Rendiconti Accademia Nazionale delle Scienze detta dei XL
Memorie di Scienze Fisiche e Naturali
ISBN 987-88-548-5718-6
DOI 10.4399/97888548571864
pag. 33–52

Half adders in binary parallel multipliers

Luigi Dadda

1. Summary

Binary parallel multipliers are composed mostly of full adders and half adders. We show that, independently from the chosen architecture, the half adders number must be at least N, the factor's length. Choosing then one of the simplest architectures half–adders–minimal, we show that such minimal number of half adders, properly arranged, can obtain properties such as: the generation of a number of non–redundant least significant product's digits, the correction of the row–overflow i.e. the optimization of the total delay (treated in a previous paper), the operation with factors in two's complement format. The case of the non–half minimal multipliers is also discussed, showing the Wallace scheme as an example. It is also shown that the set of irredundant least significant bits of the product does not cause any delay in the final Carry Propagating Adder. Finally we present the parameters of the multipliers family that includes the schemes illustrated in chapter III, obtained using a design tool based on a spreadsheet.

2. Introduction

The design of fast binary multipliers has been the subject of numerous researches in the past decades, originating several schemes. They are composed of a number of full adders and half adders in networks of different structures. The common general architecture is composed of a first part generating the partial products usually represented in a rhomboid array of N rows each containing the products of the multiplicand digits by a digit of the multiplier. The product is the value of such an array, i.e. the sum of the N rows. For reason of

speed such evaluation is done in two steps. In the first the number of rows is reduced to two using carry save additions. Those two rows represent the product in a redundant form. The final product is obtained by adding those two rows via a Carry Propagating Adder, usually adopting Carry–Look–Ahead structures for speed reasons.

The carry save operations are obtained with full adders and half adders, composed in different ways, implementing a specific algorithm.

Half adders, though in a rather small number if compared to the full adders, play a number of different roles, obtaining various specific characteristics of the multipliers. As examples, the compression phase leading to two equivalent rows can produce a number of non–redundant least significant digits, thus reducing the length of the Carry Propagating Adder. Moreover, the number of cascaded stages in the reduction process can be non–optimal, due to a "row overflow" that has been overlooked in the past and illustrated by the author in a previous writing [2], the same problem being also considered in [7] with a different methodology. It has been shown that the insertion of few half adders avoids such overflow, obtaining a faster circuit.

As a preliminary step, we will show in next chapter that in a binary multiplier there must be a minimum number of half adders, independent of its architecture. Such a minimum is N, the size of the factors in bit. This property is just a necessary condition for a binary multiplier. It permits to say that a multiplier with a smaller number of half adders cannot exist. We will show, however, that it leads to reconsider, in general, the presentation of existing algorithms. In particular we suggest to consider as a rule from the beginning both the compression phase and the Carry Propagating Adder, due to their interaction.

The notation used for representing a multiplier scheme plays an important role. If, for example, the compression network is described using the standard logic notation it would be very difficult to design and to understand it, due to its size and complexity. Some form of abstraction is needed for reducing the size and at the same time keeping the possibility to represent the main properties. A method widely used has been proposed in 1965 [3]; it represents a full adder with two dots connected by a segment, representing the sum and the carry. It is the *dot notation*. A step toward a greater compactness

was shown in [5] representing a set of variables (with the same binary weight) with an integer close to a dot. A further step has been shown in [2] with the *cell notation* where a set of variables is represented by a cell containing an integer. This leads to the use of spreadsheets and to the possibility of obtaining operations on logical variables using instructions in the language associated with spreadsheets.

In the following we will adopt the cell notation just for representing the cell arrays, implementing the algorithms "manually" and not through programming instructions. A design program based on a spreadsheet , used for obtaining the data given in chapter VII, can be downloadable from [8].

3. The minimum number of half adders in a binary multiplier

The preliminary result of this paper, i.e. the determination of the minimum number of half adders in a binary parallel multiplier, is not based on the consideration of any specific proposed scheme. It is a very general result applicable to any scheme architecture, on the assumption that it uses a reduction algorithm based exclusively on full adders and half adders. The reduction algorithm starts with an array of partial products and reduces the number of rows to two via cascaded stages using carry save techniques. Among the proposed different schemes we cite the Wallace scheme [1], the scheme proposed in [3] offering the smallest number of full adders and the scheme proposed in [7] using in a different way the same stages heights sequence of [3] and the scheme introduced in [2] for solving the problem of row–overflow.

A different approach, using signed digits, is adopted in multipliers based on the Booth algorithm, not considered in this paper.

The determination of the minimum number of half adders in a binary multiplier is based on the following steps, shown as an example in a specific simple case of fig. 1 and fig. 2. In the first the *multiple dot notation* has been used , while in fig. 2 the corresponding *cell notation* has been adopted.

We note that a multiplier scheme in dot or in cell notation can be considered a *compact hardware scheme*. Note also that in cell notation the coordinates of each cell can be assumed as the variables names

(with indexes for the variables of a given cell). A 1 for indicating a variable or an integer for indicating a set of variables in a cell, tells that those variables exist and their names are also the names of the pins where the variables can be seen.

The variables feeding a full adder are not explicitly shown but are implicitly given by the output variables in the same column in the preceding stage(s). If the three input variables are 1, the same value have the two output variables i.e. the sum and the carry. Similarly for the two variables feeding a half adder, with a relevant difference: if both input variables are 1, the sum output will be 0 and the carry will be 1, but we will denote both of them with 1, since both exist in a hardware scheme. We will call the set of $2N$ output bits in the hardware scheme a *hard–product*, to signify that it is the tool for representing the product.

