Fast Multi Operand Decimal Adders using Digit Compressors with Decimal Carry Generation

Dadda, Luigi; Nannarelli, Alberto

Publication date:
2009

Document Version
Publisher's PDF, also known as Version of record

Fast Multi Operand Decimal Adders using Digit Compressors with Decimal Carry Generation

Luigi Dadda
Politecnico di Milano, Italy

Alberto Nannarelli
Technical University of Denmark

Abstract: *We consider multi operand decimal adders designed with an architecture implementing first the addition of all the digits of each column (i.e. with the same decimal weight) and then combining in various ways such column sums for obtaining the final result. Different and efficient architectures can be conceived on the basis of compressors of a number of digits (e.g. three) generating a smaller number of digits (e.g. two) and, simultaneously, a decimal carry to be accounted for by the next (to the left) column. A suitable scheme has been proposed by Vazquez, Antelo and Montuschi, capable not only to generate the decimal carry but also to accept an incoming carry from the column at the right, if any. Such unit has been designed for decimal digit in BCD-4221 code. We show in this paper an improved theory and a compact notation of such compressor permitting the design of schemes using a large number of cells. A comparison is also made between multi-operand adders of different architectures.*

1 – Introduction

Decimal arithmetic has been used extensively in the earliest computer era [1] and it has been recently [2] revived due to the need of processing large amounts of decimal data in Internet and financial applications and in fields like statistics.

Recently, three papers have treated the same subject of this paper (the simultaneous addition of several decimal numbers) using different approaches. Kenney and Schulte [3,4] proposed three different methods, two called speculative approaches, based on purely decimal arithmetic on BCD-8421 coded numbers, a third based on an initial binary addition with subsequent corrections. Choi [5] solved the problem using a binary tree of fast carry-look-ahead decimal adders. Dadda [6] proposed a hybrid method based on parallel fast binary addition of all the digits in each decimal column, with subsequent binary-to-decimal conversion and the addition of column-sums for obtaining the final result. With such a procedure all the decimal carries in each column are accounted for.

In [7] Vazquez, Antelo and Montuschi proposed a new kind of 3-to-2 digit compression with generation of a decimal carry to be transmitted to the next decimal column. The cell, called VAM by the authors' initials in the following, has been conceived with the scope of designing fast decimal multipliers through the addition of the set of partial products. Clearly, the same cell can be applied to the simpler case of multi-operand adders.

We will show an abstract representation of the VAM cell that makes easier to draw schemes based essentially on it. Such representation could equally well be applied to the multiplier design. We consider useful to show in this paper its use in the simpler case of multi-operand decimal addition. A comparison with the results obtained in previously proposed schemes will also be shown.

2 – The basic compressor cell

Fig. 1.a) shows the compression cell introduced by Vazquez, Antelo and Montuschi in [7]. It assumes that the input and output decimal numbers are BCD-4221.

A four bit decimal column of weight 10^0 (composed from the corresponding bit-columns with weights 4,2,2,1 respectively) is shown. Two bits columns of the adjacent 10^1 (left) and 10^{-1} decimal columns are also shown.

Three digits A1, A2, A3 are input, the output being represented with two digits, S and ***C,*** **S** in the same decimal column, while ***C*** is composed from the three least significant bits weighed 4, 2, 2 in the input's column, while the most significant bit (weight = 10) is generated in the decimal column at the left (weight = 10).

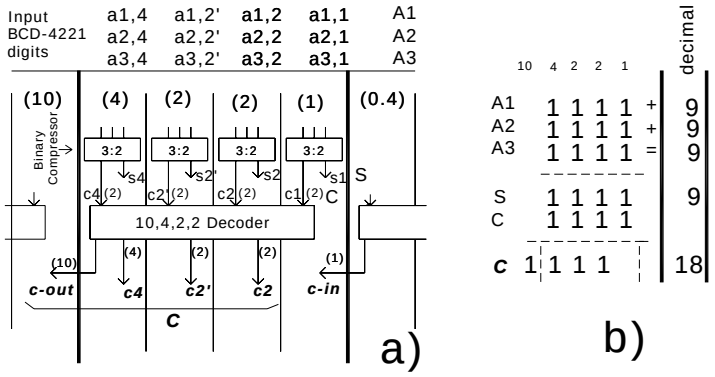

Fig 1: **a)**: The scheme of a Vazquez-Antelo-Montuschi (VAM) Decimal Compression cell, composed from a Binary Compressor and a Decoder. The inputs are the A1, A2, A3 digits, the output the S and ***C*** digits. A decimal carry **c-in** from the decimal column at the right can be placed in the least significant bit of ***C***. A similar carry **c-out** is generated for the decimal column at the left; **b)** example of addition 9+9+9.

The internal structure of the cell is composed from two parts.

- a Binary Compressor, composed from a set of four full adders (3:2), one for each binary column, each fed by the three bits of the addends and generating (in the same binary column) a sum bit and a carry bit.
- a 10, 4,2,2 Decoder, i.e. a combinational network fed from the four carry bits given by the Binary Compressor, and generating four output bits (***c2, c2', c4*** and ***c-out***) representing the input's value with output bits weighed 10, 4, 2, 2 respectively.
    - Three digits in the central column are added with the four full adders (3:2).
    - The four Sum outputs bits compose the first output digit, S.
    - The four carry bits are marked with weight "(2)" since they are generated within the same bit column.

Note that the values of the carry-bits produced by the four full adders can have value 2 (carry output true) or 0 (carry output false). The total numerical value represented by such carry-bits is therefore always an even number. The task of the d*ecoder* is to represent such value with bits having the weights 10, 4, 2, 2. No output in the rightmost binary column of weight 1 will be generated from the *decoder*.

For all inputs of the *decoder* greater than 9, a 1 will be generated for the 10 weighed output. The remaining part of the input value to the decoder, smaller that 10 and even, has to be coded with the weights 4, 2, 2.

Note that the bit weighed "10" in binary column (10) of the decimal column at the right is certainly available, for what has been said before.

An example (worst case): the three input digits are valued 9 (1111 in 4221 code), is shown in Fig. 1.b). The output digit Sum is also valued 9 (1111 in 4221 code). Since the total input value is 3*9=27, the value of the bits composing the Carry is: 27-9=18. This cannot be represented by a decimal digit (no matter in which code). It can be decided to send a 1 in a cell belonging to the next (at the left) binary column (with weight 10 since it belongs to the 10^1 decimal column). The remaining three bits in the input decimal column, weighed 4, 2 and 2 respectively, are valued 8 in total. Added to the ten in column weighed 10_{10}, sent previously, it gives 18. Adding it to S=9, we get 27 = 3*9.

Note finally that the weights written close to the inputs and the outputs can be divided by 2 (all being even). It can then be said that the *decoder* recodes the inputs according to the weights 5,2,1,1.

The just described decoder will be denoted *dec2* (or *x2* as suggested in [7]) since its inputs have weight equal to 2.

x	4	2	2	1	2x	10	4	2	2	1
0	0	0	0	0	0	0	0	0	0	
1	0	0	0	1	2	0	0	1	0	
2	0	0	1	0	4	0	1	0	0	
2	0	1	0	0	4	0	1	0	0	
3	0	0	1	1	6	0	1	1	0	
3	0	1	0	1	6	0	1	1	0	
4	0	1	1	0	8	0	1	1	1	
4	1	0	0	0	8	0	1	1	1	
5	1	0	0	1	10	1	0	0	0	
5	0	1	1	1	10	1	0	0	0	
6	1	1	0	0	12	1	0	0	1	
6	1	0	1	0	12	1	0	0	1	
7	1	1	0	1	14	1	1	0	0	
7	1	0	1	1	14	1	1	0	0	
8	1	1	1	0	16	1	1	1	0	
9	1	1	1	1	18	1	1	1	1	

Table 1: Truth –table of the *dec2* decoder

Table 1 shows the truth table of the *dec2* decoder of Fig. 1.a). The input variables are paired according to their decimal values (column x). In column 2x the corresponding doubled values are given. The columns marked *10, 4, 2, 2,* contain the coding of all the given values. Note that the value 2 is available in two adjacent columns. When a single 2 is needed, the left column 2 has been chosen for placing a *1*. Note again that the column weighed **1** is not used in the described recoding, being available for the recoding in the decimal column to the right (weighed 10^{-1}).

We will see in Chapter 4 that decoders with higher multiplicative factors can be defined

In the multi-operand adder schemes that will be illustrated in the next chapter we need to use quite a number of identical compression cells. It is thus convenient to adopt a simplified notation of such cells, in which the internal structure is ignored. A proposal for such a notation is shown in Fig.2. 2). In such a notation three input registers, A1, A2 and A3, store the three input digits, coded in accord with the code used in the compression cell, i.e. the BCD-4221 code.

A line separates the input registers A1, A2 and A3 from the output registers S and ***C***. Note that the least significant ***c1*** bit–cell is loaded with the bit ***c-in*** generated as ***c-out*** from a compression cell in the decimal column at the right, if any; otherwise it will be put to 0.

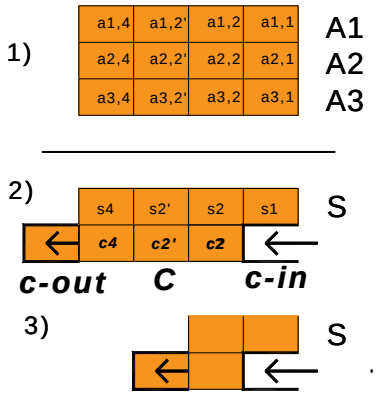

Fig. 2: 2) A compact notation of Fig. 1 compression cell.
3) A more compact notation.

In Fig. 2.3) a simplified notation is shown, in which the two-bit columns *s4 c4, s2' c2', s2 c2* are compressed in the space allotted for a single column. This obtains more compact drawings.

2.1 Characterisation of the compression cell

For using the cell in the network that we intend to design it is important to know its properties abstracting as far as possible from its internal architecture. In a decimal compressor we would expect, for instance, that knowing the decimal input digits (3 in our case) we could tell that the decimal carry is found simply by adding the three input digits and finding out that the carry is 0 or 1 or 2. This is not the case for the VAM compressor, as it can be seen by computing its result "by hand" on the basis of its internal structure, or by simulating its operation as done in [8]. .

A few examples follow. If we load three digits all equal to 5 we get a decimal carry, the register S containing 5 and the register ***C*** a zero.

If we load A1=6, A2=5 and A3=4 we get a zero carry, ***C***=8 and S=7. The whole sum, 15, is therefore stored in the registers S and ***C***. This behaviour does not invalidate the compressor, since those values, transferred to another compressor in a column of compressors will eventually generate one or more carries. The final result of a network of compressors will be composed by couples of digits, to be added in a carry-propagate decimal adder (of a carry-look-ahead type for speed reason).

This will be seen in the schemes of the next chapter.

