Data Path Optimization using Redundant Arithmetic and Pattern Matching

Sophie Belloeil, Roselyne Chotin-Avot, Habib Mehrez
University Paris VI, LIP6/SOC Laboratory,
4, place Jussieu,
75252 Paris Cedex 05, France
Email: sophie.belloeil@lip6.fr

***Abstract*— Considering the performance increase provided by redundant operators such as adders and multipliers, it appears interesting to generalize the use of those operators in high computational digital circuit design. However, using redundant arithmetic in conjunction with classical arithmetic is a complex task for circuit designers who might not have necessarily the required arithmetic knowledge. Therefore, it becomes mandatory to develop optimization CAD tools which automate the use of redundant arithmetic in circuit design. This paper presents such an optimization tool which, based on non redundant representations, introduces automatically redundant operators thanks to pattern matching techniques. To illustrate this, the optimizations of a filter and a Discrete Cosine Transform (DCT) generators are presented.**

***Index Terms*— automatic, optimization, redundant arithmetic, pattern matching, CAD tool**

I. Introduction

REDUNDANT operators such as adders and multipliers [1], [2] have very good performances in terms of timing and area [3]. Using those operators in VLSI circuit design can thus appear advantageous, enabling architecture optimizations and consequently further improvements as for circuits' performances. To illustrate this, we can consider the hand-made implementations of a DCT macro bloc [4] and a distance computation unit [5] using those architectures, which both result in a significantly increase of the timing performances with a small area overhead.

Mixing classical and redundant arithmetics in an explicit way can nevertheless appear quite tedious to non initiated designers, for whom, furthermore, the rapid pace of technological evolution puts a great "time to market" pressure. That is why redundant arithmetic is not very often used for high computational digital circuits. Such a pressure on design cycle combined with strict performance contraints make more and more useful the automation of the introduction of redundant arithmetic in high computational digital circuit design, bringing it more accessible. Research works on automatic arithmetic optimization tools have therefore existed for some time, in several areas such as high level synthesis [6], [7] and logical synthesis [8]–[11]. Our work, based on pattern matching techniques, consists in studying the chains of arithmetical operators, and proposing general rules for optimization. High computationnal digital circuits involving signal, image and control processing indeed include arithmetic data paths composed of such kind of chains of arithmetical operators.

This paper presents an optimization tool which, starting from a representation of a circuit in classical arithmetic, introduces redundant operators through the analysis of the arithmetical operators chains by applying local transformations on the circuit. Furthermore, several kind of optimizations are presented as we aim at proposing a flexible technique. The automatic optimizations of a filter and a DCT generators show the results obtained with this tool. The remainder of this paper is organized as follow: Section 2 contains a global description of the redundant arithmetic and its main advantage. In Section 3, we describe our redundant optimization tool. Section 4 presents the results of our experiments. Finally, in Section 5, possible improvements are listed and then we conclude.

II. Redundant arithmetic

A. Number representations

Redundant arithmetic involves two number representations [12]:

- **Carry-Save representation**: A digit is defined by $cs_i = cs_i^0 + cs_i^1$ with $cs_i \in \{0, 1, 2\}$ so that a number is considered as the sum of two terms: $CS = CS^0 + CS^1$
- **Borrow-Save representation**: A digit is defined by $bs_i = bs_i^0 - bs_i^1$ with $bs_i \in \{-1, 0, 1\}$ so that a number is considered as the substraction of two terms: $BS = BS^0 - BS^1$

The abbreviations CS and BS are commonly used for Carry-Save and Borrow-Save representations, as well as NR and R for classical (Non Redundant) and Redundant representations.

B. Mixed arithmetic

The arithmetic used is called **mixed arithmetic**, defined as the combination between classical and redundant representations. The sole use of redundant arithmetic is indeed not conceivable for several reasons:

- Firstly, we must preserve the NR representations of the inputs/outputs of the circuits
- Secondly, we have to deal with operators which are not arithmetic operators such as multiplexors, boolean operators, etc ...

In order to make those two arithmetics compatible, we have at disposal all kind of arithmetic operators accepting both redundant or non redundant inputs/outputs. We also are able to convert one representation to the other. For example, the conversion from a CS representation to a NR representation is the addition between the two terms composing the redundant number.

C. Architectures

1) Adder: The architecture of a **mixed** adder (adding a CS number and a NR number) is shown in Figure 1.

A similar architecture exists for a **redundant** adder (adding two CS numbers). Both adders provide a CS output made of the effective sum and the carries ($Output = S + C$).

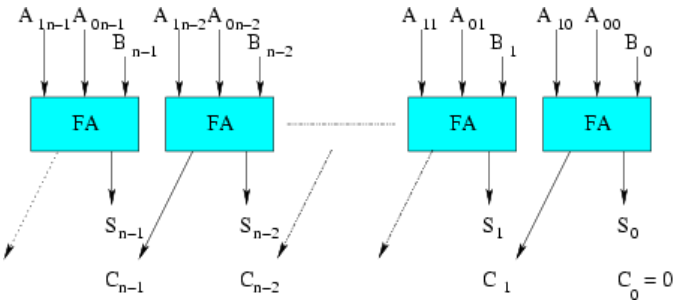


Fig. 1. Implementation of a mixed adder

The architecture presented shows the main advantage of redundant arithmetic: it allows to suppress the carry propagation in the computation of an addition.

