a multiplier by constant, a fourth table, and a final multiplier.
Two test circuits have been designed and implemented on a test chip in 0.25μm technology. Both circuits accept two 14-bit input signals, in 2's complement format. The first circuit computes only the phase, represented on 15 bits. The second circuit includes the modulus calculation, also performed on 15-bits. The maximum error of the phase processor is slightly larger than 1LSB, while the maximum modulus error is about 3.5LSB.
Experimentally, the phase processor operates correctly up to 526Mhz clock frequency, with a power dissipation of only 190mW@526Mhz. The second circuit (including both phase and modulus calculation) operates up to 476Mhz, with a power dissipation of 290mW@476Mhz. These performances compare favorably with [52] that was the best design available to date.

2.3.2 Phase Calculation

The top level of the processor for phase calculation is shown in Fig. 2.25. The inputs are two 14-bit, 2's complement values in the



$x>0$	$y>0$	x'	y'	α'
yes	yes	x	y	ϕ
yes	no	x	$-y$	$-\phi$
no	yes	$-x$	y	$\pi-\phi$
no	no	$-x$	$-y$	$\phi-\pi$

$x'<y'$	x''	y''	α''
yes	y'	x'	$\frac{\pi}{2}-\alpha'$
no	x'	y'	α'

$2y''<x''$	u	v	β
yes	$x''+y''$	$y''-x''$	$\frac{\pi}{4}-\alpha''$
no	x''	y''	α''

Fig.2.26 Input formatting block. α', α'' and β are the phases of the three vectors: (x', y'), (x'', y'') and (u, v) respectively.

range: $-1 < x,y < 1$, with one sign and thirteen fractional bits. The weight of the Lest Significant Bit (LSB) of x and y is 2^{-13}.
The input vector (x,y) is firstly processed in the input formatting stage, generating another vector (u,v). The phase of (u,v) is given by:

$$\beta = \arctan(v/u) \qquad (2.36)$$

and β is bounded as follows:

$$\arctan(1/3) < \beta \le \pi/4 \qquad (2.37)$$

The angle computation block uses a table-based approach and a logarithmic number system to compute the angle:

$$\gamma = \frac{\pi}{4} - \beta \qquad (2.38)$$

The phase ϕ of the input vector (x,y) is finally reconstructed from γ, in the output formatting stage.

2.3.3 Input Formatting

The Input Formatting block is shown in Fig.2.26. The vector (x, y) is initially reported to the first quadrant, by computing the absolute values (x', y') of the two components[2]. Then, (x', y') is reported to the first octant. To that purpose, the vector (x'', y'') is calculated, by

[2] As noted before, in order to reduce the wordlengths of the signals, the value -1 is excluded from the input range.

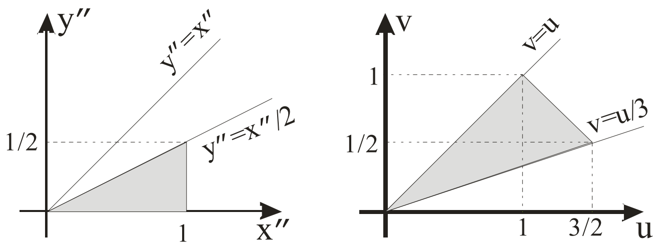


Fig 2.27. The mapping between (x'', y'') and (u, v) implemented by (2.40).

conditionally swapping each other x' and y'. The swapping is performed when $x' < y'$ and a comparator is required to check this condition.

Finally, (x'', y'') is reported to the vector (u, v) with a phase bounded as shown in (2.37). As will be shown in the following, this allows using a logarithmic transformation in the angle computation block. In order to compute (u, v), the condition $2y'' < x''$ is checked with the comparator shown in Fig. 2.26.

Let us indicate as α'' the phase of the vector (x'', y''). If: $2y'' \geq x''$, then α'' is in the range (arctan(1/2), π/4). In this case no transformations are required (that is: $u=x''$, $v=y''$).

Let us now consider the condition: $2y'' < x''$. In this case, $\alpha'' \in [0, \arctan(1/2))$. The vector ($x''$, y'') is firstly rotated by an angle −π/4. Then, the sign of the y-component of the new vector is changed, obtaining:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x'' \\ y'' \end{pmatrix} \qquad (2.39)$$

In this way we have: $\beta=\pi/4-\alpha''$, and hence $\beta \in (\pi/4-\arctan(1/2), \pi/4]$. Since it can easily be shown that: π/4−arctan(1/2)=arctan(1/3), the value of β satisfies (2.37).

Instead of implementing (2.39) (that would require the use of multipliers), in the proposed architecture a pseudo-rotation is performed, by dropping the term $1/\sqrt{2}$:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x'' \\ y'' \end{pmatrix} \qquad (2.40)$$

The phase of (u, v) is unchanged by using (2.40) instead of (2.39) while the modulus is amplified by a factor $\sqrt{2}$. The Fig.2.27 shows the mapping between (x'', y'') and (u, v) implemented by (2.40). Note that

$u \in [0,3/2)$ and is hence represented with fourteen bits, while thirteen bits are used for $v \in [0,1)$. The weight of the LSB of u and v is 2^{-13}. The operations performed in the input formatting block are encoded in the *input_control* signal, composed by four wires: the sign bits of x and y and the outputs of the two comparators of Fig.2.26.

2.3.4 Angle Computation

The angle computation block is the heart of the phase processor, and is in charge of computing the phase γ according to (2.38) and (2.36). Actually, a scaled representation G of γ is calculated:

$$G = \frac{1}{\pi}\gamma = \frac{1}{4} - \frac{1}{\pi}\beta \qquad (2.41)$$

The internal architecture of the angle computation block is shown in Fig.2.28. The logarithms of u and v are firstly computed. Then, we calculate:

$$Z = \log_2(u) - \log_2(v) \qquad (2.42)$$

Finally, the value of G is computed as:

$$G = f(Z) = \frac{1}{4} - \frac{1}{\pi}\arctan(2^{-Z}) \qquad (2.43)$$

2.3.4.a Normalization

Let us focus on the computation of the logarithm of u (the computation of $\log_2(v)$ uses exactly the same technique). The normalized argument u^* is firstly obtained as follows:

$$u^* = u \cdot 2^H \quad \text{if: } 2^{-H} \le u < 2^{-H+1} \qquad (2.44)$$

where the value of u^* is in the range:

$$1 \le u^* < 2 \qquad (2.45)$$

The logarithm of u is then computed as:

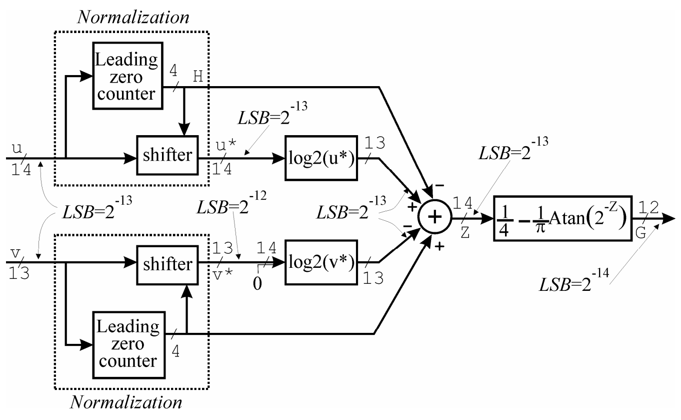


Fig.2.28. Internal architecture of the angle computation block.

$$\log_2(u) = \log_2(u^*) - H; \quad with: \; 0 \le \log_2(u^*) < 1 \qquad (2.46)$$

A leading zero counter is employed in Fig.2.28 to determine the number of leading 0's in the *u* signal. The *H* signal encodes in binary format the number of leading zeroes, and is used to drive a left-shifter the computes the normalized argument *u**.

Let us consider in more detail the special case *u*=0. This condition occurs only when the input vector is zero (*x*=0, *y*=0) since, due to the pre-processing of the input formatting block, *u*=0 implies also *v*=0 (see Fig. 2.27). When *u*=0, *H* reaches its maximum value 13 (corresponding to the weight of the LSB of *u*), but the shifting yields *u**=0, which is outside the range of equation (2.45). This is however inconsequential for the angle computation block, since the value of ϕ is undefined when the input vector is zero. In any case, the architecture of Fig.2.28 is symmetric and it will produce *Z*=0 when *u*=*v*. Therefore, from equation (2.43), *G*=0 will be computed when the input vector is zero.

2.3.4.b Logarithm tables

The value of $\log_2(u^*)$ is obtained by using the Multipartite Table Method (MTM). In order to introduce the MTM, which is described in detail in [53], let us consider a generic function *f*(*x*), whose argument is represented on *Q*-bit. The input *x* is decomposed in $K+1$ non overlapping sub-words: x_0, x_1,.., x_K of lengths q_0, q_1... q_K respectively. Hence the value of the input operand is: $x = x_0 + x_1 + \ldots + x_K$ and the

function	input word length	output word length	Number of TOs, K	Total ROM bit	Guard bits	out LSB	Approx. Error (LSB)
$\log_2(u^*)$	13	13	2	2560	1	2^{-13}	1.327
$f(Z)=\frac{1}{4}-\frac{1}{\pi}\arctan(2^{-Z})$	14	12	2	3072	3	2^{-14}	0.692
$g(Z)=\sqrt{1+2^{-2Z}}-1$	14	13	2	4608	1	2^{-14}	1.371

Tab2.11. Parameters of Multipartite Table implemented in the circuit.

length is: $Q = q_0 + q_1 + \ldots + q_K$. Using a first-order Taylor approximation we have:

$$\begin{aligned} f(x) &= f(x_0 + \ldots + x_K) \\ &\simeq f(x_0) + f'(x_0)\cdot(x_1 + \ldots + x_K) \qquad (2.47) \\ &= f(x_0) + f'(x_0)\cdot x_1 + \ldots + f'(x_0)\cdot x_K \end{aligned}$$

The term $f'(x_0)\cdot x_i$ in (2.47) is approximated as $f'(\xi_i)\cdot x_i$, where ξ_i is a sub-word of x_0 including its p_i most-significant bits, with: $p_i \le p_{i-1}$ and $p_1 \le q_0$. The equation (2.47) becomes:

$$f(x) \approx f(x_0) + \sum_{i=1}^{K} f'(\xi_i)\cdot x_i \qquad (2.48)$$

In this equation, the term $f(x_0)$ is implemented with a ROM (named table of initial values, TIV) with 2^{q_0} entries. Similarly, the terms $f'(\xi_i)\cdot x_i$ (i=1,…,K) are implemented with K ROMs (table of offsets, TO) with $2^{p_i+q_i}$ entries each. In [53] a better approximation is obtained by using a min-max approximation (instead of the Taylor expansion). Moreover, it is shown in [53] that the TOs can be made symmetric. In this way the TOs can be reduced in size by a factor of two, at the expense of a few XORs.

The approximation error of the MTM depends on the amplitude of the second derivative of $f(x)$. Hence, a direct implementation of the logarithm lookup using a simple multipartite table covering a range including zero is not feasible [54]. The derivative of the logarithm goes to infinity at x=0, so the error in the lookup becomes unbounded. Owing to the normalization (2.44), the argument of the logarithm is reported in the range [1,2) and this problem is avoided.

The MTM is ideally suited for VLSI implementation, requiring only a few small tables and a multi-operand adder. Two TOs have been used in all the Multipartite Tables of our processor, as a result of the

tradeoff between memory size and multi-operand adder complexity. The input word-length decomposition and the content of the tables have been obtained by using the algorithm presented in [53]. The Tab.2.11 shows the characteristics of the implemented Multipartite Tables. Please note that the input wordlength of the logarithm function is reported as thirteen bits. Actually, the arguments u^* and v^* are in the range [1,2). Therefore the Most Significant Bit of u^* and v^* is always one, and is not employed to perform function lookup.
As it can be seen, the implementation of the logarithm requires a total ROM size as low as 2560 bit. This is only 2.4% of the memory (about 106500 bits) that would be required by a brute-force look-up table.
In order to keep the architecture symmetric, we have used the same multipartite approximation for both $\log_2(u^*)$ and $\log_2(v^*)$. Therefore, the wordlength of v^* is augmented with a least-significant bit fixed to zero (see Fig.2.28).

2.3.4.c Arctangent table

An adder is used to compute Z according to (2.42). Since $1 \le u/v < 3$, we have:

$$0 \le Z < \log_2(3) \qquad (2.49)$$

The value of Z is represented with fourteen bits and the weight of its LSB is 2^{-13}.
A multipartite table is employed to implement (2.43). From (2.37)and (2.41) we have:

$$0 \le G < \frac{1}{4} - \frac{1}{\pi}\arctan\left(\frac{1}{3}\right) \simeq 0.148 \qquad (2.50)$$

The value of G is represented with twelve bits, and the weight of its LSB is 2^{-14}.

2.3.5 Output Formatting

The output formatting stage computes a scaled representation of the phase of the input vector (x,y)

$$F = \frac{\phi}{\pi} \qquad (2.51)$$

The value of F is reconstructed from G according to Tab.2.12 (see also Fig.2.26 and equation (2.38)). Please note that the value of γ is reported in the range: $[0,2\pi)$, while F is un unsigned number, with one

MSBx	MSBy	C1 ($x'<y'$)	C2 ($2y''<x''$)	ϕ	Range of ϕ	F
0	0	0	0	$\pi/4-\gamma$	$0\div\pi/2$	$1/4-G$
0	0	0	1	γ		G
0	0	1	0	$\pi/4+\gamma$		$1/4+G$
0	0	1	1	$\pi/2-\gamma$		$1/2-G$
0	1	0	0	$(7/4)\pi+\gamma$	$(3/2)\pi\div2\pi$	$7/4+G$
0	1	0	1	$2\pi-\gamma$		$2-G$
0	1	1	0	$(7/4)\pi-\gamma$		$7/4-G$
0	1	1	1	$(3/2)\pi+\gamma$		$3/2+G$
1	0	0	0	$(3/4)\pi+\gamma$	$\pi/2\div\pi$	$3/4+G$
1	0	0	1	$\pi-\gamma$		$1-G$
1	0	1	0	$(3/4)\pi-\gamma$		$3/4-G$
1	0	1	1	$\pi/2+\gamma$		$1/2+G$
1	1	0	0	$(5/4)\pi-\gamma$	$\pi\div(3/2)\pi$	$5/4-G$
1	1	0	1	$\pi+\gamma$		$1+G$
1	1	1	0	$(5/4)\pi+\gamma$		$5/4+G$
1	1	1	1	$(3/2)\pi-\gamma$		$3/2-G$

Tab2.12. Operations implemented in the output formatting block.

integer and fourteen fractional bits. A simple adder-subtractor is used to obtain the value of *F* in the output formatting block.

2.3.6 Modulus Calculation

Several architectures can be devised to exploit the tables used in the phase processor to compute the modulus of (x, y). In the developed circuit, we computed ρ starting from the vector (u, v). As described in the previous section, if $2y''<x''$ this vector is obtained after a pseudo-rotation. Therefore we can write:

$$\rho=\begin{cases}\frac{1}{\sqrt{2}}u\sqrt{1+(v/u)^2} & \text{if: } 2y''<x'' \\ u\sqrt{1+(v/u)^2} & \text{if: } 2y''\geq x''\end{cases} \tag{2.52}$$

The last equation can be rearranged as follows:

$$\rho=\begin{cases}\frac{1}{\sqrt{2}}u\cdot h(Z) & \text{if: } 2y''<x'' \\ u\cdot h(Z) & \text{if: } 2y''\geq x''\end{cases} \tag{2.53}$$

where *Z* is given by (2.42) and:

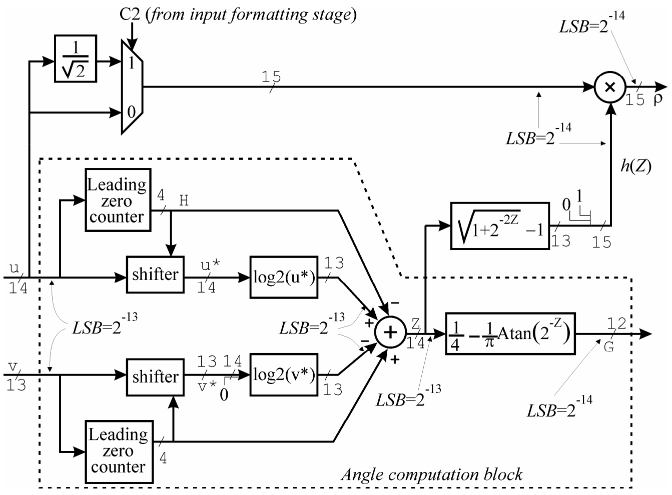


Fig. 2.29 Architecture for modulus calculation.

$$h(Z)=\sqrt{1+2^{-2Z}} \qquad (2.54)$$

Therefore, to compute ρ we need a multiplier by constant (to take into account the factor $1/\sqrt{2}$), a multiplier and a lookup table for *h*(*Z*).

From (2.49), the function *h*(*Z*) is bounded as follows: $\sqrt{10}/3<h(Z)\le\sqrt{2}$. In practice, it is more convenient to lookup the function:

$$g(Z)=\sqrt{1+2^{-2Z}}-1 \qquad (2.55)$$

which is bounded as: $0.0541\le g(Z)\le 0.414$. The function *g*(*Z*) requires an output wordlength of thirteen bits, with a LSB of 2^{-14}. The modulus ρ is finally computed as follows:

$$\rho=\begin{cases}\frac{1}{\sqrt{2}}\cdot u\cdot[g(Z)+1] & \text{if: } 2y''<x'' \\ u\cdot[g(Z)+1] & \text{if: } 2y''\ge x''\end{cases} \qquad (2.56)$$

The Fig. 2.29 shows the elements to be added to the angle computation block of Fig.2.28 to implement (2.56).

2.3.7 Error for phase calculation.

The input and output formatting stages do not include any algorithmic error. The errors of the angle calculation block are due to the approximations involved in the multipartite tables for logarithm and

Function	LSB	Maximum Absolute Error	Average Error	Error Standard Deviation
$\phi = \arctan(y/x)$	2^{-14}	1.092 LSB	0.00 LSB	0.019 LSB
$\rho = \sqrt{x^2+y^2}$	2^{-14}	3.545 LSB	0.06 LSB	0.034 LSB

Tab.2.13. Approximation errors of proposed processor.

for the function *f*(*Z*) in (2.43). Let us indicate these two sources of errors as ε_{log} and ε_f. The actual values of the multipartite tables' approximation errors are reported in Tab.2.11.
From the schematic in Fig.2.28, the worst-case error for the signal Z is given by: $\varepsilon_Z = 2\ \varepsilon_{log}$. The error at the output of the angle computation block can be bounded as follows:

$$\varepsilon_\phi \le \varepsilon_f + \varepsilon_Z \cdot \max|f'(Z)| \qquad (2.57)$$

In the range (2.49), $|f'(Z)|$ has a global maximum for *Z*=0, given by: $|f'(0)| = \frac{1}{2\pi}\log(2)$. By substituting in (2.57) we obtain:

$$\varepsilon_\phi \le \varepsilon_f + \varepsilon_{\log}\frac{\log(2)}{\pi} \qquad (2.58)$$

Using the numerical values reported in Tab.2.11 we obtain:

$$\varepsilon_\phi \le 1.278 \cdot LSB_\phi \quad \text{with: } LSB_\phi = 2^{-14} \qquad (2.59)$$

The last equation gives an upper bound for the approximation error. We have performed an exhaustive bit-level simulation of our processor, considering all the $(2^{14}-1)^2$ possible values of the input vector (*x*, *y*). The obtained results have been checked against the phase and modulus values computed by using the (non synthesizable) floating-point procedures compiled in the IEEE *math_real* VHDL package. The results of this exhaustive check are reported in Tab.2.13. As can be seen, the actual error is lower than the upper bound (2.59), and is slightly larger than 1LSB.

2.3.8 Error for modulus calculation

The largest source of error for the computation of ρ is given by the overall approximation in computing the function *g*(*Z*) in (2.55), ε_{gTOT}. This error component takes into account the approximation of the multipartite table that implements (2.55), ε_g, and the error ε_Z for the signal Z. After simple calculation one obtains:

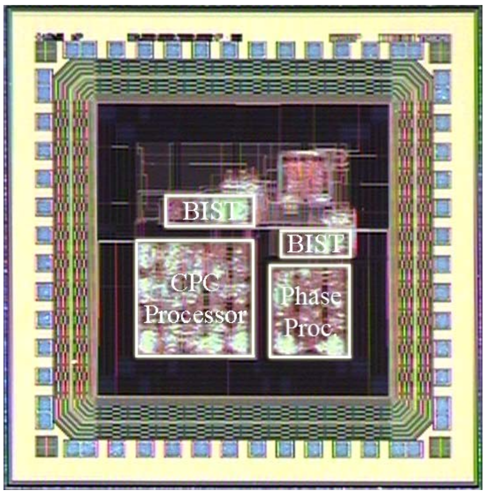


Fig.2.30 Test chip micrograph .

$$\varepsilon_{gTOT} \le \varepsilon_{\text{g}} + \varepsilon_Z \frac{\log(2)}{\sqrt{2}} \qquad (2.60)$$

The worst-case error for ρ is reached when the pseudo-rotation is performed in the input stage. In this case, the output of the multiplexer in Fig.2.29 has a maximum amplitude of: $3/\left(2\sqrt{2}\right)$ and the error ε_{gTOT} in (2.60) is amplified by this factor. Moreover, the rounding error of the multiplier by constant that computes $u\cdot\left(1/\sqrt{2}\right)$ is amplified by *h*(*Z*), which has a maximum value of $\sqrt{2}$. Taking also into account the rounding error of the final multiplier, ε_{MULT}, one obtains:

$$\varepsilon_\rho \le \varepsilon_{\text{MULT}} + \sqrt{2}\cdot\varepsilon_{\text{MULT}\_\sqrt{2}} + \varepsilon_{\text{gTOT}} \frac{3}{2\sqrt{2}} \qquad (2.61)$$

From (2.60)-(2.61), using the error values in Tab.2.11 and assuming a 0.5LSB rounding error for the two multipliers, we obtain a worst-case estimate of 4.605 LSB. The actual approximation error, reported in Tab. 3, is about 3.5 LSB.

2.3.9 Experimental Results

We have designed two circuits. The first includes only the processor for phase calculation while the second one includes the modulus calculation, as described in previous sections.

Both circuits have been implemented starting from VHDL synthesizable description, followed by synthesis and standard-cell

	This Work Phase Processor	*This Work* CPC Processor	JSSC 2003, [52] (Hwang, Fu, Willson)
Technology	TSMC 0.25µm, 5metal, 2.5V		
Transistor count	34.859	61.349	100.229
Core Area	0.178 mm2	0.343 mm2	0.484 mm2
x, y input wordlength	14	14	14
Phase output wordlength	15	15	15
Modulus output wordlength	N.A.	15	15
Phase maximum error	1.092 LSB	1.092 LSB	0.98 LSB
Modulus maximum error	N.A.	3.545 LSB	133 LSB
Latency	11 cycles	13 cycles	19 cycles
Maximum Frequency (2.5V)	482 MHz	430 MHz	406 MHz
Power Dissipation (2.5V)	180 mW @ 482 MHz (0.37mW/MHz)	276 mW @ 430 MHz (0.64mW/MHz)	470 mW @ 406 MHz (1.16mW/MHz)
Maximum Frequency (1.8V)	330 MHz	300 MHz	260 MHz
Power Dissipation (1.8V)	61 mW @ 330 MHz (0.18mW/MHz)	95 mW @ 300 MHz (0.32mW/MHz)	140 mW @ 260 MHz (0.54mW/MHz)

Tab.2.14. Circuits characteristics.

place and route. The ROMs have also been described in VHDL and synthesized into a gate-level representation.

The test circuits have been implemented on a chip in TSMC 0.25µm technology, see Fig.2.30. The test chip includes a built-in self-test logic (BIST) to make easier the measurements of maximum clock frequency and power dissipation.

A summary of the circuits' specifications is given in Tab.2.14. The IC inputs are two 14 bit, 2's complement values in the range: −1 < x,y < 1, with one sign and thirteen fractional bits. The IC outputs are two 15 bits unsigned values. The modulus ρ is in the range $0 \le \rho < \sqrt{2}$ while the normalized phase $F=\phi/\pi$ is in the range: $0 \le F < 2$.

Experimentally, the phase processor operates correctly up to 526Mhz clock frequency, with a power dissipation of only 190mW@526Mhz. The second circuit (including both phase and modulus calculation)

operates up to 476Mhz, with a power dissipation of 290mW@476Mhz.
The Tab.2.14 reports also, for comparison, the performances of the circuit proposed in [52], which was the best design available to date. As shown in Tab.2.14, the circuit of [52] (that performs only a rough estimation of the modulus) dissipates about two times the power of the developed Cartesian to polar processor (that computes the modulus with a good accuracy) and more than three times the power of the phase processor. The maximum frequency and the silicon area of the developed IC also compare favorably with [52].

2.4 Interpolator for Digital Modems

2.4.1 Introduction

A typical issue in the design of modems is the synchronization of the received symbols. In an analogue implemented modem synchronization is typically performed using a feed-back or feed-forward loop that adjusts the phase of a local oscillator. The local oscillator is then used to strobe the incoming stream once per symbol and its phase can be adjusted for optimal detection.
In modems implemented by digital techniques the incoming data stream is sampled with a local Analogue-to-Digital converter (ADC). Although in some schemes the sampling frequency can be synchronized with the incoming symbols, there are circumstances in which the sampling frequency must be considered uncorrelated with the symbols frequency. Two examples of such circumstances are digital processing of unsynchronized frequency multiplexed signals and post-processing of previously captured signals ([90], [91]).
When the processing data are not synchronized with received symbols the sample frequency must be modified for optimal detection in the digital domain. The operation in the digital domain equivalent to a change in the sampling frequency is the interpolation.
The technical papers proposed in literature facing the issues related to the interpolation for digital modems can be split in two different categories. On one side there are the papers dealing with implementation aspects of the interpolators ([90], [91]). On the other

side new algorithms for the interpolation have been developed in [92] and [94].
The interpolators developed in [90] and [91] use low-precision algorithms and provide low complexity and high-speed circuits.
On the other hand the algorithms developed in [92] and [94], while providing better approximation, have been considered too expensive in terms of hardware requirements and have been used for off-line post-processing applications. As a comparison, the piecewise parabolic interpolator proposed in [91] requires two multipliers and eight adders, whereas the techniques developed in [92] require two multipliers, twelve adders and twelve scalers, and the technique proposed in [94] requires one multiplier, one rotator and six adders.
In the recent years the speed requirements for digital modems have increased the need of precision in the interpolation process highlighting the lacks of the techniques proposed in [90] and [91]. To the best of our knowledge no work facing the efficient implementation of high-precision interpolating techniques has been proposed in Literature so far.
The purpose of our activity on this issue has been to develop a new circuital architecture for the implementation of high precision interpolating algorithms.
The technique proposed in [92] is based on a polynomial approach. It can be implemented with the Farrow structure [93] and is a scalable technique, in that the precision of the interpolation can be increased increasing the hardware complexity. The error performances of the techniques for several precision requirements are studied in [92].
The technique proposed in [92] suffers for the need of using a large number of arithmetic circuits. The technique proposed in [94] exhibits a lower architecture complexity but still is scalable in its interpolation approximation. This technique uses a trigonometric polynomial to approximate the received waveform. In its simplest form the interpolator proposed in [94] requires one multiplier, one rotator and six adders.
In this paragraph a new architecture for the implementation of the trigonometric interpolation technique developed in [94] is developed. The new architecture exploits the results obtained for the design of Digital Mixers for the implementation of the rotators. The architectural level design of the rotator is redesigned in order to take into account the precision requirement of the interpolator. Gate level

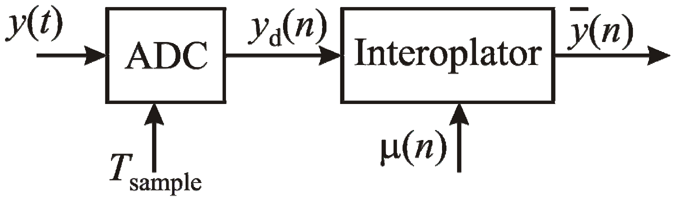


Fig. 2.31. Digital modem interpolation.

optimization is also exploited in order to further improve the performance of the circuit.

2.4.2 Interpolation Algorithm

The purpose of interpolation in a digital modem is the computation of new samples of the received continuous-time waveform. As shown in fig.2.31, starting by a continuous-time domain waveform $y(t)$ the sequence $y_d(n)$ is computing with an ADC operating with a constant sampling frequency equal to $1/T_{\text{sample}}$:

$$y_d(n) = y(t)\big|_{t=n\cdot T_{\text{sample}}} \qquad (2.62)$$

The interpolator approximates samples of the waveform $y(t)$ in the times equal to $(n+\mu(n))\cdot T_{\text{samle}}$:

$$\bar{y}(n) \simeq y(t)\big|_{t=(n+\mu(n))T_{\text{sample}}} \qquad (2.63)$$

It is worth to note that, in order to avoid aliasing in the sampling operation the bandwidth B of the waveform $y(t)$ must be limited:

$$B \leq 1/\left(2\cdot T_{\text{sample}}\right) \qquad (2.64)$$

The interpolation is hence equivalent to a re-sampling of the waveform $y(t)$ and the operation performed by the interpolator can be modeled as depicted in Fig.2.32. In order to avoid the use of a DAC and of a continuous time filter, interpolation in digital modem computes an approximation $\bar{y}_n(t)$ of $y(t)$ in each interval $[n\cdot \mathrm{T}_{\text{sample}},(n+1)\cdot \mathrm{T}_{\text{sample}}]$ whose value in the instant $(n+\mu(n))\cdot T_{\text{samle}}$ can be easily computed with a sub-set of the samples of the sequence $y_d(n)$.

In order to introduce the interpolating algorithm developed in [94] let us suppose, without[3] loss of generality, that T_{sample} is equal to 1.

[3] As a matter of fact, in the digital domain the value of T_{sample} is unknown adn it doesn't affect the result of the computation.

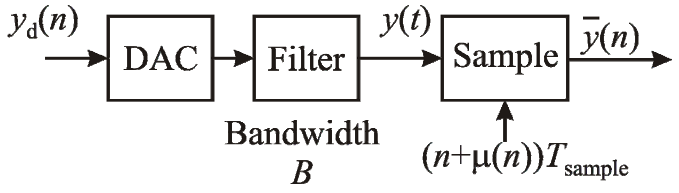


Fig.2.32. Interpolator logical scheme.

The waveform $\bar{y}_n(t)$ can be computed by a sequence $x_n(l)$ obtained by N samples of $y_d(n)$:

$$x_n(l) = y_d(l+n) \qquad l \in \left[-N/2+1, N/2\right] \tag{2.65}$$

The interpolating function $\bar{y}_n(t)$ is defined as the function having a bandwidth limited according to (2.64) crossing the samples $x_n(l)$ defined in (2.65). Furthermore the function $\bar{y}_n(t)$ is supposed to be periodic with period N.

The latter hypothesis eases the computation of $\bar{y}_n(t)$, because a periodic function can be expressed as the Fourier Series:

$$\bar{y}_n(t) = \frac{1}{N} \cdot \text{Re}\left(c_0 + 2\sum_{k=1}^{\frac{N}{2}-1} c_k W_N^{-kt} + c_{N/2} W_N^{-(N/2)t} \right) \tag{2.66}$$

In the equation (2.66) the coefficients c_k for k greater than $N/2$ are equal to zero because they are proportional to the amplitude of the armonic of $\bar{y}_n(t)$ at frequency k/N and, according to (2.64), the spectrum of $\bar{y}_n(t)$ is zero for frequencies greater than 1/2. The values of the coefficients c_k can be calculated with the following equation:

$$\bar{y}_n(t)\Big|_{t=l} = x_n(l) \qquad l = -N/2+1, \ldots, N/2 \tag{2.67}$$

which brings to [94]:

$$c_k = \sum_{l=-\frac{N}{2}+1}^{N/2} x_n(l) \cdot W_N^{k \cdot l} \qquad k = 0, \ldots, N/2 \tag{2.68}$$

The approximation of the value of $y(t)$ for $t=n+\mu(n)$ can be computed as:

$$y(t)\big|_{t=(n+\mu(n))} \simeq \overline{y}(n) = \overline{y}_n(\mu(n)) =$$

$$= \frac{1}{N} \cdot \mathrm{Re}\left(c_0 + 2\sum_{k=1}^{\frac{N}{2}-1} c_k W_N^{-k\mu(n)} + c_{N/2} W_N^{-(N/2)\mu(n)} \right) \tag{2.69}$$

By looking at the (2.68) and the (2.69) it is evident that a straightforward implementation of the interpolator proposed in [94] would require several arithmetic circuits. In [94] it is shown that a reduction of the complexity of the computation (2.69) can be achieved interpolating with the waveform $\overline{y}_n(t)$ the sequence:

$$\overline{x}_n(l) = x_n(l) + A \cdot l \qquad l \in \left[-N/2+1, N/2\right] \tag{2.70}$$

with:

$$A = \sum_{i=-N/2+1}^{N/2} (-1)^i \cdot x_n(l) \tag{2.71}$$

With a procedure similar to the one shown previously it can be shown that the approximated value $\overline{y}(n)$ can be computed as:

$$\overline{y}(n) = \overline{y}_n(\mu(n)) - A \cdot \mu(n) =$$

$$= \frac{1}{N} \cdot \mathrm{Re}\left(c_0 + 2\sum_{k=1}^{\frac{N}{2}-1} c_k W_N^{-k\mu(n)} \right) - A \cdot \mu(n) \tag{2.72}$$

with:

$$c_k = \sum_{l=-\frac{N}{2}+1}^{N/2} \hat{x}_n(l) \cdot W_N^{k \cdot l} \qquad k = 0, \ldots, N/2-1 \tag{2.73}$$

Comparing equations (2.72) and (2.69) it is evident that the improved method eliminates the need of a rotation operation at the expense of a multiplication operation.