We can, however, consider the same scheme as an *operational scheme* showing the variable's values in the *special case in which all the input variables have value 1*, i.e. the unsigned multiplication factors have their maximum values.

In a *hardware scheme* the $2N$ product bits are denoted with 1 but, notoriously, the product of two N–bits factors is composed of $N–1$ most significant bits valued 1, followed by N zeros and ending with a 1 least significant bit. Assume now that in each of the N zero–output columns we have a half adder (in one of the cascaded reduction stages). Assume also that such N half adders are placed according to a valid multiplier scheme. In fig 1a and fig, 1b schemes we have $N= 6$ half adders, placed in 6 adjacent columns 2 to 7. Correspondingly the output of the same columns in an operational scheme will be zero, giving therefore a correct maximum product.

We know, however, that valid multipliers exist having a number of half adders greater than N. Such property is found in the Wallace multiplier [1]. We will examine in detail such a multiplier in the next chapter V, fig. 4. In such a multiplier we have in the same "zero–output" columns a column 00 having 2 half adders and columns without half adders, giving anyway the correct zero output. This happens for the following reason. Let's assume that in one of the columns, i, we have two half adders in the same or in different stages. This means that a total $2^i + 2^i = 2^{i+1}$ must be subtracted from the *hard–product* for obtaining the *product*: this means that a *1* must be

subtracted in column $i + 1$. We thus obtain a reduction in column $i + 1$ using two half adders in column i, while it could be obtained with a single half adder in the same column $i + 1$ in a half–adders–minimal scheme.

In conclusion *the minimum number of half adders is N if a single half adder is placed in each of the columns 2 to N + 1.*

4. The role of half adders in a specific case

We now examine the role of half adders in multipliers schemes and for that purpose we adopt a specific scheme. Such a scheme has been introduced in a previous paper dealing mostly with the "row overflow" problem [2]. It has the following properties.

— The reduction process is implemented using only full adders obtaining the reduction to three rows, i.e. a $N : 3$ reduction (we adopt the notation used in some programming languages, in particular in spreadsheet languages). A reduction 3:2 is obtained in a *Last* stage, using also half adders.
— A *Final* stage obtains the non redundant product, using a Carry Propagating Adder, *CPA*.

The half adders are in this case used only in the *Last* and in the *Final* stages. Their total number is N, i.e. the minimum.

Such a scheme is not immune from row overflow. We will show in chapter IV the case of a half–adder–minimal multiplier immune from row overflow.

As examples, fig. 1 and fig. 2 represent multipliers for $N = 6$. Fig. 1A and fig. 2A schemes use respectively the dot notation and the cell notation. In both schemes we have 6 half adders: 1 in the *Last* stage and 5 in the *CPA*.

The two schemes are affected by row overflow. Fig.1B and fig. 2B represent the same multipliers corrected for row overflow by placing a half adder in stage 1. The number of compression stages is reduced from 3 to 2. The number of half adders is still 6.

In general the places where the half adders are situated depend on the multiplier's architecture. We found that the multipliers character-

ized by $N:3$ compression stages using exclusively full adders have the minimum number of half adders. If we want to avoid row overflows, half adders will be needed also in some of the $N:3$ compression stages. This doesn't requires an increase of the total number of half adders, since it happens that correspondingly, in the same column, the half adder in the *Last* stage is no more needed. Note that, otherwise, we would have two half adders in the same column, with the consequences described in the preceding chapter.

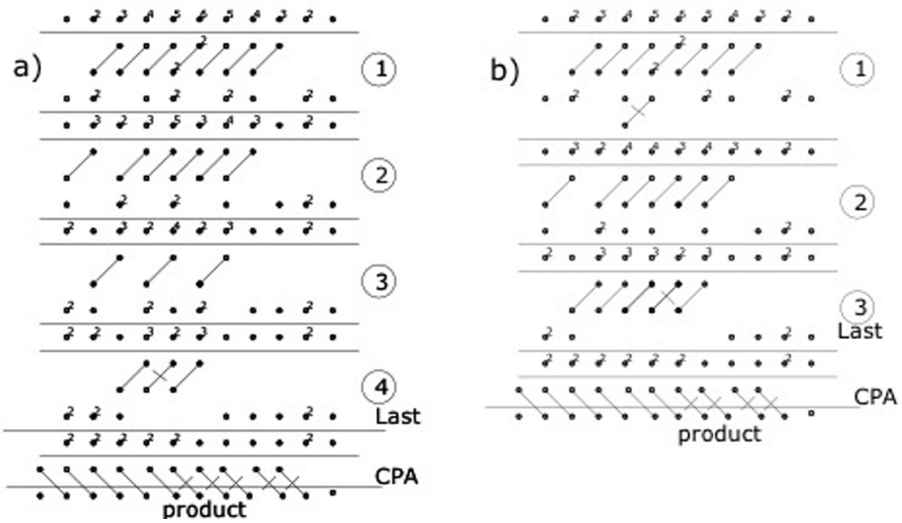

Fig. 1A: a 6x6 multiplier in dot notation, with row–overflow; fig.1B: with correction to avoid row–overflow.

In fig. 1A we notice that the first topmost row represents the partial products (usually given as a rhomboid array of 1s) by the sequence 1,2,3,4,5,6,5,4,3,2,1 representing the numbers of variables in each column: such a sequence is the *input vector* of the compression section.