We will use in some cases a property of the compression cell concerning the addition of a BCD-4221 digit valued 8 or 9, with one or two digits valued 1. In such cases no decimal carry will be generated. If a digit 9 is added to one smaller digit, no carry will be generated if the input Sum is smaller that 14.

Note finally that the compressors proposed in [6], valid for 8421-BCD coding, obtain the sum of the column to be compressed (composed by 2 or more digits) and give the value of the digit in the same column and the total value of the carries in the next column(s).

3 – Schemes of multi-operand decimal adders

Fig. 3 shows some schemes for multi-operand adders composed from the just described compressor, represented in compact form, for different operand number *N* of 8 digit each. We briefly describe the schemes in Fig. 3.

***N*=3:** each input decimal column is composed with three digits. A single cell in each column obtains a two-lines equivalent set of numbers, with a carry into the column 10^8, i.e. an *overflow*. A parallel decimal adder (not shown in the figure) will obtain the final sum (9 digit long).

N=4: in the first stage we input the first three addends to a linear array of 8 cells. In the second stage we add the fourth addend to the two outputs from the first stage. We will discuss later the case of overflows.

N=8: we have 4 stages. In the first one, we obtain 4 outputs from 6 inputs, leaving unchanged 2 of the input addends. In the second stage we associate these addends to the four outputs from the first stage, obtaining 4 outputs equivalent to the 8 input addends. Those 4 numbers are then reduced to two via two more stages (as in the N=4 case).

N=16: we reduce in the first stage 15 addends to 10 numbers, via a set of 5 linear arrays of compressors, leaving unchanged the 16th addend. In the second stage we have 11 numbers, to be operated by 3 linear arrays leaving unchanged 2 numbers (2*3+2=8). We obtain in the third stage 2*2+2=6 numbers. These will be processed as in the case N=8. The number of stages for obtaining two equivalent numbers will be 6.

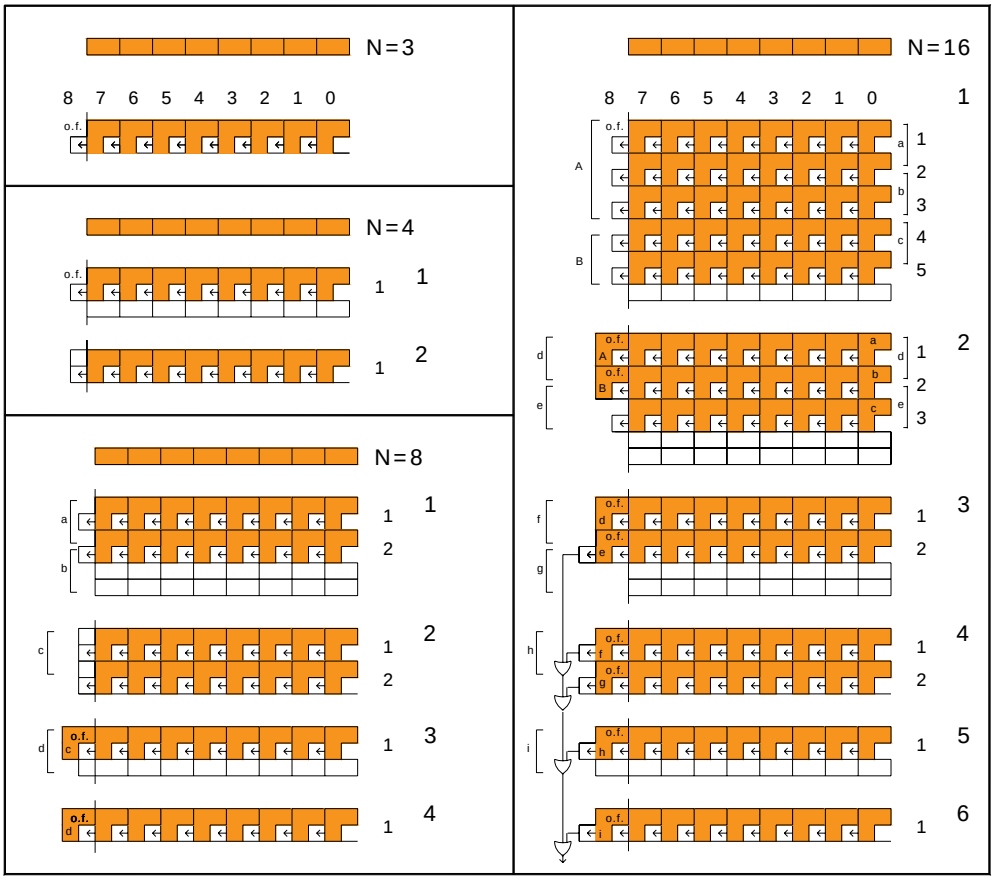

Fig. 3: Schemes for multi operand decimal adders, composed from compressors in compact notation (Fig. 3.3), for N=3, N=4, N=8 and N=16 operands of 8 digit.

3.1 Processing the overflows

In some applications we can assume that the total Sum doesn't exceed the length in digit allotted for each addend. In general, however, this will not be true. In the worst case we allow each addend to reach the maximum value of all digits equal to 9. In such a case there will be in all stages the generation of *c-out*, i.e. of overflows (o.f.). Multi-operand adders previously proposed [4,5,6] don't consider explicitly this problem since their architectures include its solution. This is not the case in the scheme described in this paper.

The task of the o.f. column(s) is to count the decimal carries generated from the last, adjacent column of the adder (whose tasks are: the addition of the input digits, the addition of the

decimal carries generated from the column at its right, the generation of the decimal carries to be added in its adjacent column at its left).

We will show a solution based on the same compression cells used for all the other columns.

Note first that the addition of N numbers composed from $n>1$ digits equal to 9, the number n_c of the decimal carries is: $n_c = N-1$. This means that for $N \leq 10$ a single o.f. column will be needed, while for N ranging from 11 to 99 two columns are needed.

A very simple (but excessively expensive) solution is to provide one additional column in the cases of Fig. 3, $N=3$, $N=4$ and $N=8$, and two columns in Fig.3, $N=16$ case. It can easily be seen that a number of compression units can be removed without affecting the operation of the circuit. This has been done in Fig. 4 schemes.

N=3: By removing the compressor in column 8 leaves there the ***c-out*** from column 7. Nothing else is needed.

N=4: The removal of the o.f. compressor from stage 1 leaves in column 8 in the same stage a single bit. A compressor unit in stage 2 would obtain the result of transferring such bit on top if the ***c-out*** bit from column 7. Such compressor is therefore useless, being possible to replace it with a simple transfer via a direct connection.

N=8: The removal of two cells in stage 1 leaves two bits in column 8: the same bits can be transferred in stage 2, aligned with the two c-out bits generated in the same stage. The top three bits in column 8 can be represented by the two bits of a full adder in stage 3, while the fourth bit is transferred to stage 3, where a new ***c-out*** bit is also generated. The three lines of column 8 stage 3 are then input to a cell in stage 4.

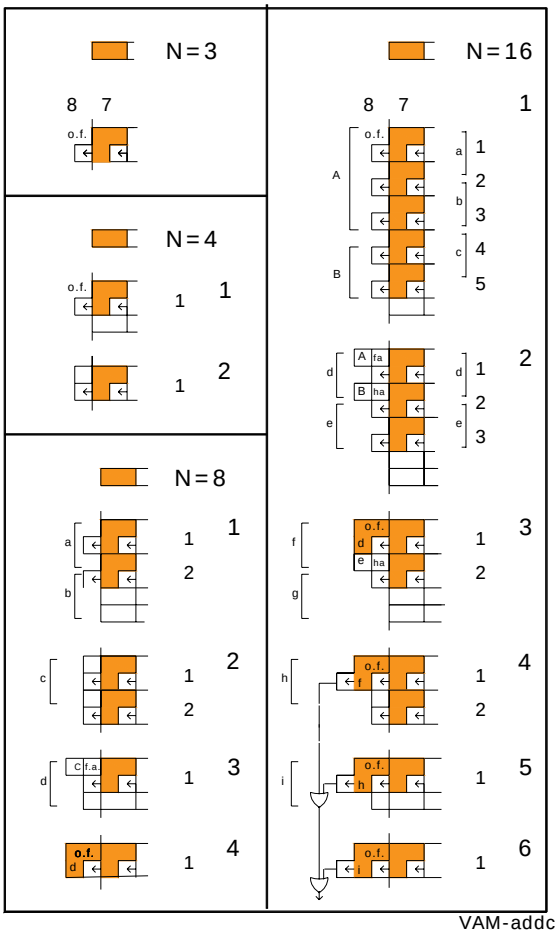

Fig. 4: Schemes of Fig. 3 (most significant digits) with optimized overflows addition

N=16: In this case, we see a full and a half adder in stage 2, representing the content of the five ***c-out*** in stage 1. In stage 3 we see two compression cells in column 8. The first one is fed by the three lines marked as "d" in stage 2, while the second is fed only with the two bits marked as "e". Note that the second c-out in "d" is the 10^{th} decimal carry generated in column 8. It can then generate a first bit in column 9. This could happen also for the ***c-out*** generated in all the following stages. Note that no ***c-out*** is assumed to be generated in all the preceding stages. Since the final value in column 9 cannot be larger than 1, we can obtain the final value in column 9 with the circuit shown in the figure, composed by cascaded two-inputs OR gates.

4 -An area optimized scheme

The above described schemes use the column compressor of Fig. 1.a) in which a set of four full adders (a 3:2 *binary compressor*) is always associated with a 10,4,2,2 *decoder*. It has been stated in [7] that a saving in area can be obtained if the compression process is split in two consecutive parts: in the first, the compression is performed using only binary compressors, the recoding being done in the second part using a network of decoders (called *x2*, equivalent to *dec2* of this paper) and binary compressors.

We will show here a modified scheme, in which the first part is identical to the scheme given in [7], while the second part is composed by decoders only. Note that the first part is such that no more binary compressors can be used. This first *stage* is followed by similar second, third, … stages, until a two-digit output (in a column with the same decimal weight of the input column) is obtained. Note also that the decoders used in all stages generate decimal carries. The value of the two decimal digits and of all the carries (generated from the decoders) must be obviously identical to the sum of the original decimal column plus the carries received from the decimal column (if any) at the right.

Coming now back to the binary compressors, we notice that the decimal carries cannot be generated by them, since each full adder composing it generates a binary sum and a binary carry within the same binary column. Their values can be affected only by changing their respective binary values (*0* or *1*) or their weights. It is important to note that the weights are not represented within a compressor. Their values depend on the weights of the inputs (that, again, are nor written in them). The weights must therefore be tagged to the box containing each compressor. In our case we will use two integers for each compressor, denoting the weight of the sum **S** and of the carry **C**, (which is twice the weight of **S).**

Only one of the two digits, **S** or **C**, needs in principle to be marked. We found useful, for practical reason, to mark both digits with their respective weights.