2.4.3 Interpolator Architecture

The interpolator developed in this section uses N=4 samples of the sequence $y_{\mathrm{d}}(n)$ to compute the value of $\overline{y}(n)$. The choice of N=4 is a

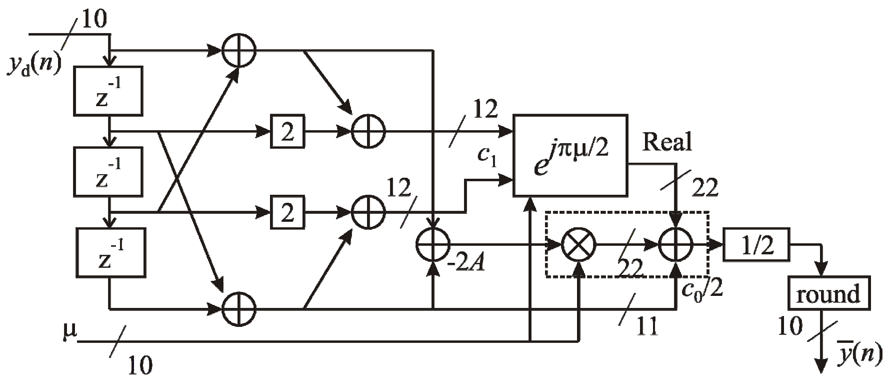


Fig. 2.33. Architecture of the interpolator.

typical choice ([91], [92], [94]) and is use here to compare achieved results with previously proposed solutions results.

For N=4 the top-level architecture needed to implement equations (2.72) and (2.73) is depicted in fig.2.33. The registers store the values of $x_n(l)$. The computation of the value of $c_0/2$, Real(c_1) and Imag(c_1) requires only multiplications by 2 and sums. The main blocks of the circuit are the multiplier and the rotator, which sizing must be carefully chosen in order to obtain a good accuracy while reducing the hardware complexity. Furthermore a higher performance architecture for the interpolator can be derived.

The proposed architecture for the rotator is shown in fig.2.34. The most significant bit of μ is used to conditionally complement and invert the values of Real(c_1) and Imag(c_1). The remaining part of the circuit rotates the input vectors of an angle lower than π/4 whose scaled representation in the range [0,1] is indicated in the figure with

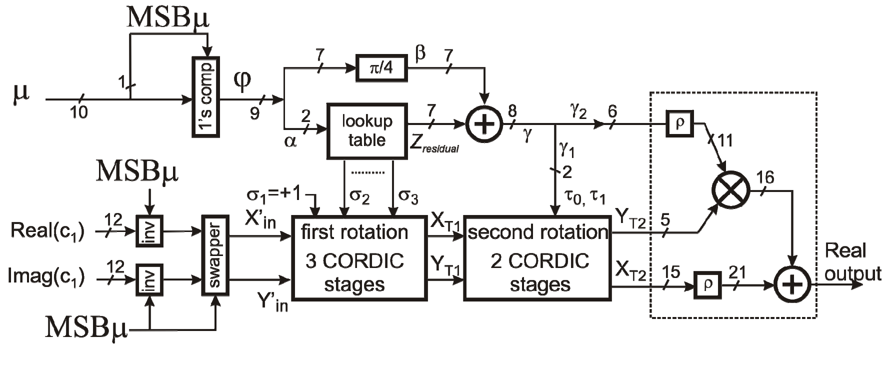


Fig. 2.34. Architecture of the rotator.

the value φ.
The two most significant bits of φ are used to drive a ROM that computes the directions of a first CORDIC based rotation phase. The remaining bits of φ are multiplied by π/4 and added to the value Z_{residual} computed by the ROM, in order to calculate the remaining angle γ. This is the angle used in the subsequent two phases to complete the rotation of the input vector.
The second rotation phase is still based on the CORCIC algorithm, but the directions of the CORDIC rotation stages are derived directly by the values of the 2 most significant bits of γ.
The final rotation phase is multiplier based. The complex vector $(X_{T2}+jY_{T2})$ is multiplied by $e^{j\gamma 2}$ to complete the rotation. Since the angle γ_2 is small, the multiplication can be computed approximating the sine(γ_2) with γ_2 and the cosine(γ_2) with 1. The multipliers by ρ are used to compensate for the module amplification introduced by the CORDIC algorithm.
The architecture in fig.2.34 is based on the same scheme of the rotator used in the digital mixer of section 2.2. Nevertheless, comparing fig.2.34 and fig.2.14 some important differences can be noted.
The first difference is in the number of CORDIC stages used. As shown in section 2.2 their number affects the maximum absolute error of the outputs. On the other hand for an interpolator the parameter that determines the precision of the circuit is the signal to noise ratio at the output of the interpolator. Since the signal to noise ratio is a cumulative value, the value of the maximum error at the output of the rotator can be a misleading parameter. Furthermore the output of the rotator is added to the terms $c_0/2$ and $-A\mu$. Hence the maximum value of the error at the output of the interpolator can be greatly different by the maximum error at the output of the rotator.
For these reasons, the sizing of the rotator has been based on a simulative approach and the choice of the maximum error at the output of the rotator can only be used as a starting point for the simulative search.
The the number of CORDIC rotation stages, as well as the other parameters of the rotator, have been chosen in order to obtain a power of noise due to the rotator sensibly smaller than the power of error at the output of the interpolator. With this choice the amount of error

Noise Source	Signal to Noise Ratio (dB)
Trigonometric Interpolator, Exact Rotation	29,5
Trigonometric Interpolator, CORDIC rotator	29,2
CORDIC rotator	37

Fig.2.35. Interpolator function.

introduced by the rotator is kept lower than the intrinsic algorithmic error of the interpolator.
The fig.2.35 shows the signal to noise ratio both at the output of the interpolator and at the output of the rotator. The values in fig.2.35 have been computed feeding the interpolator with a raised cosine spectrum sequence, sampled at twice the symbol rate and with a 100% excess bandwidth. The Signal to Noise ratio of the algorithm proposed in [94] is only 0.3dB higher than the Signal to Noise ratio of the architecture in fig.2.33 and 2.34. The low difference of the two values is due to the very high Signal to Noise value of the rotator in fig.2.34. This value has been obtained dividing the power of the interpolated signal by the power of the error at the output of the rotator. As can be seen, with the parameter chosen in fig.2.34, the power of the noise introduced by the rotator is neglectable with respect to the noise introduced by the interpolating algorithm itself.
Another important difference between the proposed rotator and the architecture developed in the section 2.2 is that there is no logic circuitry after the multiplier-based rotation phase. This is a very inportant difference, since the multipliers and the adder in the shaded box of fig.2.34 can be merged with the multiplier and the adder in the shaded box of fig.2.33, reducing both the delay and the architectural complexity of the whole interpolator.

2.4.4 Gate-Level optimization

As shown in section 2.2, the CORDIC datapath can be implemented with a chain of Carry-Save addition blocks each having a delay of three XOR gates. In order to preserve the speed of the circuit, a similar approach can be used to compute the signals X'_{in} and Y'_{in} in fig.2.34.

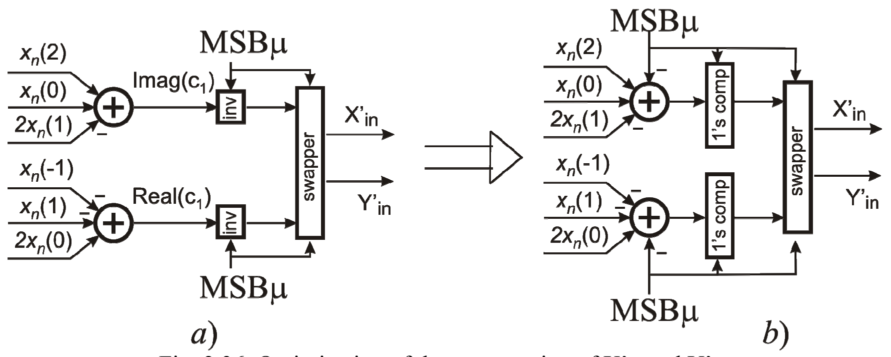


Fig. 2.36. Optimization of the computation of X'$_{in}$ and Y'$_{in}$.

In order to explain the proposed approach let us recall the operation performed to compute Imag(c_1):

$$\text{Imag}(c_1) = x_n(2) + x_n(0) - 2 \cdot x_n(1) \qquad (2.74)$$

If the value of MSBμ is equal to 1 the value of Imag(c_1) must be inverted in order to compute Y'$_{in}$ (see fig.2.36.*a*)). The inversion of Imag(c_1) can be implemented using a 1's complementer and an adder.

It is possible to avoid the need of having two subsequent sums to compute Y'$_{in}$. Infact, when MSBμ is equal to 1 we can subtract 1 by Imag(c_1) and use a single 1's complementer to invert the result (see fig.2.36.*b*)). The advantage of the circuit in fig.2.36.*b*) is in its delay. The circuit of fig.2.36.*a*) uses the cascade of two vectors merging adders to compute the components of c_1 and the outputs of the sign inverters. The circuit in fig.2.36.*b*) has a delay of only one vector merging adder to compute the sum. Furthermore, since the CORDIC datapath is a Carry-Save datapath, the sums in fig.6.*b*) can be computed in Carry-Save using the 4:2 compressors developed in section 2.2. With this topology, the computation of X'$_{in}$ and Y'$_{in}$ has a delay of one 4:2 compressor (3 XOR gates), one 1's complementer (1 XOR gate) and a swapper (one multiplexer).

As a comparison, the topology of fig.2.36.*a*) has a delay of a full adder, a vector merging adder on 12 bits, an adder on 12 bits, one XOR gate (for the sign inverters) and a multiplexer (for the swapper).

It is worth to highlight that the reduction of the delay for the computation of X'$_{in}$ and Y'$_{in}$ is due to the possibility to make Carry-Save sums in the CORDIC datapath.

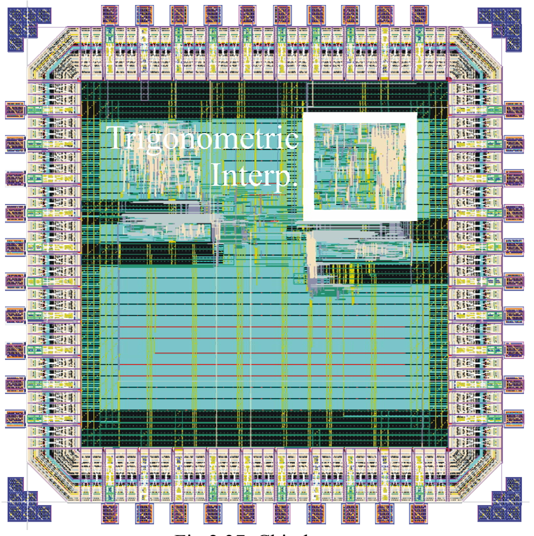


Fig.2.37. Chip layout

2.4.5 Implementaition results

The proposed interpolator has been implemented in a 0.18um technology. The circuit has been synthesized using a commercial library of standard cells. In fig.2.37 the layout of the chip is shown. At the moment this thesis is written, the chip has not yet been fabricated. Its run is scheduled for January 2007. Hence the performance of the chip reported in this paragraph is based on the synthesis and place & route characterization of the circuit.

Input/Output bits	10
Technology	TSMC 0.18um
Maximum clock speed	568MHz
Area	0.134mm^2
SNR, raised cosine input sampled at twice the symbol rate, 100% excess bandwidth	29,2dB
SNR, raised cosine input sampled at twice the symbol rate, 40% excess bandwidth	33,3dB

Fig. 2.38Performance of the proposed interpolator.

The fig2.38 summarizes the performance of the circuit. As can be seen the architecture developed, together with the gate-level optimization, were able to provide a speed of more than 550MHz with a reduced area. The overall accuracy of the circuit is also very good. As a comparison, the piecewise parabolic interpolator proposed in [56] fed with an input raised cosine spectrum signal, sampled at twice the symbol rate and with a 40% excess bandwidth, provides an SNR of 28dB, whereas the proposed circuit provide an SNR of 33dB.

Chapter 3
Forward Error Correction

3.1 Galois field Multipliers

Reed-Solomon encoding and decoding are based on Galois Field (GF(2^m)) arithmetic [95],[109]-[111], mainly the field addition and multiplication. The addition operation in GF(2^m) is equivalent to a simple bitwise XOR operation. On the other hand, the multiplication operation requires a larger and a slower hardware. Many GF(2^m) multiplier architectures have been proposed for different representation of the field elements, such as representations in standard basis [96]-[105],[108], in normal basis [99],[106], in dual basis [100],[107],[108], and in other special basis [112]-[114]. A standard basis can be constructed by choosing one root α of an irreducible polynomial $p(z)$ of degree m over GF(2). The most common representation used in Reed Solomon circuits is standard basis constructed with a polynomial $p(z)$ which is primitive; this assures that α is a primitive element of GF(2^m). In this case, in fact, it is possible to optimize hardware complexity by preserving the compatibility with many application standards (DVB, wireless, ADSL, CD, DVD). Dual basis representation is sometime used [110] when the conversions between dual and standard basis require only few XOR gates. In this thesis I will not consider the other basis since they are rarely used in Reed-Solomon applications.
An efficient multiplication scheme for standard basis multipliers with fixed $p(z)$ has been shown by Mastrovito in [103]. In this approach the multiplication $C = A \cdot B$ is performed with a matrix product $\underline{c} = \boldsymbol{M} \cdot \underline{b}$, where $\underline{c}$ and $\underline{b}$ are the components vectors of C and B, and $\boldsymbol{M}$ is a so called multiplication matrix, whose elements are obtained by XOR-ing some of the components $\underline{a}$ of A. Therefore, Mastrovito multiplication scheme breaks the multiplication in two steps. In the first step the matrix $\boldsymbol{M}$ is calculated as a function of $\underline{a}$ and of the $p(z)$ polynomial used to construct the standard basis. In the second step the

M matrix is multiplied by the components of B to calculate the multiplication result C. Sub-expression sharing can be used to substantially simplify the evaluation of ***M***. The timing and silicon area occupation of Mastrovito multiplier strongly depends on the hardware sharing technique used to evaluate ***M***. The performances that can be achieved depends moreover on the polynomial $p(z)$.
The best known performances with Mastrovito multipliers are obtained for the so called equally spaced polynomials (ESP) $p(z)$. ESP have the form $p(z)=z^m + z^{t\cdot\Delta} + ... + z^{\Delta} + 1$, where $(t+1)\cdot\Delta=m$. Mastrovito techniques for this class of polynomials are developed in [98],[100],[102]. The best performances are achieved by using the technique proposed in [102] obtaining a total XOR gate count of (m^2-Δ) with $1+\lceil \log_2 m \rceil$ XOR gates on the critical path. In [104] a systematic design approach for the technique proposed in [102] is developed by Zhang *et al.*. Note that the lowest area occupation is obtained in the case of the equally spaced trinomials (EST) for which $\Delta=m/2$. In [102] it is conjectured that m^2-$m/2$ is the lowest XOR gates count to implement a Matrovito multiplication for any irreducible polynomial $p(z)$. For $\Delta=1$ we obtain the worse performances and, in this case, $p(z)$ is also called all one polynomial (AOP). An architecture specifically developed for AOP, which re-obtains the performances of [102],[104], is shown in [99].
Unfortunately irreducible ESPs are rare; as an example there are only 81 m values lower than 1024 such that an irreducible ESP exists. In Reed-Solomon applications m is generally lower than 12, and, in these cases, no primitive ESP exists. Moreover no irreducible ESP exists for m equal to 8, which is the most common value in Reed-Solomon applications.
Apart from ESP, the best known performances are obtained for trinomials $p(z)=z^m+z^n+1$, with $n<m$. In [101] it is presented a Mastrovito technique which achieves a total XOR count of (m^2-1) with $\left(1+\left\lfloor \frac{m-2}{m-n} \right\rfloor + \lceil \log_2 m \rceil\right)$ XOR gates on the critical path. The same performances are achieved with the solution proposed by Zhang [104] for trinomials. It is apparent that the architecture complexity achievable with a trinomial is the same of the one achievable with an AOP. With respect to timing performances the best trinomial is the one with n equal to 1. In this case the timing performances of the

multiplier are the same of a multiplier based on an AOP. Primitive trinomials with n=1 exist for m equal to 4, 6 and 7. Hence, for Galois fields GF(2^4), GF(2^6) and GF(2^7) optimal multipliers for Reed-Solomon circuits exist which achieve the best known performances. On the other hand, for m equal to 5, 9, 10 and 11 primitive non-optimal (n>1) trinomials exist and they must be considered the preferred choice in Reed-Solomon application based on these fields.
Unfortunately for m equal to 8 no irreducible trinomial exists. Similarly for m equal to 12 no primitive trinomial exists. For these fields the best choice for the polynomial $p(z)$ is represented by pentanomials.
Starting from this idea, in [108] Henriquez *et al.* developed a standard basis Mastrovito technique for a class of pentanomials, and, moreover they found another class of pentanomials which allows designing efficient dual basis multipliers. Unfortunately for m=8, only one primitive polynomial, belonging to Henriquez dual basis class, exists, whereas for m=12, neither standard basis nor dual basis Henriquez techniques are applicable.
Considering dual basis multipliers an interesting solution applicable for every value of m is developed in [107] by Fenn *et al.*. The architecture complexity and the delay performances of this solution is the same of the Zhang solution for optimal and non-optimal trinomials. On the other hand, for some pentanomials Fenn solution results in better timing performances. The drawback of Fenn solution is the need of basis conversion between standard and dual basis.
The multipliers proposed in [101],[104],[107],[108],[115] have a complexity proportional to m^2 and a delay proportional to $\log_2 m$. The realization of multipliers with an asymptotic sub-quadratic complexity is possible [116]. Unfortunately the multiplier of [116] provides a delay which, depending from the technique employed, is between 2.5 times and 3 times larger than the delay of the approaches with quadratic complexity. Furthermore the technique proposed in [116] provides a reduction of the circuit complexity only for Galois Fields with large m values, rarely used for Reed-Solomon applications.
In this section a new standard basis multiplier architecture based on Mastrovito multiplication scheme is developed. For each $p(z)$, the proposed multiplier is able to implement the calculation of the $\boldsymbol{M}$ matrix exploiting as much as possible hardware sharing, thus achieving the minimum possible XOR gate count and the minimum

silicon area for the whole multiplier. This approach does not always result in the best possible delays for the signals of the ***M*** matrix. The non-optimal propagation delays of the first block are improved in the second block of the Mastrovito scheme in which a delay-driven summation technique able to compute $\boldsymbol{M}\cdot\underline{b}$ by compensating the different inputs delays is employed. Obtained overall speed compares favorably with other multipliers implementations.
The architecture proposed is easily designable with a systematic approach for every Galois field GF(2^m) and for every polynomial $p(z)$. Its complexity and timing performances are calculated for many fields GF(2^m) used in Reed-Solomon codes application and compared with state of the art techniques [104],[107],[108],[115]. For all considered GF(2^m), proposed multiplier results in the lowest delay.

3.1.1 Galois Fields Mastrovito Multipliers

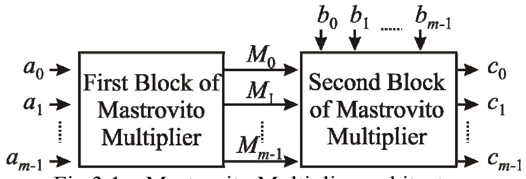
Reed-Solomon encoding and decoding circuits perform the Galois Field GF(2^m) algebraic operations using a so called standard basis to represent the field elements.
A standard basis can be constructed, given an irreducible polynomial $p(z)$ of degree m over GF(2):

$$p(z) = z^m + p_{m-1} \cdot z^{m-1} + \ldots + p_1 \cdot z + 1 \qquad p_i \in GF(2) \tag{3.1}$$

and an element $\alpha \in$GF(2^m) which is a root of $p(z)$. The standard basis $\underline{s}$ is thus defined as $\underline{s}=[1,\alpha,\alpha^2,\ldots,\alpha^{m-1}]$, and the polynomial $p(z)$ is also called field generator polynomial.
Since α is a root of $p(z)$ we can write:

$$\alpha^m = p_{m-1} \cdot \alpha^{m-1} + \ldots + p_1 \cdot \alpha + 1 \tag{3.2}$$

Equation (3.2) is known as the reduction relationship since it allows to reduce powers α^i with $i \geq m$ to the powers α^i, with $i<m$, which appear in the standard basis $\underline{s}$.
If $p(z)$ is chosen to be a primitive polynomial, the element α on which the standard basis is constructed is a primitive element of GF(2^m). This property is of main relevance in Reed-Solomon applications since the roots of the code generator polynomial are powers of a GF(2^m) primitive element.



b_0 b_1 ...... b_{m-1}
a_0
a_1
a_{m-1}
First Block of Mastrovito Multiplier
M_0
M_1
M_{m-1}
Second Block of Mastrovito Multiplier
c_0
c_1
c_{m-1}

Fig.3.1 – Mastrovito Multiplier architecture

In Reed-Solomon circuits the most critical operation is the multiplication in GF(2^m). Given two elements A and B of GF(2^m) represented in the standard basis as:

$$A = a_0 + a_1\alpha + \ ... \ + a_{m\text{-}1}\alpha^{m-1} = \underline{s} \cdot \underline{a}^T \qquad a_i \in \text{GF}(2) \qquad (3.3)$$

$$B = b_0 + b_1\alpha + \ ... \ + b_{m\text{-}1}\alpha^{m-1} = \underline{s} \cdot \underline{b}^T \qquad b_i \in \text{GF}(2) \qquad (3.4)$$

the result C of the multiplication operation between A and B can be written as:

$$\begin{aligned} C &= b_0 A + b_1 \cdot A \cdot \alpha + \ ...+ b_{m-1} \cdot A \cdot \alpha^{m-1} = \\ &= \left[A,\ A \cdot \alpha,\ ...,\ A \cdot \alpha^{m-1} \right] \cdot \underline{b}^T \end{aligned} \qquad (3.5)$$

where $\underline{b}$ represent the vector [b_0, b_1, ..., $b_{m\text{-}1}$]. Let us name M_i=[$M_{0,i}$, $M_{1,i}$, ..., $M_{m\text{-}1,\text{i}}$] the standard basis components of $A{\cdot}\underline{\alpha}^i$:

$$A \cdot \alpha^i = \underline{s} \cdot M_i^T \qquad (3.6)$$

Substituting (3.6) in (3.5) it follows:

$$\begin{aligned} \text{C} &= \underline{s} \cdot [c_0, c_1, ..., c_{m\text{-}1}]^\text{T} = \\ &= \underline{s} \cdot \left[M_0^T, M_1^T, ..., M_{m-1}^T \right] \cdot \underline{b}^T \end{aligned} \qquad (3.7)$$

Since the representation is unique, we can compute the components c_i of the multiplication result by performing the following product:

$$\left[c_0, c_1, ..., c_{m-1} \right]^T = \left[M_0^T \ M_1^T \ ... \ M_{m-1}^T \right] \cdot \underline{b}^T = \boldsymbol{M} \cdot \underline{b}^T \qquad (3.8)$$

Multiplier circuits which implements equation (3.7) use the so called Mastrovito multiplication scheme [103]. In this scheme the multiplication is realized by using two blocks as shown in Fig.3.1. The first block computes the matrix $\boldsymbol{M}$ from the a_i components of the operand A and the p_i coefficients of $p(z)$, whereas the second block evaluates the output as follows:

$$c_j = M_{j,0} \cdot b_0 + M_{j,1} \cdot b_1 + ... + M_{j,m-1} \cdot b_{m-1} \qquad (3.9)$$

The implementation of (3.9) requires, for each c_j output, m AND gates for the GF(2) multiplications and m-1 XOR gates for the additions.

Hence the complexity of the second block of Mastrovito multiplier of Fig.3,. is fixed and does not depend on the field generator polynomial. On the other hand the amount of hardware sharing used to calculate the ***M*** matrix deeply influences the complexity of the multiplier.
The simplest way to compute ***M*** is proceeding by columns. From (3.6) the first column of ***M*** is given by the components of A: $M_0=\underline{a}$. Moreover, from (3.6) it follows that each column i can be computed multiplying in GF(2^m) the column i-1 times α:

$$\underline{s}\cdot M_i^T=\alpha\cdot\underline{s}\cdot M_{i-1}^T \qquad i=1,...,m\text{-}1 \qquad (3.10)$$

Multiplication times α can easily be computed by using (3.2):

$$\begin{aligned}\alpha\cdot\underline{s}\cdot M_{i-1}^T&=M_{0,i-1}\cdot\alpha+M_{1,i-1}\cdot\alpha^2+...+M_{m-1,i-1}\cdot\alpha^m=\\&=M_{m-1,i-1}+\left(M_{0,i-1}+M_{m-1,i-1}\cdot p_1\right)\cdot\alpha+...\qquad(3.11)\\&...+\left(M_{m-2,i-1}+M_{m-1,i-1}\cdot p_{m-1}\right)\cdot\alpha^{m-1}\end{aligned}$$

Therefore we have:

$$\begin{aligned}&M_{j,i}=M_{m-1,i-1} \qquad j=0;i=1,...,m-1\\&M_{j,i}=M_{j-1,i-1}+M_{m-1,i-1}\cdot p_j \qquad (3.12)\\&j=1,...,m-1;i=1,...,m-1\end{aligned}$$

Assuming to hardwire the values p_j in the circuit, from (3.12) we have that each signal $M_{j,i}$ is the result of the XOR operation among some components of the input A. The circuit complexity depends on the amount of hardware sharing among $M_{j,i}$ functions which is related to the technique used to implement the first block of the Mastrovito multiplier. Implementing the Mastrovito multiplier by using directly (3.12), we note that some amount of hardware sharing is used, since the functions in column i are calculated from the functions of the previous i-1 column. Unfortunately this implementation does not always provide the minimum complexity circuit. Moreover we will show in the next section that, with respect to (3.12), it is possible to reduce the delay of the signals $M_{j,i}$ without increasing the number of gates needed to implement the multiplier.

3.1.2 Proposed Multiplier

In this section we introduce a new technique to implement the Mastrovito multiplication scheme (3.8). More in detail we propose a new algorithm to implement the multiplication matrix ***M*** exploiting as much as possible hardware sharing, obtaining the minimum area implementation of the Mastrovito multipliers for each pre-fixed field generator polynomial $p(z)$. This approach does not always result in the best possible delays for the $M_{j,i}$ signals. The non-optimal delays in the computation of $\boldsymbol{M}$, are improved in the second block of the Mastrovito scheme, which computes (3.9). In this block a delay-driven summation technique able to compensate the different delays of the operands $M_{j,i} \cdot b_i$ is employed obtaining a good overall speed.

In order to explain the proposed technique we firstly note from (3.5) that each element $M_{j,i}$ of the $\boldsymbol{M}$ matrix can be seen as a Boolean function of the a_i inputs. Let us indicate with $f_1, f_2, \ldots, f_L$ these functions, L being the total number of different functions in the $\boldsymbol{M}$ matrix ($m<L<m^2$). Since $M_0=\underline{a}$, the first m functions are equal to the inputs $a_0, a_1, \ldots, a_{m-1}$. From (3.12) it follows that the remaining functions are the XOR of some of the components of the $\underline{a}$ vector ($f_i=a_{i1}+\ldots+a_{il}$); therefore the overall $\boldsymbol{M}$ matrix computation is realized in hardware with a network of XOR gates. It can be easily shown that the minimum gate count of this XOR network is:

$$N_{XOR\min} = L - m \qquad (3.13)$$

In fact the first m functions (f_i with $1 \le i \le$ m), being equal to a single $\underline{a}$ vector component, do not require any XOR gate for their implementation. The computation of the remaining L-m different functions (f_i with $i > m$) requires a minimum of L-m XOR gates since a network with k XOR gates can have no more than k different output signals.

A consequence of (3.13) is that every XOR network with the minimum complexity for the computation of the functions f_i ($i > m$) must be able to evaluate each function by using only one XOR gate:

$$f_i = f_{i1} \text{ XOR } f_{i2} \quad i > m;\ i1 \text{ and } i2 \text{ lower than } i \qquad (3.14)$$

As it can be seen from (3.12), each function present in the $\boldsymbol{M}$ matrix is either equal to a previously calculated function or obtained with one 2-input XOR gate. However, in (3.12) no check is done to verify if new calculated functions are equal to previously calculated ones. Therefore (3.12) does not always give the minimum XOR network.

Our algorithm, reported in Fig.3.2, starts from (3.12) and adds two different checks on new calculated functions in order to reduce both the complexity and the delay for the computation of the $\boldsymbol{M}$ matrix. For each element $M_{row,col}$ the algorithm uses (3.12) to evaluate the new

```
initialize: $M_{i,0}=a_i$ (i=0,...,m-1)
for col=1,..,$m$-1 loop {
  "connect $M_{0,col}$ to $M_{m-1,col-1}$"
  for row=1,..,$m$-1 loop {
    if $p_{row}$=0 then
      "connect $M_{row,col}$ to $M_{row-1,col-1}$"
    else {
      new_func= $M_{row-1,col-1}$ XOR $M_{m-1,col-1}$
      if ∃ (j,i) | previously calculated $M_{j,i}$=new_func then          CHECK 1
        "connect $M_{row,col}$ to $M_{j,i}$"   (comment: no XOR gate added)
      else
        D=Delay($M_{row-1,col-1}$ XOR $M_{m-1,col-1}$);  j1'=row-1;  i1'=col-1;  j2'=$m$-1;
i2'=col-1
          foreach (j1,i1),(j2,i2)  |  $M_{j1,i1}$ XOR $M_{j2,i2}$ = new_func   {
            if Delay($M_{j1,i1}$ XOR $M_{j2,i2}$) < D  {                 CHECK 2
              D=Delay($M_{j1,i1}$ XOR $M_{j2,i2}$)
              j1'=j1; i1'=i1; j2'=j2; i2'=i2
            }
          }
        "connect $M_{row,col}$ to $M_{j1',i1'}$ XOR $M_{j2',i2'}$"   (comment: one XOR added,
                                                    delay is optimized choosing to
                                                    connect to $M_{j1',i1'}$ XOR $M_{j2',i2'}$)
}}}
```
Fig3.2 – Minimum gate first step of Mastrovito multiplier.

function (*new_func*) to be added to the matrix. The first check ("check 1" in Fig.3.2) is aimed to reduce the circuit complexity. This check compares *new_func* with all functions previously inserted in the matrix. If the function has already been inserted in the matrix, the XOR gate needed to compute the new function in (3.12) is avoided by simply hardwiring $M_{row,col}$ to the previously calculated function $M_{j,i}$ (equal to *new_func*). The second check ("check 2" in Fig.3.2) is aimed to reduce the circuit delay without increasing the circuit complexity. This check is performed when the "check 1" fails. In this case a XOR gate has to be necessarily added to the XOR network. In the approach of (3.12) the XOR gate added to the network simply has $M_{\text{row-1,col-1}}$ and $M_{\text{m-1,col-1}}$ as inputs. The "check 2" considers all couples of previously calculated functions ($M_{j1,i1}$, $M_{j2,i2}$) which verify the relationship $M_{j1,i1}$ XOR $M_{j2,i2}$ = *new_func*. Among all these couples, the algorithm actually implements in the network the solution ($M_{j1',i1'}$ XOR $M_{j2',i2'}$) which presents the minimum delay.

By considering the performances of this algorithm, we can do two observations. Since a single XOR gate is inserted in the network for each function different from previously calculated ones, the algorithm always obtains the minimum number of XOR gates, given by (3.13).

Among all possible solutions with the minimum XOR gate number, the “check 2” allows to have the best possible delay on each $M_{j,i}$ component.

Let us introduce two examples to better describe the first and the second check in the algorithm of Fig.3.2. The Fig.3.3a shows the XOR gate network obtained by employing our algorithm to the GF(2^6) field with $p(z)=z^6+z^3+1$. Note that only 3 XOR gates are needed to compute $\boldsymbol{M}$. The direct use of (3.12), in this case, would require 2 more XOR gates for the calculation of $M_{3,4}$ and $M_{3,5}$. The “check 1” in our algorithm discovers that these functions are equal to $M_{2,0}$ and $M_{1,0}$ respectively, reducing the XOR gates count.

The example of Fig.3.3b considers the GF(2^9) field with[4] $p(z)=z^9+z^6+1$. In this case the “check 1” fails for all $M_{j,i}$ terms and our algorithm gives the same XOR gates count (8) obtained when equation (3.12) is used to build the first block of Mastrovito multiplier. Nevertheless the “check 2” of our algorithm is able to improve the delay for the computation of the two signals $M_{6,7}$ and $M_{6,8}$. In fact, by using (3.12), the $M_{6,7}$ signal would be computed as $M_{6,7}=M_{8,6}$ XOR $M_{5,6}$, with a total delay of 3 XOR gates. By using the “check 2”, our algorithm discovers that $M_{6,7}$ is also equal to $M_{2,0}$ XOR $M_{5,0}$, reducing the $M_{6,7}$ delay to only one XOR gate.