Three stages are needed for reducing the input vector to 3 rows with no half adders. The *Last* stage obtains the two–rows product using 1 half adder. A final Carry Propagating Adder obtains the non redundant product, using 5 half adders for a total of 6. In fig.1B in stage 1 we place a half adder in column 7 for avoiding a row overflow. In the *Last* stage we have a second half adder in column 6. The Carry Propagating Adder is composed of 4 half adders in columns 2 to 5,

giving a total of 6 half adders. Fig. 2A and 2B, represent the same schemes of fig.1A and 1B, adopting the *cell notation*. In such figures we have added in two columns the decimal values of the corresponding rows, showing how the final product (for the maximum values of the factors) is generated. Note in fig. 2A the value of the product (3969). The output from *Last* stage is 4033 = 3969 + 64, the last term being the weight of the half adder Sum output of the half adder in column 7. The output from the CPA is further increased due to the 5 half adders in CPA. In line 19 we find the *hard–product* composed from all 1s, valued 4095. The *product* in line 21 is obtained by subtracting from the *hard–product* the value (126) of all the Sum outputs of the six half adders. The last row in fig. 2A indicates the 6 half adders and their locations:

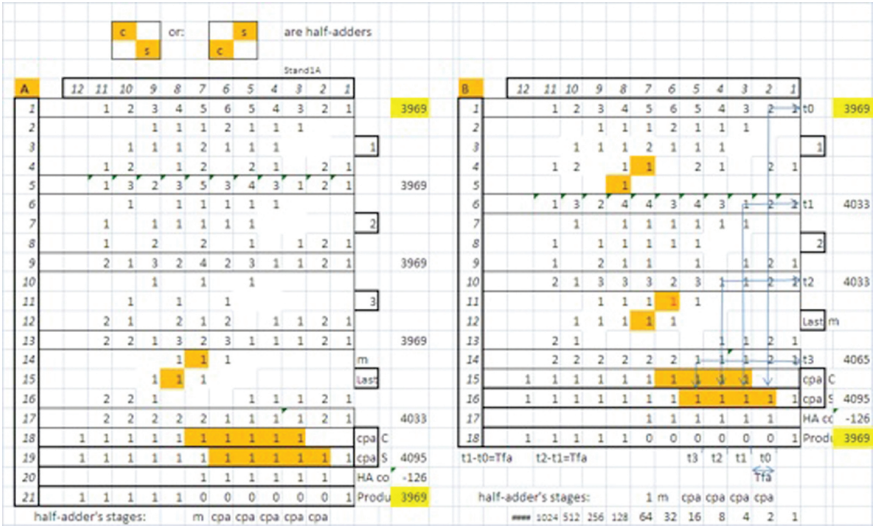

Fig.2A: 6x6 multiplier using cell notation; the scheme is affected, in stage 1, by row–overflow, requiring 4 compression stages. Fig. 2B: the row overflow is eliminated with a half adder in stage 1. Compression stages are reduced of 4 to 3. Both schemes use 6 half adders, as shown in bottom row: *1* stand for 1st stage, *m* for Last stage and *cpa* for Carry Propagating Adder. The lines pointing t3, t2, t1, to tell that the pointed bits of the product are generated at different times in the same CPA stage.

The half adders can be found in the *N:3* compression stages for two different tasks: avoiding the row overflow and generating a number of single least significant bits (reducing the CPA length). Note that we had to imply the CPA because the determination of the number and places of the half adders requires the product in the non–redundant form. This is an important peculiarity of this paper. A most important result is that the generation of the least significant non–redundant product bits requires few half adders in columns 2 to $N+1$, where in any case we have "native" half adders in *CPA*. Those half adders accomplish "naturally" the new task. The same result has been suggested in [4] by placing a single half adder in all the compression stages, no consideration being given to the CPA. We notice here that such arrangement can be obtained simply by moving the half adders from their "native" place in CPA to the Compression stages. It can moreover be seen that the two resulting schemes (fig. 2B and fig. 2C) are topologically identical, i.e. the displacement of a half adder from the CPA to a compression stage can be obtained via stretched connections. Leaving them in the CPA or moving them to compression stages is just a matter of taste.

The case of the half adders used to eliminate row overflow is rather different. Placing a half adder in a compression stage for this purpose obtains the elimination of row overflow by changing locally the structure of the scheme, generating a different (but equivalent) output vector. Such changes obtain in the *Last* stage the elimination of the half adders in the positions where previously the half adders were generated.

The scheme obtains the two results (elimination of rows overflow and generation of least significant non–redundant digits) with no additional costs, since no changes are caused in the number of full adders.

The Carry Propagating Adder will be represented, when needed, in its simplest implementation, i.e. as a Ripple Carry Adder, despite the fact that in practice, for speed reasons, a faster arry–Look–Ahead or a Parallel Prefix scheme is adopted. For our purposes, however, we can adopt the Ripple Carry scheme, since we are interested here only in its basic task i.e. the parallel addition.