2) Multiplier: The principle of multipliers' architectures is shown in figure 2.

Those architectures are divided into four parts:

- Recoding (optionnal): Recoding of the inputs in R form
- Partial Products + Sum (mandatory): Effective computation of the multiplication
- Conversion (optionnal): Conversion of the output in NR form

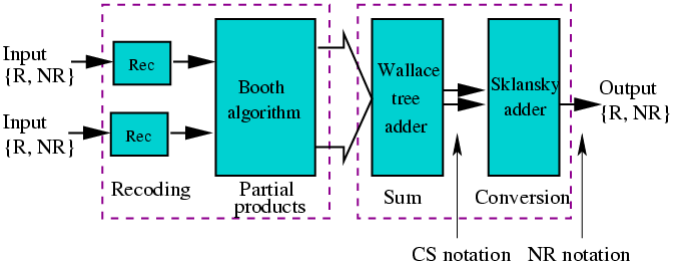


Fig. 2. Implementation of a multiplier

As we can see, the output of the Wallace tree is in CS form, so authorizing the output of the multiplier to be in a CS form allows the deletion of a CS to NR converter. CS multipliers are therefore bound to have a better critical time than classical multipliers, but a bigger area because of the input recoders (especially with two R inputs).

3) Representations: Schematic representations used for mixed/redundant operators are shown in Figure 3.

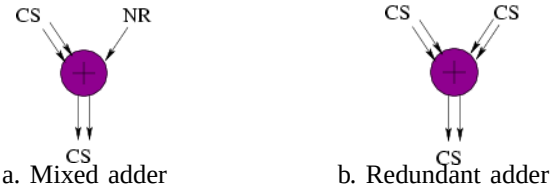


Fig. 3. Schematic representations of redundant operators

D. Advantage of redundant/mixed arithmetics

The main drawback of classical arithmetic is the carry propagation which is, indeed, the main issue for critical paths. Therefore, many researches have been done in order to minimize the carry propagation time in addition computation, such as carry-skip, carry-lookahead or parallel prefix adders [13]. All those architectures manage to minimize the carry propagation time, but do not suppress carry propagation. Furthermore, the shorter the carry propagation time is, the bigger the area is.

Whereas removing carry propagation is not possible using classical arithmetic, using redundant arithmetic allows to suppress it as shown in the mixed adder architecture in Figure 1. Consequently, the time to perform an addition of two numbers is constant and independent of the number of digits. Besides, the area of redundant and mixed adders is smaller than most classical adders area. Addition being an essential operator, the potential benefit of using redundant and mixed adders is significant for timing and for area. This benefit represents the main advantage of redundant arithmetic.

III. Automation tool

A. Aim

The aim of our CAD tool is to take advantage of redundant arithmetic by introducing automatically redundant operators in the process of circuit design.

B. VLSI design flow

Our CAD tool is part of a classical VLSI design flow as shown in Figure 4 and takes place just before logical synthesis. It is part of the **Coriolis** platform [1] [14] and it's input format is therefore made in **Stratus**, the procedural netlist description language [15] of this platform. From this functional specification of a circuit and a knowledge in arithmetic, we obtain an optimized *virtual* description (i.e. before structural mapping) of the circuit. The tool modifies the specification of the different operators and interconnections, while ensuring the feasibility of the mapping toward a target technology. After this process, the VLSI design flow remains unchanged.

C. Pattern macthing

1) Pattern: A **pattern** is a collection of arithmetical operators (typically two or three operators) and their interconnections.

[1] formerly named Tsunami

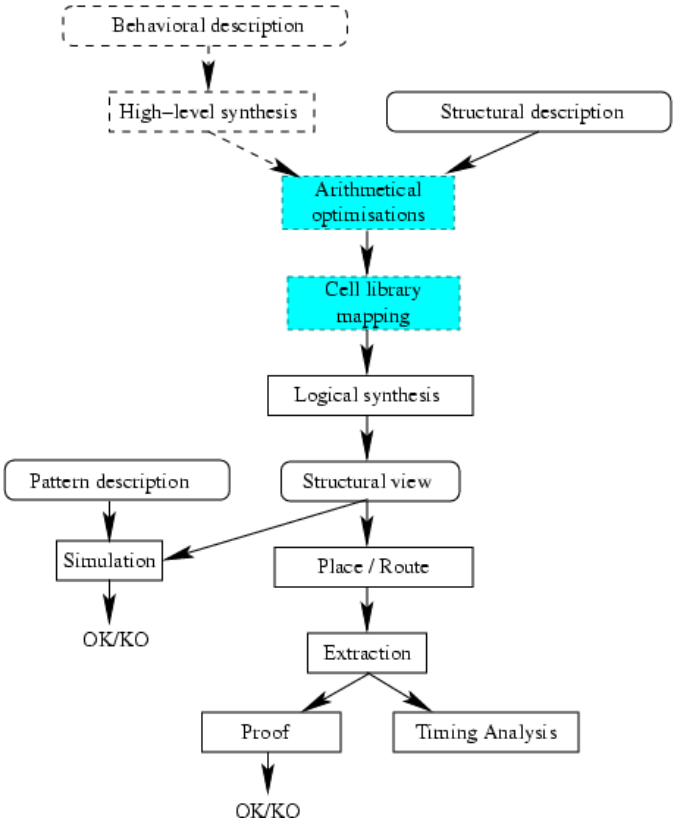


Fig. 4. Design flow

2) Principle: The aim of the tool is to search for patterns which are bound to be replaced by more competitive ones. The new patterns are composed of redundant operators, which make them more optimized than the previous ones, in terms of timing and area.

