Journal of Computer Science 6 (4): 461-469, 2010
ISSN 1549-3636
© 2010 Science Publications

A Novel Methodology for Designing Radix-2^n Serial-Serial Multipliers

Abdurazzag Sulaiman Almiladi
Department of Computer Science, King Saud University, HCC,
P.O. Box 266, Huraimla 11962, Saudi Arabia

Abstract: Problem statement: The fast growth and increase in complexity of digital and image processing systems necessitate the migration from ad hoc design methods to methodological ones. Methodologies will certainly ease the trade off selection for those systems and shortens the design time. To increase those gained values and expand the searching space more appropriate methodologies need to be developed. **Approach:** A new methodology (table methodology) to design radix-2^n serial-serial multipliers was presented. Unlike other methodologies, the table methodology was used for the full design cycle, from the algorithm to the detailed fine control. **Results:** The methodology was used to identify the drawbacks in existing radix-2^n serial-serial multipliers as well as deriving new efficient ones. **Conclusion/Recommendations:** To the author's knowledge this is the first time tables are used in this novel way in tackling the complete solution space of serial-serial multipliers. One important merit of the new methodology is that it made it clear that there is no need of parallel loading in serial-parallel architectures and hence they can be transferred to serial-serial ones and a as a consequence a huge saving of bus width, I/O pins, area and energy will be achieved.

Key words: Design methodologies, digit serial architecture, DSP

INTRODUCTION

Recently, several methodologies have been proposed to design digit-serial architectures which will be described, briefly, bellow with their features and draw backs. In this study a new methodology (which will be termed the Table Methodology (TM) in the remainder of this study) for designing radix-2^n Serial Serial Multipliers (SSMs) is introduced. This is the first time it is used in this novel way of designing SSMs, where the algorithms, architectures, Basic Cells (BCs) and their fine controls are derived in a systematic way directly from the multiplication table. One significant advantage of the TM is its richness in carrying important information and exposing them in a clear way, which leads to many significant achievements. One such important achievement is the proof, systematically, that parallel loading of one of the operands in digit serial-parallel multiplication algorithms is no longer needed and the same functionality can be achieved with only a mixed radix-2^n Serial-Serial (S/S) feeding (where one operand fed one digit at a time and the other fed two digits at a time) (Almiladi and Ibrahim, 2009). This important result will lead to a noticeable saving of bus width, I/O pins, area and energy. Another powerful aspect of TM is that it shows, clearly, the difference in design and nature (synthesis/implementation) between inherent and non-inherent S/S algorithms as will be shown in primary rules.

It is worth mentioning, that multiplication tables (before the TM) was only used as a mean of showing correctness of the derived structures. However, in the TM, tables are used to proof correctness of existent structures as well as, deriving new ones in an easy and systematic ways.

Some existent digit-serial design methodologies: Several approaches have been proposed to design digit-serial architectures based on two's complement number representation which are summarized in Fig. 1 (Wu and Cappello, 1989; Smith *et al*., 1987; Aggoun *et al*., 1998a).

The description of these approaches along with their drawbacks can be found in (Aggoun *et al*., 1998b).

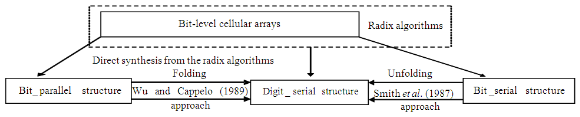
Bit-level cellular arrays
Radix algorithms
Direct synthesis from the radix algorithms
Folding
Unfolding
Bit_parallel structure
Wu and Cappelo (1989) approach
Digit _ serial structure
Smith et al. (1987) approach
Bit _serial structure

Fig. 1: Digit-serial methodologies (Aggoun *et al*., 1998b)

Recently, a new approach for designing digit-serial structures has been proposed (Aggoun *et al.*, 1998a; 1998b). It can be described by three main steps, namely, (i) writing the algorithm using the radix-2^n arithmetic, (ii) generating the Dependency Graph (DG) and selecting the projection direction and (iii) design and optimization of the radix-2^n cell.

