# A FAMILY OF REDUNDANT MULTIPLIERS DEDICATED TO FAST COMPUTATION FOR SIGNAL PROCESSING

*Yannick Dumonteix and Habib Mehrez*

LIP6 / ASIM Laboratory, UPMC,
Tour 55-65 2ème étage, 4, Place Jussieu
75252 Paris Cedex France
E-mail : Yannick.Dumonteix@lip6.fr, Habib.Mehrez@lip6.fr

**ABSTRACT**

In view of the performance achieved through the use of the redundant addition, it would appear interesting to generalize the redundant notations (Carry Save, Borrow Save). To achieve this we require, as well the addition, a multiplication satisfying these notations.
This paper presents the design of a set multipliers spanning all possible I/O combinaisons, in redundant and conventional notations. We also describe the associated architectures and details of our method, which is based on parameterizable IP cores. The fonctions developed offer superior performance than conventional multipliers.

## 1. INTRODUCTION

Technological advances allowing higher integration densities have made possible the implementation of computationally intensive algorithms for applications such as : MPEG, HDTV, DCT, TFR, digital filters. Consequently the number of arithmetic operators has been increased and operator chaining has been introduced.
At the same time, the use of redundant notations limited to the internal representations of arithmetic operators[1], becomes explicit. These notations are now used at the input/output of the operators and serve also as the links between them [2, 3].
There are two main reasons for redefining the sequences : to allows more general use of the redundant adders and to economise the redundant to non-redundant conversion present at the output of many operators : multipliers, dividers, etc.[1].
The generalization of the redundant notations implies that the arithmetic operators must support all the redundant (R) / non-redundant (NR) combinations on their input/ouputs. The development of dedicated algorithms and architectures is necessary.
At least two elementary blocs are require : addition and multiplication.
For the multiplication, several algorithms and architectures have already been proposed [2, 4]. Their performances are equivalent to conventional multipliers. However, they do not satisfy all input combinations, especially the multiplication of two redundant operands.
This paper presents the architectures and the design of a set of multipliers ranging from the full redundant to the classic. The set of architectures is based on the Wallace architecture with in conventional notation.
The first section is dedicated to the presentation of the redundant notation. In the second and third sections we detail the design of each of the redundant multipliers. In the fourth section we detail the implementation method. We conclude, in the two last sections, by a performance comparison between the redundant and classical multipliers.

## 2. NUMBER REPRESENTATIONS

In this section we summarize the various notations used in radix 2 redundant arithmetic.

**Carry Save notation (CS) :**
In this system the digits are elements of $\{0, 1, 2\}$ coded in two bits of equal weight, such that $cs_i = cs_i^0 + cs_i^1$. Similarly the numbers are composed of two terms $CS = CS^0 + CS^1$.
The CS coding on N digits allows the representation of unsigned numbers in the range $\{0, ..., 2^{N+1} - 2\}$ and signed numbers in the range $\{-2^N, ..., 2^N - 2\}$ in two's complement.

**Borrow Save notation (BS)**
The digits are elements of $\{-1, 0, 1\}$ coded in two bits, such that $bs_i = bs_i^+ - bs_i^-$. Similarly the numbers are composed of two terms $BS = BS^+ - BS^-$. This representation is inherently signed.
The two terms are in two's complement. A BS on N digits allows the representation of integers in $\{-2^N + 1, ..., 2^N - 1\}$.

## 3. FULLY REDUNDANT MULTIPLIER

An analysis of existing multipliers (Wallace, Booth, ...) shows that the critical path and the surface are directly related to the number of terms to be added. We must therefore focus on limiting of the number of terms to sum and the size of the matrix elements.

### 3.1. Partial products matrix

Given a redundant multiplication $A_{CS} * B_{CS}$. In order to take into account $A_{i,CS} = 2$ and $B_{j,CS} = 2$ in the computation of the partial products, we have generalised the usage of the shifting technique to the rows and columns.
Thus, each intersection i,j generates 3 subproducts. The first corresponds to the multiplication without a shift $(A_{i,CS} = 1) AND (B_{j,CS} = 1)$, the second to a shift according to i $(A_{i,CS} = 2) AND (B_{j,CS} \neq 0)$ and the third to a shift according to j $(A_{i,CS} \neq 0) AND (B_{j,CS} = 2)$. Their sum is equal to the intersection product : $PP \in \{0, 1, 2, 4\}$.