C	12	11	10	9	8	7	6	5	4	3	2	1			
1		1	2	3	4	5	6	5	4	3	2	1			3969
2				1	1	1	2	1	1	1	1				
3			1	1	1	2	1	1	1	1			1		
4		1	2		1	1		2	1			1			
5					1										
6		1	3	2	4	4	3	4	3	2	1	1			4035
7			1		1	1	1	1	1	1					
8		1		1	1	1	1	1	1				2		
9		1		2	1	1		1			1	1			
10		2	1	3	3	3	2	3	2	1	1	1			4039
11				1	1	1	1	1	1						
12			1	1	1	1	1	1					Last	m	
13		2	1							1	1	1			
14		2	2	2	2	2	2	2	1	1	1	1			4079
15	1	1	1	1	1	1	1						cpa	C	
16	1	1	1	1	1	1	1	1	1	1	1	1	cpa	S	4095
17						1	1	1	1	1	1		HA corr		0
18	1	1	1	1	1	0	0	0	0	0	0	1	Produc		3969
half-adder's stages:						1	m	cpa	m	2	1				
Stand1A	2048	1024	512	256	128	64	32	16	8	4	2	1			

Fig. 2C: derived from fig. 2B by displacing the half adders in columns 2, 3 and 4 from the CPA to stages 1, 2 and Last, respectively. Such displacements can be effected without changing the input connections (not shown in the figures). The row following row 18 shows the stages where one input of the corresponding half adder is connected.

5. A second case of half–adders–minimal multiplier

We show now a second case of half–adder–minimal multiplier, given by a scheme proposed in [3] characterized by a prefixed sequence of stages heights: 2, 3, 4, 6, 9, 13, 19, 28, 42, 63, 94, [. . .] Each of these terms offers the maximum value of N (the factors length in bit) for a given number S of stages. We call those multiplier *N(S)–maximal multipliers*. For a given N this multiplier needs the smallest number of Full Adders if compared with other proposed schemes.

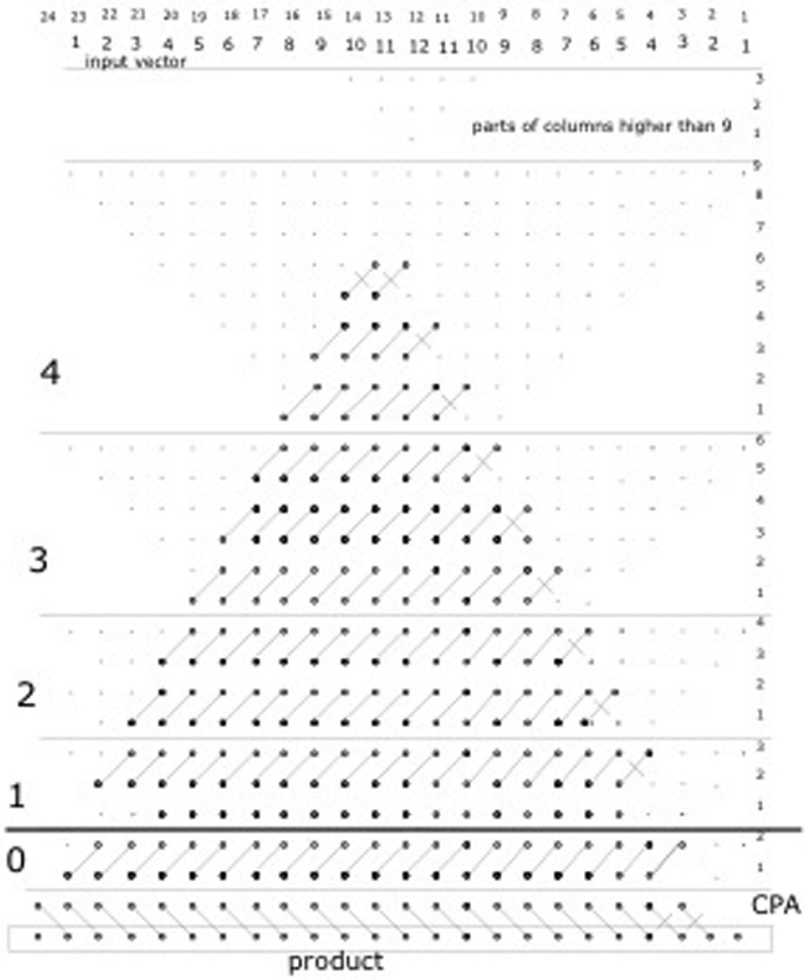

Fig. 3 The scheme for a 12x12 N–maximal binary multiplier. Stage's heights from bottom are: 2 for stage 0 (feeding the Carry Propagating Adder), 3 for stage 1, 4 for stage 2, 6 for stage 3, 9 for stage 4, the topmost. This contains the top of the input array columns higher than 9 (and smaller than 13, the next stage height).

The same scheme is also immune from row overflow [3], just because each column reduction is exactly matching the output height to a prescribed value. The scheme is derived from the original in [3] by inverting in each stage the order of the rows. The dot notation scheme

shows great simplicity. It can be seen, in particular, that the scheme has been completed with the Carry Propagating Adder in order to generate the final non–redundant product. This starts in column 3, being preceded in the least significant 3 columns by 2 half adders (in columns 2 and 3). The third half adder is in column 4, stage 1. The other half adders are in the following (in ascending order) rows. The topmost row is composed from 2 half adders.

The reduction algorithm differs significantly from the one used in the preceding case. In the latter the N_i input variables in each column were reduced as much as possible, using a number of full adders equal to $\lfloor n_i/3 \rfloor$, with no reference to the heights values. In our case instead the reduction is required for obtaining a total output column height equal (or smaller) to the height of the following output stage.

This implies that, starting from the rightmost column, some columns, i.e. the input columns whose heights are smaller than the output column height, will not be processed, the corresponding variables being simply transferred to the output columns.

If instead the input column height n_i is greater than the prescribed value n_s a number $n_{e=}n_i--n_s$ of variables cannot be simply transferred. What matters, precisely, is that their values are transferred without violating the condition of a prefixed stage height.