3) Rule: The pattern couples are called **rules**.

- The fundamental assumption is that a pattern and its substitute have the same behavior and the same inputs/outputs.
- The second assumption is that the substitute pattern of a rule is more optimized than the one it is substituted to.

4) Set: We have chosen to deal with patterns that contain chains of two operators, because working on bigger chains would become complicated to handle without leading to better results. A set of twenty-four rules has therefore been established in order to be sufficient to handle most cases of connection between adders and mutlipliers. With a smaller set, results might not be optimal. In opposition, a bigger set would not lead to better results, and would make tough the choice of an optimal rule if several patterns matched.

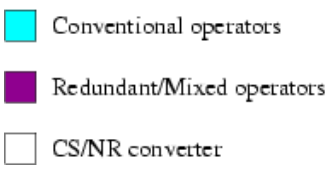


Fig. 5. Operators' color

In the reminder of this paper, colors are going to be used in order to distinguish different kinds of operators (classical or redundant). Those colors are shown in Figure 5.

The rules selected are divided into four categories. One rule of each category is shown in Figure 6. Other rules derive from the ones presented. We can notice that our main way to optimize is to consider the addition of two NR signals as a CS signal.

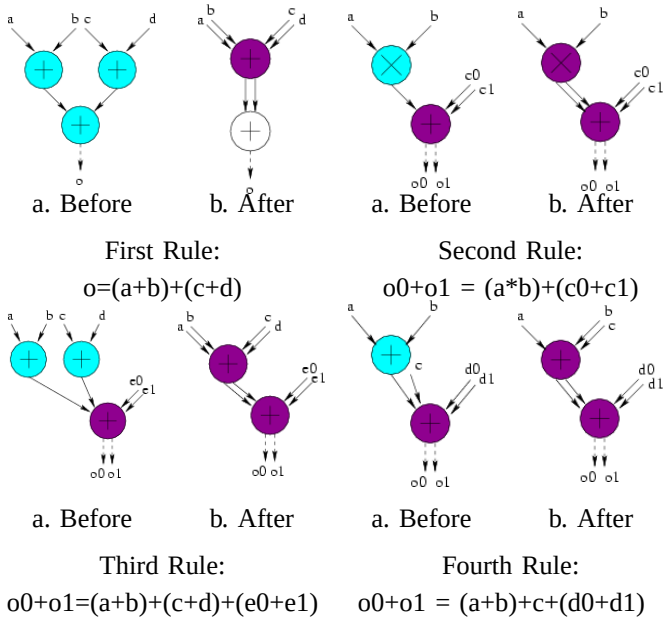


Fig. 6. Example of rules

5) Evaluation: Each pattern has been evaluated in terms of timing and area in order to verify that the *substitute* redundant pattern of each rule has better performances than the one it is substituted to. The architectures we used for adders and multipliers are [12], [13]:

- NR adder and CS/NR converter: Sklansky algorithm
- NR/R multiplier: Booth modified algorithm

The results obtained with the rules presented are shown in Table I which outlines the performances of the two patterns of each rule, in terms of timing and area, before and after optimization. Those results tend to prove that an optimization with redundant operators can improve timing and area. We can see that the improvements are up to 64.5% for the critical path, and 62.5% for the area. We can also notice that, in most cases, the bigger the number of digits is, the better the improvements are.

The evaluation of each rule has shown that, in every rule, the timing is improved with the substitution of the pattern. As for the area, only one kind of rules alters it: the ones that introduce redundant multipliers. As already discussed, those operators are indeed big. Whereas some studies suggest not to use redundant multipliers [6], we have decided to let the designer choose whether or not to use such kind of rules, depending on the goal to achieve: an optimization of the timing with a possible deterioration of the area allowed or forbidden.

6) Description of the rules: The rules are described in a configuration file. We have chosen the .xml file format which

Pattern	Number of digits	Area (mm^2)		Time (ns)	
		before	after	before	after
First	8	0.05 ref	0.03 -40%	21.6 ref	20.6 -4.6%
	16	0.12 ref	0.06 -50%	27.8 ref	23.4 -15.8%
	32	0.27 ref	0.13 -51.9%	35.4 ref	28.2 -20.3%
Second	8	0.14 ref	0.11 -21.4%	39.8 ref	30.8 -22.6%
	16	0.44 ref	0.36 -18.2%	55.5 ref	43.2 -22.2%
	32	1.5 ref	1.3 -13.3%	76.2 ref	56.5 -25.8%
Third	8	0.05 ref	0.02 -60%	20.1 ref	17.5 -12.9%
	16	0.1 ref	0.05 -50%	23.5 ref	17.5 -25.5%
	32	0.22 ref	0.09 -59.1%	27.8 ref	17.5 -37%
Fourth	8	0.03 ref	0.02 -33.3%	20.1 ref	13.2 -34.3%
	16	0.06 ref	0.03 -50%	23.5 ref	13.2 -43.8%
	32	0.13 ref	0.07 -46.2%	27.8 ref	13.2 -52.5%

TABLE I
PATTERNS PERFORMANCES

is a quite simple kind of description so that designers can easily add or delete one pattern if they want to.