Each of the two parts of a binary compressor will be represented by two adjacent squares. The input to the compressor will be in one of the two extremes of the common side of the two squares, while each output could use one of the corners of the square in which the respective weight is written.

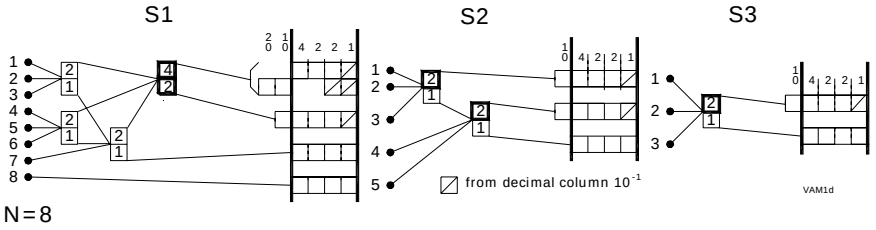

Fig. 5: Compressing an 8 digit column to 2 digit, with the generation of 7 decimal carries, obtained in three stages S1, S2 and S3, each composed with binary compressors and *dec2* and *dec4* decoders. Each square in **bold** represents a digit (with weight given by the digit written in the square) "merged" with the corresponding decoder (*dec2* or *dec4*). Fig. 6 represents the Compression Boxes for *N=16, 8, 7, 5, 4, 3.*

As an example consider, see Fig. 5, a column of 8 digits, to be added. This is done with the generation of 7 decimal carries obtained in three *stages* S1, S2 and S3, each composed with binary compressors and *dec2* and *dec4* decoders. In Fig. 5, each square in **bold** represents a digit (with weight given by the digit written in the square) and the corresponding decoder.

We feed three of the input digits to a 2:1 binary compressor, three more digits to another compressor. A third 2:1 compressor uses the 8^{th} input digit and two digit of same weight output from the two preceding compressors. The two digits weighed 2 from the first two compressors and a third digit weighed 2 from the third compressor feed a compressor tagged 4:2. One of its output is therefore a digit weighed 2 (the S digit) while the second digit (its C digit) is weighed 4. The decoder *dec2* (implicitly represented by the bold square) will feed three bits in the 4221 column and one bit weighed 10 in the "40,20,20,10" column. The second digit generated from the 4:2 compressor will feed a *dec4* decoder: this in turn will feed three bit in "4, 2, 2, 1" column and two bit in "40, 20, 20, 10" column.

As an example, we assume now for the input digits the value 9 $(1111)_{4221}$. The maximum output from the *dec4* decoder will be 4*9=36; from *dec2* it will be 2*9=18. The remaining two digits (see Fig.5) will give the value 18. In total we get 72, i.e. the input value.

Note that *dec4* will give a decimal carry valued 30, requiring two bits in the same decimal column weighed 10_{10}. The carry from *dec2* will be 10, placed in the same column. The structure of stages S2 and S3 is obvious. The number of inputs to S2 is assumed to be 5 since we consider that in decimal column of weight 10^0 we assume a carry from a 10^{-1} column.

Adding the four carries seen in Fig.5 we get a total of 7, to which we must add a further possible carry in the final decimal adder (not shown in the figure). This does not mean that the total generated carry will be 80. It will be instead 70 (with the assumed value of 9 for all the input digits). We must consider that, even in that case, the values in the digits in the stages S2 and S3 will be in the average smaller that 9. This can be verified by computing all those digits. For brevity we will not report here such results.

If we consider the summation of more than 8 digits we need to introduce compressors handling inputs of weight 8, 16, …

Note that in Fig. 5 each of the lines connecting the various components represents in general a number of wire connections. Those numbers can easily be found by inspection. Note also that the parts of the figures representing a decimal column (or a part of it) play the sole role of making easier for the reader to understand the operation of the implemented algorithm. More precisely, they permit to identify the origin and the destination of each connection between two cascaded stages.

We now consider the design of the *decoders*.

In TABLE A we see the truth tables of *dec2*, *dec4*, *dec8, dec16* and *dec32.* The rules for building such truth-tables are very simple. We start with the columns listing all the combinations of the four input variables in columns marked 4, 2', 2, 1. In column **x** we have the value of each combination, computed as the sum of the weight of bits valued 1. In column **2x** we place the double of the values listed in column **x.** In column **4x** we place the values **x** multiplied by 4. Similarly we write columns **8x**, **16x** and **32x.**

In the two decimal columns (weighed 40,20,20,10 and 4,2,2,1) following the **2x** column, we write the code of the most significant (1 bit)-digit, and the code of the least significant (3 bit) digit of the values **2x**. We do the same for the values **4x, 8x, 16x** and **32x.** In the latter two cases we need three decimal columns.

Note also that in the **8x** case we can choose the code of the most significant digit in such a way that only one of the column weighed 20 is needed, so reducing the number of functions required for the corresponding decoder. For a *dec2* four functions are required, for a *dec4* decoder five functions, six for a *dec8,* eight for *dec16* and nine for *dec32*.

The same method can be used, if necessary, for designing higher order decoders.

Data concerning area and time will be given in Chapter 7.

We consider worth noting that, due to the VAM cell architecture, the network of Binary Compressors composing a Compressor Box can be considered an overlapping of 4 identical layers. No communication exists between them within the box. The decoders generate 4 or more bits, each one being a function of 4 bits from 4 different full adders belonging to the different layers.

Inputs					Output Functions for *dec2, dec4, dec8, dec16, dec32* decoders																				
x	4	2	2	1	2x	10	4	2	2	1	4x	20	10	4	2	2	1	8x	40	20	20	10	4	2	2	1
0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0		0	0		0	0	0	0	0	
1	0	0	0	1	2	0	0	1	0		4	0	0	1	0	0		8	0		0	0	1	1	1	
2	0	0	1	0	4	0	1	0	0		8	0	0	1	1	1		16	0		0	1	1	1	0	
2	0	1	0	0	4	0	1	0	0		8	0	0	1	1	1		16	0		0	1	1	1	0	
3	0	0	1	1	6	0	1	1	0		12	0	1	0	1	0		24	0		1	0	1	0	0	
3	0	1	0	1	6	0	1	1	0		12	0	1	0	1	0		24	0		1	0	1	0	0	
4	0	1	1	0	8	0	1	1	1		16	0	1	1	1	0		32	0		1	1	0	1	0	
4	1	0	0	0	8	0	1	1	1		16	0	1	1	1	0		32	0		1	1	0	1	0	
5	1	0	0	1	10	1	0	0	0		20	1	0	0	0	0		40	1		0	0	0	0	0	
5	0	1	1	1	10	1	0	0	0		20	1	0	0	0	0		40	1		0	0	0	0	0	
6	1	1	0	0	12	1	0	0	1		24	1	0	1	0	0		48	1		0	0	1	1	1	
6	1	0	1	0	12	1	0	0	1		24	1	0	1	0	0		48	1		0	0	1	1	1	
7	1	1	0	1	14	1	1	0	0		28	1	0	1	1	1		56	1		0	1	1	1	0	
7	1	0	1	1	14	1	1	0	0		28	1	0	1	1	1		56	1		0	1	1	1	0	
8	1	1	1	0	16	1	1	1	0		32	1	1	0	1	0		64	1		1	0	1	0	0	
9	1	1	1	1	18	1	1	1	1		36	1	1	1	1	0		72	1		1	1	0	1	0	

x	4	2	2	1	16x	100	40	20	20	10	4	2	2	1	32x	200	100	40	20	20	10	4	2	2	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	16	0	0	0	0	1	1	1	0		32	0	0	0	0	1	1	0	0	1	
2	0	0	1	0	32	0	0	0	1	1	0	1	0		64	0	0	1	0	1	0	1	0	0	
2	0	1	0	0	32	0	0	0	1	1	0	1	0		64	0	0	1	0	1	0	1	0	0	
3	0	0	1	1	48	0	1	0	0	0	1	0	0		96	0	0	1	1	1	1	1	0	1	
3	0	1	0	1	48	0	1	0	0	0	1	0	0		96	0	0	1	1	1	1	1	0	1	
4	0	1	1	0	64	0	1	1	0	0	1	0	0		128	0	1	0	0	1	0	1	1	1	
4	1	0	0	0	64	0	1	1	0	0	1	0	0		128	0	1	0	0	1	0	1	1	1	
5	1	0	0	1	80	0	1	1	1	0	0	0	0		160	0	1	1	0	1	0	0	0	0	
5	0	1	1	1	80	0	1	1	1	0	0	0	0		160	0	1	1	0	1	0	0	0	0	
6	1	1	0	0	96	0	1	1	1	1	1	1	0		192	0	1	1	1	1	1	0	0	1	
6	1	0	1	0	96	0	1	1	1	1	1	1	0		192	0	1	1	1	1	1	0	0	1	
7	1	1	0	1	112	1	0	0	0	1	0	1	0		224	1	0	0	0	1	0	1	0	0	
7	1	0	1	1	112	1	0	0	0	1	0	1	0		224	1	0	0	0	1	0	1	0	0	
8	1	1	1	0	128	1	0	1	0	0	1	1	1		256	1	0	1	0	0	1	1	0	1	
9	1	1	1	1	144	1	1	0	0	0	1	0	0		288	1	0	1	0	0	1	1	0	1	

TABLE A: Output functions for the *dec2, dec4, dec8, dec16* and *dec32* decoders.

We define a *Compression Box* as the combinational network implementing a stage. It transforms a set of decimal digits of same weight into a smaller set of decimal digits of suitable weights. More precisely, assuming the input's digits weight = 1, the sequence of the output digits is composed by.

d_1 = 1 or 2 digits of weight =1 (not requiring decoder)
d_2 = 1 or 2 digits of weight =2 (requiring 1 or 2 *dec2*)
d_4 = 1 or 2 digits of weight =4 (requiring 1 or 2 *dec4*)
d_8 = 1 or 2 digits of weight =8 (requiring 1 or 2 *dec8*)
d_{16} = 1 or 2 digits of weight =16 (requiring 1 or 2 *dec16*)
……

It can be shown that an assigned set of output's digits represents the number N of input's digits:

$$N = d_1 + 2d_2 + 4d_4 + 8d_8 + 16d_{16} + \ldots.$$

Consequently, the set $\{d_1, d_2, d_4, d_8, d_{16}, \ldots\}$ of the *indices* d_k can be assumed for representing a Compression Box . In Fig. 7 we show the Compression Boxes later used in Fig. 8, 9 and 10.

Each compression box can be represented as in Fig. 7 by a rectangle where in the upper right corner we write the number of input digits, N, and at the left side we write the sequence of the indexes of the decoders outputs.

The *design of a Compression Box* for an assigned N can be done with the following algorithm.