To reduce the numbers of terms to sum, the set of subproducts is recombined in groups of three according to their shift (or weight). The number of partial products is one per intersection i,j.
This technique has a problem with the intersection of the row and column shifts. Since the two shifted subproducts are not exclusive, the partial product being coded in one bit, the result is wrong.
To correct this, it is necessary to modify input properties. If both inputs are recoded with the property : if $E_{i,CS} = 2$ then $E_{i-1,CS} = 0$, the three subproducts at the intersection of two shifts are zero. In fact, only one shifted subproduct must be null. The details of the recoding are given in 3.2.
With A on N-1 digits, B on M-1 digits and $M < N$, the total of bits to be added is $N * M + M$. This number is equivalent to the Wallace multiplier in conventional arithmetic : $N * M$, for the same input precision.

**Architecture :**
In order to limit the size of the elements of the partial products matrix, the architecture is decomposed into two blocks : figure 1. The first block (HEAD) groups the parts common to each row or

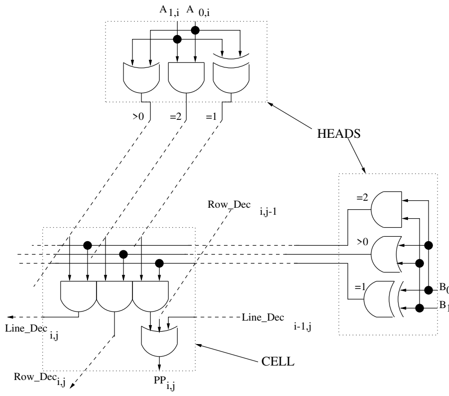


Figure 1: Partial products architecture

column. It generates a signal $E_{i,CS} \geq 1$ and two exclusive signals : $E_{i,CS} = 1$ et $E_{i,CS} = 2$. The second block (CELL) computes the subproducts (AND gates) and their recombination according to shifts (3 inputs OR gate). The row headers and column headers allow the minimisation of the hardware necessary for eleboration of the matrix elements.
In the implementation, we group the computation of the subproducts and their recombination into a single gate : (a AND b) OR (c AND d) OR (e AND f) (multiplexer without decoding). The headers correspond to half adder and OR gates. Thus, we further reduce the surface and the delay of the matrix.

## 3.2. Recoding

In their research report [4], M. Daumas and D. Matula, propose a technique of carry recoding which modifies the properties of the starting number. This recoding is based on the independant addition of the bits which make up each digit of a redundant number. For each bit we obtain a sum bit and a carry bit. The inputs are inverted if necessary.
In our case, CS to CS, the recoding $CS'_i = C'_i + S'_i$ of each digit $CS_i$ of the input number, is defined by the fallowing logic equations : carry $C'_{i+1} = CS1_i.CS2_i$ and sum $S'_i = CS1_i \oplus CS2_i$, which is equivalent to a half adder.
This technique increases the dynamic by one digit and modifies the neighborhood properties of the digits. Let $E_{CS}$ be a redundant number in Carry Save notation. Each of its digits is such that $E_{i,CS} \in \{0, 1, 2\}$. The recoding of an input digit, by the addition of the two bits building it, implies that at least one of the two resulting bits : the sum or carry, is zero. Thus, in the output CS, at least half of the bits are null.
Notice that, since the carry and sum related to the same digit have unequal weight, the digits of the output CS are comprised in $\{0, 1, 2\}$.
The modification of the neighborhood makes it impossible that two consecutive digits have the value of 2.

**Practical considerations :**
Consider a redundant number $E_{CS}$ in Carry Save notation, for which $E_{i,CS} \in \{0, 1, 2\}$, and consider the 2 recoding stages used for the multiplier. The first (R1) insures that two consecutive digits of the recoded number never have the value 2 : if $R1_{i,CS} = 2$ then $R1_{i+1,CS} \neq 2$. Thus the second (R2) implies that if $R2_{i,CS} = 2$ then $R2_{i-1,CS} = 0$.
The systematic presence of a zero next to the two's, allows the recombination of all the products, resulting from a multiplication by 2, with the neighbouring null product. As described in 3.1, this second property allows us to attain the goal of limiting the number of the terms to sum.