Therefore:

- On one hand, our tool is suitable for designers who want to use automatically redundant arithmetic without having the needed arithmetic knowledge.
- On the other hand, the simpleness of the patterns' description allows designers who have the arithmetic knowledge to modify the list of patterns.

One can consequently notice one of the advantages of using the proposed pattern matching approach: compared to other solutions, this approach is very modular and our tool can be dedicated to basic optimization, it can as well be a testing platform for new architectures for example.

D. Algorithm

Based on the pattern matching approach, our algorithm is a recursive process which skims through a model with non redundant operators, from its outputs to its inputs, and introduces redundant operators by doing local transformations. This *bottom-up* approach is inspired by [9]. This algorithm is illustrated in the example of Figure 7.

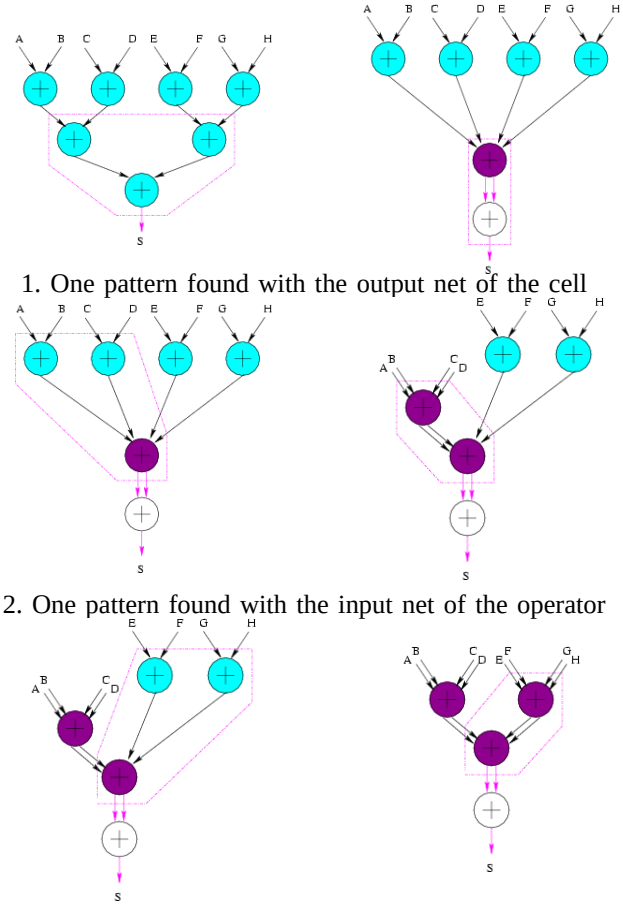
```
Algorithm ( net ) is
  if ( net is an input )
    stop

  while ( pattern = SearchPattern() )
    Replace ( pattern )

  operator = input operator of the net

  foreach input of the operator
   Algorithm ( input )

foreach output of the cell
  Algorithm ( output )
```



1. One pattern found with the output net of the cell
2. One pattern found with the input net of the operator
3. Another pattern found with the same net

Fig. 7. Example

E. Multiple operation trees

One important issue to handle is the management of multiple operation trees such as shown in Figure 8.

Let us consider the operation of Figure 8.a:

- A conventional transformation is to optimize each tree separately, which produces a circuit with the minimum area, but a non optimal optimization of the timing (because of an extra cost of CS to NR conversions): This kind of transformation is therefore called **Priority to the area** and the result is shown in Figure 8.b.
- In order to deal with that problem, another behavior of the tool has been implemented which can be called **Priority to the timing**: The rule is to treat every expression with a separate tree and without any ressource sharing. This generates a circuit with a minimal timing, but an excessive overload as for the area. The result is shown in Figure 8.c.
- One last solution has been implemented in order to be able to make trade-offs between area and timing as discussed in [11]: it allows to optimize the timing while minimizing the area penalty. This behavior is therefore called **Trade-off** and the result, shown in Figure 8.d, is the most optimal.

In view of modularity, the three possible ways to handle the issue of multiple operation trees are implemented in the tool. Designers can therefore choose which one to use depending on their needs.