This can be obtained in the following way. Let's assume that the n_e bits "in excess" are just one. We can pair the single excess bit with a bit of the same column in the following stage. We than feed a half adder, placing its Sum output in the output column and its Carry output in the next (to the left) output column. If we have instead $n_e = 2$, we do the same using a full adder. For a generic n_e we need $n_{fa} = \lfloor n_i/2 \rfloor$ full adders and a single half adder for the case of n_e odd, for a total of $n_{fa} + 1$ carries in the next left column, to be added to the column's excess bits, to be treated as just described.

The above algorithm has been applied "by hand" for obtaining the fig. 3 scheme. The same algorithm has also been programmed in the spreadsheet language, obtaining a program using the *cell notation*.

As an additional remark we note that the sequence describing its construction doesn't represent in general the actual operation of the scheme. This would happen only in the case that the output variables of each stage were stored in a set of flip–flops, obtaining then a pipelined structure permitting very high throughput (with,

necessarily, a delay due to the number of pipelined stages). Assuming that we do not adopt that structure, avoiding any pipeline step, we can see from fig. 3 that all stages would be reached by the input variables simultaneously. The pipelining in this multiplier has been treated extensively in the literature.

6. Multipliers with non–minimal half adders

The *Wallace* multiplier [1] requires a number of half adders greater than *N*. The Wallace architecture is based on transforming a group of three adjacent rows of the partial product array into two equivalent rows obtained with an array of full adders and half adders. Fig. 4 shows the Wallace reduction algorithm applied to a 7x7 multiplier. The full adders are represented by a (skewed) couple of shadowed cells , the half adders being marked with a diagonal segment with two arrows. The rows not grouped (1 or 2) are transferred with no change.

The above transformation is repeatedly applied until two rows only are obtained (in rows 20 and 21 in fig. 4 example).

The non redundant 1 row product is obtained using a Carry Propagating Adder in row 22 (containing the carries) and 23 (the sum bits).

Row 24 shows the number of half adders present in each column. In row 23 (the hard–product) we see all 1s. The true product can be obtained by subtracting from it the number representing the set of the sums outputs of all the half adders, using the weights in the bottom row. The difference is shown in the right–low part of the figure. The result of the difference (16129) converted in binary gives: 0111111000000001, i.e. th product (of two factors valued 1111111 Such result can be obtained column by column (starting with the rightmost one). In the first ten columns the difference of rows 23 and 24 is 1100000001. In column 11 we should subtract 2 from 1. We subtract just 0 and transfer 2 to the next left column after dividing by 2. The same situation in now in column 12. We then repeat the operation until a 1 is obtained in column 15, generating the correct product in row 25. We need 12 half adders in the Wallace scheme while 7 are needed in a half–adder–minimal multiplier. It is possible to perform on fig. 4 the same timing analysis done on figure 2B scheme, obtaining the same result.

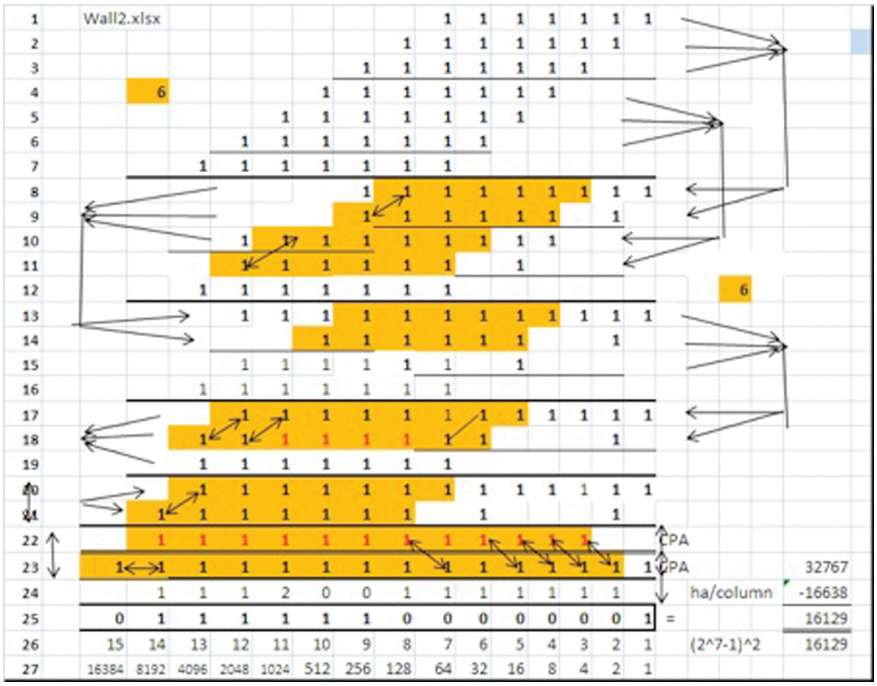

Fig. 4: a 7x7 Wallace multiplier

7. A two's complement multiplier

As a further variation of fig. 2 scheme we consider a multiplier handling signed factors in two's complement form. We have considered the problem also in [2] in a different context, showing that a good solution is obtained as in fig. 2B scheme with the variations (see also [2, 6]) depicted in figure 5.

All the terms in columns 1 to 2N–1 are positive, the only negative term is in column 2N (in other words the binary weight in column N is -2^N). We can adopt the spreadsheet program, developed for evaluating arrays for unsigned factors, with suitable modifications. First of all we change the Partial Product generator replacing the AND gates, generating the left diagonal and the bottom row, with NAND gates.Then we modify the input vector of the spreadsheet program by adding a new variable to the $N+1$ column and to the $2N$ column (empty in the original program), whose negative sign will be accounted for in the addition process.