More recently, the same design methodology proposed for designing digit serial-parallel structures in (Aggoun *et al.*, 1998b) is used in (Aggoun *et al.*, 2004) to design digit S/S structures. The new DG, which allows computation of the radix-2^n S/S multiplication was shown in (Aggoun *et al.*, 2004) for K = 4 (where K is the number of digits used). To obtain digit S/S architectures, the DG in (Aggoun *et al.*, 2004) is projected onto the line l = 0 (i.e., onto the k axis) in the direction $[1, 1]^T$. However, although most of the design cycle in (Aggoun *et al.*, 2004) was derived in a methodological way, some parts of the design are done in an ad hoc manner. For instance, making the most significant digits available to re-enter the structure, again, at the K^{th} cycle has not been done in a systematic way. In (Aggoun *et al.*, 2004) an ad hoc approach of modifying the interface by introducing K/2 shift registers to store the most significant digits of both operands is adopted. This ad hoc approach renders the original scalable architectures non scalable. Also because of the limited amount of information the DG is carrying, only a subset of the solution space can be derived systematically using it. For instance the twin pipe version of both the unidirectional and bidirectional digit SSMs cannot be derived systematically in an easy way using the DG. Also the new mixed radix-2^n SSMs, which will be explained later and proved the redundancy in digit serial-parallel architectures, cannot be observed or derived from the DG.

MATERIALS AND METHODS

The new Methodology (TM): A new methodology for designing multiplication structures is presented here. The new Methodology (TM) is effectively a mapping of different aspects of the multiplication operation into a hierarchical tabular representation. The mapping of aspects into the tabular representation is based on rules. Each aspect could have its own space (table), such as variables, temporal, spatial, control and so on. Transformation rules are also introduced as part of the new methodology which allows the manipulation of tables without losing their basic functionality.

The TM captures, in an easy way, the entire digit SSMs spectrum as well as producing new novel efficient ones. Unlike other methodologies, the TM is used for the full design cycle, from the algorithm to the detailed architecture along with all the detailed enhancements (twin pipe, area efficient and so on) and the fine controls needed for those architectures. There are two types of rules, namely primary and secondary rules. These rules will be described in detail in the next two subsections and used in generating the whole classes of S/S algorithms.

Primary rules: There are four primary transformation rules, two of them (Map and Split) will be used to construct the main multiplication table, while the other two (Fold and Merge) will be used in generating the main families of radix-2^n S/S algorithms, namely the inherent and non-inherent S/S algorithms

Constructing the main multiplication table using map and split: Mapping is a simple rule which maps an equation into a table or map one table into another, while Split rule is used to partition a table cell into smaller cells according to certain rules. In what follows, these two rules will be used in constructing the main multiplication table.

The multiplication of two numbers (U and V) can be written as:

$$P = \sum_{j=0}^{K-1}\sum_{i=0}^{K-1}(u_j \cdot v_i)2^{(j+i)n} \qquad (1)$$

where, u_j and v_i represent the j^{th} and i^{th} digits of U and V, respectively.

To perform the multiplication recursively, the above equation will be modified by introducing a dummy variable, k (where, k = i+j and represents the significance). The new multiplication equation is given by:

$$P = \sum_{k=0}^{2K-2}\sum_{j=0}^{k}(u_j \cdot v_{k-j})2^{kn} \qquad (2)$$

where, u_i and v_i = 0 for $i<0$ and $i>K-1$.

The main multiplication table can be constructed as follows:

- Use Map rule to map the above multiplication equation into a table to obtain Table 1
- The split rule is applied to Table 1 to split it into two cells (Table 2) showing inputs, operation and outputs
- Thirdly, the split rule is applied again to Table 2 to split the computation of the outer sigma into 2K-1 rows as shown in Table 3. Since the index, k, of the outer sigma represents significance, each row in Table 3 represents a different significance where the first row has significance zero and subsequent rows increase in significance until the $(2K-1)^{th}$ row which has significance (2K-2)

Table 1: The main multiplication table

$P=\sum_{k=0}^{2K-2}\sum_{j=0}^{k}(u_j\cdot v_{k-j})2^{kn}$

Table 2: The main multiplication table after the first split

$P=\sum_{k=0}^{2K-2}\sum_{j=0}^{k}(u_j\cdot v_{k-j})2^{kn}$

Table 3: The modified table (splitting Table 2 into 2K-1 rows)

P	
$\sum$	$\sum_{j=0}^{0}u_j\cdot v_{0-j}2^0$
	$\sum_{j=0}^{1}u_j\cdot v_{1-j}2^n$
	.
	.
	.
	$\sum_{j=0}^{2K-2}u_j\cdot v_{2K-2-j}2^{(2K-2)n}$

Table 4: Splitting Table 3 into the weighting of the significance and the inner sigma

P	$*2^{nk}$	
$\sum$	k = 0	
		$\sum_{j=0}^{0}u_j\cdot v_{0-j}$
	K = 1	$\sum_{j=0}^{1}u_j\cdot v_{1-j}$
.	.	.
.	.	.
	K = (2K-2)	$\sum_{j=0}^{2K-2}u_j\cdot v_{2K-2-j}$