Despite of the employ of the “check 2”, the algorithm of Fig.3.2, exploiting as much as possible hardware sharing, still evaluates the elements $M_{j,i}$ by using a network made by long chains of XOR gates in which each internal node is connected to an $M_{j,i}$ signal. The optimal delay network, on the other hand, would use a tree structure for each $M_{j,i}$ signal with many internal nodes not connected to any output, and, therefore a substantially higher number of XOR gates.

We have found that the non-optimal delays of the solution of Fig.3.2 can be compensated by the second block of Mastrovito multiplier (which implements (3.9)) by using a delay-driven tree structure. Our algorithm to implement the second block of the Mastrovito scheme exploits the non equal delays of the elements inside the same row of $\boldsymbol{M}$. The algorithm is shown in Fig.3.4. As it can be seen, firstly a set S, composed by $M_{j,i}\cdot b_i$ signals, is defined. The algorithm proceeds by deleting from S the two elements s_1 and s_2 with the minimum delay,

[4] It is worth to note that $p(z)=z^9+z^6+1$ is not irreducible in GF(2). This polynomial is considered here only in order to have a simple example. An irreducible polynomial having properties similar to z^9+z^6+1 when implemented with our approach is $z^{21}+z^{14}+1$.

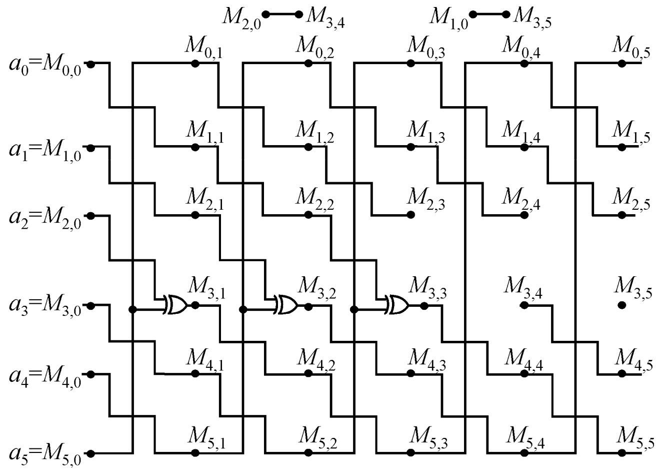


Fig.3.3a – Proposed first block of Mastrovito multiplier for GF(2^6) with $p(z)=z^6+z^3+1$.

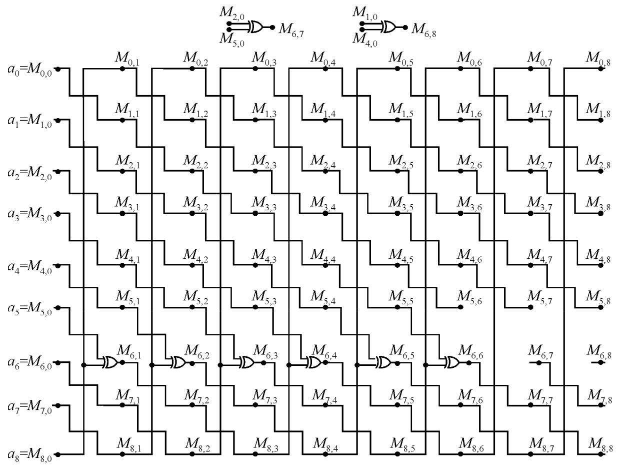


Fig.3.3b – Proposed first block of Mastrovito multiplier for $p(z)=z^9+z^6+1$.

combining s_1 and s_2 with a XOR gate, and inserting the XOR output s_3 in the set S. The algorithms ends when only one signal remains in the set S.

If we denote with d_i (i=0,...,m-1) the delays of $M_{j,i}\cdot b_i$ signals, in the appendix B it is shown that, using the proposed algorithm of Fig.3.4, the delay t (defined as the number of XOR gates on the critical path) for the computation of c_j is given by:

$$t = \left\lceil \log_2 \sum_{i=0}^{m-1} 2^{d_i} \right\rceil \qquad (3.15)$$

```
For each row j=0,...,m-1 of the matrix M:
1) let S be the set of $M_{j,i}\cdot b_i$ signals to be added to obtain $c_j$.
2) among S signals choose the two signals $s_1$ and $s_2$ with the
   minimum delay
3) remove $s_1$ and $s_2$ from S
4) obtain a new signal $s_3$ as $s_1$ XOR $s_2$
5) compute the delay of $s_3$ from the delays of $s_1$ and $s_2$, add $s_3$
   to S
6) repeat steps 2-4 until a single signal $s_L$ remains in S
7) $c_j = s_L$
```

Fig.3.4 – Delay-driven second step of Mastrovito multiplier.

In the appendix B it is also shown that this is the minimum possible delay given the delays d_i of the $M_{j,i}\cdot b_i$ signals. Of course, the proposed technique re-obtains the standard tree architecture in the case of inputs with equal delays ($d_i=d\ \forall i \Rightarrow D=d+\log_2(m)$). Clearly the number of XOR gates needed by our delay-driven tree topology (generated with the algorithm of Fig.3.4) is the same (m-1) of the number of XOR gates needed in the ripple and balanced tree implementations of the network.

As an example, the Fig.3.5 shows the delay-driven XOR network needed to calculate the slowest output c_6 for a GF(2^9) multiplier with $p(z)=z^9+z^6+1$. This is the same field considered in Fig.3.3b. The network of Fig.3.5 is constructed starting from the delays of $M_{j,i}$ signals obtained with our algorithm of Fig.3.2 (see Fig.3.3b). In Fig.3.5 the number of XOR gates on the critical path of each $M_{6,i}\cdot b_i$ signal obtained with the algorithm of Fig.3.2 is reported in parenthesis. The figure shows that a total delay of 5 XOR gates is

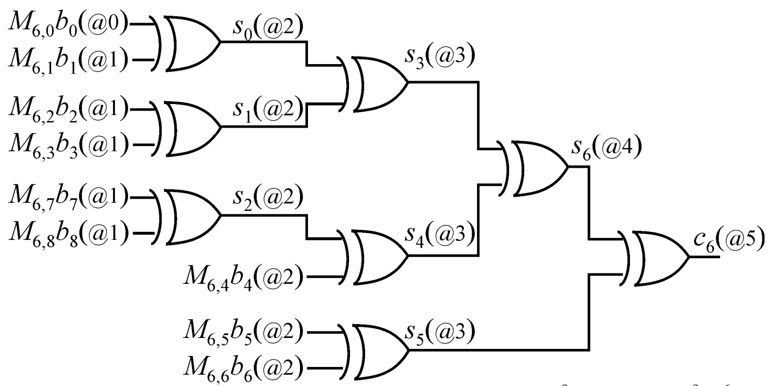


Fig.3.5 – Second block of Mastrovito multiplier for GF(2^9) with $p(z)=z^9+z^6+1$. The first block of Mastrovito multiplier is assumed to be implemented with the proposed algorithm (see Fig.3.3b).

needed to compute c_6. It is worth to note that by using a balanced tree topology joined with the employ of our algorithm of Fig.3.2 to implement the first block of the Mastrovito multiplier, a total delay of 6 XOR gates would be obtained. Moreover, the employ of the balanced tree topology and of (3.12) to build the first block of the Mastrovito multiplier leads to a total delay of 7. Therefore, in the case of the GF(2^9) field with $p(z)=z^9+z^6+1$, both our "check 2" (in the first block of Mastrovito multiplier) and the delay-driven tree topology (employed to build second block of Mastrovito multiplier) improves the multiplier speed without increasing the circuit complexity. A more detailed analysis of the performances achievable with the proposed approach is reported in the next sections.
In conclusion the proposed method for the implementation of Mastrovito multiplier scheme of Fig.3.1 provides the minimum area implementation for the first block of the multiplier and the minimum delay for the second block given the delays of the first block. Since the complexity of the second block of the Mastrovito multiplication scheme is a constant, proposed technique achieves the minimum area for the whole multiplier. The critical path delay of the whole multiplier is not optimal because of the non-optimal delays of the solution found for the first block. However, our delay driven implementation of the second block is able to compensate for the non-optimal delays of the first block achieving a good overall speed when compared with other techniques. Our "check 2", in the algorithm of Fig.3.2, helps in reducing the multiplier delay without increasing its complexity. In its entirety, our approach, while being much simpler to implement with respect to many recently proposed solutions [104],[115], provides moreover very good performances in comparison to the state of the art. The section 3.1.4 quantifies this claim.

3.1.3 Computational complexity of the algorithms

This section is devoted to study the computational complexity of the algorithms needed in our technique to construct the finite field multiplier (see Fig.3.2 and Fig.3.4). This study allows to estimate the maximum *m* value that can be treated with our approach. This aspect is relevant in cryptographic applications [117] where *m* is in the range [160, 521].

m	*r*	*p*(*z*)	computation time
500	3	$z^{500}+z^{5}+1$	14sec
500	3	$z^{500}+z^{250}+1$ (EST)	9sec
501	3	$z^{501}+z^{334}+1$ (ESPm)	1min
500	3	$z^{500}+z^{495}+1$	3min21sec
500	5	$z^{500}+z^{375}+z^{250}+z^{125}+1$ (ESP)	10sec
500	5	$z^{500}+z^{6}+z^{5}+z^{1}+1$	59min47sec
500	5	$z^{500}+z^{495}+z^{5}+z^{1}+1$	1h26min50sec
500	5	$z^{500}+z^{495}+z^{494}+z^{1}+1$	1h36min49sec

Tab.3.1 – Computation times needed to build the proposed multiplier. Times have been obtained with a 64bit processor running at 2.4GHz.

Let us start our analysis by considering the algorithm of Fig.3.2, needed to construct the first block of Mastrovito multiplier. In this case we can observe that the algorithm computational complexity is dominated by the "check 2". Let us name *r* the number of nonzero terms in *p*(*z*) and *N*=(*m*-1)(*r*-2) the total number of functions for which "check 2" is performed (in the worst case). By looking to Fig.3.2 we note that in "check 2" the generic *i*-th function (*i*=0,…,*N*-1) is compared with all couples of previously generated functions. Therefore, in the worst case, the "check 2" for the *i*-th function requires $\binom{i}{2}$ functions comparisons. If we assume a computational complexity equal to *m* for each functions comparison, we can compute the total computational complexity (ρ1) of "check 2" as:

$$\rho_1 = m \cdot \sum_{i=2}^{N-1} \binom{i}{2} = m \cdot \sum_{i=2}^{N-1} \frac{i(i-1)}{2} = m \cdot \frac{N(N-1)(N-2)}{6} \qquad (3.16)$$

Therefore the asymptotic computational complexity of the algorithm in Fig.3.2 can be estimated as:

$$\rho_1 \approx \frac{1}{6} r^3 m^4 \qquad (3.17)$$

Let us now consider the algorithm of Fig.3.4, which builds the second block of Mastrovito multiplier. This algorithm computes *m* XOR networks, one for each multiplier output. For each of these networks, at the generic *i*-th step of the algorithm (*i*=0,…,*m*-2), it is required an ordered insertion within a list of *m-i* elements. Assuming a complexity equal to *m-i* for the ordered insertion operation, we can write the computational complexity of the algorithm of Fig.3.4 as:

$$\rho_2 = m \cdot \sum_{i=0}^{m-2} m - i = \frac{1}{2} m\,(m-1)\,(m-2) \qquad (3.18)$$

From this equation we conclude that the asymptotical computational complexity of our approach is dominated by the algorithm needed to build the first block of Mastrovito multiplier. The asymptotical computational complexity of the complete multiplier will therefore be given by (3.17). Tab.3.1 reports the computation times obtained on a 2.4GHz, 64bit processor in the case of m=500 (or 501). It can be observed that, in spite of the large m value, a reasonable computation time is needed. This confirms the applicability of our technique not only in Reed-Solomon encoding/decoding (where m is never higher than 12) but also in cryptographic applications where m vales in the order of 500 are not unusual.

3.1.4 Analytical derivation of the performances

At the gate level, the building blocks for the implementation of our multiplier are the AND gates needed for the computation of $M_{j,i} \cdot b_i$ signals and the XOR gates, needed both to implement the $\boldsymbol{M}$ matrix computation and the delay-driven tree in the second block of Mastrovito multiplier. The total number of AND gates needed by our approach is m^2, while the critical path always include a single AND gate. Since these numbers are fixed and do not depend on the polynomial $p(z)$, in the following we will describe the complexity and the delay of our approach by using the total number of XOR gates (N_{XOR}) and the number of XOR gates on the critical path (D_{XOR}).

The number of XOR gates needed to implement the first block of the Mastrovito multiplier in our approach depends on the number of $M_{j,i}$ signals for which the "check 1" succeeds. If the "check 1" always fails, the number of XOR gates needed for this block is $(m-1)(r-2)$, where r is the number of nonzero terms in $p(z)$. The XOR gates count for the second block of the Mastrovito multiplier does not depend on $p(z)$ and is given by $m(m-1)$. Therefore an upper bound for the total number of XOR gates of our multiplier is:

$$N_{XOR} \leq (m + r - 2) \cdot (m-1) \qquad (3.19)$$

where the equal signs holds when the "check 1" always fails.

Giving a general upper bound for the delay of our topology is much more difficult. In the following paragraphs, starting from (3.15), we will evaluate D_{XOR} for some specific field generator polynomials $p(z)$.

3.1.5 General Trinomial ($p(z)=z^m+z^n+1$): upper bound

In this paragraph we study the case of the polynomial $p(z)= z^m+z^n+1$ assuming that both "check 1" and "check 2" always fails. Clearly an upper bound for N_{XOR} and D_{XOR} will be obtained.

By considering the multiplier complexity, the upper bound for N_{XOR} is simply obtained by particularizing (3.19) for r=3:

$$N_{XOR} \le m^2 - 1 \qquad (3.20)$$

The multiplier delay can be computed by evaluating the matrix $\boldsymbol{D}$, whose elements $D_{j,i}$ are the delays of $M_{j,i}$ signals. This matrix is shown in Fig.3.6, where the quantity k is given by:

$$k = \left\lceil \frac{m-1}{m-n} \right\rceil \qquad (3.21)$$

From Fig.3.6 it can be observed that the row with $j=n$ presents the highest delays, therefore the critical multiplier output will be c_n. From Fig.3.6 and (3.15) it follows that the delay of c_n, in terms of the number of XOR gates on the critical path, can be written as:

$$t_{cn} = \left\lceil \log_2\left(2^0 + 2^1\cdot(m-n) + \ldots + 2^{k-1}\cdot(m-n) + 2^k\cdot(m-1-(k-1)(m-n))\right)\right\rceil \qquad (3.22)$$

Since c_n is the critical output, this equation represents also an upper bound for D_{XOR}. By simplifying the second term of (3.22), we can write:

$$\begin{array}{c|cccccccccccccc|}
0 & 0 & 0 & 0 & \cdots & 0 & 1 & \cdots & 1 & 2 & \cdots & k\text{-}2 & k\text{-}1 & \cdots & k\text{-}1 \\
1 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 1 & 1 & \cdots & k\text{-}2 & k\text{-}2 & \cdots & k\text{-}1 \\
 & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\
n & 0 & 1 & 1 & \cdots & 1 & 2 & \cdots & 2 & 3 & \cdots & k\text{-}1 & k & \cdots & k \\
 & 0 & 0 & 1 & \cdots & 1 & 1 & \cdots & 2 & 2 & \cdots & k\text{-}1 & k\text{-}1 & \cdots & k \\
 & \vdots & \vdots & 0 & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\
 & & & \vdots & & & & & & & & & & & \\
m\text{-}1 & 0 & 0 & 0 & \cdots & 1 & 1 & \cdots & 2 & 2 & \cdots & k\text{-}1 & k\text{-}1 & \cdots & k\text{-}1 \\
\hline
j \uparrow \; i \rightarrow & 0 & 1 & 2 & \cdots & m\text{-}n & \cdots & \cdots & 2(m\text{-}n) & \cdots & \cdots & (k\text{-}1)(m\text{-}n) & \cdots & \cdots & m\text{-}1
\end{array}$$

Fig.3.6 – $\boldsymbol{D}$ matrix for the trinomial $p(z)=z^m+z^n+1$, assuming that "check 1" and "check 2" always fail. k is given by (3.21).

$$D_{XOR} \leq \left\lceil \log_2 \left(2^k \left(m-1-(k-2)(m-n) \right) - 2(m-n)+1 \right) \right\rceil \qquad (3.23)$$

In (3.23) the equal sign holds when the "check 1" and "check 2" fails for all $M_{j,i}$ signals.

We have evaluated the performances of all possible trinomials with $m \leq 128$. This analysis has revealed that the upper bounds (3.20) and (3.23) result always verified with the equal sign, with the exception of two classes of trinomials (the equally spaced trinomials and the missed equally spaced trinomials). For these two classes, in fact, there are some $M_{j,i}$ terms for which either "check 1" or "check 2" succeeds. The performances in these two special cases are studied in the following two sections.

3.1.6 Equally Spaced Trinomial EST ($p(z)=z^m+z^{m/2}+1$)

The general form of the ***M*** matrix for the equally spaced trinomial ($p(z)=z^m+z^{m/2}+1$) is shown in Fig.3.7. From this figure it can be observed that the "check 1" succeeds for $M_{m/2,i}$ terms with $i \in [m/2+1, m-1]$. In fact, for example, the $M_{m/2,m/2+1}$ function is given by a_{m-1} XOR ($a_{m/2-1}$ XOR a_{m-1})=$a_{m/2-1}$. The "check 1" discovers that this function is equal to a previously calculated one ($M_{m/2-1,0}$) and avoids one XOR gate in the multiplier realization. From Fig.3.7 it follows that the overall M matrix computation requires only $m/2$ XOR gates. The total complexity of the multiplier is therefore:

$$N_{XOR} = m^2 - m/2 \qquad (3.24)$$

From the timing performances point of view, the critical multiplier output is $c_{m/2}$. The computation of this signal, in fact, requires a delay-driven tree in which $m/2$ input signals have a delay equal to 0 and $m/2$ input signals have a delay equal to 1. The critical path delay of the multiplier is therefore (from (3.15)):

$$D_{XOR} = \left\lceil \log_2 \left(3/2\, m \right) \right\rceil \qquad (3.25)$$

3.1.7 Missed Equally Spaced Trinomial mEST ($p(z)=z^{3n}+z^{2n}+1$ with $m=3n$)

This section is devoted to the study of the polynomial $p(z)=z^{3n}+z^{2n}+1$. This polynomial correspond to a trinomial constructed from an equally spaced quadrinomial ($z^{3n}+z^{2n}+z^{n}+1$) in which the term z^n is missing. We name this class missed equally spaced trinomial (mEST). The Fig.3.8 shows the general form of the ***M*** matrix corresponding to this class of polynomials. From the figure it can be noted that the "check 1" always fails, while the "check 2" succeeds for the $M_{2n,i}$ terms with $i=2n+1, ...,m-1$. Since "check 1" fails, the number of XOR gates required by the multiplier will be given by (3.20):

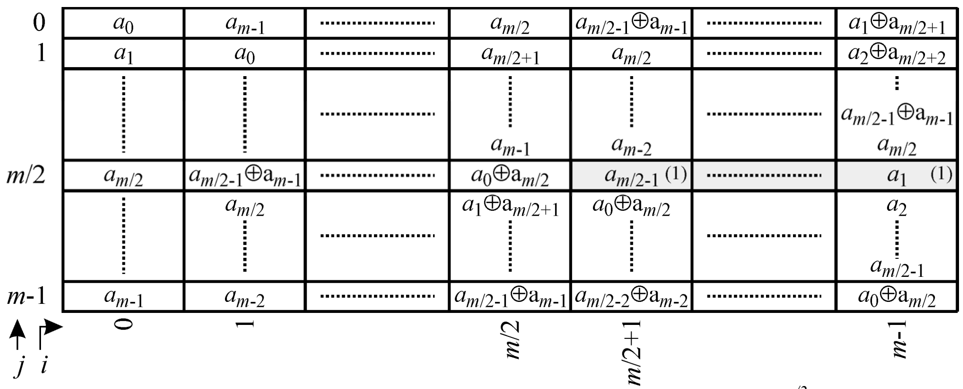
$$N_{XOR} = m^2 - 1 \qquad (3.26)$$

The multiplier delay will be lower than the upper bound (3.23) because of "check 2". With more details, from Fig.3.8 it can be observed that the row with the highest delays is the one with $j=2n$. In this row there are $2n$-1 signals with a delay equal to 1 ($i=1,...,n$ and $i=2n+1,...,m-1$). n signals with a delay equal to 2 ($i=n+1,...,2n$) and one signal with a delay equal to 0 ($i=0$). By using (3.15) we can write the multiplier delay as:

$$D_{XOR} = \left\lceil \log_2\left(8n-1\right) \right\rceil \qquad (3.27)$$

3.1.8 Equally Spaced Polynomial ESP ($p(z)=z^m+z^{t\Delta}+...+z^{\Delta}+1$ with $m=(t+1)\Delta$)

Fig.3.9 shows the general form of the ***M*** matrix corresponding to the



$j \uparrow$ / $i \rightarrow$	0	1		$m/2$	$m/2+1$		$m-1$
0	a_0	a_{m-1}	....	$a_{m/2}$	$a_{m/2-1}\oplus a_{m-1}$	....	$a_1\oplus a_{m/2+1}$
1	a_1	a_0	....	$a_{m/2+1}$	$a_{m/2}$	....	$a_2\oplus a_{m/2+2}$
	$\vdots$	$\vdots$	....	$\vdots$ a_{m-1}	$\vdots$ a_{m-2}	....	$\vdots$ $a_{m/2-1}\oplus a_{m-1}$ $a_{m/2}$
$m/2$	$a_{m/2}$	$a_{m/2-1}\oplus a_{m-1}$	....	$a_0\oplus a_{m/2}$	$a_{m/2-1}$ (1)	....	a_1 (1)
	$\vdots$	$a_{m/2}$ $\vdots$	....	$a_1\oplus a_{m/2+1}$ $\vdots$	$a_0\oplus a_{m/2}$ $\vdots$	....	a_2 $\vdots$ $a_{m/2-1}$
$m-1$	a_{m-1}	a_{m-2}	....	$a_{m/2-1}\oplus a_{m-1}$	$a_{m/2-2}\oplus a_{m-2}$	....	$a_0\oplus a_{m/2}$

Fig.3.7 – ***M*** matrix for the equally spaced trinomial (EST) $p(z)=z^m+z^{m/2}+1$.
The symbol "(1)" denotes $M_{j,i}$ terms for which "check 1" succeeds. The symbol "⊕" denotes XOR operation.

case of the equally spaced polynomial $p(z)=z^m+z^{t\Delta}+z^{(t-1)\Delta}+...+z^{\Delta}+1$ where m is assumed to be equal to $(t+1)\Delta$. In the figure the symbol "(1)" indicates the $M_{j,i}$ elements for which "check 1" succeeds. Similarly the symbol "(2)" indicates the success of "check 2".

Let us start by counting the number of XOR gates needed to compute the M matrix. From Fig.3.9 it can be observed that no simplification is done in the columns from i=1 to $i=\Delta$. Therefore the number of XOR gates needed to compute these columns is $t\Delta$. In the columns from $i=\Delta+1$ to $i=2\Delta$, the "check 1" succeeds for cells $M_{\Delta,i}$; in this case the number of needed XOR gates is $(t-1)\Delta$. In the columns from $i=2\Delta+1$ to $i=3\Delta$ the "check 1" succeeds both for $M_{\Delta,i}$ and $M_{2\Delta,i}$. The number of XOR gates is therefore $(t-2)\Delta$. In every successive Δ columns the number of XOR gates decreases by Δ. Finally, in the columns from $i=(t-1)\Delta+1$ to $i=t\Delta$, only Δ XOR gates are needed. No XOR gate is required for the remaining columns. In conclusion the total number of XOR gates needed to compute the $\boldsymbol{M}$ matrix with the proposed algorithm is:

$$N'_{XOR}=\sum_{i=1}^{t} i\cdot\Delta=(t+1)\cdot\frac{t}{2}\cdot\Delta=m\cdot\frac{t}{2} \qquad (3.28)$$

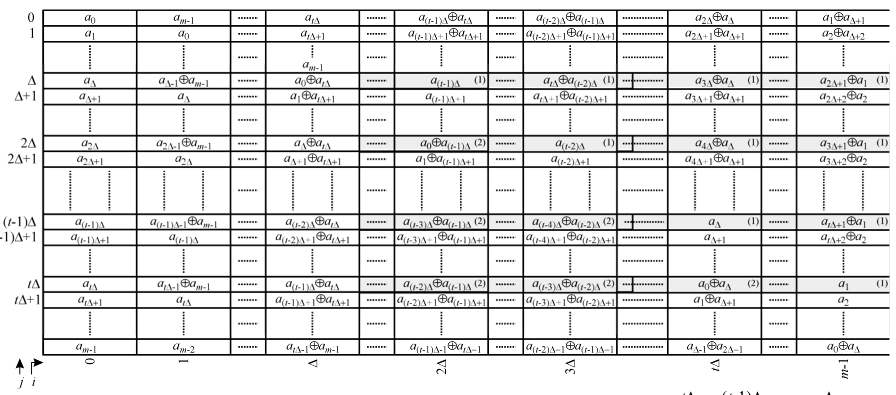
The total number of XOR gates needed for the full multiplier is therefore:

$$N_{XOR}=m\cdot(m-1)+\frac{m\cdot t}{2} \qquad (3.29)$$

In order to evaluate the multiplier delay we can observe that all $M_{j,i}$ terms in Fig.3.9 are either equal to a single variable or given by the

0	a_0	a_{m-1}	……	a_{2n}	……	$a_n\oplus a_{2n}$	……	$a_1\oplus a_{n+1}\oplus a_{2n+1}$
1	a_1	a_0	……	a_{2n+1}	……	$a_{n+1}\oplus a_{2n+1}$	……	$a_2\oplus a_{n+2}\oplus a_{2n+2}$
	$\vdots$ a_{n-1}	$\vdots$ a_{n-2}	……	$\vdots$ a_{m-1}	……	$\vdots$ $a_{2n-1}\oplus a_{m-1}$	……	$\vdots$ $a_n\oplus a_{2n}$
n	a_n	a_{n-1}	……	a_0	……	a_{2n}	……	$a_{n+1}\oplus a_{2n+1}$
	$\vdots$ a_{2n-1}	$\vdots$ a_{2n-2}	……	$\vdots$ a_{n-1}	……	$\vdots$ a_{m-1}	……	$\vdots$ a_{2n}
$2n$	a_{2n}	$a_{2n-1}\oplus a_{m-1}$	……	$a_n\oplus a_{2n}$	……	$a_0\oplus a_n\oplus a_{2n}$	……	$a_1\oplus a_{n+1}$ (2)
	$\vdots$	$\vdots$	……	$\vdots$	……	$\vdots$	……	$\vdots$
m-1	a_{m-1}	a_{m-2}	……	$a_{2n-1}\oplus a_{m-1}$	……	$a_{n-1}\oplus a_{2n-1}\oplus a_{m-1}$	……	$a_0\oplus a_n\oplus a_{2n}$
$\uparrow j$ $\rightarrow i$	0	1		n		$2n$		m-1

Fig.3.8 – $\boldsymbol{M}$ matrix for the missed equally spaced trinomial (mEST) $p(z)=z^{3n}+z^{2n}+1$. The symbol "(2)" denotes $M_{j,i}$ terms for which "check 2" succeeds. The symbol "$\oplus$" denotes XOR operation.



$$\begin{array}{r|c|c|c|c|c|c|c|c|c|c|c|c|}
0 & a_0 & a_{m-1} & \cdots & a_{t\Delta} & \cdots & a_{(t-1)\Delta}\oplus a_{t\Delta} & \cdots & a_{(t-2)\Delta}\oplus a_{(t-1)\Delta} & \cdots & a_{2\Delta}\oplus a_{\Delta} & \cdots & a_1\oplus a_{\Delta+1} \\
1 & a_1 & a_0 & \cdots & a_{t\Delta+1} & \cdots & a_{(t-1)\Delta+1}\oplus a_{t\Delta+1} & \cdots & a_{(t-2)\Delta+1}\oplus a_{(t-1)\Delta+1} & \cdots & a_{2\Delta+1}\oplus a_{\Delta+1} & \cdots & a_2\oplus a_{\Delta+2} \\
 & \vdots & \vdots & \cdots & \vdots\; a_{m-1} & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\
\Delta & a_{\Delta} & a_{\Delta-1}\oplus a_{m-1} & \cdots & a_0\oplus a_{t\Delta} & \cdots & a_{(t-1)\Delta}\ ^{(1)} & \cdots & a_{t\Delta}\oplus a_{(t-2)\Delta}\ ^{(1)} & \cdots & a_{3\Delta}\oplus a_{\Delta}\ ^{(1)} & \cdots & a_{2\Delta+1}\oplus a_1\ ^{(1)} \\
\Delta+1 & a_{\Delta+1} & a_{\Delta} & \cdots & a_1\oplus a_{t\Delta+1} & \cdots & a_{(t-1)\Delta+1} & \cdots & a_{t\Delta+1}\oplus a_{(t-2)\Delta+1} & \cdots & a_{3\Delta+1}\oplus a_{\Delta+1} & \cdots & a_{2\Delta+2}\oplus a_2 \\
 & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\
2\Delta & a_{2\Delta} & a_{2\Delta-1}\oplus a_{m-1} & \cdots & a_{\Delta}\oplus a_{t\Delta} & \cdots & a_0\oplus a_{(t-1)\Delta}\ ^{(2)} & \cdots & a_{(t-2)\Delta}\ ^{(1)} & \cdots & a_{4\Delta}\oplus a_{\Delta}\ ^{(1)} & \cdots & a_{3\Delta+1}\oplus a_1\ ^{(1)} \\
2\Delta+1 & a_{2\Delta+1} & a_{2\Delta} & \cdots & a_{\Delta+1}\oplus a_{t\Delta+1} & \cdots & a_1\oplus a_{(t-1)\Delta+1} & \cdots & a_{(t-2)\Delta+1} & \cdots & a_{4\Delta+1}\oplus a_{\Delta+1} & \cdots & a_{3\Delta+2}\oplus a_2 \\
 & \vdots\quad\vdots & \vdots\quad\vdots & \cdots & \vdots\quad\vdots & \cdots & \vdots\quad\vdots & \cdots & \vdots\quad\vdots & \cdots & \vdots\quad\vdots & \cdots & \vdots\quad\vdots \\
(t-1)\Delta & a_{(t-1)\Delta} & a_{(t-1)\Delta-1}\oplus a_{m-1} & \cdots & a_{(t-2)\Delta}\oplus a_{t\Delta} & \cdots & a_{(t-3)\Delta}\oplus a_{(t-1)\Delta}\ ^{(2)} & \cdots & a_{(t-4)\Delta}\oplus a_{(t-2)\Delta}\ ^{(2)} & \cdots & a_{\Delta}\ ^{(1)} & \cdots & a_{t\Delta+1}\oplus a_1\ ^{(1)} \\
(t-1)\Delta+1 & a_{(t-1)\Delta+1} & a_{(t-1)\Delta} & \cdots & a_{(t-2)\Delta+1}\oplus a_{t\Delta+1} & \cdots & a_{(t-3)\Delta+1}\oplus a_{(t-1)\Delta+1} & \cdots & a_{(t-4)\Delta+1}\oplus a_{(t-2)\Delta+1} & \cdots & a_{\Delta+1} & \cdots & a_{t\Delta+2}\oplus a_2 \\
 & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\
t\Delta & a_{t\Delta} & a_{t\Delta-1}\oplus a_{m-1} & \cdots & a_{(t-1)\Delta}\oplus a_{t\Delta} & \cdots & a_{(t-2)\Delta}\oplus a_{(t-1)\Delta}\ ^{(2)} & \cdots & a_{(t-3)\Delta}\oplus a_{(t-2)\Delta}\ ^{(2)} & \cdots & a_0\oplus a_{\Delta}\ ^{(2)} & \cdots & a_1\ ^{(1)} \\
t\Delta+1 & a_{t\Delta+1} & a_{t\Delta} & \cdots & a_{(t-1)\Delta+1}\oplus a_{t\Delta+1} & \cdots & a_{(t-2)\Delta+1}\oplus a_{(t-1)\Delta+1} & \cdots & a_{(t-3)\Delta+1}\oplus a_{(t-2)\Delta+1} & \cdots & a_1\oplus a_{\Delta+1} & \cdots & a_2 \\
 & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots & \cdots & \vdots \\
 & a_{m-1} & a_{m-2} & \cdots & a_{t\Delta-1}\oplus a_{m-1} & \cdots & a_{(t-1)\Delta-1}\oplus a_{t\Delta-1} & \cdots & a_{(t-2)\Delta-1}\oplus a_{(t-1)\Delta-1} & \cdots & a_{\Delta-1}\oplus a_{2\Delta-1} & \cdots & a_0\oplus a_{\Delta} \\
\uparrow j\ \ \rightarrow i & 0 & 1 & & \Delta & & 2\Delta & & 3\Delta & & t\Delta & & m\text{-}1
\end{array}$$

Fig.3.9 – ***M*** matrix for the equally spaced polynomial (ESP) $p(z)=z^m+z^{t\Delta}+z^{(t\text{-}1)\Delta}+\ldots+z^{\Delta}+1$.
The symbols "(1)" and "(2)" denotes $M_{j,i}$ terms for which "check 1" and "check 2" succeeds respectively.
The symbol "⊕" denotes XOR operation.