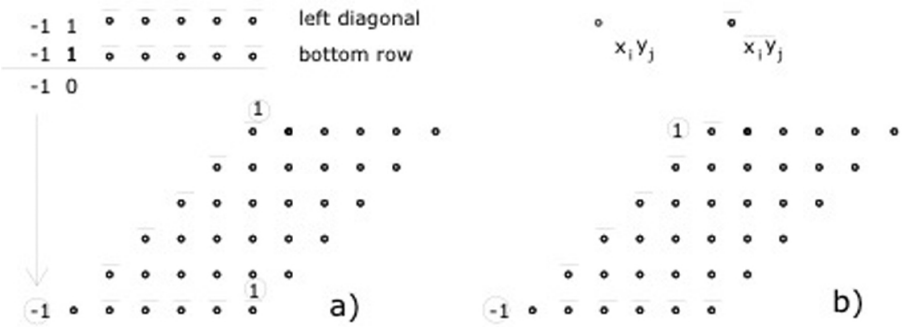

Fig. 5: a):a multiplier array for two's complement factors. The two 1s in central column are needed for complementing the bottom row and the leftmost diagonal. The leftmost –1 derives from such complementation and generates the sign of the product when affected by the carry from the preceding column. At the top–left we show the left diagonal and the bottom rows extended two columns to the left to align them with the column 2N where the negative weighed product sign bit lies. b): an equivalent more convenient version.

We obtain the result by adding a new row to the input vector generator in which we place the two 1 terms, generated by a specific program using the assigned N. The original row and the new row will be added to obtain the two's complement operation. If a standard unsigned factors operation is desired we simply place zeros in the new row. The compression algorithm designed for unsigned factors works also for two's complement factors.We found that the term in $N + 1$ column causes the number of full adders to be increased by 1, while no change in the number of compression stages occurs. Note that the additional full adder could be replaced with a special half adder as suggested in [4].

The 1 added to the column $2N$ brings the length of the 2–rows *Last* stage output to 2N columns, i.e. the product length. The final product requires in the CPA a corresponding $2N_{\text{th}}$ column composed from a half adder (whose carry will be neglected). Its Sum output (the product's sign bit) can be easily proved to be equal to the complement of the Carry generated in the preceding $(2N - -1)_{\text{th}}$ column of the CPA. In conclusion the number of half adders is unaffected, with a

modification: the leftmost one is moved from column $N+1$ to column 2N (where it can be replaced with an inverter). This is the only case, among those encountered so far, in which one of the half adders is not contiguous to the others.

The program for the design of multipliers for unsigned or two's complement factors can be downloaded from [8].

8. Parameters of a multiplier family

In the preceding chapters we have discussed the multipliers properties using working examples with small values of N, considering also a number of different families. We want in this chapter to consider a single family, i.e. the one represented in fig. 2. The main characteristic of this family is the adoption of the height of the first stage the value N, the factors length. The successive stages height is equal to the maximum value in the output vector of the preceding stage. This is quite a natural choice (we could call it the *Standard* height sequence), but not the only one possible. We have seen, as an example the scheme in fig.3, where the sequence of the successive stages heights is prefixed and assumed for all the multiplier of the corresponding family. We obtained the data using a design tool conceived for the said family, capable of giving, in cell notation, the scheme and a number of parameters. Table A shows the sequences of the stages heights for $4 \leq N \leq 212$. The sequences marked by a star (*) are characterized for having the maximum N for a given number of stages. They are the only stage heights for the multipliers proposed both in [2] and in [4].

Another family using the standard stage heights sequence is the one discussed in chapter V. The cells with grey background contain height values multiple of 3 and represent the heights of the stages in which a row overflow is generated. The total number of half adders needed for row overflow correction is given by the bottom cells in italic. Note that all the columns following the ones with a star are the only cases with no row–overflow. Table B shows for each N ($4 \leq N \leq 32$) the following data:

— the number of stages of the Compressor
— the number of non redundant least significant bits

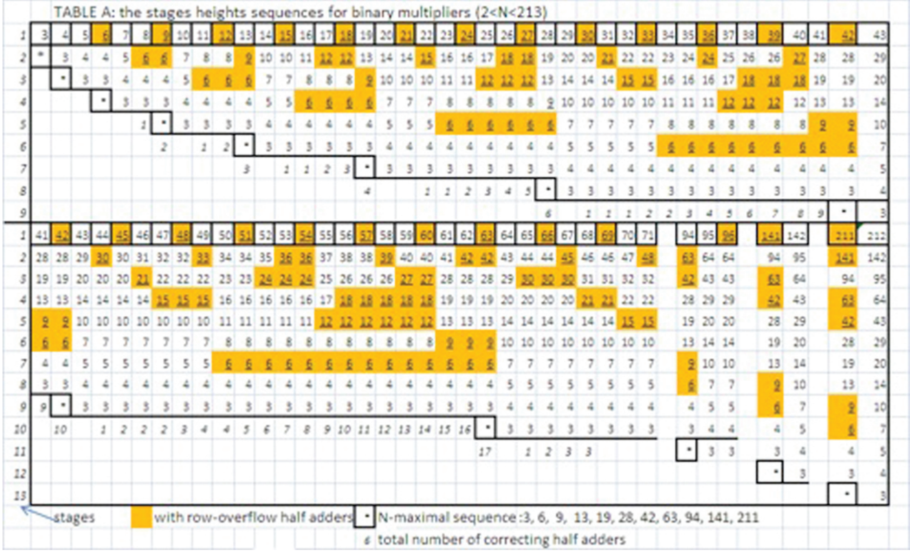

TABLE A: the stages heights sequences for binary multipliers (2<N<213)