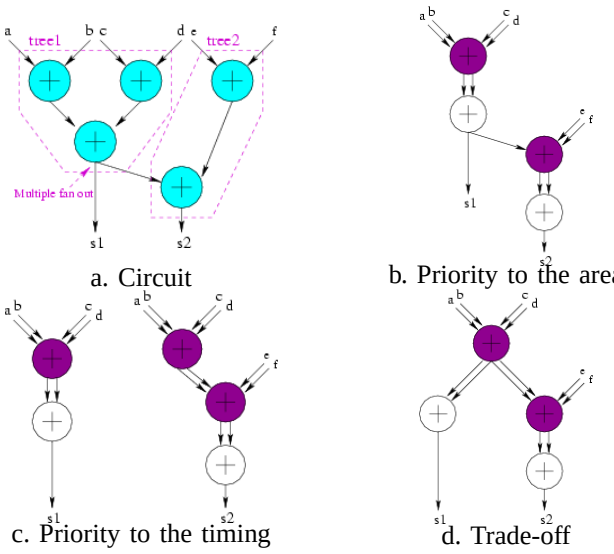


Fig. 8. Multiple operation trees

F. Conclusion

The innovative parts of our work are:

- Applying a very well known technique in order to convert operators into a redundant form
- Including *mixed* and *redundant* adders as well as *mixed* and *redundant* multipliers

Furthermore, as already discussed, we mainly aim at modularity and flexibility. Following that idea, two factors are provided :

- Several parameters exist in order to specify which behavior to choose concerning (1) the use of redundant multipliers (2) the way to handle multiple operation trees
- The set of patterns provided relies on our arithmetic knowledge and is bound to provide the best results but designers can nevertheless choose to modify it thanks to the easiness of the description

As for the treatment of subtraction, we have chosen to replace it by adding the negation (using two's complement) of the subtraction, that is:

```
a - b = a + not(b) + 1
```

The use of only the CS representation is indeed quite common and leads to good results [6]–[11]. Note that we also consider the possibility of using the BS representation in order to deal with the substraction which could lead to even better results.

IV. Experimental results

First of all, we tested our algorithm on arithmetic computations which are typically used in applications involving signal, image and control processing and are therefore representative of arithmetical blocks which can be instanciated in *real benchmarks*. Second of all, we tested our algorithm on several benchmarks: four different implementations of a filter and two different implementations of a DCT macro generator.

In order to perform those tests, we used the place and route tools of the Cadence CAD System using the Alliance [16] Standard Cell Library in 0.35 μm. Several tables are going to be presented, showing the results in terms of timing and area, before and after optimization.

A. Designs with additions, multiplications and subtractions

1) Transformation of designs with additions: Firstly, we tested our algorithm on summation tree designs of 16 bits operands. The results are summarized in Table II. Those results are quite satisfactory, reducing significantly both timing and area. One can also notice that improvements of area are more effective as the number of operands in expression increases. This is due to the good performances of redundant adders: small area combined with a constant timing.

Sum	Area (mm^2)		Time (ns)	
	Before	After	Before	After
3 operands	0.08 ref	0.05 -37.5%	27.8 ref	18.7 -32.7%
4 operands	0.12 ref	0.06 -50%	27.8 ref	25 -10.1%
5 operands	0.16 ref	0.07 -56.2%	39.7 ref	28.9 -27.2%
6 operands	0.2 ref	0.09 -55%	39.7 ref	33.8 -14.9%
7 operands	0.24 ref	0.1 -58.3%	39.7 ref	35.2 -11.3%
8 operands	0.28 ref	0.11 -60.7%	39.7 ref	35.3 -11.1%
9 operands	0.32 ref	0.12 -62.5%	51.2 ref	39.1 -23.6%
10 operands	0.36 ref	0.13 -63.9%	51.2 ref	40.1 -21.7%

TABLE II
Results for addition expressions

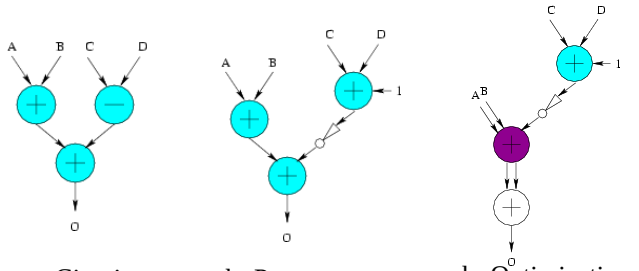
2) Transformation of designs with a mixture of additions, subtractions and multiplications: Secondly, our algorithm was tested on designs with a mixture of additions, subtractions and multiplications. We used 8 bits operands for those tests. The results are summarized in Table III. Those results show that our algorithm can also be applied to such kind of designs. Both timing and area are again significantly reduced. We can nevertheless notice that the area is less improved for designs with multiplications, due to the big area of redundant multipliers, it is even deteriorated once. We can also notice one degradation of the timing in a design with a substraction: this degradation is due to the adjunction of an inverter in the middle of the arithmetic operators chain, such as shown in Figure 9.

3) Conclusion: Those designs are quite trivial, but they are a first step in order to demonstrate the usefulness of redundant operators. They can show on which kind of designs our algorithm can be applied to. Compared to results exposed in [8]–[10], our results seem to have the same order of magnitude. More precisely: (1) concering the summation tree designs,

Equation	Area (mm^2)		Time (ns)	
	Before	After	Before	After
(a-b)+(c-d)	0.06 ref	0.03 -50%	23.9 ref	22.4 -6.3%
(a-b)+(c+d)	0.05 ref	0.03 -40%	23.9 ref	21.7 -9.2%
(a+b)-(c+d)	0.05 ref	0.04 -20%	23.1 ref	27.4 +18.6%
(a*b)+c	0.15 ref	0.14 -6.7%	59.4 ref	40.7 -31.5%
(a*b)+(c*d)	0.25 ref	0.24 -4%	59.4 ref	45.8 -22.9%
(a*b)+(c*d)+(e*f)	0.39 ref	0.36 -7.7%	67.7 ref	54.6 -19.4%
(a+b)*(c+d)	0.14 ref	0.22 +57.1%	61.3 ref	45.1 -26.4%

TABLE III

RESULTS FOR MIXED EXPRESSIONS



a. Circuit b. Pre-treatment b. Optimization

Fig. 9. Problem due to a substraction

the area is better improved with our tool, but the timing less improved, (2) concerning the designs with multiplications, the timing is better improved with our tool (the improvment of the area depends of the exemple) which strenghtens our choice of using redundant/mixed multipliers.