Determine the integer quotient $q_{1,0}= \lfloor N/3 \rfloor$: This gives the number of (2:1) Binary Compressors: $q_{1,0}$ is the number of Binary Compressors weighed (2;1), each composed from a digit weighed 2 and of a digit weighed 1).
Compute $m_{1,0}=MOD_{1i}(N;3)$; $m_{1,0}$ smaller than 3,is the remainder of the division. It is the number of un-processed input digits.
Numbers $q_{1,i}$ and $m_{1,i}$ (i=0,1,2,3,…) compose the rows q_1 and m_1 in Fig.6 scheme.
The sum of the output of the above transformation, $q_{1,0} + m_{1,0,}$ is (taking care of the weights) equivalent to the sum of the inputs.

The total number of weighed 1 digits generated in this step is $D1_{1,i}=q_{1,i}+ m_{1,i}$.
The total number of weighed 2 digits generated in this step is $D2_{1,i}=q_{1,i}$..

The above simple algorithm is the basic step to be repeatedly used for obtaining the complete Compressor Box. If we apply it to the weighed 1 digits, we will reach a situation in which the result will be 1 or 2: it is the d_1 value defined previously. If we apply it to the digits weighed 2 we will obtain the value of d_2 seen before.

Note that the algorithm will obtain also digits weighed 4, 8, 16, …. depending on the initial N those digits will be, beyond a certain index, all zeros.

The above algorithm has been implemented with a spreadsheet program, i.e. a tool for the automatic design of Compressor Boxes [8]. Fig. 6 represents the result of such program for the case $N=16$.

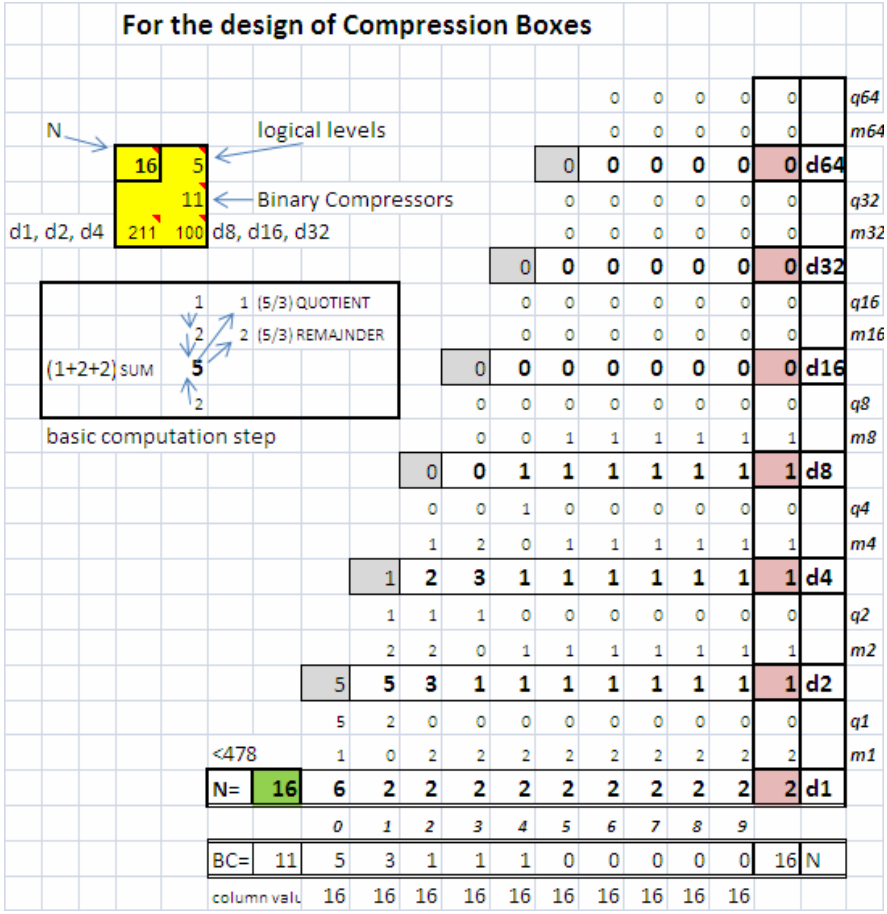

Fig.6: Results of the spreadsheet program for designing the Compression Box for $N=16$.

In the top-left corner we see the synthetic representation of the box, with the main parameters: the number N of the input digits, the number of logical levels, the number of the Binary Compressors, the set {d1, d2, d4, d8, d16, d32} previously described. The rectangle below the top-left corner in the figure, depicts an example of the basic computation step, with reference to the

main array containing the whole computation. The row marked BC at the bottom represents the BCs generated in each column (i.e. in each logical level). It is used also for computing *ll*, the number of the logical levels. The following last row in the figure, marked "columns values" represents the values computed in each column: all the results are equal the input *N*, confirming the invariance of the transformation.

Note that the drawing of the actual scheme is not fully automatic: it must be done by hand, using the data in the array: the number of BCs composing each level (represented by a column) and the fact that such numbers are certainly compatible in the sense that the outputs generated by a column find compatible inputs in the adjacent column. Note also that the remainders represent connections across the column in which has been generated.

It was noted that the rows representing variables $q_{i,j}$ represent both digits of a Binary Compressors. They are considered as representing the smaller weighed value whenever are added to the underlying $m_{i,j}$ remainder, while they represent the higher weight when added to the variables in the same column to generate the Sum represented in bold.

Note finally that the spreadsheet instruction for generating the integer quotient is:

$q_ij=ROUNDDOWN(n/3;0)$; for generating a remainder we write: $m_{i,j}=MOD(n;3)$.

4.1 Designing multi-operand adders

Using the decoders of TABLE A and the Compression Boxes (CB) of Fig. 7, we have designed multi-operand adders for *N=4, N=8* and *N=16,* four digit long, represented in Fig. 8 for *N=4*, Fig.9, for *N=8* and Fig.10 for *N=16*.

Fig. 8, 9 and 10 have been drawn according to the following criteria.

The first (top) CB of each column is fed by the digits to be added and generates at its left side the bits generated within the CB by the decoders, to be transmitted to the 4bits registers belonging to the column of the input digits, or (for the carries) to registers belonging to the next to the left column(s).

In these schemes we abstract from the internal operation of the Compression Boxes, concentrating our attention to the relation between the Compression Boxes, implemented exclusively through wiring.

A column compression drawn as in Fig. 5 gives a clear representation of the processing flow along the column. It suffers, nevertheless, from a drawback: both the input's column and the carry-column are represented repeatedly in each stage. A better representation is obtained if only the input's column is represented, assuming that the column in each column scheme is shared with the preceding column scheme, i.e. it can accept carries from the preceding column.

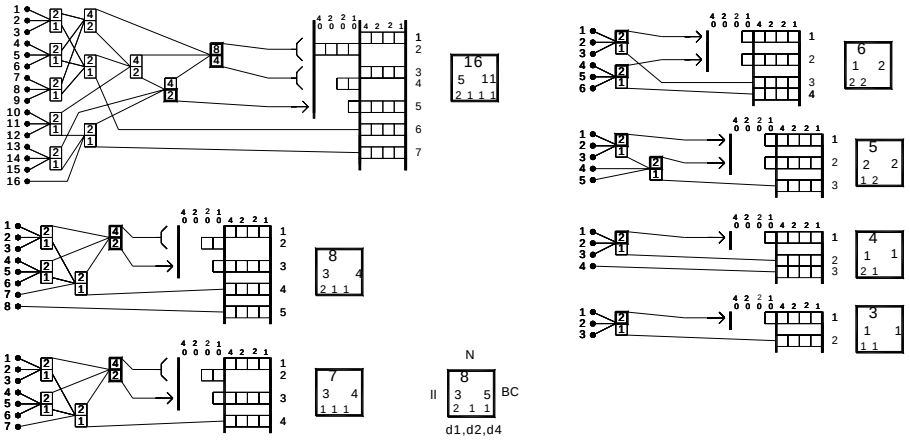

Fig.7: The Compression Boxes used in Fig.8, Fig. 9 and Fig.10

The arrows used in Fig. 8, 9 and 10 permit to determining clearly the origin of each binary variable, with the following rules:

- A line originating from a d_1 output (no decoder) is composed from 4 variables, all directed to the same input's column.
- A line originated from a d_2 (from a *dec2*) output is composed from 4 variables, three directed at the three most significant bits of the input's column, one directed to the least significant bit in the left column

Fig. 8 scheme ($N = 4$) uses only the above d_1 and d_2 outputs. Note that in each column the same digit can host the three most significant variables generated in a column and the most significant bit generated as a carry from the column at the right.

In Fig. 9 and Fig. 10 we find also outputs d_4 (from *dec4*):

- A line originating from a d_4 output is composed from 5 variables, three directed at the three most significant bits of the input's column, two directed to the two least significant bits in the next left column. Note that it is not possible to host the three least significant variables and the two most significant ones in the same digit: see Fig. 9 and 10 schemes.
- A line originating from a d_8 output is composed from 7 variables, three directed at the three most significant bits of the input's column, three directed to the next, left column as carries. One of the three variables is directed to the most significant bit in the left column, another to the least significant bit of the same left column, another to one of the two bits of the left column having the same weight (e.g. 20). See the Fig. 10 scheme.

It is worth noting that while in the Fig. 8 scheme ($N = 4$) all columns have the same composition; this is not true for Fig. 9 ($N = 8$) and Fig. 10 ($N = 16$). In Fig. 9 all columns are the same except the last rightmost one, in Fig. 10 the two rightmost columns differ among them and are different from the other columns. The least significant columns of each scheme is considerably simpler than those of the successive columns, due to the lack of carries from preceding columns.

Note that we must add to the four input's column one or more columns for computing the overflows, as done for the case of the standard schemes. Note also that the two final lines in all schemes are obtained as the final step of the reduction algorithm used for obtaining the preceding steps.

The schemes have been important for determining their respective complexity and behaviour, with the evaluation of area and timing data. Such data have been used for comparing the two types of schemes shown in this paper and other schemes published previously [4, 5, 6].