XOR of two variables. Our "check 1" assures that all $M_{j,i}$ terms equal to a single variable have a delay equal to 0. The "check 2" makes the delay of all remaining $M_{j,i}$ terms (given by the XOR of two variables) equal to 1. By looking to Fig.3.9 it can be observed that the critical row is the one with $j=t\Delta$. In this case, in fact, Δ terms have a delay of 0 while the remaining $t\Delta$ terms $M_{t\Delta,i}$ have a delay equal to 1. The critical multiplier output is therefore $c_{t\Delta}$; the delay of this signal, using (3.15) is:

$$D_{XOR}=\left\lceil \log_2\left(t\cdot\Delta\cdot 2+\Delta\right)\right\rceil=\left\lceil \log_2\left(2\left(m-\frac{\Delta}{2}\right)\right)\right\rceil \tag{3.30}$$

It is worth to note that the equally spaced polynomial reduces to the equally spaced trinomial when $t=1$. In this case, in fact, (3.29),(3.30) reduces to (3.24),(3.25).

3.1.9 Pentanomial p(z)=z^{m}+z^{n+1}+z^{n}+z+1 (with n≤m/2 1)

The general form of the M matrix corresponding to the pentanomial $z^m+z^{n+1}+z^n+z+1$ with $n\leq m/2\text{-}1$ is shown in Fig.3.10. The figure reports all columns from $i=1$ to $i=m\text{-}n$; the final n columns are not shown. It can be observed that the "check 2" succeeds for all $M_{n+1,i}$ elements with $i=n+1,\ldots,m\text{-}n\text{-}1$. For example $M_{n+1,n+1}$ element, by using "check 2", is calculated as $M_{1,1}$ XOR $M_{1,n+1}$, with a delay equal to 2. Without "check 2" this elements would be obtained as $M_{n,n}$ XOR $M_{m\text{-}1,n}$ with a total delay of 3.

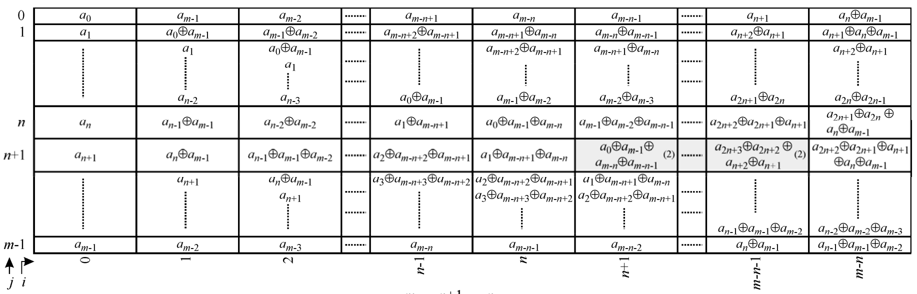


Fig.3.10 – ***M*** matrix for the pentanomial $z^m+z^{n+1}+z^n+z+1$ (with $n \leq m/2-1$). Columns from i=1 to i=m-n are shown.
The symbol "(2)" denotes $M_{j,i}$ terms for which "check 2" succeeds. The symbol "⊕" denotes XOR operation.

Since "check 1" fails for all $M_{j,i}$ elements, the number of XOR gates needed to implement the multiplier is simply obtained by particularizing (3.19) for r=5:

$$N_{XOR} = (m+3)\cdot(m-1) \qquad (3.31)$$

The multiplier delay can be evaluated by considering the ***D*** matrix whose element $D_{j,i}$ is the delay of the $M_{j,i}$ element. Fig.3.11 reports this matrix for the considered pentanomial. The figure shows that the critical multiplier output is c_{n+1}, corresponding to the ***D*** matrix row with j=n+1. This row contains one element with zero delay, one element with a delay equal to 1, m-n-2 elements with a delay equal to 2, two elements with a delay of 3 and n-2 elements with a delay equal to 4. By using (3.15) the total multiplier delay can be computed as:

$$D_{XOR} = \left\lceil \log_2\left(4m+12n-21\right)\right\rceil \qquad (3.32)$$

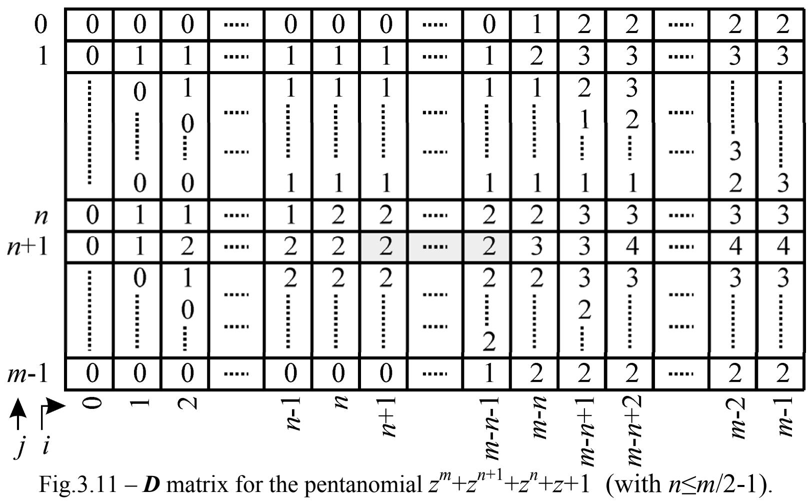


Fig.3.11 – ***D*** matrix for the pentanomial $z^m+z^{n+1}+z^n+z+1$ (with $n \leq m/2-1$).

Technique		Multiplier topology	# XOR gates	delay (# of XOR gates on the critical path)
Trinomial z^m+z+1				
Koc [101]	*TCOMP 1999*	Mastrovito	m^2-1	$\lceil \log_2(2m) \rceil$
Koc [102]	*TCOMP 2000*			
Zhang [104]	*TCOMP 2001*			
Masoleh [115]	*TCOMP 2004*	Modular Reduction		$\lceil \log_2(4m) \rceil$
Proposed		Mastrovito		$\lceil \log_2(2m-1) \rceil$
Trinomial z^m+z^n+1 $2 \leq n \leq m/2$				
Koc [101]	*TCOMP 1999*	Mastrovito	m^2-1	$\lceil \log_2(4m) \rceil$
Koc [102]	*TCOMP 2000*			
Zhang [104]	*TCOMP 2001*			
Masoleh [115]	*TCOMP 2004*	Modular Reduction		$\lceil \log_2(4(m-1)) \rceil$
Proposed		Mastrovito		$\lceil \log_2(2m+2n-3) \rceil$
Equally Spaced Trinomial (EST) $z^m+z^{m/2}+1$				
Koc [101]	*TCOMP 1999*	Mastrovito	$m^2 - m/2$	$\lceil \log_2(2m) \rceil$
Koc [102]	*TCOMP 2000*			
Zhang [104]	*TCOMP 2001*			
Masoleh [115]	*TCOMP 2004*	Modular Reduction		
Proposed		Mastrovito		$\lceil \log_2(3/2\, m) \rceil$
Trinomial z^m+z^n+1 $m/2 < n \leq m-1$ $k = \left\lceil \frac{m-1}{m-n} \right\rceil$				
Koc [101]	*TCOMP 1999*	Mastrovito	m^2-1	$\lceil \log_2(2^k m) \rceil$
Koc [102]	*TCOMP 2000*			
Zhang [104]	*TCOMP 2001*			
Masoleh [21	*TCOMP 2004*	Modular Reduction		$\lceil \log_2(2^k(m-1-(k-2)(m-n))) \rceil$
Proposed		Mastrovito		$\lceil \log_2(2^k(m-1-(k-2)(m-n))-2(m-n)+1) \rceil$
missed Equally Spaced Trinomial (mEST) $z^{3n}+z^{2n}+1$ $m=3n$				
Koc [101]	*TCOMP 1999*	Mastrovito	m^2-1	$\lceil \log_2(24n) \rceil$
Koc [102]	*TCOMP 2000*			
Zhang [104]	*TCOMP 2001*			
Masoleh [21	*TCOMP 2004*	Modular Reduction		$\lceil \log_2(16(n-1/2)) \rceil$
Proposed		Mastrovito		$\lceil \log_2(8n-1) \rceil$

Tab.3.2 – Comparison with related polynomial basis multiplier for Trinomials.

3.1.10Comparison with the state of the art

A comparison between the proposed approach and the recently proposed most effective polynomial basis multipliers is shown in Tab.3.2 and Tab.3.3. Tab.3.2 considers the trinomial classes. It can be observed that the proposed approach results in the same complexity and in a better delay with respect to previously proposed techniques. As an example, in the case of the trinomials x^m+x^n+1 with $2 \leq n \leq m/2$,

Technique	Multiplier topology	# XOR gates	delay (# of XOR gates on the critical path)
Equally Spaced Polynomial (ESP) $z^m+z^{t\Delta}+...+z^{\Delta}+1$ $m=(t+1)\Delta$			
Koc [102] (method II) *TCOMP 2000*	non-Mastrovito	m^2-Δ	$\lceil \log_2(2m) \rceil$
Zhang [104] *TCOMP 2001*			
Masoleh [115] *TCOMP 2004*	Modular Reduction		
Koc [102] (method I) *TCOMP 2000*	Mastrovito	$m(m-1)+\frac{m \cdot t}{2}$	
Proposed	Mastrovito		$\lceil \log_2(2(m-\Delta/2)) \rceil$
Pentanomial $z^m+z^{n+1}+z^n+z+1$ $n \leq m/2-1$			
Zhang [104] *TCOMP 2001*	Mastrovito	$m(m-1)+3(m-1)$	$\lceil \log_2(64m) \rceil$
Henriquez [108] *TCOMP 2003*	Modular Reduction	$m(m-1)+2n$	$\lceil \log_2(8m) \rceil$
Masoleh [108] *TCOMP 2004*		$m(m-1)$	$\lceil \log_2(8(m-1)) \rceil$
Proposed	Mastrovito	$m(m-1)+3(m-1)$	$\lceil \log_2(4m+12n-21) \rceil$

Tab.3.3 – Comparison with related polynomial basis multiplier for Equally Spaced Polynomials and a class of Pentanomials.

our technique results in an asymptotical delay of $\log_2(2m)$ while the other approaches presents an asymptotical delay of $\log_2(4m)$. A significant delay improvement can also be noted for trinomials with $m/2<n\leq m-1$, for the equally spaced trinomial and for the missed equally spaced trinomial. Remarkable is the delay obtained for EST which is the lowest delay reported in the Literature for a GF(2^m) multiplier.
The Tab.3.3 considers both the case of equally spaced polynomials and pentanomials $z^m+z^{n+1}+z^n+z+1$ (with $n\leq m/2-1$). It can be observed that the proposed technique results in the same complexity of previously proposed Mastrovito multipliers. When compared to non-Mastrovito multiplier topologies (like “method II” proposed in [102] for equally spaced polynomials or Modular Reduction techniques) proposed approach result in an higher number of XOR gates. In all considered cases our technique results in a lower delay with respect to other approaches. As an example in the considered class of pentanomials the asymptotical delay of our solution is $\log_2(4m)$, while the best available approach [115] results in an asymptotical delay of $\log_2(8m)$.
The Tab.3.4 reports a comparison considering some field generator polynomials $p(z)$ better suited for Reed-Solomon encoding and decoding. In the table, we have considered the Mastrovito multiplier technique of [104], the non-Mastrovito technique of [115], and the dual basis GF(2^m) multipliers of [107] and [108]. The multiplier of [107] has been employed in [110] for the implementation of a Reed-

Solomon decoder. For each Galois Field GF(2^m) (with $4 \leq m \leq 12$) the same $p(z)$ polynomial has been used to compare all the techniques. Among all possible irreducible polynomials $p(z)$, we restricted our attention to primitive polynomials since, in this case, the architecture complexity of Reed-Solomon circuits is sensibly reduced. For each field, $p(z)$ is chosen as the primitive polynomial which gives best performances for all considered multipliers. The used $p(z)$ are reported in Tab.3.4.

Multipliers are compared in terms of the total number of XOR gates required to implement the circuit (N_{XOR}) and the number of XOR gates on the critical path (D_{XOR}). For dual basis ([107],[108]) two numbers are reported. The first number does not take into account the extra delay and gates needed for the dual to standard basis conversions at the multiplier inputs and outputs. In parenthesis the value of the delay and of the architecture complexity considering basis conversion is reported.

The results of Tab.3.4 show that, neglecting the dual to standard basis conversion, in the fields GF(2^4), GF(2^6) and GF(2^7), where the trinomial $p(z)=z^m+z+1$ can be employed, the proposed technique achieves exactly the same performances of [104],[107]. For all other considered GF(2^m) the proposed architecture improves the timing performances with respect to Zhang [104], Fenn [107] and Masoleh [115] multipliers. In the case of GF(2^8), widely used for Reed-Solomon coding, the proposed technique has a critical path with 3 fewer XOR gates with respect to [104] and 1 fewer XOR gate with respect to [107]. Comparing with [108] we have the same critical path delay, with two less XOR gates required for circuit implementation. It

	Zhang [104]		Fenn [107]		Henriquez [108]		Masoleh [115]		proposed		primitive
m	DXOR	NXOR	DXOR	NXOR	DXOR	NXOR	DXOR	NXOR	DXO	NXOR	polynomial p(z)
4	3	15	3 (3)	15 (15)	-	-	4	15	3	15	$z^4 + z^1 + 1$
5	5	24	5 (5)	24 (24)	-	-	4	24	4	24	$z^5 + z^2 + 1$
6	4	35	4 (4)	35 (35)	-	-	5	35	4	35	$z^6 + z^1 + 1$
7	4	48	4 (4)	48 (48)	-	-	5	48	4	48	$z^7 + z^1 + 1$
8	9	77	7 (9)	77 (81)	6 (7)	79 (81)	7	77	6	77	$z^8 + z^4 + z^3 + z^2 + 1$
9	6	80	6 (6)	80 (80)	-	-	5	80	5	80	$z^9 + z^4 + 1$
10	6	99	6 (6)	99 (99)	-	-	6	99	5	99	$z^{10} + z^3 + 1$
11	6	120	6 (6)	120 (120)	-	-	6	120	5	120	$z^{11} + z^2 + 1$
12	10	165	8 (10)	165 (171)	-	-	8	165	7	165	$z^{12} + z^6 + z^5 + z^3 + 1$

Tab.3.4 – GF(2^m) multipliers performances. D_{XOR} is the number of XOR gates on the critical path and N_{XOR} is the total number of XOR.

is worth to note that, if dual-to-standard basis conversion cannot be avoided, the solution of [108] requires four more XOR gates with one more XOR gate on the critical path with respect to the proposed solution.

3.1.11Key-Equation Solving Block For a RS(255,239)

A t-error primitive Reed-Solomon (n,k) code with symbols in GF(2^m) has codewords of length n=2^m-1 and satisfies $2t$=n-k. A widely used Reed-Solomon code is the RS(255,239) which is based on GF(2^8) and can correct up to 8 erroneous symbols. In a Reed-Solomon decoder circuit, the main block, which strongly influences circuit area and limits maximum speed, is the key-equation solver, which, given the syndromes [95] polynomial $S(x)$, is able to find a solution for the following equation:

$$\Lambda(x) \bullet S(x) = \Omega(x) \mod x^{2t} \qquad (3.33)$$

where $\Lambda(x)$, S(x) and $\Omega(x)$ are polynomials over GF(2^m) of degree t, $2t$-1 and t-1 respectively. $\Lambda(x)=\sum_{i=0}^{t}\lambda_i x^i$ is called error locator polynomial, whereas $\Omega(x)=\sum_{i=0}^{t-1}\omega_i x^i$ is the error evaluator polynomial.

In order to highlight the effectiveness of the proposed multiplier in a typical application, in this section we investigate the performances achievable by using the proposed GF(2^m) multiplier scheme to design a key equation solver block for RS(255,239). For comparison the same architecture is implemented with the Henriquez [108] multiplier which was the fastest known multiplier for GF(2^8).

Many techniques can be used to solve the key-equation, in this section we consider the inversion-less ([95],[109],[110]) Berlekamp-Massey algorithm shown in Fig.3.12. This algorithm staring from the Syndromes S_i is able to compute the error locator polynomial, given by $\Lambda^{(2t)}(x)$, in a $2t$ iterative cycle which involves additions and multiplications over GF(2^m). Note that, in order to obtain the result, the algorithm introduces a new polynomial $B(x)=\sum_{i=0}^{t-1}b_i x^i$ over GF(2^8).

```
Initialize:
    $\Lambda^{(0)}(x) = B^{(0)}(x) = 1;\ \gamma^{(0)} = 1; k^{(0)} = 0$
FOR r=0,..,2t-1 LOOP
    $\delta^{(r)} = \sum_{j=0}^{r} \lambda_j^{(r)} \cdot S_{r-j} \qquad \lambda_j^{(r)} = 0, \forall j > t$
    $\Lambda^{(r+1)}(x) = \gamma^{(r)} \cdot \Lambda^{(r)}(x) + \delta^{(r)} \cdot B^{(r)}(x) \cdot x$
IF $\delta^{(r)} \neq 0$ OR $k^{(r)} \geq 0$ THEN {
        $B^{(r+1)}(x) = \Lambda^{(r)}(x)$
        $k^{(r+1)} = -k^{(r)} - 1;\ \gamma^{(r+1)} = \delta^{(r)}$ }
ELSE {
        $B^{(r+1)}(x) = x \cdot B^{(r)}(x)$
        $k^{(r+1)} = k^{(r)} + 1;\ \gamma^{(r+1)} = \gamma^{(r)}$ }

END LOOP
```

Fig.3.12 – Inversion-less BerlekampMassey algorithm equations.

The well known implementation of the considered Berlekamp-Massey algorithm is shown in Fig.3.13. The architecture starts from the syndromes polynomial $S(x)$ and computes in each clock cycle the new discrepancy value δ, updating $\Lambda(x)$ and $B(x)$ polynomials as required by the algorithm of Fig.3.12. After $2t$ clock cycles, the error locator polynomial will be stored in the registers λ_i. The blocks marked as **BC** are basis converter needed only if the Henriquez multipliers are used in the circuit. The need of basis conversion increases by two the number of XOR gates on the critical path, resulting in a slower circuit. As it can be seen the delay of the circuit in Fig.3.13 is equal to the time needed to calculate the discrepancy δ plus the delay of one

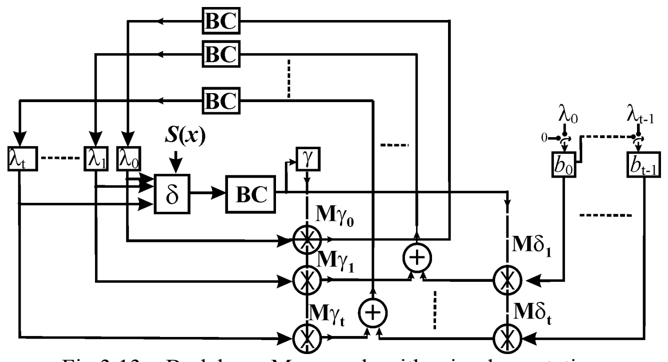


Fig.3.13 – BerlekampMassey algorithm implementation.

GF(2^8) multiplier, one GF(2^8) adder and two base converters. From Fig.3.12 it can be seen that the delay needed to calculate the discrepancy δ is equal to the sum of the delay of one GF(2^8) multiplier and one t+1 inputs GF(2^8) adder. In the architecture of Fig.3.13 the multipliers Mγ_i share the same operand γ. Considering this operand as the A input of the Mastrovito multiplier, because of (3.6),(3.7) the matrix $\boldsymbol{M}$ will be the same for all the multipliers Mγ_i. Hence all multipliers Mγ_i can share the same first block. The same technique can be also used for the multipliers Mδ_i which share the same operand δ. Please note that the same technique can be used with the Henriquez [108] multiplier.
The architecture in Fig.3.13 has been implemented for a 2.5V 1P5M 0.25μm technology, using either proposed multipliers or Henriquez [108] multipliers. This analysis has two objectives: determine the advantages obtainable by employing the proposed multiplier design in a real application; establish if starting from a previously proposed multiplier solution (like the solution of Henriquez [108]) the synthesizer is able by itself to find a circuit topology with a delay comparable with the one of our multiplier. To that purpose we have synthesized the circuits with the latest version of a state of the art commercial synthesizer by employing the highest-effort options. The implementation results are shown in Tab.3.5, where it is also reported the number of XOR gates on the critical path (D_{XOR}) as determined from the data of Tab.3.4. It can be observed that the use of the proposed multiplier allows achieving a 9.6% increase in the maximum operating speed giving about the same area occupation of the Henriquez based implementation. It is worth to highlight that the delay results are in a perfect agreement with the D_{XOR} values obtained from the simple analysis of Tab.3.4. Therefore the synthesizer is not able to re-obtain our speed-optimized implementation, starting from a non optimal description (like the one corresponding to the Henriquez approach). This can be explained with the observation that in a GF(2^m) multiplier there is a strong correlation between the delays of the different paths in the circuit.

3.1.12 Conclusions

In this section a new architecture for Galois fields GF(2^m) multipliers has been developed. The proposed multiplier is based on the Mastrovito multiplication scheme and standard basis representation.

Our multiplier can easily be designed with a systematic approach for any field GF(2^m) and any field generator polynomial $p(z)$. Our solution is extremely more simple to design and implement with respect to other recently proposed approaches (like the ones of [104] and [115]).

In the section we have analytically derived the performances of our multiplier in the case of equally spaced trinomials, missed equally spaced trinomials, equally spaced polynomials and a class of pentanomials. An upper bound is also given for a general trinomial. The comparison with the state of the art shows that in the case of trinomials our solution provides the best performances in terms of both circuit complexity and speed. Remarkable is the delay obtained for equally spaced trinomials which is the lowest delay reported in the Literature for a GF(2^m) multiplier. The comparison in the case of equally spaced polynomials and the considered class of pentanomials shows that the novel multiplier results in the lower delay.

The section also considers a comparison for the GF(2^m) fields better suited for Reed-Solomon encoding and decoding. In this case the new architecture always achieves the best performances. A sensible improvement in timing performances is shown for the widely used GF(2^8) field.

We have demonstrated the effectiveness of the proposed approach in a real application by implementing the Berlekamp-Massey algorithm for a Reed-Solomon (255,239) decoder with respectively the proposed multiplier and the fastest previously proposed multiplier. The comparison shows that the proposed solution achieves about a 10% improvement in the circuit maximum clock frequency, as predicted by our simple, gate level, analysis.

exploited multipier	D_{XOR}	Area (mm^2)	Delay (ns)	Frequency (MHz)
Henriquez [108]	19	0,164	4,37	229
Proposed	17	0,164	3,99	251

Tab.3.5 – Inversion-less Berlekamp-Massey algorithm implementation for RS(255,239) decoder. D_{XOR} is the number of XOR gates on the critical path.

3.2 Reed-Solomon Decoder

3.2.1 Introduction

The need of portable circuits able to communicate with high bandwidths is pushing in the development of high-speed and low-power Reed-Solomon decoders.

Reed-Solomon decoding is based on Galois Field (GF(2^m)) arithmetic. The most intensive procedure is the solution of so called key-equation which gives the error locator and error evaluator polynomials. The main techniques proposed to solve the key-equation are the Euclidean algorithm [118] and the Berlekamp-Massey algorithm [118],[119].

Standard Euclidean algorithm [118] requires the computation of finite-field inversion which, due to the high computational complexity, degrade the maximum bit-rate of the decoder. In [120] an inversion-free Euclidean algorithm method is proposed by improving timing performances with respect to standard Euclidean algorithm technique.

A lower complexity with respect to Euclidean algorithm is obtained by Berlekamp-Massey algorithm, which, in its original form [118],[119], requires the computation of the finite-field inversion. Solution provided in [119], known as Berlekamp architecture, evaluates at the same time both the error locator and the error evaluator polynomials. On the other hand, Blahut [118] architecture calculates error evaluator polynomial in a second step slightly increasing latency with a substantial reduction in hardware complexity. Both solutions can be rearranged to avoid the finite-filed inversion. As an example, in [121], inversion-free Blahut architecture is presented.

In [122] Chang *et al* introduce a decomposed solution for inversion-free Berlekamp-Massey algorithm. The method greatly reduces hardware complexity and delay by substantially increasing circuit latency. In [123] Sarwate *et al* propose an high-speed and low latency architecture based on inversion-free Berlekamp-Massey algorithm. A drawback of this solution is the increased circuit complexity with respect to both Blahut and Berlekamp architectures. In [124] a decoder based on Berlekamp-Massey algorithm is shown which achieves good speed and silicon area occupation.

In this section a novel technique has been used to design a high-performance Reed-Solomon decoder in 0.25µm CMOS

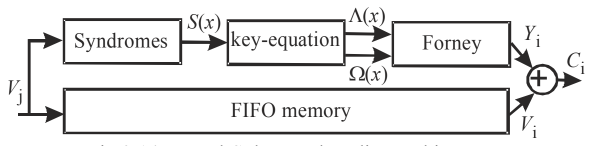


Fig.3.14 – Reed-Solomon decoding architecture.

technology. The technique introduces a new architecture for the implementation of inversion-free Berlekamp-Massey algorithm. The architecture allows to improve operating speed of standard Blahut and Berlekamp solutions with a reduced complexity with respect to Sarwate architecture.

3.2.2 Reed-Solomon decoding

In Reed-Solomon (RS) codes the bit stream to transmit is divided in symbols each formed by *m* bits assumed to be a Galois Field (GF(2^m)) value. In a RS (*n,k*) code, symbols are organized in blocks each of a fixed length *n*. A block is made up with *k* data symbols, and *n-k* parity symbols, used to recover errors. RS codes can correct up to *t* wrong symbols in the block of *n* symbols; *k* being chosen so that *n-k*=2*t* and *n* being equal to 2^m-1.

RS decoding procedure can be divided in three steps: syndromes calculation, key-equation solution, error detection and correction (Forney algorithm), as shown in Fig.3.14.

In the first step 2*t* syndromes are evaluated:

$$S_i = \sum_{j=0}^{n-1} V_j \cdot \left(\alpha^i\right)^j \qquad i = 0,..,2t-1 \qquad (3.34)$$

where S_i is the *i*-th Syndrome, V_j is the *j*-th symbol of the received block, and α is the primitive element of GF(2^m). Syndromes S_i can be arranged in Syndromes polynomial:

$$S(x) = \sum_{i=0}^{2t-1} S_i \cdot x^i \qquad (3.35)$$

In the second step the error locator polynomial $\Lambda(x)$ and the error evaluator polynomial $\Omega(x)$ are calculated by solving the key-equation:

$$\Lambda(x) \cdot S(x) = \Omega(x) \mod x^{2t} \qquad (3.36)$$

The last step of the decoding procedure is the error detection and correction, which can be realized by using the Forney algorithm. Roots of $\Lambda(x)$ are related to the position of the symbols affected by

```
Initialize:
  $\Lambda^{(0)}(x) = B^{(0)}(x) = 1;\ \gamma^{(0)} = 1; k^{(0)} = 0;\ \Omega^{(0)}(x) = \Theta^{(0)}(x) = S(x)$
FOR r=0,..,2t-1 LOOP
    $\delta^{(r)} = \sum_{j=0}^{r} \lambda_j^{(r)} \cdot S_{r-j} \qquad \lambda_j^{(r)} = 0, \forall j > t$
    $\Lambda^{(r+1)}(x) = \gamma^{(r)} \cdot \Lambda^{(r)}(x) + \delta^{(r)} \cdot B^{(r)}(x) \cdot x$
    $\Omega^{(r+1)}(x) = \gamma^{(r)} \cdot \Omega^{(r)}(x) + \delta^{(r)} \cdot \Theta^{(r)}(x) \cdot x$
IF $\delta^{(r)} \neq 0$ OR $k^{(r)} \geq 0$ THEN {
    $B^{(r+1)}(x) = B^{(r)}(x)$;  $\Theta^{(r+1)}(x) = \Theta^{(r)}(x)$
    $k^{(r+1)} = -k^{(r)} - 1$;  $\gamma^{(r+1)} = \delta^{(r)}$ }
ELSE {
    $B^{(r+1)}(x) = x \cdot B^{(r)}(x)$;  $\Theta^{(r+1)}(x) = x \cdot \Theta^{(r)}(x)$
    $k^{(r+1)} = k^{(r)} + 1$;  $\gamma^{(r+1)} = \gamma^{(r)}$ }
END LOOP
```
Fig.3.15 – BerlekampMassey algorithm equations.

error so that if $\Lambda(\alpha^{-w})=0$ then the *w*-th symbol has to be corrected. Error magnitude is given by:

$$Y_w = \frac{\Omega\left(\alpha^{-w}\right)}{\alpha^{-w}\Lambda'\left(\alpha^{-w}\right)} \tag{3.37}$$

where $\Lambda'(x)$ is the formal derivative of the polynomial $\Lambda(x)$, and is defined as $\Lambda'(x) = \lambda_1 + \lambda_3 \cdot x^2 + \lambda_5 \cdot x^4 + \dots$.

FIFO memory in Fig.3.14 is used to store received V_j symbols during the time needed to complete the three decoding steps.

3.2.3 Key-equation block

In this section we will introduce a novel architecture of inversion-free Berlekamp-Massey algorithm (BMA) for the implementation of key-equation block.

Inversion-free BMA uses four polynomials:

$$\Lambda^{(r)}(x) = \sum_{i=0}^{t} \lambda_i^{(r)} x^i \ , \ \mathrm{B}^{(r)}(x) = \sum_{i=0}^{t-1} \beta_i^{(r)} x^i \tag{3.38}$$

$$\Omega^{(r)}(x) = \sum_{i=0}^{t-1} \omega_i^{(r)} x^i \ , \ \Theta^{(r)}(x) = \sum_{i=0}^{t-2} \vartheta_i^{(r)} x^i \tag{3.39}$$

in a 2*t* iterative steps algorithm. The algorithm is shown in Fig.3.15, where multiplication and addition operations are defined in GF(2^m). The value $\Lambda^{(2t)}$(x) and $\Omega^{(2t)}$(x), obtained at the end of the cycle, are the error locator polynomial Λ(x) and the error evaluator polynomial Ω(x)

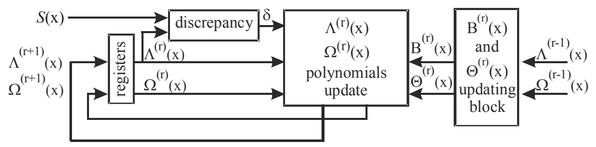


Fig.3.16 – Berlekamp architecture for BMA implementation.

respectively. Please note that the algorithm shown in Fig.3.15 calculates both the error locator polynomial and the error evaluator polynomial as proposed by Berlekamp [119],[123]. In recent years most researchers have used the formulation of BMA given by Blahut [118] in which only $\Lambda^{(r)}(x)$ and $B^{(r)}(x)$ are calculated iteratively, the polynomial $\Omega(x)$ being calculated, according to key equation (3.36), as the terms of degree lower than t of the polynomial multiplication $\Lambda(x)\cdot S(x)$.

The main steps in algorithm of Fig.3.15, are the evaluation of discrepancy $\delta^{(r)}$, and the updating of $\Lambda^{(r)}$(x) and $\Omega^{(r)}$(x) polynomials. As you can see, the discrepancy $\delta^{(r)}$ is evaluated from $\Lambda^{(r)}$(x) and syndromes S_i with the following equation:

$$\delta^{(r)} = \sum_{j=0}^{r} \lambda_j^{(r)} \cdot S_{r-j} \qquad (3.40)$$

where $\lambda^{(r)}{}_j$ is assumed to be equal to zero for $j>t$.

Polynomials $\Lambda^{(r)}$(x) and $\Omega^{(r)}$(x) polynomials are updated according to:

$$\Lambda^{(r+1)}(x) = \gamma^{(r)} \cdot \Lambda^{(r)}(x) + \delta^{(r)} \cdot \mathrm{B}^{(r)}(x) \cdot x \qquad (3.41)$$

$$\Omega^{(r+1)}(x) = \gamma^{(r)} \cdot \Omega^{(r)}(x) + \delta^{(r)} \cdot \Theta^{(r)}(x) \cdot x \qquad (3.42)$$

Berlekamp [119],[123] architecture for the implementation of BMA directly use the algorithm described in Fig.3.15 by employing the feed-back loop based architecture shown in Fig.3.16. In this architecture, in each clock cycle, discrepancy $\delta^{(r)}$ is evaluated with a delay of one multiplier and one t+1 input adder (see (3.40)), and subsequently, polynomial $\Lambda^{(r+1)}$(x) and $\Omega^{(r+1)}$(x) are updated with another delay of one multiplier and one 2-input adder (see (3.41) ,(3.42)). Therefore the total critical path delay includes two multipliers and two adders. Blahut architecture [118] reduces circuit complexity with respect to Berlekamp solution by maintaining the same critical path delay.