B. Filters

In order to test the performances of our tool, we optimized a Finite Impulse Response filter (FIR). Figure 10 presents the architecture of a FIR.

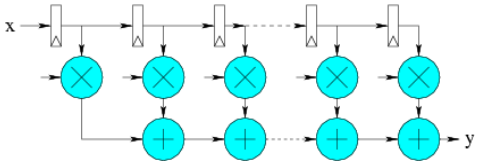


Fig. 10. Architecture of a filter

Given the architecture presented (with no multiple operation tree), we used our optimization tool with the option **Priority to the timing**. We optimised four types of this filter : (1) with four or eight coefficients, (2) with input data and constants encoded on eight or sixteen bits. Table IV shows the results obtained, resulting in highly increased performances in timing and area, compared to the classic architecture implementation. This is because: (1) all multipliers outputs are converted into CS form, which suppresses the CS to NR converters, and (2), all slansky adders are therefore transformed into redundant adders. As a consequence, all arithmetical operators used have a better timing and a better area than the previous ones. Note also that the bigger the number of coefficients is, the better the improvments are, for the timing as well as for the area. Concerning the number of bits, it seems that the timing is better improved as it grows as opposed to the area which is less improved.

Type	Area (mm^2)		Time (ns)	
	Before	After	Before	After
4coef/8bits	0.65 ref	0.46 -29.2%	65.6 ref	49.4 -24.7%
4coef/16bits	1.95 ref	1.5 -23.1%	87.7 ref	64.6 -26.3%
8coef/8bits	1.32 ref	0.9 -31.8%	103.7 ref	68.9 -33.6%
8coef/16bits	3.97 ref	2.94 -25.9%	134.4 ref	84.6 -37.1%

TABLE IV

OPTIMIZATIONS OF A FIR

C. Discrete Cosine Transform generator

We optimized a Discrete Cosine Transform generator [4] also. Several algorithms have been proposed in order to compute the 1-D DCT:

- The Loeffer Signal Flow Graph [17] has been widely used: this implementation permits to compute the 1-D DCT of 8 pixels in only one cycle. The corresponding architecture is shown in Figure 12.
- A graph partitionning is possible in order to obtain a rate of 1 pixel/cycle. This architecture is shown in Figure 11.

Both implementations contain multiple operation trees. Those designs are therefore good examples in order to test the performances of the three kinds of options of the tool: **Priority to timing**, **Priority to area** and **Trade-off**.

1) Partitionning graph: Figure 11 shows the architecture of the 1-D DCT partitionning graph.

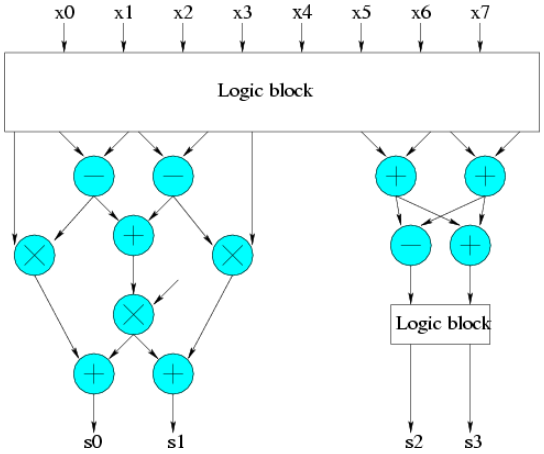


Fig. 11. Architecture of the partitionning graph

Table V shows the results obtained with the different behaviors as for timing and area:

- **Time**:
  - Improvement of the critical time whatever the option of the tool is
  - Better improvement with options **Priority to the timing** and **Trade-off** because more patterns can be subsituted
- **Area**:
  - **Priority to the area**: Improvement of the area
  - **Priority to the timing**: Deterioration of the area because of the lack of ressource sharing
  - **Trade-off**: Improvement of the area thanks to the ressource sharing

Note that the architecture generated with the option **Trade-off** of our tool is the same one than the hand-coded one presented in [4].

Mode	Area (mm^2)		Time (ns)	
	Before	After	Before	After
Priority to the area	1.88 ref	1.65 -12.2%	133.6 ref	124.7 -6.7%
Priority to the timing	1.88 ref	1.95 +3.7%	133.6 ref	113.1 -15.4%
Trade-off	1.88 ref	1.61 -14.4%	133.6 ref	110.2 -17.5%

TABLE V

OPTIMIZATIONS OF THE PARTITIONNING GRAPH

2) Complete Signal flow graph: Figure 12 shows the architecture of the 1-D DCT complete signal flow graph.