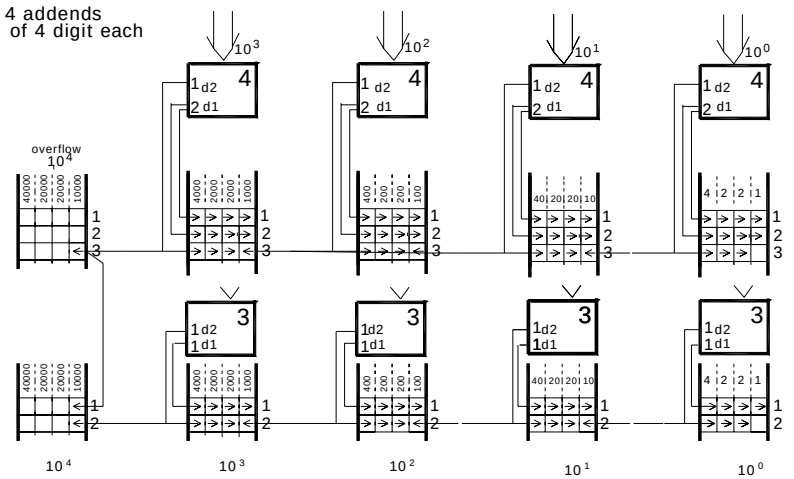

Fig. 8: The adder of N=4 four digit numbers

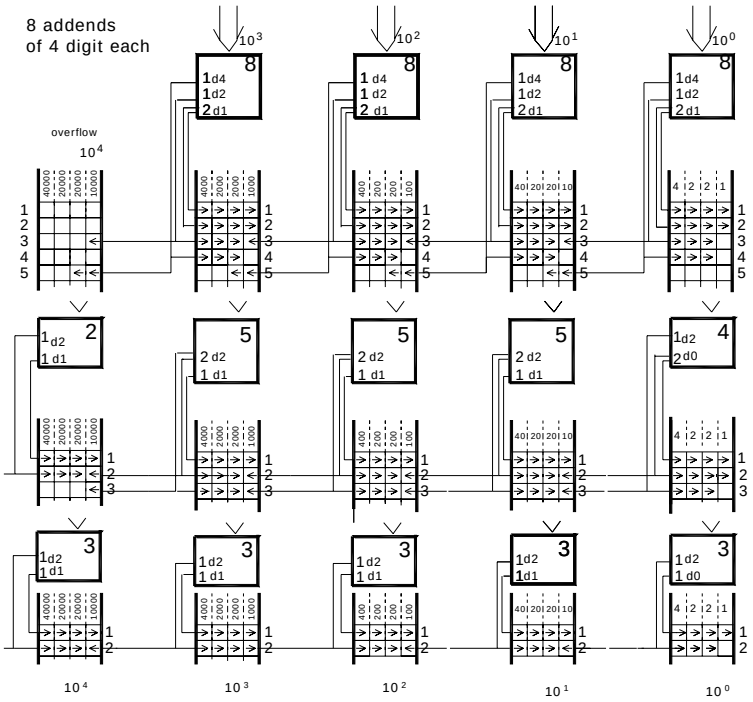

Fig. 9: The adder of N=8 four digit numbers

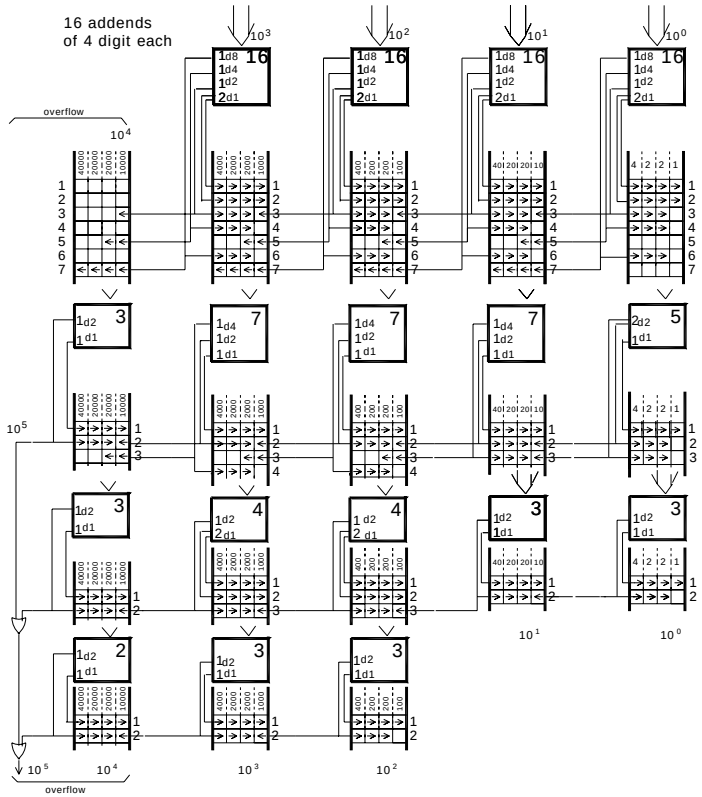

Fig. 10: The adder of N=16 four digit numbers

For greater N we will need to adopt decoders whose carries outputs must be placed in two columns at the left of the input's column. This is shown in TABLE A for *dec16* and *dec32* truth tables.

5 –Schemes using Binary Compressors and x2 (dec2) only decoder

In the schemes of adders just discussed we adopt decoders that use digit multiples with multiplicity factors m = 2, 4, 8, 16, 32. It has been found that such decoders have a complexity moderately increasing with m.

The Adder scheme proposed in [7] uses one type only of decoder, namely *x2 (dec2).* We are going now to show how such schemes (not fully treated in [7][1]) can be developed, in order to compare them with the schemes using all decoders shown in the preceding chapter. The basic arrangement (shown in [7]) consists in cascading 2 or more *x2* decoders: precisely, given the multiplicity m of a digit, the number of *x2* needed being equal to $log_2 m$.

Each of the outputs of a Compression Box will therefore feed $log_2 m$ cascaded *x2*. Note that the digit issued from the last decoder of the cascade is characterized by a multiplicity factor $m=1$. In Fig. 11 schemes each *x2* unit is represented with a small square. The input digit is applied to the (top) corner. The output digit appears at the opposite (bottom) corner. The ***c-in*** bit is applied to the right corner, the ***c-out*** bit appears at the opposite, left corner.

It is important to note that Compression Boxes in Fig. 11 schemes do not include any decoder. Decoding is done on their output, consisting in BCD-4221 digits b_j associated to multiplicity factors j (2, 4, 8, 16, ..). In outputs b_1 no decoder is needed. In b_2 a single dec2 decoder is needed, 2 of them for b_4 outputs, 3 for b_8, 4 for b_{16}, ..., as in Fig.11 cases. All columns are identical, except those having the task of computing the overflows digits.

Note that the **c-out** binary outputs feed the corresponding dec2 in next column. This could suggest that a propagation occurs in such connecting lines, but it is not true. In effect, see Fig. 1, a **c-in** bit is associated with the three most significant bits composing the C digit (characterized by a weight (or multiplicity factor) and sent downward to the next decoder if any, or to the next Compression Box.

The carries generated from the last N^{th} column represent the total overflow. In performing the overflow calculation we must take into account the binary value of each carry. Representing each **c-out** with a digit and adding them for obtaining the total overflow would be certainly correct but also expensive. A better solution can be obtained by associating more carries in a number of BCD-4221 digits. This has been done in Fig.11, taking into account the binary value of each carry.

Those generated by the last (at the bottom of the cascaded decoders) have weight "1", the next (moving upward) decoders generate bit of weight 2, the next generate a bit weighed 4, and so on. In the case N=16 of Fig.10 we reach the maximum weigh equal to 8 (from the first decoder in the d_{16} output).

[1] In paper [7] the case is illustrated by figure 6a, which seems to be drawn with the following criteria: first use a number of Binary Compressors. At a certain point (it isn't said at which point the first phase is finished) start using VAM cells (each one of them generating a decimal carry-bit) combining them with BCs, until two-terms only are generated. This appears an interesting, very flexible approach. The problem of treating the carries from the preceding column does not seem to be explicitly considered.

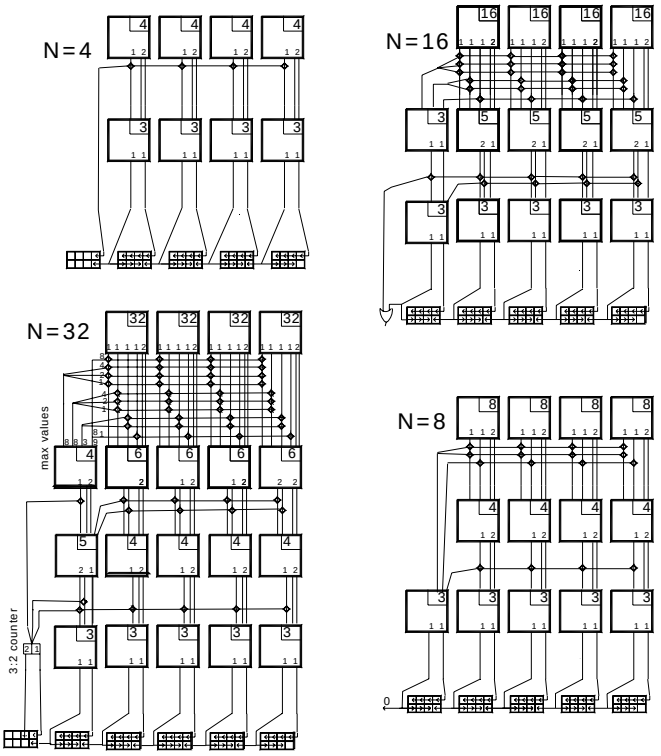

Fig.11 Adder schemes using decoders of type dec2 (or x2).

6 – Optimisation

In the development of the schemes so far shown we had the opportunity to identify and to apply a few methods intended to improve their operation. We show in the following the ones that seem the most efficient and of general applicability.

The ***carry-merging*** takes advantage of a general property of the carries generated from all types of decoders illustrated in Section 3: it was shown that three least significant bits belong to the same decimal column of the compressor of a VAM cell; 1-bit carry is generated from dec2 decoders, 2-bit carry from dec4, 3-bit carry from dec8, 4-bit carry from dec16. We show in Fig. 8, 9 and 10 how those carries are treated. We see in particular that the 1-bit carry from dec2 can be hosted in the 3 bit digits generated by dec2 in the next column.

In case of dec4 and dec8 their 2bit or 3bit carries require a specific digit to be allotted in the left column, while for dec16 the most significant carry bit must be placed in the second left column. In case of dec16 present in all columns, the intermediate 3 carry bits can be placed in a second row, while the most significant carry bit can be hosted with the 3 bit generated in the same column by the respective dec16 decoder. This means that for dec16 two rows will suffice for hosting all the carry bits generated by the dec16 decoders present in each column. Also for dec4 and dec8 we have to provide two rows each.

However, if a dec8 is also present, the 2bits carry of a dec4 can be hosted: the least significant 10^1 weighed carry bit with the 3bit digit generated from dec4 or dec8 (and also from dec2 if still available), while the $2*10^1$ bit can be hosted from the equal weight bit in the dec8, 3 bit carry digit. Fig.12 shows an example taken from Fig. 10.