The number of clock cycles needed to implement the BMA directly influences the latency of the circuit. As shown in Fig.3.1 the higher the latency, the larger is the size of the memory needed for the

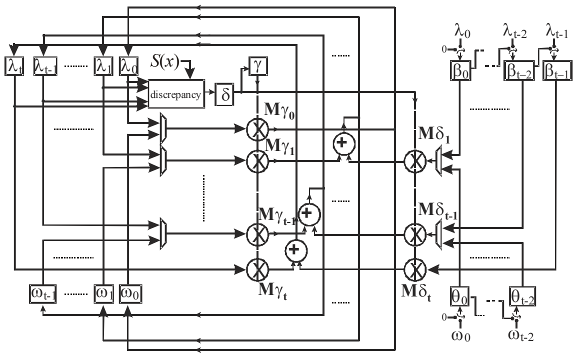


Fig.3.17 – proposed BMA implementation.

decoder. Berlekamp solutions provides the lowest latency (2*t* clock cycles) requiring the minimum memory size.
The timing performances of BMA circuit can be improved introducing architectures in which the calculation of the discrepancy $\delta^{(r)}$ and the calculation of $\Lambda^{(r+1)}(x)$ and $\Omega^{(r+1)}(x)$ do not belong to the same combinatorial path. Two recently proposed techniques [122],[123] use this principle to achieve a delay of one multiplier and one 2-input adder. In the solution proposed by Chang *et al* [122] the discrepancy calculation is divided from the computation of $\Lambda^{(r+1)}(x)$ and $\Omega^{(r+1)}(x)$ by decomposing BMA. Unfortunately decomposed technique substantially increases the latency for the key-equation solving, requiring a bigger FIFO memory in the structure of Fig.3.14. Sarwate *et al* [123] propose a low latency BMA implementation, but, in this case, the implementation itself requires a large area.
In this section a trade-off solution for the BMA implementation is proposed which achieve low latency and reduced area with a high speed architecture.
The main idea is to insert a register to break the loop between the discrepancy calculation and the updating of $\Lambda^{(r+1)}(x)$ and $\Omega^{(r+1)}(x)$ polynomials, as shown in the architecture of Fig.3.17. Although this modification doubles elaboration latency, since each step of BMA now requires two clock cycles, it allows two significant improvements. First of all, with respect to solution of Fig.3.16, the critical path is broken and includes only the computation of the discrepancy $\delta^{(r)}$, providing a minimum cycle time given by the delay of one multiplier and one *t*+1-input adder. Moreover the updating of $\Lambda^{(r+1)}(x)$ can share the same hardware with the updating of $\Omega^{(r+1)}(x)$. In

clock cycle	circuit operations
1	from $\Lambda(x)$ and $S(x)$ initial values calculate $\delta^{(0)}$
2	calculate $\Lambda^{(1)}(x)$
3	calculate $\Omega^{(1)}(x)$; $\gamma^{(1)}$; $\delta^{(1)}$
…	…
4t	calculate $\Lambda^{(2t)}(x)$
4t+1	calculate $\Omega^{(2t)}(x)$

Tab.3.6 – proposed BMA implementation circuit operations.

fact, in the architecture of Fig.3.17 the circuits employs the same block, with $2t+1$ GF(2^m) multipliers ($M\gamma_0$-$M\gamma_t$ and $M\delta_1$-$M\delta_t$) and t adders, to update $\Lambda^{(r+1)}(x)$ and $\Omega^{(r+1)}(x)$.

The operation of the architecture of Fig.3.17 is the following. In the first clock cycle, the first discrepancy $\delta^{(0)}$ is evaluated and stored. In subsequently even clock cycles, multipliers are driven with $\Lambda^{(r)}(x)$ and $B^{(r)}(x)$ coefficients, and $\Lambda^{(r+1)}(x)$ is updated according to (3.41). In odd clock cycles, while new discrepancy $\delta^{(r+1)}$ is evaluated based on previously updated $\Lambda^{(r+1)}(x)$ coefficients, $\Omega^{(r+1)}(x)$ is updated according to (3.42). In these clock cycles multipliers are driven with $\Omega^{(r)}(x)$ and $\Theta^{(r)}(x)$ coefficients. Tab.3.6 summarizes circuit operation, showing that the total latency for the join computation of $\Lambda(x)$ and $\Omega(x)$ is $4t+1$ clock cycles.

It is worth to note that hardware complexity of the architecture of Fig.3.17 can be further reduced with an efficient use of Mastrovito multipliers developed in section 3.1. In fact all the multipliers $M\gamma_i$ ($i=0,...,t$) have the same operand γ. Considering γ as the A operand of Mastrovito multipliers, the matrix M will be the same for all multipliers. Hence all multipliers $M\gamma_i$ can share the same first block of Mastrovito scheme. The same technique can be also used for multipliers $M\delta_i$ ($i=1,...,t$) which share the same operand δ.

The comparison between proposed BMA implementation, and

BMA technique	Multipliers	Latency	critical path
BMA Blahut [118,121]	3t+2	3t	$2\cdot T_{mult} + T_{add2} + T_{add(t+1)}$
BMA Berlekamp [119,124]	5t+1	2t	$2\cdot T_{mult} + T_{add2} + T_{add(t+1)}$
BMA decomposed [122]	3	$1+2t+2t^2$	$T_{mult} + T_{add2}$
BMA Sarwate [124]	6t+2	2t	$T_{mult} + T_{add2}$
BMA proposed	3t+2	4t	$T_{mult} + T_{add(t+1)}$

Tab.3.7 – BMA implementations comparison.

	transistors count	f_{max} (MHz)	bit-rate (Mbps)	latency	tech.
proposed BM based	84015	200	1500	288	0.25 μm
Euclidean based [120]	122630	41	309	287	0.25 μm
inv-free Euclidean based [120]	220841	75	562	321	0.25 μm

Tab.3.8 – Reed-Solomon decoder performances comparison.

previously proposed solutions [118],[119],[121],[122],[123] is shown in Tab.3.7. Comparing proposed technique with standard Blahut [118] and Berlekamp [119],[123] techniques we note that our solution about halves critical path delay. With respect to Blahut technique, we need t more latency clock cycles, which, can be compensated, with a little increase in circuit complexity, by inserting t pipeline registers in FIFO memory of Fig.1. On the other hand, comparing with Berlekamp technique, proposed solution results moreover in a substantially lower number of multipliers. The comparison with the solution of Sarwate *et al*, shows that our technique provides a slightly higher critical path delay, while requiring about the half of GF(2^m) multipliers. Finally, decomposed technique, results in the minimum number of multiplier, with, unfortunately a quadratic increase of latency with respect to t, which could heavily increase the size of FIFO memory, especially for codes able to correct an high number of symbols.

3.2.4 Circuit implementation

A (255,239) Reed-Solomon decoder has been designed for a 0.25μm, 1P5M, 2.5V, CMOS technology. The circuit implements novel developed BMA architecture employing high-speed GF(2^m) multiplier scheme shown in section 3.1. The total latency is 288 clock cycles. This latency is compensated by a FIFO memory block which being designed using technique proposed in [124], uses two 128x8 SRAM memories and 17 pipeline registers. The circuit can operate with a maximum clock frequency of 200MHz for an input throughput of 1.6Gbps. Due to the presence of parity symbols the output throughput results 1.5Gbps. The implementation of the proposed circuit requires 84000 transistors with an area occupation of 0.38mm^2.

Circuit performances are summarized and compared with recently proposed techniques in Tab.3.8. As you can see proposed circuit substantially improves both transistor count and maximum operating frequency by providing about the same latency.

Chapter 4
VLSI Design

4.1 Sense Amplifier Flip-Flop

High performance flip-flops are key elements in the design of contemporary high-speed integrated circuits. In these circuits high clock frequencies are generally gained by using a fine grain pipeline in which only few logic levels are inserted between pipeline stages. Because of the high number of pipeline stages, the power dissipation of the clock tree and the flip-flops is a substantial portion of the total power budget. Moreover, the amount of clock cycle time taken by the flip-flops (given by the sum of the clock-to-output and the setup times [130]) is today comparable with the propagation delay of the few logic levels between the pipeline stages. Finally, the ability to absorb clock skew and clock jitter is becoming more and more relevant [131],[132],[135]. Therefore the design of high performance flip-flops, with reduced power dissipation, reduced clock-to-output time, near-zero or negative setup-time and clock skew absorption property (soft clock edge) is a major concern in modern high performance applications.

Recently several high-speed flip-flops structures have been proposed. The topology developed in [133] by Partovi *et al.* uses a latch which is made transparent during a brief sampling window following clock rising edge. This structure (named Hybrid Latch Flip-flop - HLFF) is able to provide clock-skew absorption (soft clock edge). However, it suffers from sizing problems since a too large transparency window increases the hold-time and results in possible race problems, whereas a too small transparency window could not allow the latch to switch.

Improved pulsed latch implementations are proposed in [134],[135]. The Itanium 2 pulsed latch, proposed by Naffziger *et al.* in [135], consists of a transparent passgate latch clocked with a local pulse generator that provides a relatively wide transparency window. The pulse generator can be shared among many passgate latches to reduce the power dissipation. It is also shown in [135] that the Naffziger pulsed latch is faster than the HLFF. Like the HLFF, also the Naffziger pulsed latch suffers from conflicting requirements for the width of the clock pulse produced by the local pulse generator.

In [136] Klass *et al.* reduce the sizing problems of the pulsed latches by employing a conditional shut off of the transparency window. The developed flip-flop (named Semidynamic flip-flop – SDFF), exhibits a shorter hold-time with respect to the pulsed latches and a reduced sensitivity to the sampling window duration.

The sense amplifier based flip-flop (SAFF), initially proposed in [137]-[138], is composed by a fast differential sense amplifier stage, followed by a slave latch. The sense amplifier stage can be seen as a latch whose sampling window closes as soon as the stage switches. This guarantees that the circuit is able to switch independently on circuit sizing. In addition, the SAFF is characterized by a near-zero setup-time, a reduced hold-time, a low clock load and true single phase operation. These characteristics make the sense amplifier based flip-flops good candidates to substitute conventional transmission gate flip-flops in standard cells design approaches.

The main drawback of the SAFF proposed in [137]-[138] is the slave element, composed by a SR NAND latch. While this circuit requires a minimum transistor number, it results in asymmetrical delays with a slow high-to-low clock-to-output delay.

The SAFF proposed by Nikolic *et al.* in [139] yields improved performances by using a symmetric slave latch composed by two inverters and two complex CMOS gates. The performance gain is paid with an increased number of transistors in the output stage, composed by 16 MOS devices.

In [140] Kim *et al.* propose a SAFF circuit that uses a slave latch realized with two N-C^{2}MOS circuits and two cross-coupled weak inverter pairs, needed to make the flip-flop static. The Kim SAFF is very fast, with the output falling transition having a single gate delay with respect to the active clock edge, and requires 14 MOS in the output latch. The SAFF proposed in [140] still has some disadvantages. The first one is the glitching on the output nodes, which is more pronounced for lightly load condition. The second disadvantage is due to the use of cross-coupled inverter latches, that require an appropriate device sizing for a correct operation and suffer from crow-bar current that increases the power dissipation.

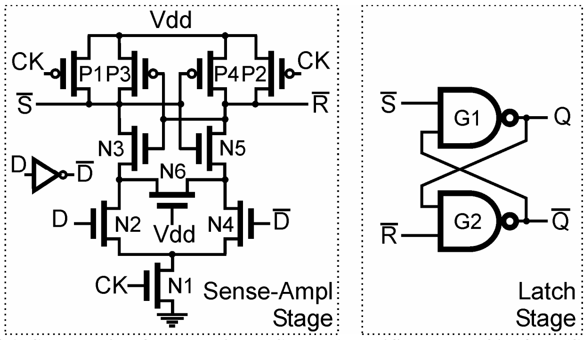


Fig 4.1. Schematic of conventional Sense-Amplifier based flip-flop (SAFF) with NAND SR slave latch.

In this section we propose a new sense amplifier based flip-flop in which the slave latch overcomes the limitations of Kim design while keeping its advantages. The new slave latch requires 12 MOS and can be considered as a hybrid solution between the NAND-based SR latch [137]-[138] and the N-C^{2}MOS approach [140].

4.1.1.a Sense Amplifier Based Flip-flops

Fig.4.1 shows a schematic diagram of the conventional NAND-based SAFF [138]. The circuit is composed by a sense amplifier master stage followed by a NAND-based set-reset slave latch. The circuit operation is the following. When the clock signal *CK* is low, both $\overline{S}$ (set) and $\overline{R}$ (reset) nodes are precharged to *Vdd* and the transistors N3 and N5 are ON. In this phase the latch stage holds the flip-flop state. At the rising edge of *CK*, the sense amplifier senses the differential inputs (D, $\overline{D}$), and one of the precharged nodes ($\overline{S}$ or $\overline{R}$) is pulled down to 0, thorough either N3 or N5, while the other precharged node remains at *Vdd*. The output latch stores the new data acquired by sense amplifier. Note that as soon as the sense amplifier switches, one of N3 or N5 switches off and, therefore, any subsequent transition of (D, $\overline{D}$) inputs is not able to modify the value of the set and reset nodes. The NMOS N6, driven by *Vdd*, provides fully static operation [138] by guarantying a pull-down path for either $\overline{S}$ or $\overline{R}$ if D changes while the clock is at the high level.

In the Flip-flop of Fig.4.1, the differential inputs are sensed in a short transparency window which opens at clock rising edge and closes as soon as a new data sample is acquired by the sense amplifier stage when one of N3 or N5 switches off. Therefore we have a self-timed transparency window closing mechanism, that assures short hold-time

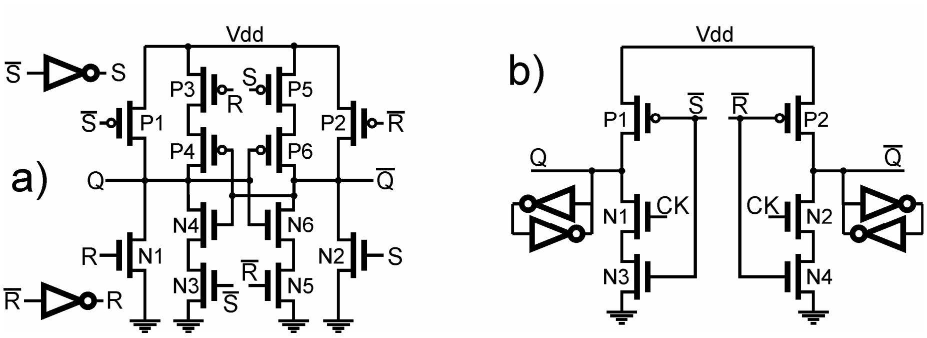


Fig 4.2. High-speed SAFF slave latches. a) Nikolic slave latch; b) Kim slave latch.

and intrinsic insensitivity to process and temperature variations, making the flip-flop suitable for standard cells design approaches.

A drawback of the conventional NAND-based SAFF is the high clock-to-output delay due to the slow output latch stage. Let us assume that, in Fig.4.1, *D* is high while the current *Q* value is zero. At the clock rising edge we have a first delay needed to drive $\overline{S}$ low through N1,N2,N3. Once $\overline{S}$ is gone low, we have a second gate delay due to the switching of G1, which drives *Q* high; and a third gate delay to switch $\overline{Q}$ low through G2. Please note that the delay of $\overline{Q}$ depends not only on the capacitive load on $\overline{Q}$, but also on the capacitive load on the other output *Q*. Hence, the delays of *Q* and $\overline{Q}$ are not independent. In general, for the conventional NAND-based SAFF of Fig.4.1, we have a two gate delay for the low-to-high output transition, and a three gate delay for the high-to-low output transition.

Two high-speed slave latches have been proposed in the literature to make the speed of the SAFF comparable or higher than the speed of the HLFF and the SDFF.

The first approach, shown in Fig.4.2a, has been proposed by Nikolic *et al.* in [139]. The circuit employs two inverters to evaluate the signals *S* and *R*. The four signals *S*, *R*, $\overline{S}$ and $\overline{R}$ are used to drive four devices N1, N2, P1 and P2 which are devoted to switch *Q* and $\overline{Q}$ output nodes. The remaining eight devices N3-N6, P3-P6 are minimum sized, and hold the latch state for *CK*=0, providing a fully static operation with a ratioless sizing.

Let us examine with more detail the operation and the performances of the Nikolic slave latch. If $\overline{S}$ and $\overline{R}$ are high, the latch is in the hold state. In fact, S and R are both low, and the devices N1, N2, P1 and P2

are OFF. The devices N3, N5, P3, P5 are ON, and, consequently, N4, N6, P4 and P6 hold the latch state. Let us now assume that the flip-flop master stage switches, driving $\overline{S}$ low. Device P1 is turned on, and node *Q* is quickly driven high. Note that this transition is ratioless and without crow-bar current since N1 is OFF (*R*=0) and $\overline{S}$ also shuts off device N3 which opens the remaining pull-down path for *Q* node. Moreover signal *S* goes high, N2 turns on driving $\overline{Q}$ low. This second transition is also ratioless and without crow-bar current owing to P5 device. Hence, in addition to the sense amplifier delay, we have one gate delay for the low-to-high output transition, and two gates delay for the high-to-low output transition.
The circuit in Fig.4.2a has the same number of delay stages as NAND-based latch. However, it is worth to note that in the Nikolic circuit all the critical pull-down and pull-up networks are composed by a single device, providing significantly higher speed, especially in the case of high output capacitive loads. Moreover, the delays independence between *Q* and $\overline{Q}$ is obtained. Unfortunately, the worst-case three stages delay still limit performances in the case of medium or low output capacitive loads.
The output latch proposed by Kim *et al.* [140] is shown in Fig.4.2b. The circuit includes two N-C^{2}MOS half-latches driven by $\overline{S}$ and $\overline{R}$ and a couple of two cross-coupled inverters used to achieve a fully static operation. The circuit operation is the following. For *CK*=0, P1, P2, N1, N2 are OFF and *Q* and $\overline{Q}$ hold their state because of the cross-coupled inverters. For *CK*=1 the slave stages are transparent, and the outputs *Q* and $\overline{Q}$ become equal to *S* and *R*, respectively. In order to investigate the speed performances of the Kim latch, let us assume, without loss of generality, that *D* is high at the rising edge of *CK*. In this case $\overline{S}$ is pulled down while $\overline{R}$ remains high. Hence, P1 turns on driving *Q* high. The other flip-flop output, $\overline{Q}$, is quickly pulled down through N2 and N4. Note that the clock-to-output delay for a low-to-high output transition includes both the sense amplifier and the output stage delays. On the other hand, the high-to-low output transition is only one gate delay, because the output latch immediately catches the precharged value at the rising edge of *CK*. As a consequence we have a two stage delay for the low-to-high output transition and only a single stage delay for the high-to-low output

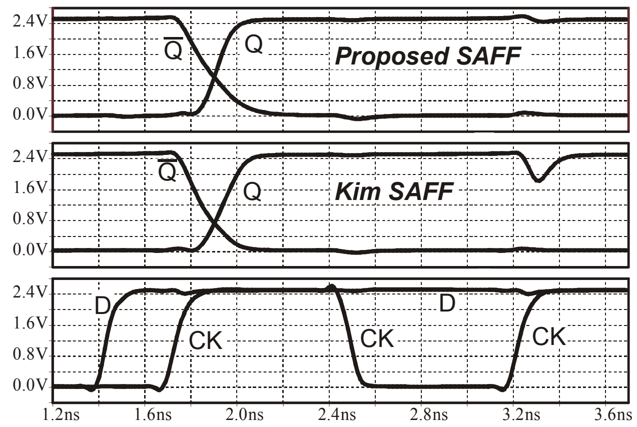


Fig 4.3. Transient SPICE simulation of Kim SAFF and Proposed SAFF.

transition. This characteristic makes the Kim SAFF faster than the Nikolic circuit [139].

Unfortunately the high-speed single stage delay of NC^{2}MOS circuit of Fig.4.2b gives also an unwanted glitching on the output nodes. To explain this phenomenon, let us suppose, for example, that both *Q* and *D* are high when a clock rising edge occurs. In this condition, output *Q* should remain high. Immediately after *CK* rising edge, however, the $\overline{S}$ node is high, and transistor N3 is ON. This device will remain ON until the sense amplifier fully discharges $\overline{S}$. In this time interval, therefore, the pull-down path through N1 and N3 is active, and tries to pull down the value of *Q*. Hence, a glitch appears on the *Q* output, whose amplitude depends on both the *Q* output capacitive load and the device sizing. The glitch is unwanted in many applications and results in additional power dissipation.

SPICE simulation of Fig.4.3 shows the glitching at the output of Kim SAFF. The simulation has been performed for a 2.5V, 0.25μm technology, assuming an output capacitance of 30fF (equivalent to a fanout of 11 symmetrical and minimum-area CMOS inverters). As you can see, the *Q* output of the Kim circuit exhibits a glitch of about 700mV generated by the second clock rising edge. It should be noted that other single stage delay flip-flops, such as the SDFF [136], exhibit the same glitching behavior.

A second disadvantage of N-C^{2}MOS output stage of Fig.4.2b is due to the use of cross-coupled inverter latches on the output nodes. These inverters have to be ratio-sized with N-C^{2}MOS structures to guarantee a correct circuit operation. Moreover, because of the cross-coupled

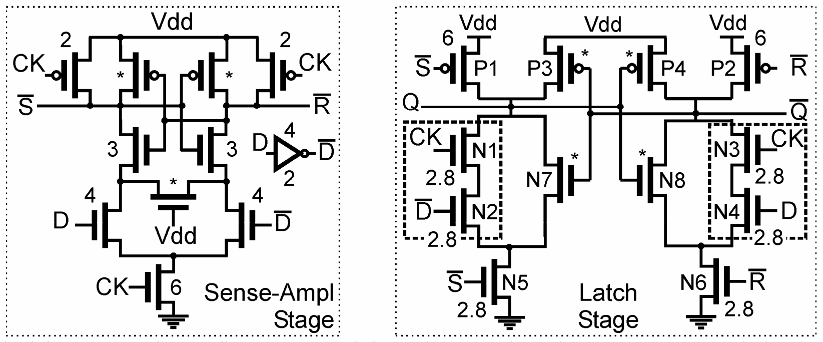


Fig 4.4. Proposed SAFF. Slave latch exhibits ratioless design with reduced power dissipation. The reported numbers are transistors widths in μm for a 0.25μm technology. The minimum sized devices, indicated with an asterisk (*), have a width of 0.58μm.

inverter latches, both the low-to-high and the high-to-low output transitions exhibit crow-bar current that increases the power dissipation.

4.1.2 Proposed SAFF

4.1.2.a Circuit operation

A schematic diagram of the proposed SAFF is shown in Fig.4.4. The sense amplifier is the same as in Fig.4.1. The output stage, instead, can be considered as a hybrid solution between the conventional NAND-based SR latch and the N-C^{2}MOS circuit of Fig.4.2b. If we neglect the transistors N1-N4 (enclosed in the two dashed boxes in Fig.4.4) the new output stage reduces to the NAND-based SR latch.

The transistors N1-N4 allow to speed-up the high-to-low output transition, similarly to what happens in the N-C^{2}MOS SAFF of Fig.4.2b. To describe the circuit operation, let us assume that *D* is high at the rising edge of *CK*. The sense amplifier drives $\overline{S}$ to zero, while $\overline{R}$ remains high. In this way N5 turns off and P1 turns on, driving *Q* high. Note that the shut-off of N5 assures at the same time a ratioless design, without crow-bar current, and the independence of the transition delay from the capacitive load on the other $\overline{Q}$ output. The output $\overline{Q}$ is quickly pulled down through N3, N4 and N6. Hence the high-to-low output transition requires only one stage delay, because the output latch immediately catches the precharged value at the rising edge of *CK*. Note that, after pull-up of *Q*, N8 turns on, keeping $\overline{Q}$ at zero even if input *D* changes after clock rising edge. Devices

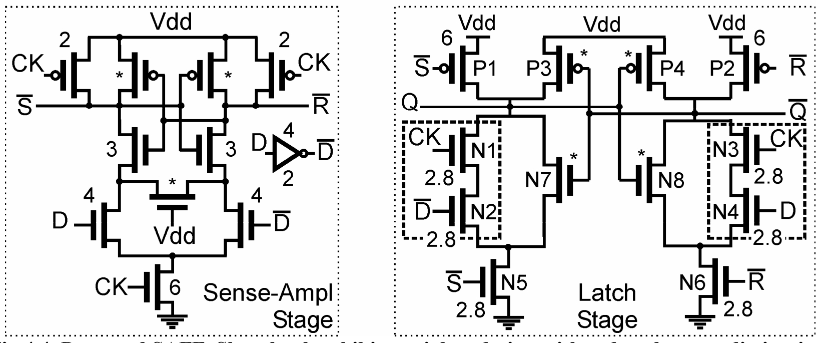


Fig 4.4. Proposed SAFF. Slave latch exhibits ratioless design with reduced power dissipation. The reported numbers are transistors widths in μm for a 0.25μm technology. The minimum sized devices, indicated with an asterisk (*), have a width of 0.58μm.

inverter latches, both the low-to-high and the high-to-low output transitions exhibit crow-bar current that increases the power dissipation.

4.1.2 Proposed SAFF

4.1.2.a Circuit operation

A schematic diagram of the proposed SAFF is shown in Fig.4.4. The sense amplifier is the same as in Fig.4.1. The output stage, instead, can be considered as a hybrid solution between the conventional NAND-based SR latch and the N-C^{2}MOS circuit of Fig.4.2b. If we neglect the transistors N1-N4 (enclosed in the two dashed boxes in Fig.4.4) the new output stage reduces to the NAND-based SR latch.
The transistors N1-N4 allow to speed-up the high-to-low output transition, similarly to what happens in the N-C^{2}MOS SAFF of Fig.4.2b. To describe the circuit operation, let us assume that *D* is high at the rising edge of *CK*. The sense amplifier drives $\overline{S}$ to zero, while $\overline{R}$ remains high. In this way N5 turns off and P1 turns on, driving *Q* high. Note that the shut-off of N5 assures at the same time a ratioless design, without crow-bar current, and the independence of the transition delay from the capacitive load on the other $\overline{Q}$ output. The output $\overline{Q}$ is quickly pulled down through N3, N4 and N6. Hence the high-to-low output transition requires only one stage delay, because the output latch immediately catches the precharged value at the rising edge of *CK*. Note that, after pull-up of *Q*, N8 turns on, keeping $\overline{Q}$ at zero even if input *D* changes after clock rising edge. Devices

SAFF	clock-to-output delay L->H	clock-to-output delay H->L	ratioless	Q and $\overline{Q}$ delays independence	glitch-free output
Conventional [137-138]	2 stages	3 stages	yes	no	yes
Nikolic [139]	2 stages	3 stages	yes	yes	yes
Kim [140]	2 stages	1 stage	no	yes	no
Proposed	2 stages	1 stage	yes	yes	yes

Tab 4.1. Summary of the characteristic of different sense amplifier flip-flops.

P3,P4,N5,N6,N7,N8, hold the previous Q and $\overline{Q}$ values during the sense amplifier precharge, making the proposed flip-flop fully static.
It is worth highlight that the inclusion of transistors N2 and N4 is able to avoid the glitch problem shown before for the N-C^{2}MOS output stage of Fig.4.2b. Let us suppose that a clock rising edge occurs with both Q and D high. Immediately after the CK rising edge the $\overline{S}$ node is still high, and transistor N5 is ON. However, the pull-down through the speed-up network N1-N2 does not take place, since N2 is OFF. As a consequence, the output Q is stable at Vdd, without glitch. The absence of glitching at the flip-flop outputs gives a safe operation and reduces the power dissipation. The SPICE simulation of Fig.4.3 confirms that the proposed circuit is completely glitch-free.
The characteristic of the proposed SAFF are summarized and compared with the previously proposed SAFF circuits in Tab.4.1. As can be seen, the proposed circuit is able to keep the fast operation of the Kim SAFF while avoiding its drawbacks. The best-case high-to-low transition is slower in the proposed SAFF with respect to the N-C^{2}MOS SAFF. In fact, the proposed circuit uses three series NMOS to pull-down Q and $\overline{Q}$, whereas the N-C^{2}MOS SAFF shows only two series NMOS in the output latch. However, considering the worst-case low-to-high output transition, the proposed SAFF is slightly faster than the N-C^{2}MOS SAFF, owing to absence of crow-bar current.

4.1.2.b Circuit sizing

The circuit shown in Fig.4.4 has been implemented in a 0.25µm, 2.5V technology. The transistor sizing has been obtained by optimizing the power-delay product, as proposed in [142], for a load capacitance of 40 fF. The device widths are reported in Fig.4.4.
In the sense-amplifier stage, the width of the two clock driven PMOS are chosen in order to allow the complete precharge of $\overline{S}$ and $\overline{R}$ nodes during the low clock phase. The other two PMOS devices, needed to guarantee a fully static operation, are minimum sized. The sizing of

the devices in the pull-down network of the sense-amplifier stage is critical to reduce the low-to-high output delay. Therefore, these devices are substantially larger than the PMOS of the pull-up network. The two most critical devices of the latch stage are P1 and P2, that determine the low-to-high output delay. The devices N1-N6 can be made substantially smaller than P1 and P2, since they determine the high-to-low output delay which is less critical in the proposed SAFF. Finally, devices N7, N8, P3, P4 act as keeper for the Q and $\overline{Q}$ state during the low clock phase. As show in Fig.4.4, these devices are minimum sized to optimize power-delay product. It is worthwhile to note that using weak keeper transistors in the output stage reduces the noise immunity, since the coupling noise on the output could change the state of the flip-flop. This potential failure mode is common to all topologies where an outputs node is directly connected to a keeper. This happens in the HLFF, the SDFF and the SAFF topologies. These clocking elements, therefore, should be employed when the output fanout is limited and the cross coupling noise is carefully considered during the design flow. Most general purpose standard cell libraries forbid flip-flops that can be "back-driven" from the noise on the output nodes. The proposed flip-flop can be made safer by inserting an output isolation inverter on Q and $\overline{Q}$ at the expense of another stage of delay. In this case, the different number of delay stages between the low-to-high and the high-to-low transitions can be compensated by employing slightly asymmetrical output inverters.

4.1.2.c Optimized circuit for single output flip-flop

In many applications, using a single output flip-flop suffices, and having both Q and $\overline{Q}$ outputs is redundant.

The Fig.4.5 shows the proposed SAFF circuit optimized for the single output configuration.

In this case we can substantially reduce the power dissipation by eliminating the speed-up network for $\overline{Q}$, and reducing the sizing of P2 and N6. This results in a reduction of the capacitive load on $\overline{R}$, which, in turn allows reducing the sizing of the devices driving $\overline{R}$ in the sense-amplifier stage. The PMOS P5 in the output stage is introduced to reduce the crow-bar current during high-to-low transition of Q, improving both delay and power dissipation. Please note that the clocked PMOS P5 in the output stage could also be included in the

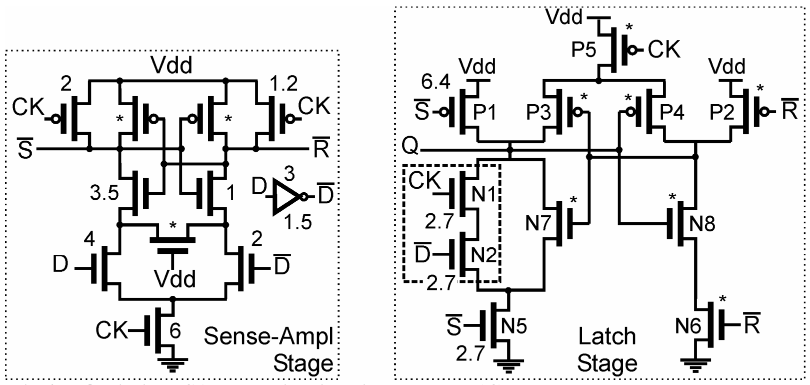


Fig 4.5. Optimized SAFF considering single output (*Q*) case. The reported numbers are transistors widths in μm for a 0.25μm technology.
The minimum sized devices, indicated with an asterisk (*), have a width of 0.58μm.

dual output circuit of Fig.4.4. However, in the dual output circuit, the device P2 is not minimum sized and quickly pulls-up node $\overline{Q}$, shutting down P3. Hence, in the circuit of Fig.4.4, the introduction of PMOS P5 would provide a minimal improvement, that would be more than compensated by the increased clock load.

4.1.2.d Proposed SAFF with asynchronous clear and preset

The circuit schematic of Fig.4.6 shows how asynchronous clear (*CLR*) and preset (*PST*) signals can be added to the proposed SAFF circuit of Fig.4.4. The same approach can be employed to introduce asynchronous clear and preset to the single output SAFF of Fig.4.5.

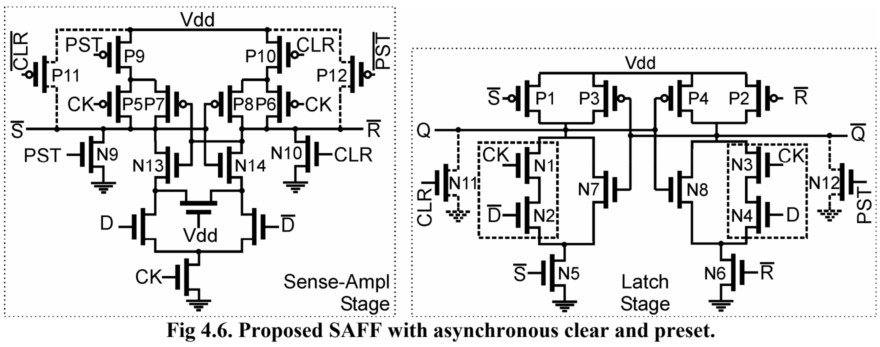


Fig 4.6. Proposed SAFF with asynchronous clear and preset.
The dashed transistors are optional devices which reduce clear and preset delay times.

The dashed devices (N11,N12,P11,P12) are optional devices which can be added to the circuit in order to speed-up the clear-to-output and the preset-to-output propagation delays. The insertion of the speed-up devices P11 and P12 requires the addition of two inverters, not shown in Fig.4.6, needed to evaluate $\overline{CLR}$ and $\overline{PST}$ signals.