- The split rule is applied to Table 3 to split the computation into the weighting of the significance and the inner sigma as shown in Table 4
- Finally, split rule is applied to Table 4 to split each row into K columns so that each cell contains one partial product computation as shown in Table 5

This will allow the computation of the sigma in each row in an iterative manner by distributing the partial products along columns. In Table 5, the index of the operand, U, is mapped to the column index. The following notes is noteworthy:

- This way each column has different digits of V
- Since k = i+j, each column has the same multiplicand, U

Table 5: The modified table (split each row of Table 4 into K columns)

P	$*2^{nk}$				
$\sum$	k = 0				
	$\sum$				
		u_0v_0			
	K = 1	u_0v_1	u_1v_0	-	-
	.	.	.	.	.
	.	.	.	.	.
	.	.	.	.	.
	K = K-1	u_0v_{K-1}	u_1v_{K-2}	u_2v_{K-3}	$u_{K-1}v_0$
			u_1v_{K-1}	u_2v_{K-2}	$\ldots u_{K-1}v_1$
				u_2v_{K-1}	$\ldots$
					$\ldots$
	k = 2K-2				$\ldots u_{K-1}v_{K-1}$

Table 6: The main multiplication Table (for K = 4)

T	P	0	1	2	3	P
0	P_0	u_0v_0				P_0
1	P_1	u_0v_1	u_1v_0			P_1
2	P_2	u_0v_2	u_1v_1	u_2v_0		P_2
3	P_3	u_0v_3	u_1v_2	u_2v_1	u_3v_0	P_3
4	P_4		u_1v_3	u_2v_2	u_3v_1	P_4
5	P_5			u_2v_3	u_3v_2	P_5
6	P_6				u_3v_3	P_6
7	P_7					P_7

- The index of V increases by 1 in each successive row
- The index of V decreases by 1 in each successive column
- Each column has exactly K partial products

As can be seen from Table 5, each row represents a specific significance and the sum accumulated horizontally along the rows in the east/west direction for unidirectional/bidirectional algorithms and the carry can be transferred to any cell in next row (significance). This means that the columns order in Table 5 is insignificant and hence can be interchanged. This is another merit of the TM over the DG.

Table 6 which is a specific case of Table 5 (for K = 4), is the main multiplication table that will be used in the remainder of this study to derive all classes of possible radix-2^n S/S algorithms and architectures.

Designing inherent S/S algorithms by applying the Folding rule: The Folding rule is applied only to cells which are engaged in generating partial products along a row. These cells are termed the active row. In this rule, each active row is folded on its centre. To apply this rule, all rows are scanned and each active row is folded on its centre. For instance, the first row, row 0, has only one active cell, u_0v_0, so no action will be taken. Row 1, has two (even) active cells and their centre lies between them, so the right cell, u_1v_0, will be folded on the left cell, u_0v_1 as shown in Table 7. Row 2, has three (odd) active cells and their centre is the middle cell, u_1v_1. So, the cells on the right of the middle cell, u_2v_0 will be folded on the cells on its left, u_0v_2, as shown in Table 7. The remaining rows will be processed in the same manner.

It is worth noting that the number of columns after applying the Folding rule will not be changed. Furthermore, the Folding rule will ensure that the digits of the two input operands of a multiplication operation have the same temporal-spatial mapping (distribution). These algorithms, where both operands have the same temporal-spatial mapping will result in inherent S/S algorithms. Table 7, will be processed more in later, using the secondary rules) to result in more efficient algorithms.

Designing non-inherent S/S algorithms using merge rule: In merging, the cells along the rows of two adjacent columns are combined together to form one column. If the merge rule is applied to Table 6, it will produce Table 8 (the last two columns have been added to explain an important merit of the TM).

It is clear from Table 8 that the merging process will cause the digits of the two input operands to have different temporal-spatial mappings. Such tables will result in non-inherent S/S algorithms. The reason those algorithms are termed non-inherent S/S algorithms is that they are originally a serial-parallel algorithms where the multiplicand, U, is distributed and latched one significance per column, while the multiplier, V, is propagating through the columns and multiplied by the relevant multiplicand digit. However, it is obvious, from the right most two columns in Table 8, that no more than one new operand from the multiplicand, U and the multiplier, V, is needed at any cycle. This means that the same functionality can be achieved by serial feeding. As a consequence, there is no need for parallel loading the multiplicand, U and hence the name non-inherent S/S algorithms (i.e., serial-parallel algorithms transformed to S/S ones). Table 8, will be processed more in later, (using the secondary rules) to result in more efficient algorithms.

Secondary rules: All algorithms (tables) generated by the previous rules can be processed further to yield a variety of other more efficient multiplication algorithms (tables) by applying seven secondary rules.