**Architecture :**
The figure 2 illustrates the architecture of the coding used in the

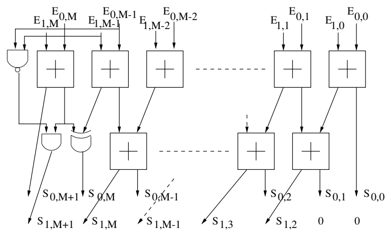


Figure 2: Recoding in two stages

multiplier. The input is signed. Each stage corresponds to a row of half adders and computes the addition of two terms with no carry propagation. In the MSB's, the regularity is not respected, in order to limit the dynamic extention and handle the sign. For a number $E_{CS}$ on N digits, the recoded output : $S_{CS}$, is on N+1 digits.

## 3.3. Sum

The addition of the partial products is achieved by a Wallace reductor. The result is in Carry Save. This multiplier carries out shifts according to both rows and columns. This characteristic implies a sign bit at the end of each row and column. Direct sign extension being costly, we have generalized the sign extention technique presented by Fadavi-Ardekani [5]. The technique is as follows : all sign bits are considered negative when added. In this case, the sum of the sign extensions can be computed.
$K = 2^N \sum_{i=0}^{N}(i+1)2^i + 2^M \sum_{j=0}^{M}(j+1)2^j$

This result will be added to the partial products (without sign extention) in the form of a constant. When a sign bit is positive, it is sufficient to assign it a value '1' to eliminate its effect in K.

## 4. VARIANTION OF THE MULTIPLIER

The initial goal was to propose a set of multipliers allowing redundant/non-redundant input/output combinations. We will base this study on the full redundant multiplier presented above.
Obtaining either redundant or non-redundant results is straightforward. The result of the Wallace reductor is in CS, it's enough to convert the result (carry lookahead adder).
Changing the input notations implies more specific modification in the architecture. The different cases are detailled below.

**Fully redundant multiplier with BS input(s) :**
For the full redundant multipliers, we use the first stage of recoding to carry out a 2's complement of the negative term of the BS input. The 2's complement is obtained by inverting all the $E_i^-$. Addition of 1 is done by setting the bit $S_{1,0}$ (always 0) of the output CS. Only the recoding is modified by changing to the BS notation. $CS * BS$ multiplier performances stay equal to that of $CS * CS$. Furthemore, in the case of the $BS * BS$ multiplier, the second input must be converted to the CS notation. Using a second recoding is the simplest solution. The area increases but the delay stay the same.

**Mixed $CS * NR$ multiplier :**
This multiplication is a particular case of the $CS * CS$. Since only one output is redundant, there is no intersection between shifts. So recoding is no longer necessary and the hardware requirements are reduced : figure 3. The number of the subproducts at each intersection is reduced from 3 to 2. Each element of the matrix is this output of a gate (a AND b)OR(c AND d). The headers are half adders.

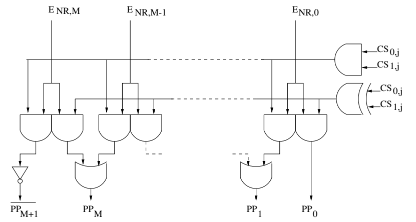


Figure 3: Signed $CS_i * E_{NR}$ products

Notice that degrading the architecture of the $CS * CS$ gives a Wallace multiplier : an AND matrix for the products without header or recoding.

**Mixed $BS * NR$ multiplier :**
As before we can carry out a 2's complement of the negative component of the BS. However, there exists a simpler structure based on the decomposition of the partial products into headers and cells : figure 4. The idea is simple : first of all, each $BS_i$ is decomposed into two signals : multiplication by 1 ($BS_i^+$ AND (NOT $BS_i^-$)) and multiplication by -1 ((NOT $BS_i^+$) AND $BS_i^-$). Then we compute the partial products (the AND/OR gates). For the multiplication by -1, the 2's complement is obtained by inverting each $NR_j$ and by adding 1 to the complete sum.

But for the sign digit of the BS it is the inverse : X1 = ((NOT $BS_i^+$) AND $BS_i^-$) et X-1 = ($BS_i^+$ AND (NOT $BS_i^-$)).
Each cell corresponds to a $2- > 1$ multiplexer. The multiplication by -1 is obtained by the select line of the multiplexer.

