A FAST AND LOW-POWER DISTANCE COMPUTATION UNIT DEDICATED TO NEURAL NETWORKS, BASED ON REDUNDANT ARITHMETIC

Yannick Dumonteix, Yann Bajot 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, Yann.Bajot@lip6.fr and Habib.Mehrez@lip6.fr

ABSTRACT

This paper presents the design of a fast and low power consumption distance computation unit : $\sum_i (A_i - B_i)^2$. It is dedicated to the digital RBF neural network implementation.
The proposed architecture is composed of two parts. The first computes the distance $(A_i - B_i)^2$, and the second performs the sum of these distances. It is based on an efficient squarer in redundant arithmetic. Thank to this operator, the distance measure circuits developed offer better performances than those based on classical arithmetic. The average gain is equal to 11% in delay and 18% in power consumption.

1. INTRODUCTION

Today, with the advent of sub-micron technologies, it is possible to integrate a complete RBF neural network on a chip. This allows the development of embedded applications using a neural network such as face or voice recognition. In this case, the important criteria are real-time processing and power consumption.
A RBF network [1, 2] is composed of three layers (figure 1). The first distributes the inputs to the hidden layer neurons. These neurons constitute the heart of the network. A vector prototype is associated to each hidden neuron.
To evaluate a hidden neuron output, the first step is to compute the distance between the input vector and the prototype associated with the neuron. The second step is to apply to the computed distance a non-linear function like a Gaussian (figure 1). The output layer computes the decisions according to the weighted sum of the outputs of the hidden layer.

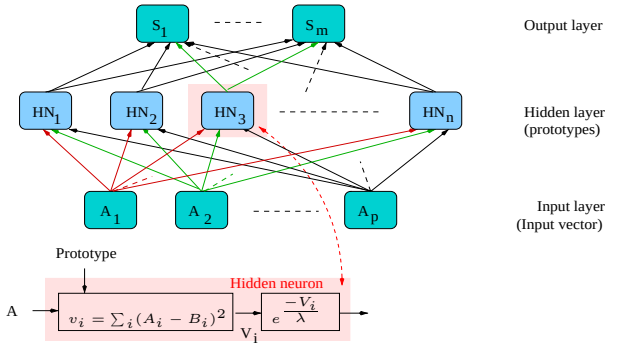

Figure 1: Radial Basis Function (RBF)

In this paper we are interested in the distance computation : $\sum_i (A_i - B_i)^2$. The aim of this work is to propose a fast and low power consumption implementation. Considering the excellent performances of the fundamental operators (the adder [3] and the multiplier [4]) in redundant arithmetic we propose to use this arithmetic to increase the performances in delay and power consumption of the distance computation.
The first section is dedicated to the presentation of the redundant arithmetic notations. In the second section we detail the design of the distance computation unit (DCU) and the squarer in redundant arithmetic. The third section describes the implemented DCU and squarer performances.

2. NUMBER REPRESENTATIONS

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

2.1. 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 $CS = CS^0 + CS^1$. This representation allows the coding of signed and unsigned numbers. 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.

2.2. 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 the range $\{-2^N + 1, ..., 2^N - 1\}$.

2.3. Notice that

The common use of classical and redundant arithmetic, requires the compatibilty of various representations. A CS can be the sum of two conventional numbers (NR) : CS^0 and CS^1, which will be coded in unsigned binary notation. In signed CS notation , CS^0 and CS^1 will be in 2's complement. With a number in BS notation, BS^+ and BS^- will be in 2's complement.

3. ARCHITECTURE

3.1. Distance computation unit (DCU) : $\sum_i (A_i - B_i)^2$

The figure 2 presents the corresponding architecture of the distance computation unit (DCU) : $\sum_i (A_i - B_i)^2$. It is composed of two parts. The first computes the distance $(A_i - B_i)^2$, and the second performs the sum of these distances by accumulation / storage.
The figure shows that the first part corresponds to a squarer with an input in Borrow-Save notation. Its output is in classical notation.
In order to reduce more the hardware the accumulation is included directely in the squarer (*real implementation* in figure 2). Thus, the architecture is essentialy composed of a redundant squarer. As the redundant operator ouput is in redundant notation (CS), it is necessary to convert it in classical notation : ouput adder in the *real implementation*.