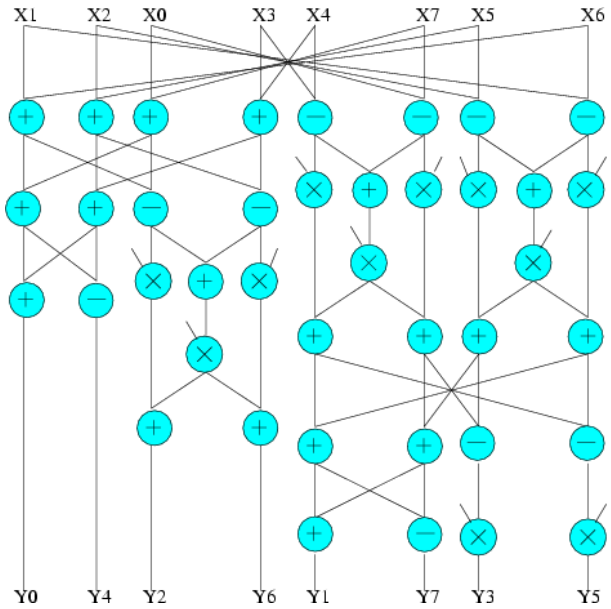


Fig. 12. Architecture of the complete signal flow graph

Table VI shows the results for this implementation:

- **Time**:
  - Improvement of the critical time whatever the option of the tool is
  - Better improvement with options **Priority to the timing** and **Trade-off** because more patterns can be substituted
- **Area**:
  - **Priority to the area**: Improvement of the area
  - **Priority to the timing**: Major deterioration of the area because of the lack of ressource sharing
  - **Trade-off**: Minor deterioration of the area because the ressource sharing is not sufficient

Mode	Area (mm^2)		Time (ns)	
	Before	After	Before	After
Priority to the area	3.96 ref	3.9 -1.5%	115.7 ref	114.9 -0.7%
Priority to the timing	3.96 ref	7.4 +86.9%	115.7 ref	106.3 -8.1%
Trade-off	3.96 ref	4.1 +3.5%	115.7 ref	100.5 -13.1%

TABLE VI

OPTIMIZATIONS OF THE COMPLETE SIGNAL FLOW GRAPH

3) Conclusion: The examples of the Discrete Cosine Transform generators corroborate the two following assumptions: (1) the use of the Carry-Save representation leads to good results, even for benchmarks with several substractions, and (2) several multipliers in a benchmark can restrict a lot the reduction of the area. Those two examples contain several operators with multiple fan out also, which leads to the following conclusion as for the different kind of treatments:

- **Time**: Options **Priority to the timing** and **Trade-off** always give a better result than option **Priority to the area**
- **Area**:
  - Option **Priority to the area** always gives a better result than option **Priority to the timing**
  - The result obtained with option **Trade-off** is dependant from the context, it can be better or worse than with option **Priority to the area**

Furthermore, we can conclude that the choice of those options change significaly the results obtained with our tool. It is therefore important to have the control of the option to use in function of the performances to achieve.

V. FUTURE WORKS

A. Borrow-Save

The use of the two's complement for the treatment of the subtraction is common and leads to good results. However, the introduction of an inverter cell is not always optimal: the design of Table III illustrated in Figure 9 is a good example. Therefore, the use of the BS representation can be an interesting idea which needs to be studied in order to improve the results obtained.

B. Timing and area analysis

Another idea is to be able to estimate automatically the performances of the patterns in terms of:

- Area: addition of all the areas of the operators
- Timing: critical path considering the critical paths of the operators

This would improve the modularity of the tool, allowing to automatically evaluate all the patterns. A designer could therefore easily test new patterns. Furthermore, it would improve the perennity of the tool. The set of rules that have been tried indeed depends on the architectures of the arithmetic operators as well as on the technology used. Our tool could therefore be obsolete because of a new architecture of adder for example. Being able to estimate automatically all rules would avoid that issue. In addition, applying such an evaluation on the circuits to optimize would allow to create new behaviors of the tool such as searching for patterns only while skimming through the critical path. This process would minimize the changes done to the design while ensuring an improvement of the critical path.

C. Optimal solution

The chosen approach is an heuristic based on positive cases (the set of rules). It therefore is a greedy algorithm generating an optimized solution but not necessarily the best one. Considering the time to process the complete signal flow graph of the 1-D DCT in classical arithmetic: 27,5 seconds, the optimizations take from 46,6 seconds to 200 seconds given the behavior chosen as for multiple operation tree. The overcost not being important, the implementation of an algorithm generating all possible solutions and choosing the optimal one can appear as an interesting approach.

D. Treatment of the non arithmetic blocks

Making a specific treatment to non arithmetic blocks can be a good idea for heterogeneous designs containing both arithmetic and non arithmetic operators. A process such as the one shown in Figure 13: duplication of a non arithmetic block, e.g. a multiplexor (in order to use two mixed adders instead of two classical adders), would allow to improve performances.