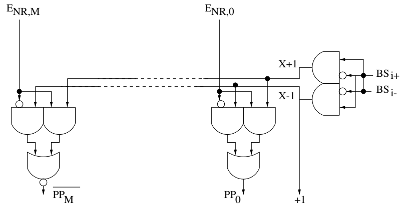


Figure 4: Signed $BS_i * E_{NR}$ products

## 5. IMPLEMENTATION METHODE

To implement all the multipliers presented in this paper, we have made a parametrizable generator. The generator was designed under GenOptim.

**GenOptim CAD environment :**
Genoptim allows the designer to concentrate exclusively on the algorithm. Futhermore, it makes it possible to perform modifications without intervention on the source code of the generator. Consequently, the developer of a generator is never bothered by electrical, placement and timing constraints. GenOptim allows the user to quickly design efficient macro-cell generators independant of the target technology. These macro functions (shifters, multiplexers, adders, etc.) are very common in the VLSI design world and may be used in many different contexts[6],[7]. GenOptim uses

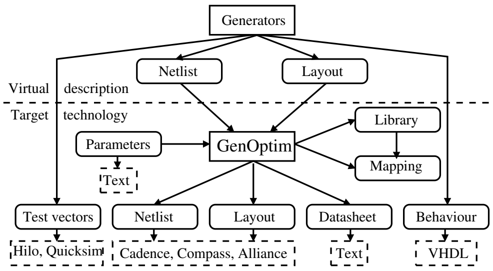


Figure 5: GenOptim description

a set of C functions which allow the user to describe the netlist, the layout, the test vectors and the behavioral views. As shown in figure 5, the design starts with a virtual description (netlist, and layout of virtual cells) of the circuit. GenOptim then maps the virtual description to the target library and optimizes the circuit. A succesfully fabricated image convolution has been reported in [7].

**Generator description :**
This generator allows us to specify the notations, the size and sign of the inputs and the output representation. It automatically creates a netlist view, test vectors and a file summarizing all the circuit characteristics. A future version will give a pre-placement. The generator is built from two distinct parts. The first part is different

| multiplier | delay in ns | % of ref | recoding or adder % of ref | area unrouted $\mu M^2$ | % of ref | recoding or adder % of ref | canal routing | % of ref | net | instance | transitor | full adder | half adder |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 bits | | | | | | | | | | | | | |
| $NR * NR -> NR$ | 14.6 | ref | 38.3% | 97650 | ref | 37% | 208950 | ref | 371 | 328 | 2243 | 36 | 6 |
| $NR * CS -> CS$ | 10.8 | -26% | . | 77427 | -21% | . | 186381 | -11% | 258 | 236 | 1845 | 36 | 5 |
| $NR * BS -> CS$ | 10.9 | -25% | . | 77679 | -20% | . | 185562 | -11% | 280 | 250 | 1711 | 36 | 6 |
| $CS * CS -> CS$ | 13.4 | -8% | 13.5% | 115794 | +19% | 21% | 280665 | +34% | 356 | 355 | 2754 | 41 | 5 |
| 16 bits | | | | | | | | | | | | | |
| $NR * NR -> NR$ | 23.5 | ref | 34.0% | 378882 | ref | 21% | 1011681 | ref | 1411 | 1379 | 9091 | 196 | 14 |
| $NR * CS -> CS$ | 15.8 | -32% | . | 354627 | -6.5% | . | 998613 | -1.3% | 1146 | 1100 | 8563 | 196 | 13 |
| $NR * BS -> CS$ | 16.5 | -30% | . | 323001 | -6.5% | . | 986742 | -2.5% | 1130 | 1076 | 8473 | 196 | 14 |
| $CS * CS -> CS$ | 20.6 | -12% | 8.5% | 468090 | +23% | 12% | 1460472 | +44% | 1364 | 1304 | 11572 | 209 | 13 |
| 32 bits | | | | | | | | | | | | | |
| $NR * NR -> NR$ | 34.0 | ref | 31.4% | 1496565 | ref | 12% | 5543964 | ref | 5523 | 5395 | 35949 | 900 | 31 |
| $NR * CS -> CS$ | 25.2 | -26% | . | 1533987 | +2.5% | . | 5619309 | +1.6% | 4937 | 4748 | 36723 | 900 | 29 |
| $NR * BS -> CS$ | 26.6 | -22% | . | 1528443 | +2.1% | . | 5607801 | +1.2% | 4999 | 4810 | 37075 | 900 | 30 |
| $CS * CS -> CS$ | 33.1 | -3% | 5.9% | 1913751 | +28% | 6% | 10579698 | +90% | 5488 | 5176 | 47352 | 929 | 29 |

for each generator and describes the algorithm in terms of virtual cells. The other is common to all generators. It calls GenOptim, which maps the virtuals cells on the target library and solves automatically the electrical problems by analyzing the circuit and the cell library used.