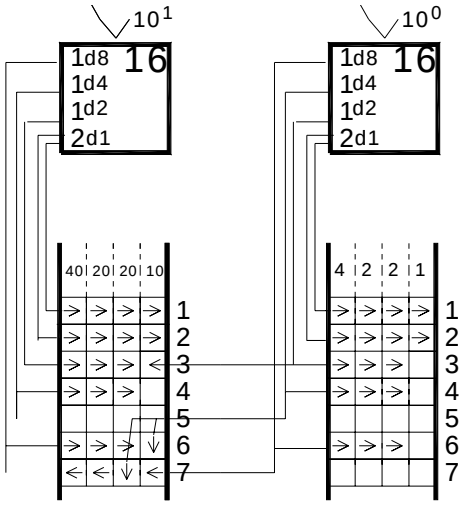

Fig.12: Showing how a 2bit carry from a dec4 in column 10^0 can be stored in column 10^1 in line 6 (from a dec8 in same column) and in line 7 as a $2*10^1$ bit within a 3bit carry from a dec8 in column 10^0.

The ***Logical levels minimization*** is a scope of any procedure aiming at reducing the delay in Compression Boxes.

The design of a Compression Box for a prescribed number *N* of BCD-4221 digits of same weight can be performed according to the algorithm described in Chapter 3. The scheme is composed from a number of Binary Compressors (each composed from four full adders). The output is a set of *n* digits with different weights, more precisely *d1* digits of weight 1, *d2* digits of weight 2, etc.:

$N = 2^0*d1+2^1*d2+.....$

$dj=\{1,2\}$.

In the following Table B we have listed the main parameters of Compression Boxes for *N=2* to *N=39*. Those parameters include the non-zero *dj*, the number *BC* of the Binary Compressors, the number *ll* of logical levels, the number *n* of the outputs digits (the sums of the *dj*). All those parameters have been computed automatically as shown previously (fig.6).

For a number of cases a second lines of parameters is given: the delays of the *n* outputs. When *dj=2* the greater delay value must be used in the evaluation of the critical path delay. Note also that the delay in *d1* is not relevant since in that output, weighted 1, no decoding is needed. Looking at the maximum delays (and to the related logical levels), we notice that these parameters do not increase monotonically with N.

We have pinpointed the case of:

N=6 (ll=1) following N=5 (ll=2)
N=9 (ll=2) following N=7 and 8 (ll=3)
N=12 (ll=3) following N=11 (ll=4)
N=18 (ll=4) following N=15, 16 and 17 (ll=5)
N=27 (ll=4) following N=23 to 26 (ll=5)
N=34 (ll=6) following N=31, 32 and 33 (ll=7)

We call these cases "(naturally) minimal". It seems worth noting that their respective schemes appear more regular than their neighbours: look, as an example, at the schemes for N=6 and N=5 in Fig.7.

In order to take advantage of the minimal cases we implement them and use for some of the cases preceding each of them. For example we could replace cases N=33, N=32, N=31 and N=30 with the naturally minimal N=34, feeding with zeros some (1, or 2, or 3 or 4) input variables, correspondingly. It is however more convenient to remove the corresponding inputs (and the attached circuitry) in order to save area. More precisely after the removal of an input variable (from

the triplets of variables feeding a Binary Compressor) we replace all the four full adders of a Binary Compressor with half adders. The rest of the circuitry can be left as it is.

The simplest rule is to remove only one variable from a triplet. We could decide to remove two or three variables from a triplet (in the latter case the corresponding BC would be entirely removed). We will not consider here for brevity the choice of the optimal strategy.

Table B Parameters of Compression Boxes

N	d1	d2	d4	d8	d16	BC	ll	n
2	2					0	1	2
3	1	1				1	1	2
	2	1					2	
4	2	1				1	1	3
	0,2	1					2	
5	1	2				2	2	3
	4	3,1					2	
6	2	2				2	1	4
	2,2	1,1					2	
7	1	1	1			4	3	3
	4	5	4				1.33	
8	2	1	1			4	3	4
	0,4	5	4				1.67	
9	1	2	1			5	2	4
	4	3,3	2				2	
10	2	2	1			5	2	5
	0,4	3,3	2				2	
11	1	1	2			7	4	4
	6	7	6,2				1.75	
12	2	1	2			7	3	5
	2,4	5	4,2				1.67	
13	1	2	2			8	3	5
	6	5,5	4,2				2	
14	2	2	2			8	3	6
	4,4	3,5	4,2				1.67	
15	1	1	1	1		11	5	4
	6	7	8	7			1.6	
16	2	1	1	1		11	5	5
	4,4	7	8	7			1.6	
17	1	2	1	1		12	5	5
	2	7,5	6	5			1.4	
18	2	2	1	1		12	4	6
	4,4	3,5	6	5			1.5	
19	1	1	2	1		14	4	5
	6	6,7	6	5				
20	2	1	2	1		14	4	6
	0,6	7	6,6	5			1.75	

N	d1	d2	d4	d8	d16	BC	ll	n
21	1	2	2	1		15	4	6
	6	1,7	6,6	5				
22	2	2	2	1		15	4	7
	0,6	1,7	6,6	5			1.5	
23	1	1	1	2		18	5	5
	6	7	8	5,7				
24	2	1	1	2		18	5	6
	2,6	7	8	3,7			1.6	
25	1	2	1	2		19	5	6
	6	5,7	8	5,7				
26	2	2	1	2		19	5	7
	0,6	5,7	8	5,7			1.6	
27	1	1	2	2		21	4	6
	6	7	6	5,3				
28	2	1	2	2		21	4	7
	0,6	7	6,6	5,3			1.75	
29	1	2	2	2		22	4	7
	7	6,7	6,6	3,5				
30	2	2	2	2		22	4	8
	2,6	1,7	6,6	5,3			1.75	
31	1	1	1	1	1	26	7	5
	8	9	10	11	10		1.57	
32	2	1	1	1	1	26	7	6
	2,6	9	10	11	10		1.57	
33	1	2	1	1	1	27	7	6
	8	7,9	10	11	**10**			
34	2	2	1	1	1	27	6	7
	6,4	7,5	8	9	8			
35	1	1	2	1	1	29	6	6
36	2	1	2	1	1	29	6	7
37	1	2	2	1	1	30	6	7
38	2	2	2	1	1	30	6	8
39	1	1	1	2	1	31	6	6

VAMtables1

N: input n° BC: B**inary Comprssors** **ll logical layers** **n: output n°**

d1: output n° we9ghed 1 **d2: output n° weighed 2** **d4: output n° weighed 4**

d8: output n° weighed 8 **d16: output n° weighed 16**

DELAYS: in the following row. 1 for a carry, 2 for the sum from a Full Adder

In col. ll: the ratio max.Delay / ll (max=2) naturally minimal CB

TABLE B: Parameters of Compression Boxes.

It is then convenient to call the new CBs with their original number-name followed “f”, so that N=16 CB will be replaced with N=16f CB.

In TABLE C we show the non-minimal cases N=7, 8, 11, 16 and 32, followed by the parameters of the new "faster" schemes marked 7f, 8f, 11f, 16f and 32f. We notice that the corresponding max delay (excluding *d1*) is 2 units below the original ones.

The *BC, ll* and *n* values have been corrected accordingly

Table C: Fast Compression Boxes parameters

N	d1	d2	d4	d8	d16	BC	ll	n
7	1	1	1			4	3	3
	4	*5*	*4*					
7f	1	2	1			4	2	4
	4	*3,3*	*2*					
8	2	1	1			4	3	4
	0,4	*5*	*4*					
8f	1	2	1			4.5	2	4
	4	*3,3*	*2*					
11	1	1	2			7	4	4
	6	*7*	*6,2*					
11f	2	1	2			6.5	3	5
	2,4	*5*	*4,2*					
16	2	1	1	1		11	5	5
	4,4	*7*	*8*	*7*				
16f	2	2	1	1		11	4	6
	4,4	*3,5*	*6*	*5*				
32	2	1	1	1	1	26	7	6
	2,6	*9*	*10*	*11*	*10*			
32f	2	2	1	1	1	26	6	7
	6,4	*7,5*	*8*	*9*	*8*			

VAMtables1

TABLE C: Parameters of standard Compression Boxes for N=7, 8, 11, 16, 32 and of the corresponding fast versions N=7f, 8f, 11f , 16f, 32f.

The inputs removal reduces also the delays of all the paths including it. The delays of the outputs will not be affected if we remove just few inputs, as in our examples.

The method is however characterized by an increase of the number of the outputs, requiring therefore more inputs for the next compression box in the considered column and consequently a greater number of stages and a larger total column delay. This problem has to be carefully considered, checking the effects of each change.

The experience shows that in several practical cases the method leads to acceptable solutions. A case in which a difficulty might arise (not included in TABLE C) is N=6. It is faster then N=5 (the logical levels are respectively 1 and 2). N=6 gives 4 outputs, while N=5 gives 3 outputs. Assuming as a faster Box for N=5 the scheme of N=6 will certainly obtains a smaller delay, but also it would require an additional stage for reducing the four outputs to three. This would offset the delay reduction achieved by the N=6 box.

In drawing the new faster schemes we have followed the same rules used for the standard schemes using also very few Binary Compressors composed from Half Adders. The parameters given in TABLE B for N=2 do not include any delay, since a column of 2 digit doesn't require any processing. When a half adder is used in a Binary Compressor it must, instead, operate and must be represented as a full adder in TABLE B: with a Sum and a Carry outputs characterized by delays equal to 1 and to 0.15 time-units, respectively.

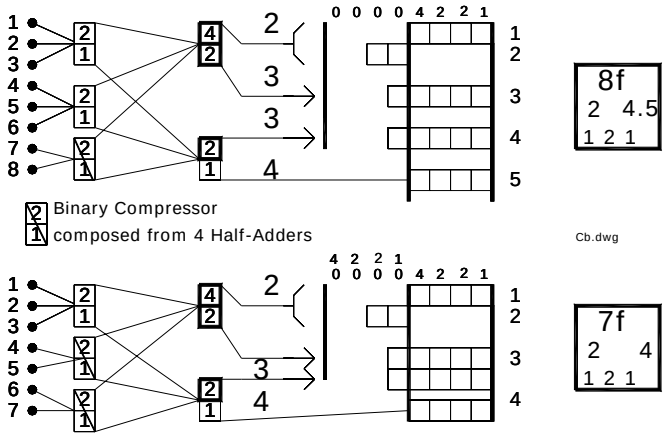

Fig.13: Fast schemes for N=7 and N=8 (see in Fig.6 the corresponding standard versions)

Note also that the outputs in scheme N=8f are composed from *{d1*, *d2*, *d4}={1, 2, 1}*, while from the standard N=8 scheme (Fig.6) they are: *{d1*, *d2*, *d4}={2,1,1}* . This requires an additional *dec2*, besides and additional "half Adder" binary Compressor, see fig.13.

In the fast N=7f scheme the outputs are composed as in the N=8f scheme, while in the N=7 standard scheme the output is composed as *{d1, d2, d4}={1, 1, 1}*. The increase in the number of digits output from a N=7f compressor must be carefully considered in an adder design, since it could entail the need for an additional reduction stage.