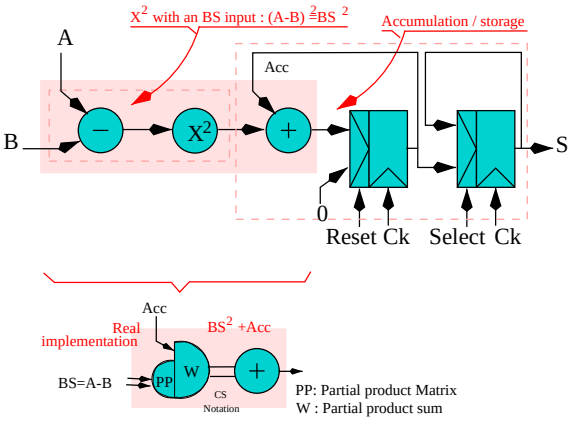

Figure 2: Distance computation unit architecture (DCU)

3.2. Squaring unit : X^2

Any standard multiplier can be used for computing $X^2 = X * X$. That is also true for redundant arithmetic.

3.2.1. Full redundant multiplier :

In [4] the authors give a solution to implement a full redundant multiplier : $A_{CS} * B_{CS}$. This multiplier architecture performs shift operations on rows and columns, in order to take into account digit values ($A_{i,CS} = 2$ and $B_{j,CS} = 2$) used in the computation of partial products.
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 sub-products is recombined in groups of three according to their shift and weight. The number of partial products is one per intersection i,j. Since the two shifted sub-products 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. Both inputs are recoded with the property : if $E_{i,CS} = 2$ then $E_{i-1,CS} = 0$; the three sub-products at the intersection of two shifts are zero.

3.2.2. Squarer : partial product computation

In classical arithmetic, a special-purpose squarer can be build in hardware, in order to reduce significantly delay, area and power consumption [3]. The partial products matrix can be considerably simplified before performing multi-operand addition. A term $a_i * a_i$ is reduced to a_i and a pair of terms $a_i * a_j$ $a_j * a_i$ can be replaced by $2 * a_i * a_j$. The partial product number is almost reduced by two.
The figure 3 presents the general architecture of a redundant squaring unit, after application of this method on a redundant multiplier. We show the described input recoding and the modified partial product matrix.

Partial product matrix :

Each black arrow corresponds to a shift according to the row or the column. Since specific partial products $a_i * a_i$ are element of {0,1,4}, shifts are equal to two.
In order to limit the size of the partial products matrix elements, the structure is decomposed into two blocks. The first block (*generic header recoding* in figure), groups the parts common to each row or column. It generates a signal $E_{i,CS} \geq 1$ and two exclusive signals $E_{i,CS} = 1$ and $E_{i,CS} = 2$. The second block (*generic cell of partial product matrix* in figure), computes the sub-products (AND gates) and their recombination according to shifts (3 inputs OR gate).

Input recoding :

In [4], the recoding architecture corresponds to two rows of half adders which compute the addition of two terms without carry propagation : *generic recoding cell* in figure 3.
In addition to the input property modification, the recoding reduces the value set of digit to {0,1}. It is true for the two first LSB digits and the MSB digit. This property allows to reduce the partial products matrix and the matrix cell in many cases : only one or two sub-products by partial product (zero or only one shift in figure 3).
In order to reduce more the matrix hardware, we have developed a new input recoding (figure 4). The architecture is composed of two stages. The first ensures that two consecutive digits of the recoded number never have the value 2, and that one on two digits is defined by only one bit. The second implies that if the digit d_i equals 2, the digit d_{i-1} equals 0. That is true because the sum and the carry of a half-adder are exclusive. This is the desired recoding property.
The particularity of this input recoding is that half of the digits are defined by a single bit. It allows to increase the number of matrix cells which recombine only one or two sub-products, and in the same time to reduce the hardware of the matrix.