## 6. RESULTS

This part focuses on the performances analysis of the various architectures proposed : table above. The various circuits are compared with a classical multiplier : $NR * NR -> NR$ (Wallace), used as the reference.
All the circuits presented have been placed and routed with the Alliance CAD system [8] using the Alliance Standard Cell library in $0.5\mu M$. This library has no full adder or half adder cells, which increases the complexity of the routing.
In view of the number of nets, transistors and instances, we can say that for a given input dynamic, the circuits have the same complexity. For the $CS * CS -> CS$ , the transitor overhead is essentially due to the difference in the matrix cell and the necessary recoding.
Notice that, for all the proposed architectures : a single gate is used for the matrix elementary cell. From the number of full and half adders, we can see that the Wallace tree is identical in each circuit and has no impact on the performances variations.

**Surface evaluation :**
For many reasons, we are interested only by the non-routed area. First, the actual placing doesn't take into account the architecture regularity. Second, the lack of full and half adder gates increases the problems of local routing. With more metal layers the surface would become equal to the sum of the cell areas.
Up to 32 bit, the mixed multiplier areas are equivalent to that of the $NR * NR -> NR$ (from -20% to 3%). The change in the gain is due to the difference between matrix cells combined with the difference of growth between the partial product matrix ($N^2$) and the ouput adder of the reference multiplier ($N.log_2 N$).
In the case of the $CS * CS -> CS$, the timing gain obtained by the increase in area is inferior to 20%.

**Timing results :**
Our architectures offer better delay performance than to conventional multipliers. Initially this benefit increases with the input dynamic, subsequently it decreases. This effect is very important in the fully redundant multiplier, and is due to the electrical problems introduced by the *bad* placing.
Notice that, if the recoding of the $CS * CS -> CS$ is separated by a register, the delay gain is superior or equal to 20%.

## 7. CONCLUSION

This paper presents a multiplier generator which supports all the redundant/classical input/output combinations. The architectures developed offer better delay performance and equivalent areas, compared to conventional multipliers.
However, to reduce the routing and the electrical problems it is necessary to add a pre-placement to the generator. This is particulally true for the fully redondant multiplier.
Our goal is to generalize the redundant arithmetic to computationally intensive applications. Multipliers of this type have an advantage in terms of inherent quality as well as in quality of the redundant adders. In addition, it is possible to take advantage of operator chaining.
A major issue in this generator is the use of advanced arithmetic and the implementation according to the module generation concept. This is of great interest as it allows parameterization, portability and design reuse.

## 8. REFERENCES


[1] Muller J-M., *Arithmétique des ordinateurs, opérations et fonctions élémentaires*, Masson, 1989.

[2] Olivier Peyran, *Synthèse d'Architectures Intégrées Utilisant des Arithmétiques Redondantes*, Phd repport INP Grenoble France, december 1997.

[3] E. Antelo et al *High Performance Rotation Architectures Based on the Radix-4 CORDIC algorithm*, IEEE Transaction on computers, Vol.46, $N^o$ 8, August 1997.

[4] M.Daumas et al *Recoders for Partial Compression and Rouding*, Research report, January 1997. ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/ RR1997/RR1997-01.ps.Z

[5] J. Fadavi-Ardekani, *MxN Booth Encoded Multiplier Generator Using Optimized Trees*, IEEE Transaction On VLSI Systems, vol.1, $N^o$ 2, june 1993.

[6] M. Aberbour et al *Architecture and design methodology of the RBF-DDA neural network*, ISCAS98.

[7] A. Houelle et al *A real time image convolver*, ICSPAT96.

[8] A. Greiner et al *ALLIANCE : A complete set of CAD Tools for teaching VLSI Design*, Third EuroChip Workshop, 1992.