The outputs from the N=8f and N=7f schemes are identical to those of a standard N=9 scheme, also in their delays. The generating networks are nevertheless (slightly) different. The N=9 standard scheme is composed exclusively of full adders, while N=8f and N=7f include 1 and 2, respectively, Binary Compressors made of half adders.

An example

Let us consider the scheme of Fig. 10, i.e. an Adder for N=16. It has been drawn using the simplest procedure: starting from the first, less significant input column of 16 digits, then choosing the N=16 Compression Box of Table B, placing it as in Fig.10; and then tracing the connection scheme and the various output of the CB. The following CB in the same column is chosen after counting the digits generated by the first CB, i.e. 5. In the second column we do the same, taking into account also the carries generated by the first column.

In Fig. 15 we show the scheme for N=16, drawn by choosing, when useful and possible, the fast version of the CBs. We start by choosing the fast version of the N=16 CB, since the N=16 CB previously chosen is not a "naturally minimal" CB: it is the 16f CB of Fig.12, offering a maximum delay of 6 units, instead of 8. The number of outputs in the same column, see Fig. 15, is 6 (instead of 5).

We place a CB for N=6 in sequence of the CB N=16f in the same column, followed by a N=4 CB and finally with the N= 3 CB to complete the first column.

We start then the second column using again a CB for N=16f. Their output will be merged with the carries generated by the first column, for a total of 7 digits. Note that two digit can be merged according to the Fig. 12 procedure.

Since the N=7 CB is not minimal, we decide to adopt the fast CB for N=7f, see Table C. We will find its output, merged with the carries, being 5. We are not going to replace it with the nearest minimal CB, N=6, since this would require two more additional CBs (N=4 and N=3) to complete the column, with a total of 5 CB with no advantages in time. The second column will then be completed with 2 more CBs: for N=5 followed with a N=3 CB as the last. Following the same rules we complete the Adder scheme as shown in Fig.14.

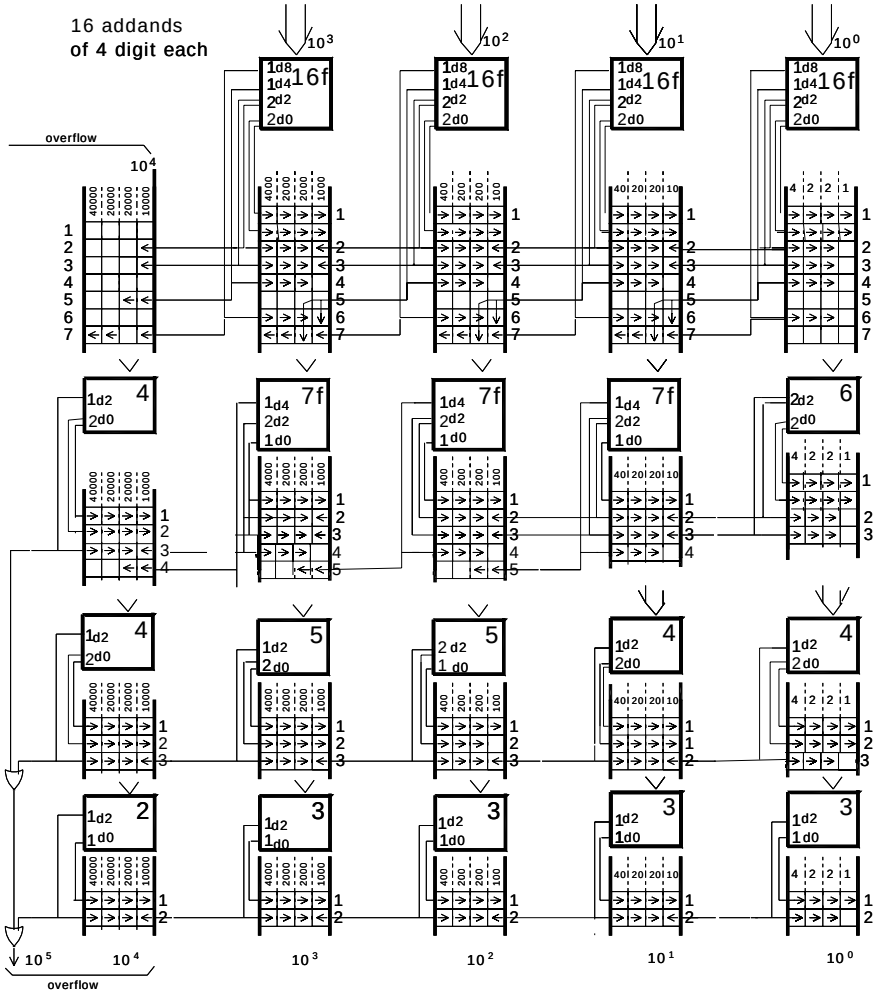

Fig.14: A faster adder of N=16 four digit numbers (a variant of Fig. 10 scheme)

We show in Fig. 15 the scheme for the computation of delay for the schemes of Fig. 9 (using non minimal CBs) and of Fig.14, just described. The scheme of the delay computation in Fig.15 is composed from the scheme in each stage (corresponding to Compression Boxes) and to the addition of the delay of all stages.

In each stage we find:

- In the first row the set of CB outputs (e.g.: 2,1,1,1)
- In second row the sequence of the corresponding delays, from Table B (e.g.:3,7,8,7) in units equal to 50ps
- In third row the delay in ps (the values of the preceding row multiplied by 50)
- In fourth row the sequence of the delays in the corresponding decoders (e.g. 200, 210, 200 ps). Note that *d1* has no decoder)
- In the fifth row the additions, in each column, of the two preceding rows, i.e. the total delay generated in the stage on all the decoders outputs.

In the rightmost cell of the last row we get the sum of the maximum values generated in each stage. The area is computed with a program using data in Table B, Table C and Table D, the final area values being reported in Fig.15 at the top-right corners.
We see that the total delay in Fig.10 scheme is 1660ps, while the delay in Fig.14 scheme is 1460ps.

The area of a Compression Box can be computed on the basis of data contained in Table B and in Table C: they give the *number of BC* and the number of each *decoders* (dec2, dec4, dec8,

dec16, dec32). Table C gives the numerical values of time and area data for a CMOS Standard Cell 90nm library [9]. The details of the library and the tools used to obtain the values of Table C are given in the next section.

	STAGE 1					STAGE 2				STAGE 3				STAGE 4			
canonical	**16**	**2**	**1**	**1**	**1**	**7**	**1**	**1**	**1**	**5**	**1**	**2**		**3**	**1**	**1**	8417
ADD		*3*	*7*	*8*	**7**		*4*	*5*	**4**		*4*	*3*			*2*	*1*	µm^2
Vam delay		150	350	400	350		200	250	200		200	150			100	50	
xj *delay*			200	210	200			200	210			200				200	ps
		150	550	610	550		200	450	410		200	350			100	250	**1660**
fast	**16f**	**2**	**2**	**1**	**1**	**7f**	**1**	**2**	**1**	**5**	**1**	**2**		**3**	**1**	**1**	9179
ADD		*4*	*5*	*6*	*5*		*4*	*3*	*2*		*4*	*3*			*2*	*1*	µm^2
VAM delay		200	250	300	250		200	150	100		200	150			100	50	
xj *delay*			200	210	200			200	210			200				200	ps
		200	450	510	450		200	350	310		200	350	0		100	250	**1460**

Fig.15: The delay and area computation scheme for Fig. 10 and Fig.14 adders

The fast solution gives a total delay of 1460 ps, 13% smaller than the standard one. The area of the fast method (9179 μm^2) is 9% higher than the standard scheme. The schemes for N=32 and N=32f are shown in the Appendix. The area of the fast scheme is 15509 μm^2, 6.4% higher than for the standard scheme. The delay of the fast scheme, 1600ps, is 12% smaller than in the standard scheme.

7 – Delay and Area: comparison of different schemes

In this section we evaluate delay and area for the schemes presented above. The evaluation is based on the characterization of the basic components: the Binary Compressor (composed of four full adders), and the decoders for digits of multiplicity (or weight) 2, 4, 8 or 16. The characterization is carried out by synthesizing the components with Synopsys Design Compiler for minimum delay. The library used is the STM 90 nm standard cell library [9]. The results are shown in TABLE D.

	BC	dec2	dec4	dec8	dec16	
Delay	100[2]	200	210	200	200	ps
Area	360	290	240	370	380	μm^2

TABLE D: Delay and area for building blocks

.

Once the basic components have been characterized, the delay and area of the different schemes are determined by a spread-sheet or any similar tool. This approach with respect to logical effort, used for example in [7], is easier to apply because it does not require to know what specific gates implement a logic function. Moreover, this method takes into account the synthesizer optimization strategies, such as grouping and buffering, that are not considered in logical effort.

We assume, as the simplest solution, that Binary Compressors and decoders are implemented separately: the decimal compressor, Fig. 1, can be obtained by connecting in cascade a binary compressor with a *dec2* decoder.

An important area reduction could be obtained by synthesizing in a single step the whole adder scheme. A good improvement can be reached by synthesizing suitable small groups of

[2] The fast path (carry-in to output) in the full-adder composing the BCs is 50 ps.

elementary components. In our case it can be shown that a convenient choice is to merge a decoder with the related binary compressor. We will adopt this choice.

The merging of a binary compressor (4 full adders) and a dec2 (x2) decoder shows a negligible effect on delay and a reduction in area of 123μm^2.

We have then computed separately each Compression Box of the various schemes, connecting then the ones composing the various columns. Note that the data related to the generation of the two final digits of each column are included in the column data. In the calculation of the delays of each stage, the delay of each stage output is computed independently and the highest, worst value is adopted for the stage delay. The minimum delay has been obtained adopting the "fast" version when appropriate.

Table E shows the results obtained for the three schemes using the VAM approach: VAM standard (A), VAM minArea using decoders for various multiplicity factors (B), VAM minArea using x2 decoders only (C).

Comparison

data for 1 column; compression to 2 addends						
addends N=	3	4	8	16	32	
VAM standard	1	2	6	14	30	**cells**
A)	1	2	4	6	8	stages
delay ps	*245*	*490*	*980*	*1470*	*1960*	*dec2 merged*
area µm^2	528	1056	3168	7392	*15840*	
VAM minArea						*dec2,dec4,*
B) delay ps		*490*	*850*	*1460*	***1600***	*dec8,dec16*
area µm^2		1056	3433	9179	15509	*merged*
VAM dec2 only						*dec2 first in*
C) delay ps		*500*	*1020*	*1550*	*2050*	*column merged*
area µm^2		1696	6636	9601	17730	
hybrid method {6}						
D) *delay ps*	*400*	*556*	*836*	*1390*	*1850*	
area µm^2	815	1156	2626	6060	12630	
copyVAMtablesA			-1.6	-5	13.5	%delay
			-30.7	-51.4	-22.7	%area

TABLE E: comparison among three VAM schemes and the hybrid scheme.