Notice that :

1. To take into account BS input, we use the first stage of recoding to carry out a 2's complement of the negative term BS^-.
2. In classical arithmetic, the partial product matrix is the same except the partial products 7x. These ones are due to shifts.

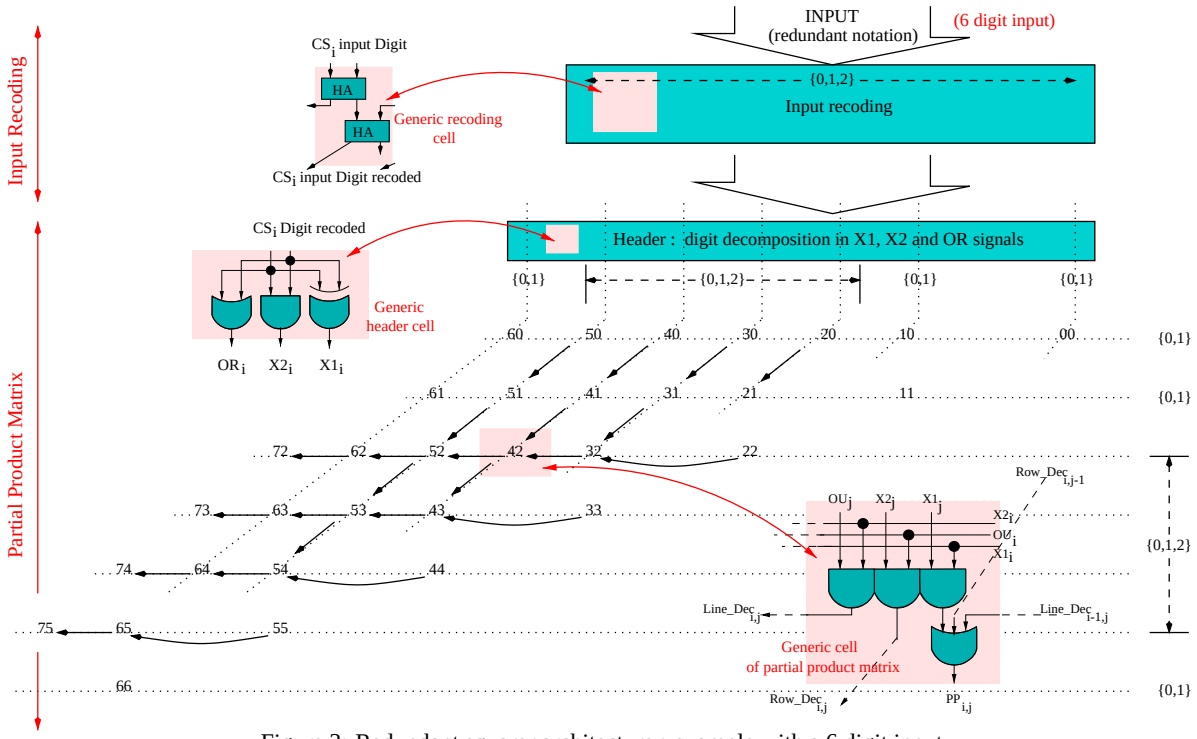

Figure 3: Redundant squarer architecture : example with a 6 digit input

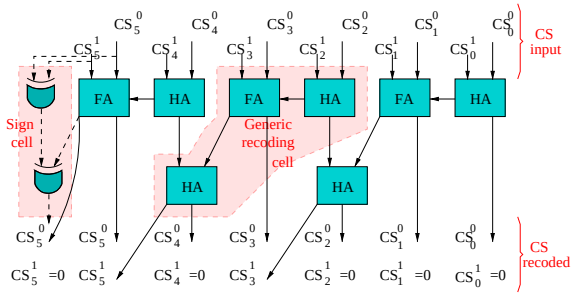

Figure 4: Example of a 6 digit Carry-Save recoding

3.2.3. Partial product sum

The addition of the partial products is achieved by a Wallace reductor [3]. The result is in Carry Save notation. This square unit carries out shifts according to both rows and columns. With a redundant multiplier, this characteristic implies a sign bit at the end of each row and column. For the square unit, the partial product matrix modification reduces the sign bits only to column (or row).

Direct sign extension being costly, we have generalized the sign extension 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.

This result will be added to the partial products (without sign extension) in the form of a constant (K). When a sign bit is positive, it is sufficient to assign it a value '1' to eliminate its effect in K. In practice, the value of each sign bit is complemented. For the figure 3 example, the sign bits are : 60, 61, 72, 73, 74 and 75.

4. PERFORMANCES

This part focuses on the performances analysis of the proposed architectures. The design features of the square unit and the distance computation unit, are presented in classical and redundant arithmetic : table 1.

To situate the performance gains, the various circuits are compared with a classical multiplier : $NR * NR \rightarrow NR$, used as the reference. Classical and redundant implementation are then compared. For each real value corresponding to delay, consumption, or surface, we indicate the gain in % of the reference result (ref). The percentage limited by brackets given for the redundant circuits, corresponds to the gain of this circuits face to classical circuits.