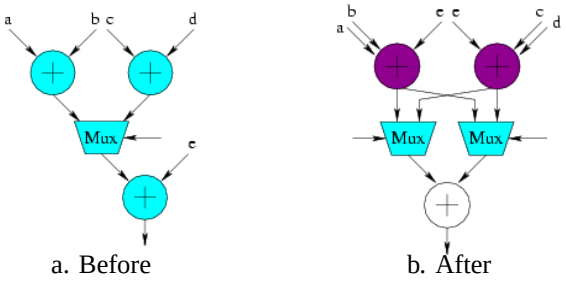


Fig. 13. Treatment of a multiplexor

VI. Conclusion

This paper describes an algorithm based on pattern matching techniques, that introduces redundant operators in high computational digital circuits. The idea of using pattern matching techniques in order to find operators that can be redundant is new and leads to good results.

More powerful solutions for this problem have existed for some time, such as including the use of redundant arithmetic in high level synthesis [6], [7]. Those approaches are nevertheless far less flexible than our approah. The advantages of using the proposed pattern matching approach are indeed based on flexibility and modularity as first of all several parameters exist in order to guide the behavior of the tool and second of all the set of rules can easily be modified in order to test new architectures.

Experimental results indicate that our work can be used effectively on several designs with mixture of additions, substraction and multiplication. This encourages us, first of all to investigate the different kinds of improvements which can be applied to this algorithm, and second of all, to create new benchmarks (such as a distance computation unit) on order to corroborate our results.

References

[1] Marc Daumas and David W. Matula “A Booth multiplier accepting both a redundant or a non redondant input with no additional delay.” *ASAP’00: Proc. of IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 205–214, july 2000.

[2] Yannick Dumonteix and Habib Mehrez “A family of redundant multipliers dedicated to fast computation for signal processing.” *ISCAS’00: Proc. of IEEE International Symposium on Circuits and Systems*, volume 5, pp. 325–328, mai 2000.

[3] Yannick Dumonteix “Optimisations des chemins de données arithmétiques par l’utilisation de plusieurs systèmes de numérations” *PhD thesis*, Université Pierre et Marie Curie Paris VI, october 2001.

[4] Yannick Dumonteix, Roselyne Chotin, and Habib Mehrez “Use of redondant arithmetic on architecture and design of a high performance DCT macro-bloc generator.” *DCIS’00: Proc. of 15th Conference on Design of Circuits and Integrated Systems*, pp. 428–433, november 2000

[5] Yannick Dumonteix, Yann Bajot, and Habib Mehrez “A fast and low-power distance computation unit dedicated to neural networks, based on redundant arithmetic.” *ISCAS’01: Proc. of IEEE International Symposium on Circuits and Systems*, volume 4, pp. 878–881, mai 2001.

[6] Anne Mignotte, Jean-Michel Muller, and Olivier Peyran. “Mixed arithmetic: Introduction and design structure.” *2nd International conference on Massively Parallel Computing Systems*, 1996.

[7] Olivier Peyran “Synthèse d’architectures intégrées utilisant des arithmétiques redondantes” *PhD thesis*, Institut National Polytechnique de Grenoble, Laboratoire d’Informatique du Parallélisme de l’École Normale Supérieure de Lyon, december 1997.

[8] Taewhan Kim, William Jao, and Steve Tjiang. “Circuit optimization using carry-save-adder cells.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):974–984, october 1998.

[9] Taewhan Kim, William Jao, and Steve Tjiang “Arithmetic optimization using carry-save-adders” *DAC’98: Proc. of the 35th Design Automation Conference*, pp. 433–438, june 1998.

[10] Junhyung Um and Taewhan Kim and C. L. Liu “Optimal Allocation of carry-save-adders in arithmetic optimization” *ICCAD’99: Proc. of IEEE International Conference on Computer-Aider Design*, pp. 410–413, november 1999.

[11] Youngtae Kim and Taewhan Kim “Accurate exploration of timing and area trade-offs in arithmetic optimization using carry-save-adders” *ASP-DAC’01: Proc. of the conference on Asia South Pacific design automation*, pp. 622–628., january 2001.

[12] A. Avizienis. “Signed-digit number representation for fast parallel arithmetic.” *IRE Trans. Electronic Computers*, 10:389–400, 1962.

[13] I. Koren. “Computer Arithmetic Algorithms.” Prentice Hall, 1993.

[14] Christophe Alexandre, Hugo Clement, Jean-Paul Chaput, Marek Sroka, Christian Masson and Remy Escassut “TSUNAMI: An Integrated Timing-Driven Place And Route Research Platform.” *DATE’05: Proc. of the conference on Design, Automation and Test in Europe*, pp. 920–921, march 2005.

[15] Sophie Belloeil, Damien Dupuis, Christian Masson, Jean-Paul Chaput and Habib Mehrez “Stratus : a procedural circuit description language based upon Python.” *ICM ’07: Proceedings of the 19th International Conference on Microelectronics*, december 2007

[16] A. Greiner and F. Pecheux. “Alliance: A complete set of cad tools for teaching vlsi design”, 1992.

[17] C. Loeffler, A. Lightenberg and G.Moschytz. “Practical Fast 1D-DCT Algorithls with 11 Multiplications” *ICASSP’89: Proc. of Intl. Conf. On Acoustics, Speech and Signal Processing*, pp. 988-991, 1989.