Data of a different architecture [6] have been added (D) for comparison purposes. The architectures offering for each N (the addends number) the best performance (both in delay and area) are marked with grey.

Comparing the three VAM based architectures:

- For **N**=3 and **N**=4 the delay of the VAM standard scheme is smaller than the corresponding values of the hybrid scheme. Take into account that the input numbers are in BCD-4221 format for the VAM architectures, in BCD-8421 for the hybrid one. In VAM architectures a single Fig.1 VAM cell suffices.
- For **N=32** the delay of the B) VAM schemes is considerably smaller than in all other schemes. The areas in the D) schemes appear smaller than those required in the VAM schemes. The delays in the N=8 and N=16 B) VAM are close to the values in the hybrid schemes, while the areas are considerably higher.

The above data contradict the denomination *minimal Area* given in [7] to such scheme. There is, in effect, no proof of this property. The scheme was rather inspired by the idea of using, as far as possible, Binary Compressors, smaller and faster than Decimal Compressors (due to the added decoders).

Conclusions

We have shown how the basic compression scheme proposed by Vazquez, Antelo and Montuschi [7], with suitable notations and additional improvements, can be conveniently applied to the design of multi-operand decimal adders. We have computed the values of the basic parameters (delay, area) and have compared them with the corresponding parameters of the scheme based on binary-decimal arithmetic using binary-to-decimal conversions (hybrid schemes). For numbers N of addends equal to 8 and16 the hybrid schemes offer smaller delay and area, while for N=32 the VAM minArea scheme offers a smaller delay and correspondingly requires a larger area.

References

[1] K.K. Richards, *Arithmetic Operation in Digital Computers,* Van Nostrand, 1955

[2] M.F. Cowlishaw, *Decimal Floating Point Algorithms for Computers,* Proc. 16th IEEE Symp. on Computer Arithmetic, pp.104-111, June 2003.

[3] M.A. Erle, M.J. Schulte, *Decimal Multiplication via Carry-Save Addition,* Proc. IEEE Int'l Conf. Application Specific Architectures and Processors, pp. 348-358, June 2003

[4] Kenney, R.D., Schulte, M.J. *High speed multioperand decimal adders,* IEEE Trans. On Computers, vol.54, n.8, pp.953-963, August 2005

[5] Choi, H., Kim, Y.D., You, Y. *Dynamic Decimal Adder Circuits Design by using the Carry Look-Ahead,* Proc. Design and Diagnostic of Electronic Circuits and Systems, pp. 242-244, Apr. 2006

[6] Dadda, L. *A Multioperand Decimal Adder*: *a mixed binary and BCD approach,* IEEE Trans. on Computers, vol. 56, n. 9. pp. 1320-1328, September 2007

[7] Vazquez, A., Antelo, E., Montuschi, P. *A new family of high performance Parallel Decimal Multipliers,* Proc. of 18th Symposium on Computer Arithmetic, pp. 195-204, 25-28 June, 2007

[8] Dadda, L. *Spreadsheet program for designing the Compresssion Boxes using the Vazquez-Antelo-Montuschi cell.* http://www.alari.ch/

[9] STMicroelectronics. 90nm CMOS090 Design Platform. [Online]. Available: http://www.st.com/stonline/prodpres/dedicate/soc/asic/90plat.htm

Appendix

A: Computing the critical paths delays

We show here how a given Compression Box scheme can be redrawn as a spreadsheet scheme for computing the critical path delay of each output variable. The method, shown here as an example applied to the N=16 Compression Box (shown in Fig. 7) has been used for obtaining the delay data for the Compression Boxes shown in Table B.

The topology of a delay-scheme is identical to the topology of the given logical-scheme: compare the following delay-scheme of a N=16 CB as in Fig.7.

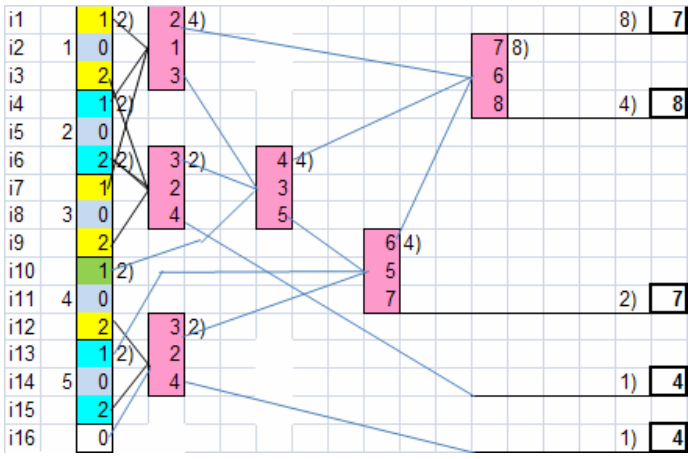

Fig. A1: The delay-scheme of a N=16 Compression Box, whose logical scheme is shown in Fig. 7

Each box in Fig.7 representing a Binary Compressor (with an attached weight) is replaced in Fig. A1 with three aligned spreadsheet cells. The Central cell is implemented with an instruction MAX(iA;iB;iC) generating the maximum value of the three input variables iA, iB, iC. The top-cell generates the sum of the max input value with the carry-delay of the assumed full adders. The bottom cell generates the sum of the max-input with the sum–delay. This property assures that the outputs values are the total max delay of the paths connecting the Compression Box inputs with its outputs.

The delays shown in Fig. A1 assume the same rule given for Table B. All the input variables are assumed to be applied simultaneously at time 0.

The spreadsheet program permits also to simulate the effects of additional delays for one or more input variables.

B: Schemes for Adders of N=32 decimal numbers

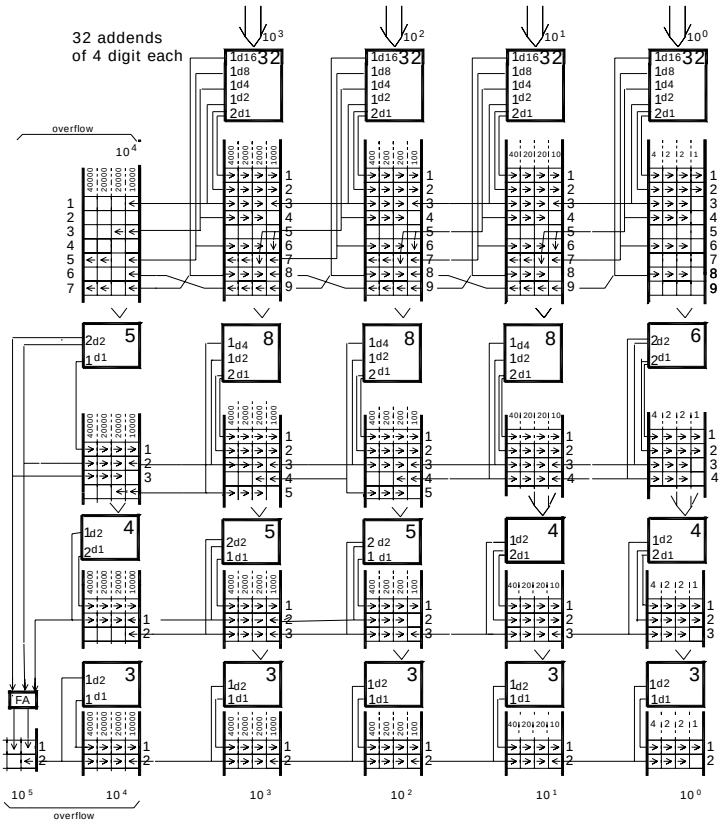

Fig. A1: Standard scheme for N=32 decimal numbers

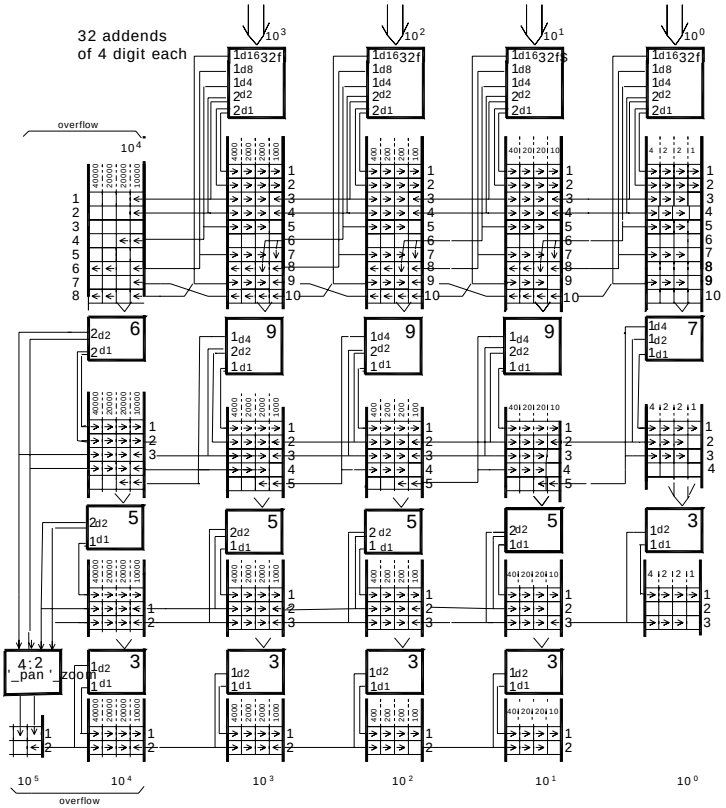

Fig.A2: A faster adder for N=32 decimal numbers

		STAGE 1					STAGE 2				STAGE 3				STAGE 4		
canonical	**32**	**2**	**1**	**1**	**1**	**8**	**2**	**1**	**1**	**5**	**1**	**2**		**3**	**1**	**1**	14567
ADD		*6*	*9*	*10*	*11*		*4*	*5*	*4*		*4*	*3*			*2*	*1*	µm^2
VAM delay		300	450	500	550		200	250	200		200	150			100	50	
xj *delay*			200	210	200			200	200			200				200	ps
		300	650	710	**750**		200	**450**	400		200	**350**			100	**250**	**1800**
variant	**32f**	**2**	**2**	**1**	**1**	**9**	**1**	**2**	**1**	**5**	**1**	**2**		**3**	**1**	**1**	15509
ADD		*4*	*7*	*8*	*9*		*4*	*3*	*2*		*4*	*3*			*2*	*1*	µm^2
VAM delay		200	350	400	450		200	150	100		200	150	0		100	50	
xj *delay*			200	200	200			200	200			200	200			200	ps
		200	550	600	**650**		200	**350**	300		200	**350**	200		100	**250**	**1600**

Fig.A3: delay and area computation for Fig. A1 and Fig.A2 schemes