1	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
2	•	3	4	4	5	6	6	7	8	8	9	10	10	11	12	12	13	14	14	15	16	16	17	18	18	19	20	20	21	22	22	23	24	24	25	26	26	27	28	28	29
3		•	3	3	4	4	4	5	6	6	6	7	7	8	8	8	9	10	10	10	11	11	12	12	12	13	14	14	14	15	15	16	16	16	17	18	18	18	19	19	20
4				•	3	3	3	4	4	4	4	5	5	6	6	6	6	7	7	7	8	8	8	8	8	9	10	10	10	10	10	11	11	11	12	12	12	12	13	13	14
5						1	•	3	3	3	3	4	4	4	4	4	4	5	5	5	6	6	6	6	6	6	7	7	7	7	7	8	8	8	8	8	8	8	9	9	10
6							2		1	2	•	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	5	5	5	5	5	6	6	6	6	6	6	6	6	6	7
7											3		1	1	2	3	•	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5
8																	4			1	1	2	3	4	5	•	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4
9																										6		1	1	1	2	2	3	4	5	6	7	8	9	•	3

1	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71		94	95	96		141	142		211	212
2	28	28	29	30	30	31	32	32	33	34	34	35	36	36	37	38	38	39	40	40	41	42	42	43	44	44	45	46	46	47	48		63	64	64		94	95		141	142
3	19	19	20	20	20	21	22	22	22	23	23	24	24	24	25	26	26	26	27	27	28	28	28	29	30	30	30	31	31	32	32		42	43	43		63	64		94	95
4	13	13	14	14	14	14	15	15	15	16	16	16	16	16	17	18	18	18	18	18	19	19	19	20	20	20	20	21	21	22	22		28	29	29		42	43		63	64
5	9	9	10	10	10	10	10	10	10	11	11	11	11	11	12	12	12	12	12	12	13	13	13	14	14	14	14	14	14	15	15		19	20	20		28	29		42	43
6	6	6	7	7	7	7	7	7	7	8	8	8	8	8	8	8	8	8	8	8	9	9	9	10	10	10	10	10	10	10	10		13	14	14		19	20		28	29
7	4	4	5	5	5	5	5	5	5	6	6	6	6	6	6	6	6	6	6	6	6	6	6	7	7	7	7	7	7	7	7		9	10	10		13	14		19	20
8	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5	5		6	7	7		9	10		13	14
9	9	•	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4		4	5	5		6	7		9	10
10		10		1	2	2	2	3	4	4	5	6	7	8	9	10	11	12	13	14	15	16	•	3	3	3	3	3	3	3	3		3	4	4		4	5		6	7
11																							17		1	2	3	3					•	3	3		3	4		4	5
12																																					•	3		3	4
13																																								•	3

stages 🟧 with row-overflow half adders • N-maximal sequence :3, 6, 9, 13, 19, 28, 42, 63, 94, 141, 211

6 total number of correcting half adders

Table A

- — the number of full adders (excluding those used in the Carry Propagating Adder)
- — the number of half adders (in this case equal to N, the factors length)
- — in an adjacent column, the number of row overflow correcting half adders
- — N the factors length in bit

In an asociated column, the number of row–overflow correcting half adders in stage 1.

- — The following columns (marked 2, 3, 4, 5, 6, 7, 8 and partitioned in two sub columns) present for each N the successive stages heights with (when needed) the number of half adders for row–overflow correction).

The data in the tables cover the range $4 \leq N \leq 32$. Tables for up to $N = 67$ can be found in [8].

Table C shows for each N a row containing the location of the N half adders with the symbols used in the preceding figure 2. Since

5		69	10		10		7		5		4		3		2		2		2	
stg	lsb	fa	ha	hac	N	1	2		3		4		5		6		7		8	
2	3	5	4		4		3		5											
3	4	11	5		5		4		3											
3	4	18	6	1	6	1	4		3											
4	5	28	7		7		5		4		3									
4	5	39	8	1	8		6	1	4		3									
4	5	52	9	2	9	1	6	1	4		3									
5	6	68	10		10		7		5		4		3							
5	6	85	11	1	11		8		6	1	4		3							
5	6	104	12	2	12	1	8		6	1	4		3							
5	6	125	13	3	13		9	1	6	2	4		3							
6	7	149	14		14		10		7		4		4		3					
6	7	174	15	1	15	1	10		7		4		4		3					
6	7	201	16	1	16		11		8		4	1	4		3					
6	7	230	17	2	17		12	1	8		4	1	4		3					
6	7	261	18	3	18	1	12	1	8		4	1	4		3					
6	7	294	19	4	19		13		9	2	4	2	4		3					
7	8	330	20		20		14		10		4		5		4		3			
7	8	367	21		21	1	14		10		4		5		4		3			
7	8	406	22	1	22		15	1	10		4		5		4		3			
7	8	447	23	1	23		16		11		4		6	1	4		3			
7	8	490	24	2	24	1	16		11		4		6	1	4		3			
7	8	535	25	3	25		17		12	1	4		6	2	4		3			
7	8	582	26	4	26		18	1	12	1	4		6	2	4		3			
7	8	631	27	5	27	1	18	1	12	1	4		6	2	4		3			
7	8	682	28	6	28		19		13		4	3	6	3	4		3			
8	9	736	29		29		20		14		4		7		5		4		3	
8	9	791	30	1	30	1	21		14		4		7		5		4		3	
8	9	848	31	1	31		21	1	14		4		7		5		4		3	
8	9	907	32	1	32		22		15	1	4		7		5		4		3	

Table B

the program can be used for two's complement factors, two different locations are shown for the leftmost half adder: in column $N + 1$ for unsigned factors, in column $2N$ for two's complements factors, as discussed in chapter VI.