If both *CLR* and *PST* are low, then N9-N12,P11,P12 devices are OFF, whereas P9 and P10 are ON and the circuit reduces to the flip-flop of Fig.4.4. Note that the addition of clear and preset devices does not increase the number of series devices on timing critical pull-up and pull-down networks. Let us neglect, for the time being, the speed-up (dashed) transistors. For *CLR*=1 (*PST*=0), in the sense amplifier stage, $\overline{R}$ is pulled-down by N10. This transition is ratioless, without crow-bar current, since *CLR*=1 also turns off P10; moreover the turn off of P10 assures that the *CK* signal is unable to change the $\overline{R}$ logic level through P6. After $\overline{R}$ goes to 0, the MOS P2 in output latch turns on, driving $\overline{Q}$ high. Note that, the shut-off of the N6 device (driven by $\overline{R}$=0) guarantees at the same time a ratioless transition, without crow-bar current, and the independence of $\overline{Q}$ output level from *CK* and *D* logic values. In the sense amplifier P7 and P9 are ON, pulling-up $\overline{S}$. Once $\overline{S}$ is gone high, *Q* is pulled down through N5 and N7. The time needed to clear the flip-flop state is hence two stages delay for $\overline{Q}$ transition and three stages delay for *Q* transition. A similar operation is obtained for the preset condition (*PST*=1, *CLR*=0).

Please note that the slowest high-to-low propagation delay depends on two minimum sized devices: P7 (which pulls-up $\overline{S}$) and N7 (which pulls-down *Q*). Therefore, without speed-up devices (dashed devices of Fig.4.6), it takes a long time to clear or preset the flip-flop. The speed-up devices can be inserted in the circuit to reduce the clear and preset times. The device N11 in the latch stage speeds-up the high-to-low transitions of *Q*. In fact, as soon as *CLR* goes high, N11 starts pulling down *Q*. Since, in this phase, $\overline{S}$ could be still high, P1 could be ON, slowing *Q* transition due to the presence of crow-bar current. The device P11 in the sense amplifier stage avoids this problem by quickly pulling-up $\overline{S}$ turning off P1. The devices N12 and P12 play a similar role for the preset operation.

4.1.3 Comparison with High Speed Flip-flops

The new SAFF flip-flops, both with and without output isolation inverters, have been designed for a 2.5V, 1P5M, 0.25μm technology (FO4 delay in considered technology is 85ps). To evaluate the effectiveness of the proposed SAFF we have also designed, for the same technology, the following circuits: conventional SAFF [137],[138], transmission gate flip-flop in PowerPC topology [141], SDFF [136], Naffziger pulsed latch [135], Nikolic SAFF [139], Kim SAFF [140]. In order to perform a fair comparison, we eliminated the scan logic from both the transmission gate PowerPC and the Naffziger circuits. The pulsed latch with inverter isolated D input was selected between the two pulsed latch topologies proposed in [135]. Similarly, we added an inverter to the transmission gate PowerPC flip-flop, to eliminate the passgate input on *D* signal. All investigated SAFF topologies include the inverter needed to evaluate $\overline{D}$.

A comparison between circuit performances is not easy, due to different flip-flop fundamental characteristic, like the immunity to the cross coupling noise and the glitch free operation. To make a fair comparison, we decided first of all to divide the flip-flop circuits in two main categories. The circuits belonging to the "general purpose" category are better suited for general purpose standard cell applications. Three attributes identify a flip-flop circuit belonging to this category: the immunity to the output coupling noise, the glitching free output operation and the capability to switch the flip-flop independently from circuit sizing, process, voltage and temperature (PVT) variations. The "high performance" category includes the flip-flop circuits which, lacking of at least one of the above attributes, should be employed only when ultimate performances are needed. The increased performances are generally paid with a more careful design flow, which may require full system transistor level simulations to guarantee the correct circuit operation.

The flip-flop structures considered in our comparison are shown in Tab.4.2. It can be observed that the general purpose category includes the PowerPC transmission gate flip-flop and the proposed SAFF circuits (both with one and two outputs) with output isolation inverters, needed to increase the output noise immunity. The high performances category includes remaining flip-flops and proposed SAFF without output inverters.

	output noise immunity	glitch free operation	robustness against sizing and PVT var.	comp. outputs	number of transistors	sum of widths (μm)	normalized area	
Tgate (PowerPC) [141]	yes	yes	yes	no	22	51.7	1.00	gen. purpose
Prop. SAFF 1 out with inv	yes	yes	yes	no	25	58.0	1.12	
Prop. SAFF 2 out with inv	yes	yes	yes	yes	28	82.7	1.60	
Pul. latch (Naffziger) [135]	yes	yes	no	no	21	38.6	0.75	high performance
SDFF [136]	no	no	no	no	23	32.7	0.63	
Prop. SAFF 1 out	no	yes	yes	no	23	44.5	0.86	
Conv. SAFF [137-138]	no	yes	yes	yes	20	54.9	1.06	
Nikolic SAFF [139]	no	yes	yes	yes	28	67.8	1.31	
Kim SAFF [140]	no	no	no	yes	26	58.4	1.13	
Prop. SAFF 2 out	no	yes	yes	yes	24	62.9	1.22	

Tab 4.2. Flip-flops characteristics and area occupation parameters. The areas are normalized to the transmission gate flip-flop.
The grayed background distinguishes complementary outputs flip-flops.

The transistor sizing for all considered flip-flops has been obtained by optimizing the power-delay product, as proposed in [142], for a load capacitance of 40fF. The transistor sizing summary is reported in Tab.4.2. The reported values reveal that, among the general purpose flip-flops, the lower silicon area is obtained with the transmission gate flip-flop. The proposed single output SAFF is only a 12% larger than the transmission gate flip-flop. Considering the high performance circuits with complementary outputs, the circuit proposed in this paper requires less silicon area than the Nikolic SAFF, while is larger than the Kim SAFF due to the presence of two more NMOS on the critical path, that require an adequate sizing. The SDFF topology requires the minimum area among the single output topologies. The proposed high performance single output SAFF is slightly smaller than the transmission gate flip-flop, but requires a significant larger area when compared with the SDFF and the Naffziger pulsed latch.
The timing performances are shown in Tab.4.3, where the setup-time values have been obtained by employing the optimal setup-time definition proposed in [142]. Since the sum of the setup-time and the clock-to-output delay is the real amount of clock cycle time taken by the flip-flop, the speed-up shown in Tab.4.3, is defined as the inverse of the sum between the worst case clock-to-output delay plus the setup-time, normalized to the transmission gate flip-flop.

The Tab.4.3 shows that the conventional SAFF and the transmission gate flip-flops are the slowest circuits. The timing performances of the conventional SAFF are limited by the ineffective NAND-based output latch, whereas the transmission gate flip-flop pays for the large setup time due to the propagation delay of the master latch. The analysis of general purpose flip-flops reveals that the proposed SAFF circuits with output isolation inverters are between 50% and 60% faster than the PowerPC transmission gate flip-flop. This speed advantage can be mainly attributed to the reduced setup time.

Considering the high performance circuits, it can be noted that the SDFF is slower than the Kim and the proposed SAFF solutions. This is due to both the sizing constrains, needed to guarantee the correct transparency window duration, and the crow-bar current during transitions. The Naffziger pulsed latch is, in turn, slightly slower than the SDFF. The Nikolic SAFF has three stages on the critical path, like the conventional SAFF. However, the final stages of Nikolic SAFF are very fast, without stacked transistors; this gives a speed advantage, especially when driving heavy output loads. For low output loads, the Kim SAFF reveals faster than the Nikolic flip-flop, owing to the reduced number of stages. The proposed SAFFs are the fastest circuits, having only a two stages delays without crow-bar current during the output transition and a reduced setup time. Proposed solutions appear also interesting because they join the high speed performances with the absence of output glitching and the robustness against sizing and PVT variations. It is worthwhile to note that both the SDFF and the Kim SAFF are affected by output glitching. Moreover, among the high performance flip-flops, the Naffziger

	worst case clock-to-output delay				setup time (ps)	Speed-up			
	num. stages	time (ps) @ 20fF	time (ps) @ 40fF	time (ps) @ 80fF		@20fF	@40fF	@80fF	
Tgate (PowerPC) [141]	2	106	125	169	142	1.00	1.00	1.00	gen. purpose
Prop. SAFF 1 out with inv	3	145	156	189	10	1.60	1.61	1.56	
Prop. SAFF 2 out with inv	3	142	156	199	10	1.63	1.61	1.49	
Pul. latch (Naffziger) [135]	3	224	243	281	-66	1.57	1.51	1.45	high performance
SDFF [136]	2	163	183	222	-20	1.73	1.64	1.54	
Prop. SAFF 1 out	2	114	132	168	10	2.00	1.88	1.75	
Conv. SAFF [137-138]	3	193	246	349	10	1.22	1.04	0.87	
Nikolic SAFF [139]	3	144	155	174	10	1.61	1.62	1.69	
Kim SAFF [140]	2	123	142	179	10	1.86	1.76	1.65	
Prop. SAFF 2 out	2	111	132	170	10	2.05	1.88	1.73	

Tab 4.3. Flip-flops timing performances. The grayed background distinguishes complementary outputs flip-flops.
The speed-up refers to the inverse of normalized sum of the setup-time plus the worst case clock-to-output time.

	best clk-to-out delay num. stages	best clk-to-out delay time (ps) @ 20fF	hold time (ps)	internal race immunity (ps)	
Tgate (PowerPC) [141]	2	102	-88	190	gen. purpose
Prop. SAFF 1 out with inv	2	137	35	102	
Prop. SAFF 2 out with inv	2	133	35	98	
Pul. latch (Naffziger) [135]	3	210	190	20	high performance
SDFF [136]	1	78	85	-7	
Prop. SAFF 1 out	1	74	35	39	
Conv. SAFF [137-138]	2	115	35	80	
Nikolic SAFF [139]	2	116	35	81	
Kim SAFF [140]	1	70	35	35	
Prop. SAFF 2 out	1	74	35	39	

Tab 4.4. Race immunity performances. The grayed background distinguishes complementary outputs flip-flops.

pulsed latch is the only immune to the output coupling noise.
The race immunity parameters are reported in Tab.4.4. In this table the internal race immunity [143] is given by the difference between the best-case clock-to-output delay and the hold-time. This quantity represents the maximum clock skew that can be tolerated by a shift-register structure. By examining the Tab.4.4, it can be noted that the highest race immunity is obtained with the general purpose flip-flops. The transmission gate flip-flop benefits from the negative hold time, while in the proposed SAFF circuits the propagation delay introduced by the output inverters leads to a race immunity of about 100 ps. Among high performance flip-flops, the SDFF exhibits a fast best-case clock-to-output transition and a large hold-time. This results in a slightly negative internal race immunity, which could constraint the insertion of buffers to prevent race conditions. The Naffziger pulsed latch structure appears also to be critical from the point of view of hold time violations, with an internal race immunity as low as 20 ps. The Nikolic SAFF shows a good trade-off between the setup and the hold times, with a good internal race immunity.
The average energy dissipation per clock cycle (Ed) and Energy-delay product (EDP) are reported in Tab.4.5. In this table different flip-flop input switching activities (α) have been considered for an output load of 40fF. The switching activity is defined here as the ratio between the average D input frequency and the clock frequency.

Considering the general purpose flip-flops, it is interesting to note that the power dissipation of the proposed single output SAFF is comparable with the power dissipation of the PowerPC transmission gate flip-flop. The increased speed of the proposed circuit leads to an EDP about 35% lower than the transmission gate flip-flop. Considering the high performance circuits, the lowest power dissipation is obtained with the proposed single output SAFF. The Naffziger pulsed latch and the conventional SAFF also reveal attractive for low power applications. It is worthwhile to note that, in the Naffziger pulsed latch, the pulse generator can be shared among many passgate latches to further reduce the power dissipation.
The power dissipation of the proposed two outputs SAFF is 15% lower than the power dissipation of the Kim SAFF, for α=0. Since the input is constant for α=0, this improvement is due to the glitch-free operation of proposed circuit. The power saving reduces for larger α values and is about 7% for α=0.5. In this latter case the input *D* changes every clock period, and no glitching is exhibited in the Kim SAFF. In this case the power reduction can hence be attributed to the ratioless output stage of the proposed SAFF. Among the complementary outputs circuits, the proposed SAFF provides the lowest EDP.
The results reported in Tab.4.3 and Tab.4.5 for the proposed SAFF flip-flops, show that the single output circuits are able to achieve the same speed of the double outputs circuits with a substantially lower power dissipation.

	Ed (pJ) @ 40fF			EDP (pJ·ns)	Norm. EDP	
	α=0	α=0.25	α=0.5	α=0.25 @ 40fF	α=0.25 @ 40fF	
Tgate (PowerPC) [141]	0.42	0.74	1.07	0.199	1.00	gen. purpose
Prop. SAFF 1 out with inv	0.48	0.76	0.99	0.126	0.64	
Prop. SAFF 2 out with inv	0.63	0.95	1.28	0.158	0.80	
Pul. latch (Naffziger) [135]	0.54	0.74	0.92	0.130	0.66	high performance
SDFF [136]	0.82	0.95	1.07	0.155	0.78	
Prop. SAFF 1 out	0.48	0.68	0.82	0.097	0.49	
Conv. SAFF [137-138]	0.56	0.74	0.91	0.190	0.96	
Nikolic SAFF [139]	0.75	0.93	1.11	0.153	0.77	
Kim SAFF [140]	0.73	0.92	1.12	0.140	0.70	
Prop. SAFF 2 out	0.62	0.83	1.04	0.118	0.59	

Tab 4.5. Energy dissipation performances. The grayed background distinguishes complementary outputs flip-flops.

Speed-up devices configuration	worst delay clock-to-output (ps) @ 40fF	best delay clock-to-output (ps) @ 20fF	setup time (ps)	hold time (ps)	Ed (pJ) @ 40fF α=0.25	clear-to-output delay (ps) @ 40fF H->L	L->H
No speed-up device	134	74	10	35	0.84	835	150
N11 and N12 only	137	79	10	35	0.85	265	150
All speed-up devices	142	79	10	35	0.87	160	150

Tab 4.6. Performances of proposed SAFF with asynchronous clear and preset considering different speed-up devices configurations.

The Tab.4.6 reports the performances of the proposed SAFF with asynchronous clear and preset, shown in Fig.4.6. In this table both the clock-to-output and the clear-to-output delays have been reported for different configuration of the speed-up devices. Let us start by examining the clear-to-output delays. Without speed-up devices the high-to-low delay is very high (835ps). The addition of N11 and N12 speed-up devices is able to substantially decrease this delay, up to 265ps. Finally, the inclusion of P11, P12 and the inverters for the computation of $\overline{CLR}$ and $\overline{PST}$ reduces the crow-bar current for the high-to-low output switching, making the high-to-low delay comparable with the low-to-high delay. Comparing the data in Tab.6 with the performances of the proposed SAFF without clear and preset devices (see Tab.4.3-4.5) it is worth to note that the inclusion of clear and preset devices does not influence significantly the flip-flop performances.

4.1.4 Experimental Verification

The proposed SAFF flip-flops have been used in the design of a high-speed Direct Digital Frequency Synthesizer (DDFS) and of a Signature Analyzer (SA), used for the built-in-self test of the DDFS. The chip micrograph is shown in Fig.4.7. The system has been realized in 0.25µm, 1P5M, 2.5V technology by using a standard cell

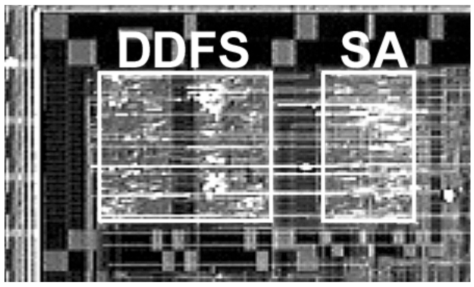


Fig 4.7. Direct-Digital-Frequency Synthesizer (DDFS) and Signature-Analyzer (SA) designed by using new SAFF.

approach with automatic placement and routing of synthesized netlist. In order to achieve better performances, both the single output and the dual output SAFF without output isolation inverters have been employed. The standard cell library does not include any other flip-flop. During circuit synthesis freedom is given to the synthesizer to choose between the two flip-flop cells.

Before to proceed with the fabrication, many DDFS versions (with different internal wordlengths and clock frequencies) were simulated at the transistor level. Owing to the low flip-flop fanout and to the reduced circuit area (about 0.1 mm^2), all versions showed correct operation, despite of the potential output coupling noise problem previously discussed.

Experimentally, fabricated prototypes resulted in a correct operation. This let us believe that the proposed safer flip-flops with output isolation inverters, should have no problems to substitute transmission gate flip-flops in general purpose standard cell libraries.

It is worthwhile to note that the use of proposed SAFF has been a key element in the design of the DDFS, allowing reaching a measured 600 MHz maximum clock frequency with only six pipeline levels in the circuit.

The Signature Analyzer is able to reach an higher clock frequency with respect to the DDFS. In particular, the Signature Analyzer includes a frequency divider which employs a 14 bit LFSR and a pipelined comparator to divide the input clock frequency by 16383. Experimentally, the circuit exhibits correct behavior up to 1.2 GHz clock frequency, highlighting the effectiveness of the proposed flip-flops in high speed standard cells based applications.

4.1.5 Conclusions

The section introduces a new sense amplifier based flip-flop. The slave latch of the new flip-flop is able to keep the advantages of the state of the art N-C^2MOS approach [140] while avoiding its disadvantages.

A fast asynchronous clear and preset functionality can be achieved without compromising the flip-flop performances.

The proposed flip-flop gives a very good power-delay product with glitch-free operation, and is useful in high performance applications. As an example, the use of the new flip-flop in a 0.25μm technology allowed to reach a 600 MHz operation in the DDFS circuit proposed in [144]. A 14 bit LFSR designed with a standard-cell approach in the same technology experimentally shows correct operation up to 1.2GHz.

4.2 Truncated Multipliers

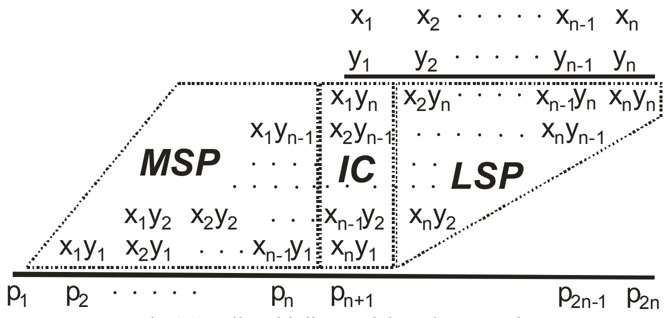
Multiplication is the main operation in many signal processing algorithms (filtering, convolution, Euclidean distance, FFT, ...). As a consequence low complexity parallel multipliers are desirable both in general purpose DSP processors and application specific architectures for digital signal processing.
Let X and Y be two n bit unsigned fractional numbers:

$$X = \sum_{i=1}^{n} x_i \cdot 2^{-i} \qquad Y = \sum_{i=1}^{n} y_i \cdot 2^{-i} \tag{4.1}$$

A multiplier calculates the product $P=X\cdot Y$ as follows:

$$P = \sum_{k=1}^{2n} p_k \cdot 2^{-k} = \sum_{j=1}^{n}\sum_{i=1}^{n} x_i y_j \cdot 2^{-i-j} \tag{4.2}$$

The multiplier partial products matrix is shown in Fig.4.8.
Many applications require a multiplier output that is also fractional with n bit precision [145]-[147]. A “fixed-width” multiplier can be easily realized by using only $p_1 ... p_n$ outputs of a full multiplier. In order to reduce the truncation error, output rounding is often carried-out. Rounding is obtained [148] by adding $2^{-(n+1)}$ to the full-width multiplier output before truncation:



x_1 x_2 · · · · · x_{n-1} x_n
y_1 y_2 · · · · · y_{n-1} y_n
MSP
IC
LSP
x_1y_n x_2y_n $x_{n-1}y_n$ x_ny_n
x_1y_{n-1} x_2y_{n-1} x_ny_{n-1}
x_1y_2 x_2y_2 $x_{n-1}y_2$ x_ny_2
x_1y_1 x_2y_1 $x_{n-1}y_1$ x_ny_1
p_1 p_2 · · · · · p_n p_{n+1} p_{2n-1} p_{2n}

Fig.4.8. Full multiplier partial products matrix

$$P_{round} = \left\lfloor 2^{-(n+1)} + \sum_{k=1}^{2n} p_k \cdot 2^{-k} \right\rfloor_n = Round_n\left(\sum_{k=1}^{2n} p_k \cdot 2^{-k}\right) \qquad (4.3)$$

where we indicated with the symbol $\lfloor\ \rfloor_n$ the truncation to n most-significant bits and with $Round_n()$ the n-bit rounding operator.

Many techniques have been proposed which exploit "fixed-width" property to reduce hardware complexity with respect to rounded full-width multiplier [146],[149]-[152]. In order to simplify the review and the comparison of these techniques, let us divide the partial product matrix in the three subsets MSP, IC and LSP shown in Fig.4.8.

In [149], Kidambi *et. al* simplify the multiplier partial product matrix by deleting both IC and LSP parts. A pre-computed constant is added to the final multiplier output in order to compensate for the introduced error. The fixed-width multiplication is hence approximated as follows:

$$P_{Kidambi} = \sum_{j=1}^{n-1} y_j 2^{-j} \sum_{i=1}^{n-j} x_i 2^{-i} + K \qquad (4.4)$$

This technique provides a hardware complexity about halved with respect to a full multiplier. However, the introduced error is high, reducing practical applications.

The approximation error of fixed bias correction (4.4) is investigated in [150] by Lim. It is shown that the error rapidly increases with multiplier size n. The error can be reduced by deleting the less partial products (for instance, retaining the partial products belonging to the IC subset) before adding the fixed bias K. Obviously, this results in a trade-off between precision and hardware complexity.

An improved fixed-witdh multiplication algorithm, named partial product "conditional correction", is also proposed in [150]. This algorithm, basically, exploits a correlation between the sums of partial products in columns belonging to the IC or LSP subsets. Neither hardware implementation nor performance analysis of the algorithm is given in [150].

The conditional correction algorithm is further developed in [151] by Jou *et. al*, where a multiplier architecture is proposed in which only the LSP subset of partial products is discarded. The partial products in the IC subset are summed to compute an intermediate quantity S_{IC}:

$$S_{IC} = x_1 \cdot y_n + x_2 \cdot y_{n-1} + ... + x_n \cdot y_1 \qquad (4.5)$$

The sum S_{IC} is then used to calculate a correction factor that estimates the sum of dropped partial products. The correction factor is then added to the MSP subset of the partial products matrix, to obtain multiplier output:

$$P_{Jou} = \sum_{j=1}^{n-1} y_j 2^{-j} \sum_{i=1}^{n-j} x_i 2^{-i} + h(S_{IC}) \qquad (4.6)$$

The function $h(S_{IC})$ is implemented in [151] with reduced hardware complexity. It is worthwhile to note, however, that the technique [151] still suffers from large errors and, moreover, it uses ripple a architecture to calculate the correction function. This results in low speed and increased glitching, giving large power dissipation.

In [152] Van *et. al* propose a more accurate fixed-width multiplier architecture. Also in this case the LSP subset of partial products is neglected and the result is computed by adding a correction factor to the MSP part of the partial products matrix. The correction factor, however, is computed as a function of the single partial products of IC subset (and not as a function of their sum S_{IC}):

$$P_t = \sum_{j=1}^{n-1} y_j 2^{-j} \sum_{i=1}^{n-j} x_i 2^{-i} + f(x_1 y_n, x_2 y_{n-1}, \ldots\ x_{n-1} y_2, x_n y_1) \qquad (4.7)$$

Please note that this approach was developed in [152] only for signed multipliers. Moreover, the error compensation circuit that implements the function *f()* still has a ripple architecture with poor delay and power performances.

In the following we will name "fixed-width multipliers with multiple-input error compensation" the architecture based on equation (4.7). The partial products of IC subset will be named as "input correction vector" $I=(x_1 y_n, \ldots, y_1 x_n)$, while the function *f(I)* will be named as "error compensation function".

Curticapean *et al.* [146] use the same multiple-input error compensation architecture proposed in [152]. An improved error compensation function is discovered in [146], giving better error performances with respect to previously proposed architectures. It is worthwhile to note that in the paper [146] no motivation is given about improved error performances. Moreover, the error compensation network remains based on a slow and power hungry ripple architecture.

In this section a new approach to design high-performance unsigned fixed-width multipliers is proposed. The multiplier architecture is still based on (4.7), like [146],[152]. A new error compensation function is developed, that can be optimized in order to minimize either the maximum absolute error or the mean square error. The proposed error compensation function gives better accuracy with respect to previously published approaches. Our error compensation function, moreover, requires few gates and is easily implemented with a tree architecture, ideally suited for implementation with fast tree-based multipliers [148]. As a consequence, proposed approach improves speed, power and accuracy with respect to previously proposed fixed-width unsigned multipliers. Results for circuit implementation in 0.35μm technology and a comprehensive comparison with previously proposed techniques are also reported in the paper. The paper, in addition, investigates more in general the error performances achievable using multiple-input error compensation architecture (4.7), giving lower bounds for both maximum absolute error and mean square error.

4.2.1 Fixed-Width Multipliers Errors

4.2.1.a Error metric

The accuracy of a fixed-width multiplier can be evaluated considering the introduced error ε with respect to the output of the $2 \cdot n$ bit complete multiplier.

$$\varepsilon = P - P_t \tag{4.8}$$

Where $P = X \cdot Y$ is the output of the complete multiplier, given by (4.1), and P_t is the output of the fixed-width multiplier. As error metric we consider either the normalized maximum absolute error (ε_{max}) or the normalized mean square error (ε_{ms}) defined as:

$$\varepsilon_{max} = \max\left(|\varepsilon|\right) / LSB \tag{4.9}$$

$$\varepsilon_{ms} = \mathrm{E}\left\{\varepsilon^2\right\} / LSB^2 \tag{4.10}$$

Where E{} is the average operator, while $LSB = 2^{-n}$. Another parameter useful to characterize fixed-width multiplier accuracy is the normalized mean error (ε_m), given by:

$$\varepsilon_m = \mathrm{E}\left\{\varepsilon\right\} / LSB \tag{4.11}$$

4.2.1.b Errors in full-width Rounded Multipliers

The simplest way to obtain a fixed-width multiplier is through a rounded, full-width multiplier (see (4.3)). Rounding introduces a quantization error, that is well known to provide ε_{max}=1/2 and ε_{ms}=1/12 [150]. These values are a lower bound for the errors achievable with any fixed width multiplier, since full-width multiplier rounding is the most accurate fixed-width technique. On the other hand, since the full set of partial products shown in Fig.4.8 has to be calculated and summed, full-width rounded multiplier provides the same circuit complexity of a standard multiplier and no gain is obtained by using an *n*-bit output width.

4.2.1.c Error bounds for fixed-width multipliers with multiple-input error compensation

Let us consider a fixed-width multiplier designed according to equation (4.7). In this case the approximation error, form equations (4.2),(4.7), can be written as:

$$\varepsilon = P - P_t = S(x_1,...,x_n;y_1,...,y_n) - f(I) \qquad (4.12)$$

where S($x_1,...,x_n;y_1,...,y_n$)=S($X;Y$) is the sum of the partial products of the IC and LSP subsets:

$$\mathrm{S} = \sum_{j=1}^{n} y_j \cdot 2^{-j} \sum_{i=n-j+1}^{n} x_i \cdot 2^{-i} \qquad (4.13)$$

From (4.12) we note that the introduced error depends on the choice of the error compensation function *f(I)*. In previous papers different error compensation functions have been proposed to easily implement fixed-width multipliers, but no analysis has been carried out to investigate the lower errors bound achievable by using equation (4.7). To find error lower bounds, let us indicate as *fmax* and *fms* the two error compensation functions which minimize error metrics ε_{max} and ε_{ms} respectively. To obtain *fmax* and *fms* let us note that every value I_0 of the input correction vector *I* can be obtained with different values of *X* and *Y*. As an example, for *n*=4, I=(x_1y_4, x_2y_3, x_3y_2, x_4y_1) and the value I_0=(0,1,1,1) can be obtained with three different input values: (X;Y)=(0,1,1,1;1,1,1,0), (X;Y)=(1,1,1,1;1,1,1,0), (X;Y)=(0,1,1,1;1,1,1,1). We will indicate in the following as Ω(I_0) the set of *X* and *Y* values for which I=I_0, and as N(I_0) the number of elements in Ω(I_0) set.

Let us consider, for a given input correction vector I_0, all the X and Y values belonging to $\Omega(I_0)$. The maximum, minimum and average values of $\mathrm{S}(X,Y)$ when X and Y assume all possible values in $\Omega(I_0)$ are given by:

$$\mathrm{Smax}(I_0)=\max_{(X,Y)\in\Omega(I_0)}(\mathrm{S}(X,Y)) \qquad (4.14)$$

$$\mathrm{Smin}(I_0)=\min_{(X,Y)\in\Omega(I_0)}(\mathrm{S}(X,Y)) \qquad (4.15)$$

$$\mathrm{Savg}(I_0)=\frac{1}{N(I_0)}\sum_{(X,Y)\in\Omega(I_0)}\mathrm{S}(X,Y) \qquad (4.16)$$

The function *fmax* can be obtained by minimizing the absolute error for each value I_0 of input correction vector. From this consideration it follows that *fmax*(I_0) should be chosen as close as possible to (Smax(I_0)+Smin(I_0))/2. Since the error compensation function should have the same *n* bit precision of the fixed-width multiplier output, the best choice for *fmax*(I_0) is obtained by rounding the previous value to *n* bit:

$$fmax(I_0)=Round_n\left[\frac{\mathrm{Smax}(I_0)+\mathrm{Smin}(I_0)}{2}\right] \qquad (4.17)$$

A lower bound for the maximum absolute error is given by:

$$\varepsilon_{max}\geq\max_{I_0}\left[\frac{\mathrm{Smax}(I_0)-\mathrm{Smin}(I_0)}{2}\right] \qquad (4.18)$$

If the error correction function is designed according to (4.17), then ε_{max} can be larger than the lower bound (4.18) due to quantization error, for a maximum amount of *LSB*/2.
To obtain *fms* function, let us consider the mean square error when X and Y inputs belong to $\Omega(I_0)$:

$$\varepsilon_{ms}(I_0)=\frac{1}{\mathrm{N}(I_0)}\sum_{(X;Y)\in\Omega(I_0)}(\mathrm{S}(X;Y)-f(I_0))^2 \qquad (4.19)$$

The overall mean square error can easily be obtained as a sum of the $\varepsilon_{ms}(I_0)$ values, weighted by the number of elements in $\Omega(I_0)$:

$$\varepsilon_{ms}=\sum_{I_0}\varepsilon_{ms}(I_0)\cdot\frac{\mathrm{N}(I_0)}{2^{2n}} \qquad (4.20)$$

Equation (4.20) shows that ε_{ms} is the sum of positive quantities ($\varepsilon_{ms}(I_0)$). Hence the minimum value of ε_{ms} is obtained if, for any input correction vector I_0, $f(I_0)$ minimizes $\varepsilon_{ms}(I_0)$.
With simple algebra, equation (4.19) can be rewritten as:

$$\varepsilon_{ms}(I_0) = [\mathrm{Savg}(I_0) - f(I_0)]^2 + \frac{1}{\mathrm{N}(I_0)} \sum_{(X,Y)\in\Omega(I_0)} [\mathrm{S}(X;Y) - \mathrm{Savg}(I_0)]^2 \qquad (4.21)$$

From equation (4.21) it follows that $\varepsilon_{ms}(I_0)$ is minimized if $f(I_0)$=Savg(I_0). Since *f(I)* should be rounded to *n* bit, the function *fms(I)* is calculated as:

$$fms(I_0) = Round_n[\mathrm{Savg}(I_0)] \qquad (4.22)$$

A lower bound for the mean square error is given by:

$$\varepsilon_{ms} \geq 2^{-2n} \cdot \sum_{I_0} \sum_{(X;Y)\in\Omega(I_0)} [\mathrm{S}(X;Y) - \mathrm{Savg}(I_0)]^2 \qquad (4.23)$$

Again, if the error compensation function is designed according to (4.22) then ε_{ms} can be larger than the lower bound (4.23) due to quantization error, for a maximum amount of $(LSB/2)^2$.

4.2.2 Fixed-Width Multipliers with Multiple-Input Error Compensation

4.2.2.a Multiplier architecture

The architecture of proposed fixed width multiplier is shown in Fig.4.9. The PP Generation block evaluates all partial products belonging to MSP and IC subsets of Fig.4.8 using simple AND gates. The partial products of the input correction vector feed the error compensation block that evaluates error compensation function. The output of error compensation block is given by *m* bits: $c_1, \ldots, c_m$ all having the weight LSB=2^{-n}. The Carry-save Addition Tree sums the partial products of the MSP subset and the outputs of the Error compensation block, according to the partial product matrix shown in Fig.4.9b. The Carry-save Addition Tree can be implemented using any one of the well known multi-operand addition techniques [148],[153].