All the circuits presented have been placed and routed with the Cadence CAD system, using the Alliance Standard Cell Library [6] in $0.35\mu m$. This library associated to the Alliance CAD, permits to have area, delay and power consumption estimations. The power consumption computation is based on a tool presented in [7].

4.1. Squaring unit : X^2

Comparison with the reference :

Following the input size the gains are from 36% to 46% in area, from 26% to 15% in delay and 58% to 48% in power consumption for the classical squarer. For the redundant squarer, the gains are better : 45% in area, 31% in delay and from 65% to 46% in power consumption.

Important gains in area, delay and power consumption have been introduced by the choice to develop specific architecture for the squarer. That is true in classical and redundant arithmetic.

Comparison between classical and redundant implementations :

		Classic arithmetic						redundant arithmetic								
Operator	input size	Area μM^2		Delay ns		Consumption $\mu W/Mhz$		Area μM^2			Delay ns			Consumption $\mu W/Mhz$		
$A*B$	8	6890	(ref)	8.7	(ref)	119	(ref)	-			-			-		
$A*B$	16	26288	(ref)	12.9	(ref)	399	(ref)	-			-			-		
$A*B$	24	65929	(ref)	15.2	(ref)	809	(ref)	-			-			-		
$A*B$	32	137886	(ref)	17.0	(ref)	1448	(ref)	-			-			-		
A^2	8	4410	-36%	6.4	-26%	50	-58%	3773	-45%	(-14.4%)	6.0	-31%	(-6.3%)	42	-65%	(-16.0%)
A^2	16	15092	-43%	9.6	-26%	197	-51%	14822	-44%	(-1.8%)	8.5	-34%	(-11.5%)	171	-57%	(-13.2%)
A^2	24	36027	-45%	12.5	-18%	433	-46%	37515	-43%	(+4.1%)	10.2	-33%	(-18.4%)	397	-51%	(-8.3%)
A^2	32	74130	-46%	14.5	-15%	759	-48%	77175	-44%	(+4.1%)	12.0	-30%	(-17.2%)	785	-46%	(+3.4%)
$(A-B)^2$	8	6259	-9%	10.7	+23%	147	+23%	7166	+4%	(+14.5%)	9.9	+14%	(-7.5%)	130	+9%	(-11.5%)
$(A-B)^2$	16	19293	-27%	14.3	+11%	478	+20%	22491	-14%	(+16.6%)	13.0	+1%	(-9.0%)	386	-3%	(-19.2%)
$(A-B)^2$	24	44222	-33%	16.9	+11%	961	+19%	51480	-22%	(+16.4%)	14.8	-3%	(-12.5%)	760	-6%	(-20.9%)
$(A-B)^2$	32	86350	-37%	20.4	+20%	1713	+18%	99849	-28%	(+15.6%)	17.5	+3%	(-14.2%)	1426	-2%	(-16.8%)