We underline the fact that the main result obtained by the design program is the scheme of the multiplier, represented in cell notation. Such a notation generates quite a compact scheme. This is particularly important for high values of N. The compactness is certainly very useful, but is far from a standard logical scheme. Knowing the basic principles of the cell notation, however, it is a good start for the design process, permitting to master the whole project. It is not a replacement of a description via languages like Verilog or VHDL. It can be considered a preliminary description, also because it offers a coherent

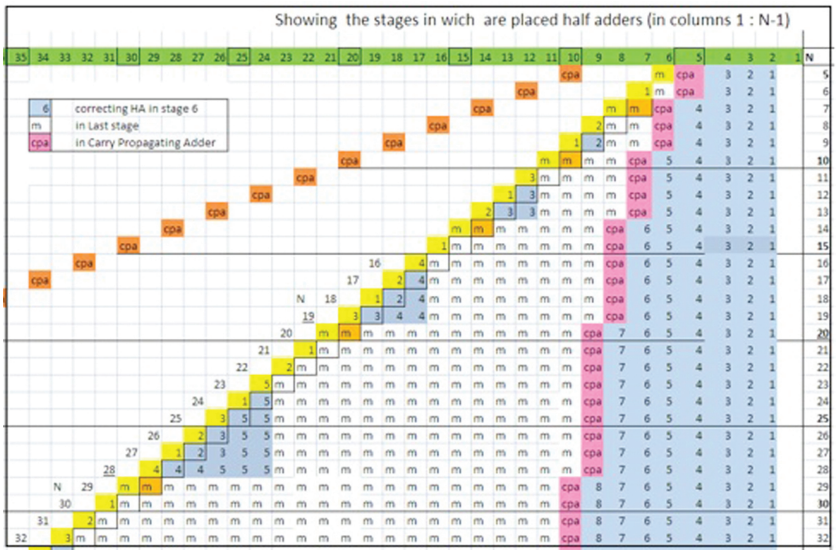

Table C

set of variables names, given by the cells addresses. It is easier to write the VHDL expressions for the sum and carry of the numerous full adders and half adders. Note also that a cell could be represented conveniently in VHDL with the *Array* instruction.

Last but not least the above described set of data can be considered a first example for an adjourned presentation style of multipliers and other similar arithmetic operators.

9. Conclusion

We have shown first that a minimum number of half adders has to be used in parallel binary multipliers, independent of their specific architectures. Two different families of half–adders–minimal multipliers and one family of non–half–adders–minimal multipliers have been shown.

Having chosen a specific family of half–adders–minimal multipliers, we have then found that in such multipliers we can solve the row–overflow problem and obtain the generation of a number of non redundant least significant digits in the product, using the available

minimal number of half adders, without increasing the number of full adders. The length of the Carry Propagating Adder is therefore reduced accordingly. Moreover, the computation of such least significant product bits is done in parallel with the existing compression stages, so reducing the total time for the computation of the final product. This last property has been so far overlooked.

We have shown that a multiplier transformed for factors in two's complement format requires the same number of half adders with one of them placed in the product's most significant sign bit and one more full adder.

We found also essential to consider the role of the Carry Propagating Adder, independently from its actual architecture. The main reason is that a number of "native" half adders is always present in it.

In most of the examples we have adopted a *cell notation*, introduced first in [2] as an evolution of the know *dot notation*. We could draw much more compact schemes with the possibility to handle them by computer programs using *spreadsheets*. We have finally shown in three tables a set of data for each multiplier of a specific family, obtained from a computer program capable of generating the corresponding schemes, in the range $4 \leq N \leq 67$.

References

[1] Wallace, C.S., *A Suggestion for a Fast Multiplier*, IEEE Trans. Electronic Computers, vol. 13, pp.14–17, Feb. 1964

[2] Dadda L., *A new notation for Arithmetic Array Design and its application to Binary Parallel Multipliers*, Rendiconti Accademia Nazionale delle Scienze, Memorie di Scienze fisiche e Naturali, 127° (2009), Vol. XXXIII, P. II t. I, pp.47–62.

——, *Some schemes for Parallel Multipliers*, Alta Frequenza, Vol. 19, pp.349–356, 1965.

[3] Bickerstaff, A.C., Schulte, M.I., Swartzlander, E.E., *Parallel Reduced Area Multiplication*, of VLSI Signal Processing, 9, 181–191, 1995.

[4] Dadda, L., *Parallel Decimal Multioperand Adders: a mixed Binary and Decimal approach*, IEEE Transactions on Computers.

[5] Parhani, B., *Computer Algorithms and Hardware Design* Oxford University Press, p. 179, 2000.

[6] Walters, S.M., Swartzlander, E.E., *A Reduced Complexity Wallace Multiplier Reduction*, IEEE Transactions on Computers,, vol. 59, n.8, pp. 1134–1137, Aug. 2011.

[7] Dadda. L., *Spreadsheet Programs for the design of binary parallel multipliers*, ALaRI Università della Svizzera italiana, 2011.

[8] http://www.alari.ch/people/dadda/home/DirRaDatwoCpa.xlsx.

Luigi Dadda
Politecnico di Milano, Università della Svizzera italiana
Uno dei XL
luigi.dadda@polimi.it