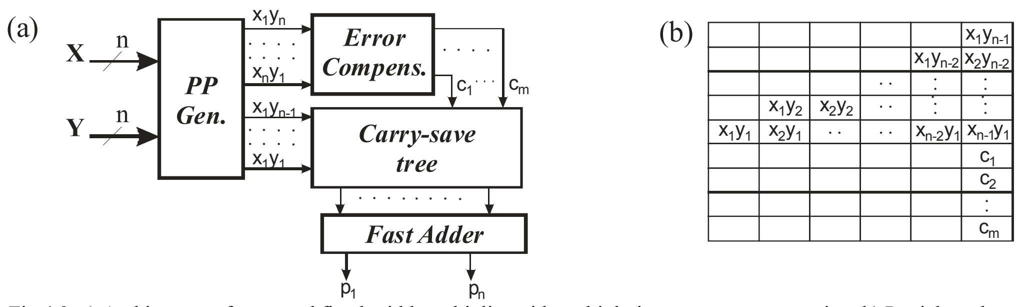


Fig.4.9. a) Architecture of proposed fixed-width multiplier with multiple-input error compensation; b) Partial product matrix of carry-save tree.

4.2.2.b Error compensation function

The accuracy of fixed-width multipliers with multiple-input error correction depends on the choice of error compensation function. On the other hand electrical performances (speed, power, silicon area) depend on implementation of error compensation function.

Designing the error compensation function according to either (4.17) or (4.22), the best possible accuracy is obtained. This solution, however, calls for a lookup table to implement either the *fmax* or *fms* functions. Lookup table complexity grows exponentially with *n*, rapidly becoming an impractical solution. Therefore the effective fixed-width multiplier implementations ([146],[151],[152]) employ error compensation functions which approximate *fmax* or *fms*, with worse error performances, giving a substantially lower complexity which linearly increases with *n*. However the techniques [146],[151],[152] require a ripple architecture for the implementation of the error compensation function, with poor delay and power performances.

In this section we propose the use of an architecture, denoted as dual-tree in the following, that is faster and less power hungry, while keeping the linear complexity of previously proposed approaches. Moreover our architecture gives a better approximation of *fms* and *fmax*.

In order to introduce our approach let us examine with more details the peculiarities of *fms* and *fmax*. With this purpose let us recall that the error compensation function has to approximate the sum of the partial products in the IC and LSP subsets. Since the inputs of the error compensation block of Fig.4.9 are the partial products of the IC

I=(x_1y_6, x_2y_5, x_3y_4, x_4y_3, x_5y_2, x_6y_1)	Savg*(I)*	*fms(I)*
(0, 0, 0, 0, 0, 0)	0.224	0
(0, 0, 0, 0, 0, 1)	0.8136	1
(0, 0, 0, 1, 0, 0)	0.9045	1
(0, 0, 1, 1, 0, 0)	1.6962	2

Table4.7. Input correction vector *I*, ideal correction factor Savg*(I)* and rounded value *fms(I)* for a 6-bit fixed-width multiplier using multiple-input error compensation, optimized to reduce mean-square error. Savg*(I)* and *fms(I)* are normalized to LSB=2^{-n}.

subsets (having a weight of LSB/2) we expect that the weight of each element of the input correction vector on the *fms* and *fmax* functions values is higher than LSB/2. As an example let us consider a 6-bit fixed-width multiplier, with optimized mean square error. Tab.4.7 reports the ideal correction factors Savg*(I)*, given by (4.16), and the rounded value *fms(I)* given by (4.22), both normalized to LSB.
As can be seen, when all bits of input correction vector are zero, a small bias (Savg=0.224 LSB) should be ideally calculated by error compensation block (note that the bias, however, is smaller that LSB/2 and is hence rounded to zero). The bias takes into account the probability that some partial products of the LSP subset could be high, even if the input correction vector is zero. When the input correction vector becomes I=(0,0,0,0,0,1), with only partial product x_6y_1 high, the correction factor increases up to 0.8136 LSB, which is rounded to 1 LSB. The "effective" weight of partial product x_6y_1, defined as the difference between the correction factors corresponding to I=(0,0,0,0,0,1) and I=(0,0,0,0,0,0), is 0.5896 LSB. Note that this "effective" weight is higher than the weight of the partial product x_6y_1 (given by LSB/2) due to the correlation between x_6y_1 and the partial products including either x_6 or y_1 in the LSP subset (see [150]). This phenomenon is even more evident if we consider the third row in Table 4.7, when only the partial product x_4y_3 is high. Here the "effective" weight (0.6805 LSB) is higher than the "effective" weight of x_6y_1 since the partial products, in the LSP subset, correlated with x_4y_3 have a weight higher than the partial products correlated to x_6y_1.
From the above discussion we can conclude that, in the calculation of optimal error compensation function, different weights should be attributed to each partial product. In particular, "inner" partial products (like x_4y_3 and x_3y_4 in our 6-bits example) should have a larger weight with respect to "outer" partial products (like x_6y_1 and x_1y_6). Therefore the use of only the sum S_{IC} to calculate the compensation factor, as proposed in [150],[151], give only a first order

approximation of the best possible error compensation, because equal weights are assumed for all the input correction vector elements.
It is worthwhile to note that *fms* isn't simply a linear function of the input correction vector partial products. In fact, as shown in the forth row of Table 4.7, the *Savg* value corresponding to the case in which x_4y_3 and x_3y_4 are both high, is higher than the sum of the bias plus the "effective" weights of the two partial products.

4.2.2.c Dual tree architecture

The architecture of proposed error Compensation block, shown in Fig.4.10, takes into account the different weights of the input correction vector elements by using two different addition trees. The input correction vector partial products are subdivided in two disjoined sets: a so called "standard addition" set and a "modified addition" set. The elements of the standard addition set, are added by using a standard one-counter [148], implemented with common full adders (FA) and half adders (HA). Since the weight of added partial products is LSB/2, the carries of the tree, having a weight of LSB, are considered as the first c_1, …, $c_{m'}$ outputs of the error compensation block of Fig.2 ($m' < m$).

The $c_{m'+1}$, …, c_{m-2} error compensation elements are obtained as the carries at the output of the second addition tree. This tree receives as inputs the elements of the modified addition set and uses modified half adder (mHA) cells to evaluated error compensation elements. As shown in Fig.4.11, modified half-adders are realized by using one AND gate and one OR gate and hence correspond to the so-called AO gate introduced in [151] and then used also in [146]. By looking to the truth table of Fig.4.11, it can be seen that the modified half adder is similar to the common half adder, except when both A and B inputs are one; in this case the result computed by the modified half adder is 3 (*Sum*=1 and *Cout*=1) whereas a common half adder would provide the standard sum between A and B equal to 2. It can easily be seen that the modified half-adder operator is associative (and also commutative). As a consequence multi-operand addition with modified half adders can be implemented with tree architectures [148],[153] (like standard addition) as opposed as the ripple architecture used in previous papers. From this consideration it simply follows that, if we sum k input bits using a modified half adder tree the result $S_m = 2\cdot(c_{m'+1} + \ldots + c_{m-2}) + S_2$ is given by:

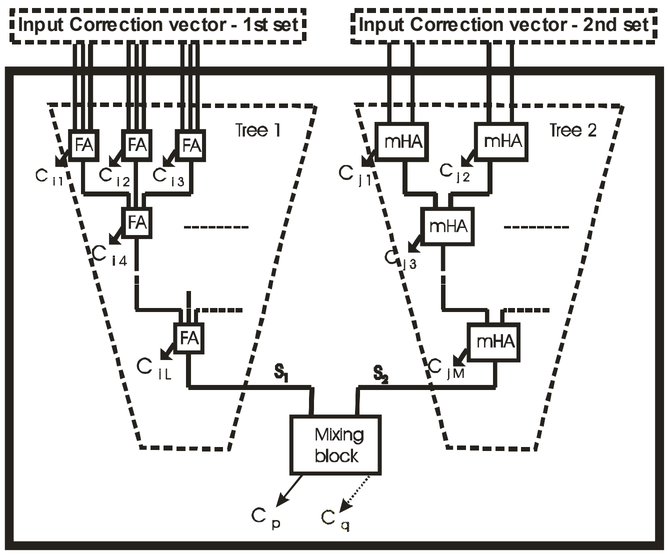
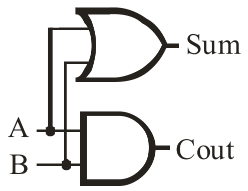


Fig.4.10. Architecture of dual-tree error compensation block. The Tree 1 sums lower weight partial products using standard full adders FA. The Tree 2 uses modified half adders mHA to take into account the contribution of partial products with higher weights. The sums of the two trees are merged in a "mixing block" in order to obtain one or two (see text) additional outputs.



A	B	Cout	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Fig.4.11. Modified half-adder.

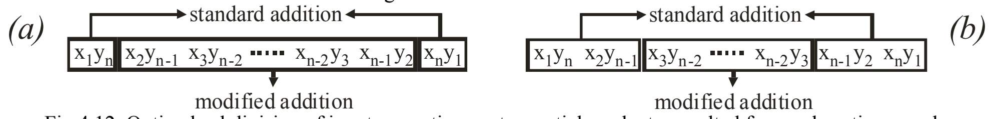

(a)
standard addition
x_1y_n x_2y_{n-1} x_3y_{n-2} ..... $x_{n-2}y_3$ $x_{n-1}y_2$ x_ny_1
modified addition
(b)
standard addition
x_1y_n x_2y_{n-1} x_3y_{n-2} ..... $x_{n-2}y_3$ $x_{n-1}y_2$ x_ny_1
modified addition

Fig.4.12. Optimal subdivision of input correction vector partial products, resulted from exhaustive search.
a) "Type 1" architecture that optimizes maximum absolute error. This subdivision is also optimal for the mean-square error when n≤5.
The optimal mixing block is an OR gate.
b) "Type 2" architecture that optimizes mean-square error for n≥6. The optimal mixing block is a modified half-adder.

$$S_m = \begin{cases} 2 \cdot U - 1, & \text{if } U > 0 \\ 0, & \text{if } U = 0 \end{cases} \qquad (4.24)$$

where U is the number of input bits equal to 1. Hence, we have a "modified one counter" that sums the input bits, giving them weight 2. The S_1 and S_2 sum outputs of the two trees, having a weight of LSB/2, cannot be directly considered as error compensation elements. Therefore S_1 and S_2 are elaborated by a "mixing block" (see Fig.4.10) which further correct multiplier output by using c_{m-1} and c_m elements. We consider three possible "mixing block" addition schemes. First solution adds S_1 and S_2 by using standard addition and truncates the result. In this case S_1 and S_2 are mixed with an AND gate providing a single additional output at the error compensation block: c_{m-1}=S_1 AND S_2 (c_m=0 in Fig.4.10). In second solution S_1 and S_2 are still added by using standard addition, but the result is rounded. It can be easily shown that this results in mixing S_1 and S_2 by using an OR gate (c_{m-1}=S_1 OR S_2, c_m=0). The third solution adds S_1 and S_2 with a modified half-adder and rounds the result. In this case the mixing block evaluates both c_{m-1}=S_1 OR S_2 and c_m=S_1 AND S_2.

4.2.2 c.1 Dual tree architecture optimization and error performances

Exhaustive search has been used to obtain, for a given multiplier size n, the optimal subdivision of input correction vector partial products (between standard and modified summation trees) and the optimal mixing block configuration. We realized two optimizations. In the first one we assumed as a goal function the absolute error (ε_{max}), whereas the second optimization was carried-out to minimize the mean square error (ε_{ms}). Since the search space grows exponentially with n, a maximum value of n=12 has been considered.

The optimizations resulted in the two solutions highlighted in Fig.4.12. In "type 1" solution, the standard addition set contains only the two partial products $x_1 y_n$ and $x_n y_1$, and the mixing block is an OR gate. In "type 2" solution the standard addition set is $\{x_1 y_n, x_2 y_{n-1}, x_{n-1} y_2, x_n y_1\}$, and a modified half adder is used as mixing block. Type 1 resulted the best architecture to optimize ε_{max} for all the values of n investigated; in addition this is also the architecture that achieves the best mean square error for $n<6$. For $6 \leq n \leq 12$, and assuming ε_{ms} as error metric, type 2 is the best architecture. As expected, in both type 1 and type 2 solutions, "outer" partial products are assigned to the standard addition tree, whereas "inner" partial products, which should have a higher weight on output correction, are assigned to the modified addition tree.

The error performances of the multipliers proposed in this paper are compared in Table 4.8a and Table 4.8b with the full-width rounded multiplier, the recently proposed fixed-width multipliers of [146],[151] and the ROM lookup table approach (coefficients of *ROM_max* have been found according to (4.17) to minimize absolute error, whereas *ROM_ms* is calculated from (4.22) and minimizes the mean square error). In Table 4.8a the comparison is based on the normalized absolute error metric, while Table 4.8b compares the normalized mean-square errors.

(a)

n	architecture	ε_{max}	$\Delta\varepsilon_{max}$	ε_m
4	rounded	0.500	-38%	-0.062
4	ROM max	0.813	+0%	-0.078
4	Curticapean [146]	0.875	+8%	-0.090
4	Jou [151]	1.313	+62%	0.449
4	this paper (type 1)	**0.813**	**+0%**	**-0.008**
8	rounded	0.500	-62%	-0.008
8	ROM max	1.316	+0%	-0.193
8	Curticapean [146]	1.555	+18%	-0.032
8	Jou [151]	2.012	+53%	0.651
8	this paper (type 1)	**1.512**	**+15%**	**0.122**
12	rounded	0.500	-72%	-0.001
12	ROM max	1.796	+0%	-0.216
12	Curticapean [146]	2.222	+24%	-0.010
12	Jou [151]	2.680	+49%	0.718
12	this paper (type 1)	**2.180**	**+21%**	**0.166**
16	rounded	0.500	N.A.	0.000
16	ROM max	N.A.	N.A.	N.A.
16	Curticapean [146]	2.889	N.A.	-0.003
16	Jou [151]	3.347	N.A.	0.740
16	this paper (type 1)	**2.847**	**N.A.**	**0.181**

(b)

n	architecture	ε_{ms}	$\Delta\varepsilon_{ms}$	ε_m
4	rounded	0.083	-27%	-0.062
4	ROM ms	0.113	+0%	-0.008
4	Curticapean [146]	0.127	+13%	-0.090
4	Jou [151]	0.313	+176%	0.449
4	This paper (type 1)	**0.113**	**+0%**	**-0.008**
8	rounded	0.083	-55%	-0.008
8	ROM ms	0.184	+0%	-0.013
8	Curticapean [146]	0.234	+27%	-0.032
8	Jou [151]	0.598	+226%	0.651
8	This paper (type 2)	**0.216**	**+18%**	**0.017**
12	rounded	0.083	-66%	-0.001
12	ROM ms	0.245	+0%	-0.034
12	Curticapean [146]	0.333	+36%	-0.010
12	Jou [151]	0.780	+219%	0.718
12	This paper (type 2)	**0.300**	**+23%**	**0.016**
16	rounded	0.083	N.A.	0.000
16	ROM ms	N.A.	N.A.	N.A.
16	Curticapean [146]	0.425	N.A.	-0.003
16	Jou [151]	0.905	N.A.	0.740
16	This paper (type 2)	**0.384**	**N.A.**	**0.016**

Table 4.8. Error performances of fixed-width multipliers: a) assumes maximum absolute error ε_{max} as error metric; b) assumes mean-square error ε_{ms} as error metric.
$\Delta\varepsilon_{max}$ is the percentage increase of ε_{max} with respect to *ROM max*, $\Delta\varepsilon_{ms}$ is the percentage increase of ε_{ms} with respect to *ROM ms*. ε_m is the mean error.
Note that ε_{ms} is normalized to LSB^2, while ε_{max} and ε_m are normalized to LSB.

From Table 4.8 we note that the *rounded* multipliers, using the complete partial products matrix of Fig. 4.8, provide the best achievable error performances, with constant maximum and mean square errors. In ROM-based approach, on the other hand, both ε_{ms} and ε_{max} increase with n. As a consequence, multiple-input error compensation appears to be a useful approach only for reduced n values.

Results of Table 4.8 show that error performances of proposed circuits are near to the *ROM* lower bounds. For n=12, only a 21% absolute error increase and a 23% mean square error increase is found. Comparing proposed approaches with previously proposed ones, we note a substantial decrease of the error with respect to Jou solution [151]. On the other hand, the improvement with respect to Curticapean solution [146] is smaller, and is more noticeable in terms of ε_{ms}.

In the last rows of Table 4.8 we compare 16-bit fixed-width multipliers. Extending the results found from optimization, we used "type 1" architecture when maximum absolute error was considered as error metric, and "type 2" architecture to optimize ε_{ms}. You can see that also for n=16 proposed multipliers provide reduced ε_{ms} and ε_{max} values with respect to Jou and Curticapean solutions.

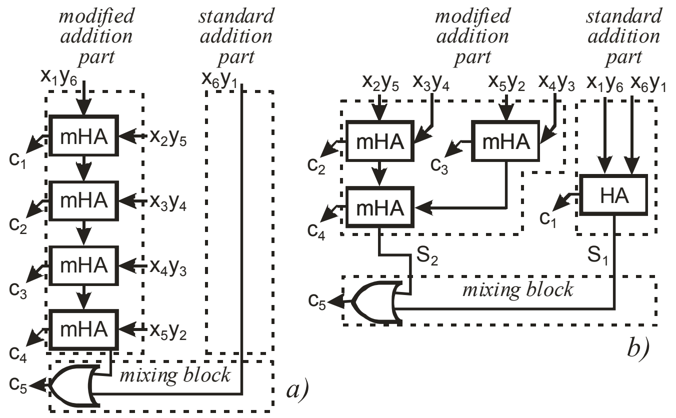


Fig.4.13. Error compensation circuits in 6-bit fixed-width multipliers.
a) Error compensation circuit proposed in [2];
b) Dual-tree "type 1" architecture, that minimizes maximum absolute error.

4.2.3 Dual Tree Architecture Implementation

Proposed dual-tree approach can be seen as an extension of the error compensation solutions proposed in [146] and [151]. As an example, Fig.4.13a shows that the error compensation circuit proposed in [146], for n=6. This circuit is equal to a dual-tree circuit in which ripple architecture is used to sum n-1 partial products with modified addition. Only one partial product is included in standard addition set, using an OR gate as "mixing block". Similar considerations can be done for the error compensation circuit provided in [151].
Our "type 1" dual-tree compensation block is shown in Fig.4.13b. Comparing Fig.4.13b with Fig.4.13a, it can be observed that dual-tree approach, while not requiring additional hardware, is significantly faster with only three OR gates on the critical path. Similar considerations apply to "type 2" dual-tree compensation block, displayed in Fig.4.13c.
The actual implementation of our dual-tree architecture, however, can be further simplified. From (4.24) you can see that the one could obtain the result of the modified tree by using a standard one-counter (based on full and half adders), multiplying the result by two and then subtracting one. Hence (neglecting for the time being both one subtraction and mixing block), we can eliminate the modified tree altogether, by sending the partial products originally assigned to the modified tree directly to the carry-save adder, with a weight *LSB*. For "type 1" architecture, it can be demonstrated that final subtraction and mixing block correspond to the inclusion of a NOR and an AND gate,

as shown in Fig.4.14a. For "type 2" architecture, Fig.4.14b highlights that the effect of final subtraction and mixing block simply correspond to send the sum bit of the standard tree directly to the output of the error compensation block.

By comparing Fig.4.14 with Fig.4.10 it can be seen that optimized implementation of dual-tree architectures results in a substantial hardware saving. This is clearly shown in Fig.4.14, where optimized "type 1" and "type 2" architectures are reported, for n=6. Optimized "type 1" dual-tree architecture (Fig.4.14a) requires an half-adder, a four-input NOR gate and a two-input AND gate. Optimized "type 2" dual-tree architecture (Fig.4.14b) is even simpler, and is actually composed by just a single half-adder.

4.2.3.a Circuits Performances

We implemented rounded full-width multipliers, Jou [151], Curticapean [146], and the optimized dual-tree fixed width multipliers proposed in this paper using a three metal 0.35μm technology with 3.3V supply voltage. Multiplier size for n equal to 4, 8, 12 and 16 has been considered.

In order to have a realistic and accurate indication of architecture performances, we implemented the carry-save tree of all multipliers using the three-dimensional reduction method (TDM) proposed in [153]. TDM is a state of the art technique to add elements of partial products matrix with a tree based carry-save approach, compensating for different delays in partial products generation, and exploiting delays asymmetries in full-adders to improve overall timing.

The obtained results are shown in Table 4.9. As you can see, we have that proposed circuits are faster than complete rounded multiplier for every value of n, whereas Jou and Curticapean ripple error compensation architectures produces multipliers slower than the complete one for n≥12. As an example, proposed circuit with "type 2"

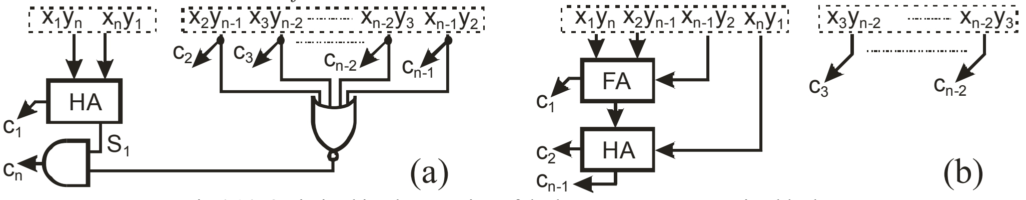

standard addition
modified addition
x_1y_n x_ny_1
x_2y_{n-1} x_3y_{n-2} $x_{n-2}y_3$ $x_{n-1}y_2$
c_2 c_3 c_{n-2} c_{n-1}
HA
c_1
S_1
c_n
(a)
standard addition
modified addition
x_1y_n x_2y_{n-1} $x_{n-1}y_2$ x_ny_1
x_3y_{n-2} $x_{n-2}y_3$
FA
HA
c_1 c_2 c_{n-1}
c_3 c_{n-2}
(b)

Fig.4.14. Optimized implementation of dual-tree error compensation blocks: a) "type 1" architecture; b) "type 2" architecture.

n	Architecture	Delay ns	Area 10^3 μm^2	Power μW/MHz
4	Rounded	4.61	7.95	9.39
4	Jou [151]	4.50	4.20	4.73
4	Curticapean [146]	4.50	4.20	5.01
4	**This paper (type 1)**	**4.15**	**3.84**	**5.06**
8	Rounded	7.06	30.56	47.96
8	Jou [151]	6.99	16.69	26.41
8	Curticapean [146]	6.99	16.69	26.68
8	**This paper (type 2)**	**6.53**	**15.20**	**23.59**
8	**This paper (type 1)**	**6.54**	**15.27**	**23.60**
12	Rounded	8.69	68.60	112.89
12	Jou [151]	9.12	36.67	60.66
12	Curticapean [146]	9.12	36.67	60.88
12	**This paper (type 2)**	**7.95**	**33.94**	**55.89**
12	**This paper (type 1)**	**7.92**	**34.18**	**56.17**
16	Rounded	9.84	120.28	198.64
16	Jou [151]	11.15	63.83	111.47
16	Curticapean [146]	11.15	63.83	111.60
16	**This paper (type 2)**	**9.02**	**59.86**	**99.90**
16	**This paper (type 1)**	**8.88**	**60.22**	**98.98**

Table 4.9. Performances of fixed-width implemented in 0.35 μm CMOS technology.

architecture for *n*=16 provides 9.8% decrease in propagation delay with respect to complete rounded multiplier and more than 20% improvement with respect [146],[151].
Considering area occupation, the developed dual-tree multipliers result slightly more efficient than Jou and Curicapean solutions, with silicon area reduction of about 6% for *n*=16. Obviously, the advantage with respect to complete rounded multiplier is much more evident, with area reduction of about 50%.
Finally, because of reduced glitching in partial products generation, proposed circuits exhibit a lower power dissipation with respect to Jou and Curticapean solutions for *n*>4. For instance, power saving is about 11% for *n* equal to 16. Power dissipation is almost halved with respect to complete rounded multiplier.

4.2.4 Conclusion

The section introduces a new technique to design unsigned fixed-width multipliers. New approach improves accuracy, silicon area, timing performances and power dissipation with respect to previously proposed techniques. Simulation results for a 0.35μm technology show a decrease of the propagation delay up to 20%, with more than 10% power dissipation reduction and a substantial improvement of mean square error.

Appendix A: DDFS

Let us focus our attention on the approximation of the sine function *f*(*x*) (see (2.2)). To that purpose let us rewrite the MTM approximation (2.4) for the symmetric case of Fig.2.3b:

$$f(x) \simeq \tilde{A}(x_0) + B_1(\xi_1)\cdot\left(x_1 - \frac{\delta_1}{2}\right) + \ldots + B_K(\xi_K)\cdot\left(x_K - \frac{\delta_K}{2}\right) \qquad \text{(A.1)}$$

where:

$$\delta_i = \left(2^{q_i} - 1\right)\cdot 2^{-s_i} \qquad \text{(A.2)}$$

$$s_i = \sum_{j=0}^{i} q_j \qquad \text{(A.3)}$$

Following [23], the $\tilde{A}(x_0)$ and $B_i(\xi_i)$ coefficients are computed by minimizing the maximum approximation error in (A.1) as:

$$\tilde{A}(x_0) = \frac{f(x_0) + f(x_0 + \Delta_0)}{2} \qquad \text{(A.4)}$$

$$B_i(\xi_i) = \frac{f(\xi_i + \delta_i) - f(\xi_i) + f(\xi_i + \delta_i + \sigma_i) - f(\xi_i + \sigma_i)}{2\delta_i} \qquad \text{(A.5)}$$

where:

$$\sigma_i = 2^{-p_i} - 2^{q_i - s_i} \qquad \text{(A.6)}$$

$$\Delta_0 = \sum_{j=1}^{K} \delta_j = 2^{-q_0} - 2^{-Q} \qquad \text{(A.7)}$$

The equation (A.4) defines directly the content of the Table of Initial Values ($TIV(x_0) = \tilde{A}(x_0)$). The content of each TO*ᵢ* in Fig.2.3b can be easily computed by starting from (A.5):

$$\text{TO}_i(\xi_i, x_i) = B_i(\xi_i)\cdot\left(x_i + 2^{-s_i-1}\right) \qquad \text{(A.8)}$$

The MTM approximation error depends on the second derivative of *f*(*x*). Consequently, following [23], the worst case error ε_i, due to the *i*-th Table of Offsets, can be evaluated as:

$$\varepsilon_i = \left.\frac{f(\xi_i + \delta_i) - f(\xi_i) - f(\xi_i + \delta_i + \sigma_i) + f(\xi_i + \sigma_i)}{4}\right|_{\xi_i = 1 - 2^{-p_i}} \qquad \text{(A.9)}$$

A upper bound for the total approximation error can be obtained by summing the errors due to the single Tables of Offsets ($\varepsilon_{appr}=\varepsilon_1+...+\varepsilon_K$). In addition to this algorithmic error, we have to consider the rounding error of each table. By using g guard bits the total rounding error ε_{rnd} is:

$$\varepsilon_{rnd} = (K+1)\cdot 2^{-R-g} + 2^{-R}\cdot(1-2^{-g}) \qquad \text{(A.10)}$$

where 2^{-R+1} is the weight of the output LSB, and the rounding technique of Das Sarma and Matula is used for $g>0$ [23].

The total error ε can be estimated as:

$$\varepsilon \simeq \varepsilon_{appr} + \varepsilon_{rnd} \qquad \text{(A.11)}$$

The ROM size for each table in Fig.2.3b can be evaluated as follows:

$$ROMsize\_TIV = (R-1+g)\cdot 2^{q_0} \qquad \text{(A.12)}$$

$$ROMsize\_TO_i = w_i \cdot 2^{p_i+q_i-1} \qquad \text{(A.13)}$$

In (A.13) w_i represents the output wordlength of the i-th Table of Offsets, given by:

$$w_i = R+g-2-\left\lfloor \log_2\left(\frac{1}{B_{i\_max}\cdot\delta_i}\right)\right\rfloor \qquad \text{(A.14)}$$

where B_{i_max} is the maximum slope of the i-th Table of Offsets. Since the slopes $B_i(\xi_i)$ are related to the first derivative of $f(x)$, in our case we have:

$$B_{i\_max} = B_i(\xi_i)\big|_{\xi_i=0} \qquad \text{(A.15)}$$

Appendix B: GF

This section is aimed to demonstrate that, given the delays of $M_{j,i}\cdot b_i$ signals, the algorithm of Fig.3.4 achieves the minimum possible delay. Without loss of generality let us assume that the delay of an XOR gate is equal to one and that $M_{j,i}\cdot b_i$ signals have integer delays.

LEMMA 1. Let $N(t)$ be the maximum number of zero delay inputs for a XOR network with output delay t. We state that $N(t)$ is equal to 2^t.

PROOF: The proof is by mathematical induction. Since a network with delay 0 contains no XOR gate, the initial step of the induction ($N(0)=1$) is straightforward. Let us demonstrate the induction hypothesis $N(t)=2\cdot N(t-1)$. Consider a XOR network with delay t. The final XOR of this network splits the network in two sub-networks each of delay t-1. The maximum number of inputs of such networks is $N(t-1)$, and therefore it remains demonstrated that the total XOR network with delay t has a maximum of $2\cdot N(t-1)$ inputs.

THEOREM 1. Given n inputs $s_1, \ldots, s_n$ with delays $d_1, \ldots, d_n$, for any XOR network which sums all s_i signals it holds true the following inequality:

$$2^t \geq \sum_{i=1}^{n} 2^{d_i} \qquad \text{(B.1)}$$

where t is the delay of the XOR network.

PROOF: Let us assume that it is possible to build a XOR network which sums $s_1, \ldots, s_n$ with a delay t such that:

$$2^t < \sum_{i=1}^{n} 2^{d_i} \qquad \text{(B.2)}$$

Since the input s_i of this network has a delay d_i, we can imagine that s_i is obtained from a XOR network with 2^{d_i} inputs with a zero delay. Let us assume to build a new XOR network by joining the original network and the XOR network which computes s_i. Applying this transformation for all the s_i signals, we obtain a new network with $\sum_{i=1}^{n} 2^{d_i}$ inputs, all having zero delay. By looking to (B.2) we note that we have built a XOR network with a delay equal to t and more than 2^t inputs. Since this is in contrast with the lemma 1, the equation (B.1) remains demonstrated.

Please note that a consequence of (B.1) is that the minimum delay t_{min} to sum n inputs $s_1, \ldots, s_n$ with delays $d_1, \ldots, d_n$ is given by:

$$t_{min} = \left\lceil \log_2 \sum_{i=1}^{n} 2^{d_i} \right\rceil \qquad \text{(B.3)}$$

where $\lceil a \rceil$ represent the lowest integer higher than a.

The algorithm of Fig.3.4, which construct the XOR network for the summation of $M_{j,i} \cdot b_i$ signals, starts by building a set S, composed by $M_{j,i} \cdot b_i$ signals, and proceeds in iterative steps by deleting from S the two elements s_1 and s_2 with the minimum delay, combining s_1 and s_2 with a XOR gate, and inserting the XOR output s_3 in the set S. We will show that this algorithm achieves exactly the minimum delay given by (B.3). To that purpose let us introduce the following theorem:

THEOREM 2. Let δ_i be the delays of the signals in the set S at a given k-th step of the algorithm of Fig.3.4, and assume that:

$$\sum_{i=1}^{n} 2^{\delta_i} \le 2^t \qquad \text{(B.4)}$$

for a given value t.

The delays χ_i of the S signals at the k+1-th step will verify the same inequality:

$$\sum_{i=1}^{n-1} 2^{\chi_i} \le 2^t \qquad \text{(B.5)}$$

PROOF: Let us assume to order δ_i delays in non decreasing order ($\delta_i \le \delta_{i+1}$). We can rewrite the (B.4) in the following way:

$$2^{t-\delta_2} - \sum_{i=3}^{n} 2^{\delta_i - \delta_2} - 1 \ge 2^{\delta_1 - \delta_2} > 0 \qquad \text{(B.6)}$$

Since t and δ_i with $i \ge 3$ are all greater than or equal to δ_2, the first member of (B.6) is integer, and as a consequence we can rewrite (B.6) in the following way:

$$2^{t-\delta_2} - \sum_{i=3}^{n} 2^{\delta_i - \delta_2} - 1 \ge 1 \qquad \text{(B.7)}$$

Rearranging (B.7) we obtain:

$$2 \cdot 2^{\delta_2} + \sum_{i=3}^{n} 2^{\delta_i} \le 2^t \qquad \text{(B.8)}$$

The algorithm of Fig.3.4 deletes form S the signals with delays δ_1 and δ_2 and inserts a new signal with delay δ_2+1. Let χ_1 be the delay of the new added signal, we can therefore write:

$$\sum_{i=1}^{n-1} 2^{\chi_i} = 2^{\delta_2+1} + \sum_{i=3}^{n} 2^{\delta_i} \qquad \text{(B.9)}$$

From (B.9) and (B.8) it follows the (B.5).

From theorems 1 and 2 it follows that the optimal delay for summing the initial signals $M_{j,i} \cdot b_i$ is the same of the optimal delay for summing the signals in the set S at a generic step of the algorithm of Fig.3.4. Since this remains true also for the last step of the algorithm, in which S contains only one element, it is clear that the proposed solution achieves exactly the minimum delay given by (B.3).

References

[1] B.G. Goldberg, “Direct Digital Frequency Synthesis Demystified”, LLH Technology Publishing, 1999.

[2] V. F. Kroupa, “Direct Digital Frequency Synthesizer”, New York: IEEE Press Book, 1998.

[3] J. Vankka, K. Halonen, “Direct Digital Synthesizers: Theory, Design and Applications”, Kluver Academic Publishers, 2001.

[4] J. Tierney, C. M. Rader, B. Gold, “A digital frequency synthesizer”, *IEEE Trans. Audio Electoracust.*, vol. AU-19, pp. 48-57, Mar. 1971.