Table 1: Squarer and distance calculation unit performances

Globally area and power consumption are the same for the classical and redundant squarer. Following these criteria and up to 24 bits, the redundant squarers are more interesting. The difference between the two implementations concerns delay results. The gains introduce by the redundant squarers are from 7% to 19%. The average is equal to 14%. The redundant squarer appears as more interesting than the classical squarer.

4.2. Distance computation unit (DCU) : $\sum_i (A_i - B_i)^2$

Comparison with the reference :

The interest of this comparison is to situate the DCU performances. In classical arithmetic, the delay and power consumption are superior, respectively from 11% to 23%, and from 18% to 23%. In the same time, the area is inferior by 30%. For the redundant DCU, delay and power consumption are almost equal to the performances of the multipliers. In the same time, the area decrease from -4% to 28%.

Comparison between classical and redundant implementations :

The DCU in redundant arithmetic offers superior performances in delay : from 10% to 15%, and from 12% to 21% in power consumption. In the same time, the area overhead is limited to 17%.
The difference between the DCU architecture and the squarer is essentially the input adder in classical arithmetic, and the output adder in redundant arithmetic. The gain evolution between the two implementations (squarer and DCU) is due to this difference. Note that both adders are implemented with a Sklansky Architecture [3]. The area overhead is due to the difference of the input sizes of these both adders (2 times), the squarer areas are the same. The modification in the delay gain is also due to the input size difference $O(log_2(size))$.

5. CONCLUSION

In this paper, we have presented the design of a fast and low power consumption distance computation unit (DCU) : $\sum_i (A_i - B_i)^2$. This computation unit is very important for the RBF neural network implementation. It is composed of two parts. The first computes the distance $(A_i - B_i)^2$, and the second performs the sum of these distances.

The proposed architecture is based on a squarer in redundant arithmetic. This operator offers better results than a squarer in classical arithmetic : according to the input size, from 10% to 20% in time and from 16% to -3% in power consumption for the same area.
Compared to conventional implementation based on classical arithmetic, the DCU circuits implemented offer superior performances in delay : from 10% to 15%, and from 12% to 21% in power consumption. In the same time, the area overhead is limited to 17%.

Using the redundant arithmetic permits to increase the timing performances and to reduce the power consumption with a small area overhead.

6. REFERENCES

[1] B. Granado et al. "An embedded system for the real time execution of GRBF networks". *"Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems"*, 1999.

[2] M. Aberbour et al. "Architecture and design methodology of the RBF-DDA neural network". *"International Symposium on Circuits and Systems (ISCAS)"*, 1998.

[3] B. Parhami. *"Computer Arithmetic, Algorithms and Hardware Designs"*. "Oxford University Press", 1999.

[4] Y. Dumonteix et al. "A family of redundant multipliers dedicated to fast computation for signal processing". *"International Symposium on Circuits and Systems (ISCAS)"*, May 2000.

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

[6] A. Greiner et al. "A complet set of CAD Tools for teaching VLSI Design". *"Third EuroChip Workshop"*, 1992. http://www-asim.lip6.fr/alliance/.

[7] J. Dunoyer. *"Modèles et Méthodes Probabilistes Pour L'évaluation de la Consommation des Circuits Intégrés VLSI"*. PhD thesis, Université Pierre et Marie Curie Paris, 1999. http://asim.lip6.fr/publications/1999/.