[5] H. T. Nicholas and H. Samueli, "An analysis of the output spectrum of direct digital frequency synthesizers in the presence of phase accumulator truncation", *Proc. of 41st Annual Frequency Control Symp.*, pp.495-502, May 1987.

[6] A. Torosyan, A.N. Willson Jr., “Analysis of the output spectrum for direct digital frequency synthesizers in the presence of phase truncation and finite arithmetic precision”, *Proc. 21th Symp. on Image and Signal processing and analysis*, pp. 458-463, June 2001.

[7] F. Curticapean, J. Niittylahti., “Exact analysis of spurious signals in direct digital frequency synthesizers due to phase truncation”, *Electronics Letters*, vol.39, n.6, pp. 499-501, Mar. 2003.

[8] J.M.P. Langlois, D. Al-Khalili, “Phase to sinusoid amplitude conversion techniques for direct digital frequency synthesis”, *IEE Proc.-Circuits Devices Syst.*, vol. 151, no. 6, pp. 519-528, Dec. 2004.

[9] A. Madisetti, A.Y. Kwentus, A.N. Willson, “A 100-MHz, 16-b, Direct Digital Frequency Synthesizer with a 100-dBc Spurious-Free Dynamic Range”, *IEEE Journal of Solid-State Circuits*, vol. 34, no.8, pp.1034 1043, Aug. 1999.

[10] F. Curticapean, K. I. Palomaki, J. Niittylahti., “Quadrature direct digital frequency synthesizer using angle rotation

algorithm", Proc. of *IEEE Int. Symp. on Circuits and Systems* (ISCAS 2003), vol.II, pp.81-84, May 2003.

[11] Y. Song, B. Kim, "Quadrature direct digital frequency syntesizer using interpolation based angle rotation", *IEEE Trans. on VLSI Systems*, vol.12, n.7, pp.701-710, Jul.2004.

[12] A. Torosyan, D. Fu, A. Willson, "A 300 MHz quadrature direct digital synthesizer/mixer in 0.25 μm CMOS", *IEEE Journal of Solid-State Circuits*, vol 38, n.6, pp.875–887, Jun.2003.

[13] Y. Song, B. Kim, "A quadrature digital synthesizer/mixer architecture using fine/coarse coordinate rotation to achieve 14-b input, 15-b output, and 100-dBc SFDR", *IEEE Journal of Solid-State Circuits*, vol.39, no.11, pp.1853-1861, Nov.2004.

[14] D. De Caro, N. Petra, A.G.M. Strollo, "A 380MHz Direct Digital Synthesizer/Mixer with Hybrid CORDIC Architecture in 0.25μm CMOS", *IEEE Journal of Solid-State Circuits*, accepted for publication.

[15] J.M.P. Langlois, D. Al Khalili, "Novel approach to the design of direct digital frequency synthesizers based on linear interpolation", *IEEE Trans. on Circuits and System II: Analog and Digital Signal Processing*, vol.50, n.9, pp.567-578, Sept. 2003.

[16] D. De Caro, E. Napoli, A.G.M. Strollo, "Direct Digital Frequency Synthesizers with Polynomial Hyperfolding Technique", *IEEE Trans. on Circuits and System II*, vol.51, n.7, pp.337-344, Jul. 2004.

[17] A. Bellaouar, M.S. O'Brecht, A.M. Fahim, M.I. Elmasry, "Low-Power Direct Digital Frequency Synthesis for Wireless Communications", *IEEE Journal of Solid-State Circuits*, vol. 35, no.3, pp.385-390, Mar. 2000.

[18] D. De Caro, A.G.M. Strollo, "High Performance Direct Digital Frequency Synthesizers Using Piecewise Polynomial Approximation", *IEEE Trans. on Circuits Systems I*, vol.52, no.2, pp.324-337, Feb. 2005.

[19] D. De Caro, A.G.M. Strollo, "High Performance Direct Digital Frequency Synthesizers in 0.25μm CMOS Using Dual-Slope Approximation", *IEEE Journal of Solid-State Circuits*, vol.40, no.11, pp.2220-2227, Nov.2005.

[20] H. T. Nicholas III, H. Samueli, "A 150-MHz direct digital frequency synthesizer in 1.25-micron CMOS with -90dBc spurious performance", *IEEE Journal of Solid-State Circuits*, vol. 26, no.12, pp. 1959-1969, Dec. 1991.

[21] B. D. Yang, J. H. Choi, S. H. Han, L. S. Kim, H. K. Yu, “An 800 Mhz low-power direct digital frequency synthesizer with on-chip D/A converter”, *IEEE Journal of Solid-State Circuits*, vol.39, n.5, pp.761 774, May 2004.

[22] D. De Caro, N. Petra, A.G.M. Strollo, “A 630MHz Direct Digital Frequency Synthesizer with 90dBc SFDR in 0.25μm CMOS”, Proc. of *IEEE Int. Solid State Circuits Conf.* 2006, San Francisco, pp.258 259, Feb.2006.

[23] F. De Dinechin, A. Tisserand, “Multipartite table methods”, *IEEE Trans. on Computers*, vol.54, n.3, pp.319-330, Mar. 2005.

[24] F. Curticapean, J. Niittylahti, “A hardware efficient direct digital frequency synthesizer”, Proc. *IEEE Int. Conf. on Electronics*, Circuits and Systems, Malta, pp. 51–54, Sept. 2001.

[25] F. Curticapean, J. Niittylahti, “Low power direct digital frequency synthesizer” Proc. 43rd *IEEE Midwest Symp. on Circuits and Systems*, Lansing, MI, USA, pp. 822–825, Aug 2000.

[26] J. F. Ardekani, “MxN Booth Encoded Multiplier Generator Using Optimized Wallace Trees”, *IEEE Trans. on VLSI Systems*, vol.1, no.2, pp.120-125, Jun. 1993.

[27] J. M. P. Langlois, D. Al Khalili, “Low Power Direct Digital Frequency Synthesizer in 0.18μm CMOS”, Proc. of *Custom Integrated Circuits Conf.* (CICC 2003), pp.21-24, Sept. 2003.

[28] Y. Song, B. Kim, “A 14-b Direct Digital Frequency Synthesizer With Sigma-Delta Noise Shaping”, *IEEE Journal of Solid-State Circuits*, vol.39, no.5, pp.847-851, May 2004.

[29] V. Stojanovic, V.G. Oklobdzija, “Comparative Analysis of Master–Slave Latches and Flip-Flops for High-Performance and Low-Power Systems”, *IEEE Journal of Solid-state Circuits*, vol.34, no.4, pp.536-548, Apr. 1999.

[30] S. Heo, R. Krashinsky, K. Asanovic, “Activity-Sensitive Flip-Flop and Latch Selection for Reduced Energy”, Proc. of *Conf. on Advanced Research in VLSI (ARVLSI)*, pp. 59-74, Mar. 2001.

[31] S. Xue, B. Oelmann, “Comparative Study of Low-Voltage Performance of Standard-cell Flip-Flops”, Proc. of *IEEE Int. Conference on Electronics*, Circuits and Systems (ICECS), vol.2, pp.953-957, Sept. 2001.

[32] D. De Caro, E. Napoli, N. Petra, A.G.M. Strollo, “A High-Speed Sense-Amplifier based Flip-flop”, proc. of *European Conference on Circuits Theory and Design* (ECCTD 2005), vol.II, pp.99-102, Sept. 2005.

[33] A.G.M. Strollo, D. De Caro, E. Napoli, N.Petra, “A Novel High-Speed Sense Amplifier based Flip-flop”, *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol.13, no.11, pp.1266–1274, Nov.2005.

[34] J. Yuan, C. Svensson, “New Single-Clock CMOS Latches and Flipflops with Improved Speed and Power Savings”, *IEEE Journal of Solid-State Circuits*, vol.32, no.1, Jan.1997.

[35] A.G.M. Strollo, D. De Caro, N. Petra, “A 630MHz, 76mW, Direct Digital Frequency Synthesizer Using Enhanced ROM Compression Technique”, *Journal of Solid State Circuits*, accepted for publication.

[36] M. Sala, F. Salidu, F. Stefani, C. Kutschenreiter, A. Baschirotto, “Design considerations and implementation of a DSP-based car-radio IF processor”, *IEEE Journal of Solid-State Circuits*, vol. 39, no. 7, pp. 1110-1118, Jul. 2004.

[37] N. Boutin, “An arctangent type wideband PM/FM demodulator with improved performance”, *IEEE Trans. on Consumer Electr.*, vol. CE-38, pp. 5-9, Feb. 1992.

[38] A. Chen, S. Yang, “Reduced complexity CORDIC demodulator implementation for D-AMPS and digital IF-samples receiver”, Proc. *IEEE Globecom*, vol. 3, Nov. 1998, pp. 1491-1496

[39] Intersil HSP50210 Datasheet.

[40] M. Krstic, A. Troya, K. Maharatna, E. Grass, “Optimized low-power synchronizer design for the IEEE 802.11a standard”, Proc. of *ICASSP03*, vol. II, pp. 321-324.

[41] J. Lindeberg, J. Vankka, J. Sommarek, K. Halonen, “A 1.5V direct digital synthesizer with tunable delta-sigma modulator in 0.13um CMOS”, *IEEE Journal of Solid-State Circuits*, vol. 40, no. 9, pp. 1978-1982, Sept. 2005.

[42] H.Y. Ko, Y. C. Wang, A. Y. Wu, “Digital Signal Processing Engine Design for Polar Transmitter in Wireless Communication Systems”, Proc. of *IEEE ISCAS Conference*, pp. 6026-6029, 2005

[43] W. B. Sander, S. V. Schell, B. L. Sander, “Polar Modulator for Multi-mode Cell Phones”, Proc. of *IEEE Custom IC Conf.*, pp. 439-445, 2003

[44] Fairchild TMC2330A datasheet.

[45] H. Ngo, V. Asari, “A pipelined architecture for real-time correction of barrel distorsion in wide-angle camera images”, *IEEE Trans. on Circuits and Syst. for Video Technology*, vol. 15, no. 3, pp.436-444, Mar. 2005.

[46] D. Halupka, N. J. Mathai, P. Aarabi, A. Sheikholeslami, “Robust Sound Localization in 0.18 um CMOS”, *IEEE Trans. on Signal Processing*, vol. 53, no. 6, pp. 2243 - 2250, June 2005.

[47] J. Volder, “The CORDIC trigonometric computer technique”, *IRE Trans. Electr. Comput.*, vol. EC-8, pp. 330-334, Sept. 1959.

[48] R. Andraka, “A survey of CORDIC algorithms for FPGA based computers,” in *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on FPGAs*, Monterey, California, 1998, pp. 191-200.

[49] I. Janiszewski, H. Meuth, B. Koppe, “pipeline efficient hybrid vectoring implementation”, Proc. of *IEEE Intern. Frequency Control Symp. and PDA Exhibition*, pp. 643-648, 2002.

[50] D. Fu, N. Willson, “An high-speed processor for digital for rectangular to polar conversion with applications in digital telecommunications”, Proc. *IEEE Globecom*, vol. 4, Dec. 1999, pp. 2172-2176

[51] D. D. Hwang, D. Fu, N. Willson, “A 400 Mhz processor for the efficient conversion of rectangular to polar coordinates for digital telecommunications applications”, Proc. *IEEE Symp. VLSI circuits*, June 2002, pp. 248-251.

[52] D. D. Hwang, D. Fu, N. Willson, “A 400 Mhz processor for the conversion of rectangular to polar coordinates in 0.25 um CMOS”, *IEEE Journal of Solid-State Circuits*, vol. 38, no. 10, pp. 1771-1775, Oct. 2003.

[53] F. De Dinechin, A. Tisserand, “Multipartite table methods”, *IEEE Trans. on Computers*, vol. 54, n. 3, mar. 2005, pp. 319-330.

[54] D. Harris, “An exponentiation unit for an OpenGL lighting engine”, *IEEE Trans. on Computers*, vol. 53, n. 3, mar. 2004, pp. 251-258.

[55] L. K. Tan, H. Samueli, "A 200 MHz direct digital synthesizer/mixer in 0.8 μm CMOS”, *IEEE Journal of Solid State Circuits*, vol. 30, pp. 193–200, Mar. 1995.

[56] B.G. Goldberg, “Direct Digital Frequency Synthesis Demystified”, LLH Technology Publishing, 1999.

[57] J. Vankka, K. Halonen, “Direct Digital Synthesizers: Theory, Design and Applications”, Kluver Academic Publishers, 2001.

[58] S. Nahm, K. Han, W. Sung, "A CORDIC-based digital quadrature mixer: comparison with a ROM-based architecture”, Proc. *IEEE Int. Symp. on Circuits and Systems* (ISCAS 1998), vol.4, pp. 385-388, 1998.

[59] G. Gielis, R. Van de Plassche, J. Van Valburg, "A 540 MHz 10-b polar to cartesian converter”, *IEEE Journal of Solid State Circuits*, vol. 26, pp. 1645 1650, Nov. 1991.

[60] Y. Ahn, S. Nahm, W. Sung, "VLSI design of a CORDIC-based derotator”, *Proc. IEEE Int. Symp. on Circuits and Systems* (ISCAS 1998), vol.2, pp. 449-452, 1998.

[61] J.E. Volder, “The CORDIC Trigonometric Computing Technique”, *IRE Transactions on Electronic Computers*, vol.EC-8, no.3, pp.330-334, Sept.1959.

[62] A. Torosyan, D. Fu, A. N. Wilson, "A 300 MHz direct digital synthesizer/mixer in 0.25 μm CMOS”, *IEEE Journal of Solid-State Circuits*, vol. 38, pp. 875–887, June 2003.

[63] D. Fu, A. N. Wilson, "A high-speed processor for digital sine/cosine generation and angle rotation”, Proc. 42nd *Asilomar Conf. on Signal, Systems and Computers*, vol.1, pp. 177-181, 1998.

[64] Y. Song, B. Kim, "A quadrature digital synthesizer/mixer architecture using fine/coarse coordinate rotation to achieve 14-b input, 15-b output, and 100-dBc SFDR", *IEEE Journal of Solid-State Circuits*, vol.39, no.11, pp.1853-1861, Nov.2004.

[65] F. Curticapean, J. Niittylahti, "An improved digital quadrature frequency down-converter architecture", 35th *Asilomar Conf. on . Signals, Systems and Computers*, pp. 1318-1321, Nov. 2001.

[66] D. De Caro, N. Petra, A.G.M. Strollo, "A 380MHz, 150mW Direct Digital Synthesizer/Mixer in 0.25μm CMOS", Proc. of *IEEE Int. Solid-State Circuits Conf.*, San Francisco, pp.258-259, Feb.2006.

[67] H. T. Nicholas and H. Samueli, "An analysis of the output spectrum of direct digital frequency synthesizers in the presence of phase accumulator truncation", Proc. of 41st *Annual Frequency Control Symp.*, pp.495-502, May 1987.

[68] A. Torosyan, A.N. Willson Jr., "Analysis of the output spectrum for direct digital frequency synthesizers in the presence of phase truncation and finite arithmetic precision", Proc. 21th *Symp. on Image and Signal processing and analysis*, pp. 458-463, June 2001.

[69] F. Curticapean, J. Niittylahti., "Exact analysis of spurious signals in direct digital frequency synthesizers due to phase truncation", *Electronics Letters*, vol.39, n.6, pp. 499-501, Mar. 2003.

[70] J. Vankka, "Methods of mapping from phase to sine amplitude in direct digital synthesis" *IEEE Trans. Ultrasonic Ferroel. Freq. Control*, vol.44, n.2, pp. 526-534, Mar.1997.

[71] H.M. Ahmed, "Signal Processing Algorithms and Architectures", Ph.D. Thesis, Dept. of EE, Stanford University, Dec. 1981.

[72] H.M. Ahmed, "Efficient Elementary Function generation with multipliers", Proc. of 19th *Symp. On Computer Arithmetic*, pp.52-59, Sept. 1989.

[73] T.G. Noll, "Carry-Save Arithmetic for High Speed Digital Signal Processing", Proc. of *IEEE Int. Symp. on Circuits and Systems*, vol.2, pp.982-986, 1990.

[74] N. Takagi, T. Asada, S. Yajima, "Redundant CORDIC Methods with a Costant Scale Factor for Sine and Cosine Computation", *IEEE Transactions on Computers*, vol.40, no.9, pp.989 995, Sept. 1991.

[75] T.B. Juang, S.F. Hsiao, M.Y. Tsai, "Para CORDIC: Parallel CORDIC Rotation Algorithm", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol.51, no.8, Aug. 2004.

[76] S. Wang, V. Piuri, E. E. Swartzlander, "Hybrid CORDIC Algorithms", *IEEE Transactions on Computers*, vol.46, no.11, pp.1202 1207, Nov.1997

[77] J.F. Ardekani, "MxN Booth Encoded Multiplier Generator Using Optimized Wallace Trees", *IEEE Transactions on VLSI Systems*, vol.1, no.2, pp.120 125, Jun. 1993.

[78] M. Nagamatsu, S. Tanaka, J. Mori, K. Hirano, T. Noguchi, K. Hatanaka, "A 15ns 32x32 b CMOS Multiplier with an Improved Parallel Structure", *IEEE Journal of Solid State Circuits*, vol.25, no.2, pp.494 497, Apr. 1990.

[79] J. Mori, M. Nagamatsu, M. Hirano, S. Tanaka, M. Noda, Y. Toyoshima, K. Hashimoto, H. Hayashida, K. Maeguchi, "A 10 ns 54x54 b Parallel Structured Full Array Multiplier with 0.5 μm CMOS Technology", *IEEE Journal of Solid State Circuits*, vol.26, no.4, pp.600 606, Apr. 1991.

[80] G. Goto, T. Sato, M. Nakajima, T. Sukemura, "A 54x54 b Regularly Structured Tree Multiplier", *IEEE Journal of Solid-State Circuits*, vol.27, no.9, pp.1229 1236, Sept. 1992.

[81] N. Ohkubo, M. Suzuki, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, Y. Nakagome, "A 4.4 ns CMOS 54x54 b Multiplier Using Pass-Transistor Multiplexer", *IEEE Journal of Solid State Circuits*, vol.30, no.3, pp.251 257, Mar. 1995.

[82] S.F. Hsiao, M.R. Jiang, J.S. Yeh, "Design of high speed low-power 3-2 counter and 4-2 compressor for fast multipliers", *Electronics Letters*, vol.34, no.4, pp.341-342, Feb 1998.

[83] D. Ghosh, S.K. Nandy, K. Parthasarathy, "TWTXBB: A Low Latency, High Throughput Multiplier Architecture Using a New 4-2 Compressor", Proc. of 7th *Int. Conf. on VLSI Design*, pp.77 82, Jan.1994.

[84] M. Suzuki, N. Ohkubo, T. Shinbo, T. Yamanaka, A. Shimizu, K. Sasaki, Y. Nakagome, "A 1.5-ns 32-b CMOS ALU in Double Pass-Transistor Logic", *IEEE Journal of Solid State Circuits*, vol.28, no.11, pp.1145 1151, Nov. 1993.

[85] B. Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", Oxford University Press, 1999.

[86] Y.H. Hu, "The Quantizzation Effects of the CORDIC Algorithm", *IEEE Transactions on Signal Processing*, vol.40, no.4, pp.834 844, Apr.1992.

[87] S.Y. Park, N.I. Cho, "Fixed-point Error Analysis of CORDIC Processor Based on the Variance Propagation Formula", *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol.51, no.3, pp.573-584, Mar.2004.

[88] N.H.E. Weste, K. Eshragian, "Principles of CMOS VLSI Design", Addison-Wesley Pub. Co., Jan. 1993, ISBN 0201533766.

[89] D. De Caro, N. Petra, A.G.M. Strollo, "A 380MHz Direct Digital Synthesizer/Mixer with Hybrid CORDIC Architecture in 0.25um CMOS", *Journal of Solid State Circuits*, accepted for publication.

[90] F. M. Gardner, "Interpolation in digital modems - Part I: fundamentals", *IEEE Trans. Comm.*, vol. 41, pp. 502-508, Mar. 1993.

[91] L. Erup, F. M. Gardner, and R. Harris, "Interpolation in digital modems - Part II: implementation and performance," *IEEE Trans. Comm.*, vol. 41, pp. 998-1008, June 1993.

[92] J. Vesma and T. Saramäki, "Interpolation filters with arbitrary frequency response for all-digital receivers," in Proc. 1996 *IEEE Int. Symp. Circuits Syst.*, pp. 568-571, May 1996.

[93] C. Farrow, "A continuously variable digital delay element," in Proc. *IEEE Int. Symp. Circuits Syst.*, pp. 2641-2645, June 1988.

[94] Dengwei Fu; Willson, A.N., Jr., "Trigonometric Polynomial Interpolation for Timing Recovery", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Volume 52, Issue 2, Feb. 2005.

[95] R.E. Blahut, "Theory and Practice of Error Control Codes", Addison Wesley, 1983.

[96] K.J. Surendra, L. Song, K.K. Parhi, "Efficient Semisystolic Architectures for Finite-Field Arithmetic", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.6, no.1, Mar.1998.

[97] L. Song, K.K. Parhi, I. Kuroda, T. Nishi-tani, "Hardware/Software Codesign of Finite Field Datapath for Low-Energy Reed-Solomon Codecs", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol.8, no.2, Apr.2000.

[98] M.A. Hasan, M.Z. Wang, V.K. Bhargava "Modular Construction of Low Complexity Parallel Multipliers for a Class of Finite Fields GF(2m)", *IEEE Transactions On Computers*, vol. 41, no. 8, Aug. 1992

[99] C.K. Koç, B. Sunar, "Low-Complexity Bit-Parallel Canonical and Normal Basis Multipliers for a Class of Finite Fields", *IEEE Transactions On Computers*, vol. 47, no. 3, Mar. 1998

[100] H. Wu, M.A. Hasan, "Low-Complexity Bit-Parallel Multipliers for a Class of Finite Fields", *IEEE Transactions On Computers*, vol. 47, no. 8, Aug. 1998.

[101] B. Sunar, C.K. Koç, "Mastrovito Multipliers for All Trinomials", *IEEE Transactions On Computers*, vol. 48, no. 5, May. 1999.

[102] A. Halbutogullari, C.K. Koç, "Mastrovito Multiplier for General Irreducible Polynomials", *IEEE Transactions On Computers*, vol. 49, no. 5, May. 2000.

[103] E.D. Mastrovito, "VLSI Architectures for Computations in Galois Fields", PhD Thesis, Linkoping Univ., Sweden, 1991.

[104] T. Zhang, K.K. Parhi, "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials", *IEEE Transactions On Computers*, vol. 50, no. 7, july 2001.

[105] L. Song, K.K. Parhi, "Low Complexity Modified Mastrovito Multipliers over Finite Fields GF(2m)", Proceedings of. *IEEE International Symposium on Circuits and Systems*, vol.1, May 1999.

[106] A. Reyhani-Masoleh, M. Anwar Hasan, "A New Construction of Massey-Omura Parallel Multiplier over GF(2m)", *IEEE Transactions On Computers*, vol. 51, no. 5, May. 2002.

[107] S.T.J. Fenn, M. Benaissa, D. Taylor, "GF(2m) Multiplication and Division Over the Dual Basis", *IEEE Transactions On Computers*, vol. 45, no. 3, march 1996.

[108] F. Rodrìquez-Henrìquez, C.K. Koç, "Parallel Multipliers Based on Irreducible Pentanomials", *IEEE Transactions On Computers*, vol. 52, no. 12, Dec. 2003.

[109] I.S. Reed, M.T. Shih, T.K. Truong, "VLSI design of inverse-free Berlekamp-Massey algorithm", *IEE Proceedings of Computers and Digital Techniques*, Vol. 138, Sept. 1991.

[110] H. Chang, C.B. Shung, C. Lee "A Reed-Solomon Product-Code (RS-PC) Decoder Chip for DVD Applications", *IEEE Journal of Solid State Circuits*, vol. 36, no.2, Feb.2001.

[111] D.V. Sarwate, N.R. Shanbhag, "High-Speed Architectures for Reed-Solomon Decoders", *IEEE Transactions On Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 7, Oct. 2001.

[112] K.Y. Chang; D. Hong; H.S. Cho, "Low complexity bit-parallel multiplier for GF(2m) defined by all-one polynomials using redundant representation", *IEEE Transactions on Computers*, vol.54, no.12, pp.1628–1630, Dec. 2005.

[113] H. Fan; Y. Dai, "Fast bit-parallel GF(2n) multiplier for all trinomials", *IEEE Transactions on Computers*, vol.54, no.4, pp.485–490, Apr. 2005.

[114] A. Reyhani-Masoleh, "Efficient algorithms and architectures for field multiplication using Gaussian normal bases", *IEEE Transactions on Computers*, vol.55, no.1, pp.34 47, Jan. 2006.

[115] A. Reyhani-Masoleh, M.Anwar Hasan, " Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over GF(2^m)", *IEEE Trans. on Computers*, col.53 no.8, pp.945-959, Aug. 2004.

[116] B. Sunar, "A generalized method for constructing subquadratic complexity GF(2k) multipliers", *IEEE*

Transactions on Computers, vol.53, no.9, pp.1097–1105, Sept. 2004.

[117] A.J. Menezes, I.F. Blake, X. Gao, R.C. Mullin, S.A. Vanstone, T. Yaghoobian, "Applications of Finite Fileds", Kluwer Academic, 1993.

[118] R.E.Blahut, "Theory and Practice of Error Control Codes", Addison Wesley, 1983.

[119] E.R.Berlekamp, "Algebraic Coding Theory", McGraw-Hill, 1968.

[120] H. Lee et al, "VLSI design of Reed-Solomon decoder architectures", *Proc of ISCAS 2000*, vol.5, pp.705-708.

[121] I.S. Reed et al , "VLSI design of inverse-free Berlekamp-Massey algorithm", IEEE Proc.-E, Vol.138, No.5, Sept.1991.

[122] H. Chang et al, "A Reed-Solomon Product-Code (RS-PC) Decoder ..", *IEEE JSSC*, vol. 36, no.2, Feb.2001.

[123] D.V. Sarwate et al, "High-Speed Architectures for Reed-Solomon Decoders", *IEEE Trans. On Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 7, Oct. 2001.

[124] A.G.M. Strollọ et al, "VLSI Design of a (255,239) Reed-Solomon Decoder", 16th *European Conference on Circuits Theory and Design* (ECCTD 2003), Sept. 2003.

[125] L. Song et al, "Low-Complexity Modified Mastrovito Multipliers Over Finite Fields GF(2M)", *Proc. of ISCAS* 1999.

[126] E.D. Mastrovito, "VLSI Architectures for Computations in Galois Fields", PhD Thesis, Linkoping Univ., Sweden, 1991.

[127] T. Zhang et al, "Systematic Design of Original and Modified Mastrovito Multipliers for General Irreducible Polynomials", *IEEE Trans. On Computers*, vol. 50, no. 7, july 2001.

[128] S.T.J. Fenn et al, "GF(2m) Multiplication and Division Over the Dual Basis", *IEEE Trans. On Computers*, vol. 45, Mar.1996.

[129] F. Henrìquez et al, "Parallel Multipliers Based on Irreducible Pentanomials", *IEEE Trans. On Computers*, vol. 52, Dec. 2003.

[130] S.H. Unger, C.J. Tan, “Clocking schemes for high-speed digital systems”, *IEEE Trans. on Computers*, vol.C-35, pp.880-895, Oct.1986.

[131] V.G. Oklobdzija, V.M. Stojanovic, D.M. Markovic, N.M. Nedovic, “Digital System Clocking: High-Performance and Low-Power Aspects”, Wiley-IEEE Press, 2003.

[132] N. Nedovic, V.G. Oklobdzija, W.W. Walker, “A Clock Skew Absorbing Flip-flop”, *Proc. of Int. Solid-State Circuits Conf.* (ISSCC), 2003.

[133] H. Partovi, R. Burd, U.Salim, F. Weber, L. Di Gregorio, D. Draper., “Flow-Through Latch and Edge-Triggered Flip flop Hybrid Elements”, *Proc. of Int. Solid-State Circuits Conf.* (ISSCC), 1996, pp.138-139.

[134] J. Tschanz, S. Narendra, C. Zhanping, S. Borkar, M. Sachdev, Vivek De, “Comparative delay and energy of single edge-triggered and dual edge-triggered pulsed flip-flops for high-performance microprocessors”, *Int. Symp. On Low Power Electronics and Design* (ISLPED 2001), pp.147 152, Aug.2001.

[135] S.D. Naffziger, G. Collon-Bonet, T. Fischer, R. Riedlinger, T.J. Sullivan, T. Grutkowski, “The Implementation of the Itanium 2 Microprocessor”, *IEEE Journal of Solid-State Circuits*, vol.37, no.11, pp.1448-1459, Nov.2002.

[136] F. Klass, C. Amir, A. Das, K. Aingaran, C. Truong, R. Wang, A. Mehta, R. Heald, G. Yee, “ A New Family of Semidynamic and Dynamic Flip-Flops with Embedded Logic for High-Performance Processors”, *IEEE J. Solid-state Circuits*, vol.34, no.5, pp.712-716, May 1999.

[137] M. Matsui, H. Hara, Y. Uetani, L. Kim, T. Nagamatsu, Y. Watanabe, A. Chiba, K. Matsuda, T. Sakurai, “A 200 Mhz 13mm2 2D DCT macrocell using sense-amplifying pipeline flip-flop scheme”, *IEEE J. Solid-state Circuits*, vol.29, pag.1482-1490, Dec. 1994.

[138] J. Montanaro et al., “A 160 Mhz 32-b 0.5-W CMOS RISC microprocessor”, *IEEE J. Solid-state Circuits*, vol.31, no.11, pag.1703-1714, Nov. 1996.

[139] B. Nikolic, V.G. Oklobdzija, V. Stajanovic, W. Jia, J.K. Chiu, M.M. Leung, “Improved Sense-Amplifier Based Flip-

flop: Design and Measurements”, *IEEE J. Solid-state Circuits*, vol.35, no.6, pag.876-883, Jun. 2000.

[140] J. Kim, Y. Jang, H. Park, “CMOS sense-amplifier based flip-flop with two N-C2MOS output latches”, *Electronics Letters*, vol.36, no.6, pag.498-500, Mar. 2000.

[141] G. Gerosa, S. Gary, C. Dietz, P. Dac, K. Hoover, J. Alvarez, H. Sanchez, P. Ippolito, N. Tai, S. Litch, J. Eno, J. Golab, N. Vanderschaaf, J. Kahle, “A 2.2W, 80MHz superscalar RISC microprocessor”, *IEEE J. Solid-state Circuits*, vol.29, no.12, pp.1440-1452, Dec. 1994.

[142] V. Stojanovic, V.G. Oklobdzija, “Comparative Analysis of Master–Slave Latches and Flip-Flops for High-Performance and Low-Power Systems”, *IEEE J. Solid-state Circuits,* vol.34, no.4, pp.536-548, Apr. 1999

[143] D. Markovic, B. Nikolic, R.W. Brodersen, “Analysis and design of low-energy flip-flops”, *Proc. of Int. Symp on Low Power Electronics and Design*, pp.52-55, Aug.2001.

[144] A.G.M. Strollo, D. De Caro, E. Napoli, N.Petra, “High-speed Direct Digital Frequency Synthesizers in 0.25-μm CMOS”, *Proc. of Custom Integrated Circuits Conference* (CICC2004), Orlando (USA), Oct. 2004.

[145] S. R. Kuang, J. M. Jou, and Y. L. Chen, “The design of an adaptive on-line binary arithmetic coding chip”, *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol.45, pp.693–706, July 1998.

[146] F. Curticapean, J. Niittylahti, “A Hardware Efficient Direct Digital Frequency Synthesizer”, proc. of *IEEE Int. Conf. on Electronics, Circuits and Systems* (ICECS 2001), vol.1, pp51-54, 2-5 Sept. 2001 Malta.

[147] A.G.M. Strollo, E. Napoli, D. De Caro, "Direct Digital Frequency Synthesizers using First-Order Polynomial Chebyshev Approximation", 28th *European Solid-State Circuits Conference* (ESSCIRC 2002), Florence (Italy), Sept. 24-26 2002, pp.527-530.

[148] B. Parhami, “Computer Arithmetic: Algorithms and Hardware Designs”, Oxford University Press, 1999.

[149] S.S. Kidambi, F. El-Guibaly, A. Antonious, “Area-Efficient Multipliers for Digital Signal Processing Applications”,

IEEE Trans. on Circuits and Systems—II: Analog and Digital Signal Processing, vol.43, no.2, pp.90-95, Feb. 1996.

[150] Y.C. Lim, "Single-Precision Multiplier with Reduced Circuit Complexity for Signal Processing Applications", *IEEE Trans. on Computers*, vol.41, no.10, pp.1333-1336, Oct. 1992.

[151] J.M. Jou, S.R. Kuang, R.D. Chen, "Design of Low-Error Fixed-Width Multipliers for DSP Applications", *IEEE Trans. on Circuits and Systems—II: Analog and Digital Signal Processing*, vol.46, no.6, pp.836-842, Jun. 1999.

[152] L. Van, S. Wang, W. Feng, "Design of the Lower Error Fixed-Width Multiplier and Its Application", *IEEE Trans. on Circuits and Systems—II: Analog and Digital Signal Processing*, vol.47, no.10, pp.1112-1118, Oct. 2000.

[153] V.G. Oklobdzija, D. Villeger, S.S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach", *IEEE Trans. Computers*, vol.45, no.3, pp.294-306, Mar. 1996.