The convention in this study is to use temporal mapping for rows and spatial mapping for columns (Although, the reverse can be used as multiplication is commutative). Accordingly, seven secondary rules are developed and used in this study.

Shift time forward: Since different rows represent different time cycles, by shifting contents of cells along a column down implies shifting the content forward by one cycle in time. The shift time forward can be performed in either of the following two ways:

- In a recursive way: Where at each stage, the first column will be fixed and the rest are shifted down one step (time unit). The output of each stage, the shifted block, will be processed in the same manner
- Order way: Each column will be shifted down according to its order. For example the first column which has order zero will not be shifted, the second column will be shifted by one step and so on

Systolise: This rule is similar to the previous rule, except that it is applied to every other column instead of each column. In this rule, column2j (where j represents the column index and ($0 \le j \le K/2-1$)) will be fixed and all columns on its right are shifted down one step. This rule will be used to systolise bi-directional structures and hence the name of the rule.

Shift time backward: As different rows represent different time cycles, shifting contents of cells along a column up implies shifting the content backward by one cycle in time. The shift time backward can be performed following the same procedure used for Shift time forward, with replacing a shift down action by a shift up one.

Table 7: General inherent S/S algorithm

T	P	0	1	2	3	P
0	p_0	$u_0\ v_0$				p_0
1	p_1	$u_0v_1+u_1v_0$				p_1
2	p_2	$u_0v_2+u_2v_0$	u_1v_1			p_2
3	p_3	$u_0v_3+u_3v_0$	$u_1v_2+u_2v_1$			p_3
4	p_4		$u_1v_3+u_3v_1$	u_2v_2		p_4
5	p_5			$u_2v_3+u_3v_2$		p_5
6	p_6				u_3v_3	p_6
7	p_7					p_7

Table 8: Generic non-inherent S/S algorithm

T	P	0	1	P	New u	New v
0	p_0	$u_0\ v_0$		p_0	u_0	v_0
1	p_1	$u_0v_1+u_1v_0$		p_1	u_1	v_1
2	p_2	$u_0v_2+u_1v_1$	$u_2\ v_0$	p_2	u_2	v_2
3	p_3	$u_0\ v_3+u_1v_2$	$u_2v_1+u_3v_0$	p_3	u_3	v_3
4	p_4	u_1v_3	$u_2v_2+u_3v_1$	p_4	-	-
5	p_5		$u_2v_3+u_3v_2$	p_5	-	-
6	p_6		u_3v_3	p_6	-	-
7	p_7			p_7	-	-

It is significant to make the following notes about the above secondary rules:

- The shift time rules can be interpreted in terms of adding or removing delay elements from data path of multiplication structures
- When data paths move in the same direction (unidirectional algorithms), the Shift Time rules are equivalent to the unidirectional cut set rules (Kung, 1988)
- When data paths move in opposite directions (bidirectional algorithms), the Shift Time rules are equivalent to the bidirectional cut-set rules (Kung, 1988)

Cell duplicate: This rule duplicates the contents of a group of cells to another empty ones. As will be seen in later, this rule is used to generate twin pipe algorithms. Twin pipe algorithms are simply derived by taking a copy of any active cells and pasting them into inactive cells to duplicate the same computation. If a new computation is required at every l cycles (where l = the maximum number of cycles in columns engaged in generating partial products), the cells that correspond to this computation are pasted at the l^{th} cycle. In twin pipe algorithms a new result will be available every l cycles as opposed to classical one pipe structures where 2 K cycles are needed to complete the computation:

Slide back: This rule is used in deriving area efficient algorithms, where a ~50% silicon saving is achieved. To apply this rule the following conditions have to be satisfied:

- There should be K columns in the original algorithm
- Whenever a cell in the region K/2 to K-1 become busy in generating partial products, a cell or more in the region 1 to K /2 in the same significance line should be free to generate the same number of partial products

As a consequence, the region K/2 to K-1 can be shifted back by K/2 positions along the significance line and the $(K/2)^{th}$ columns and above is accommodated by columns 0 to K/2-1 at the last K cycles. In this way a full utilization of columns 0 to K/2-1 can be achieved and ~50% of area will be saved. However, for this technique to work, the K/2 most significant digits need to be made available to the algorithm again at the last K cycles, which can be achieved, systematically as well, by applying the next rule (Data-feedback).

Data-feedback: This rule is applied, only, to algorithms (tables) after applying the Slide back rule. For this rule to be applied the following condition has to be satisfied: (The data needed by the last K/2 columns (BCs) have to be available at column K/2-1, which is the last active column (A column is active if it is performing some computation) after applying the slide back rule, at cycle $k \leq K+1$).

As will be seen later, to apply this rule, the data will be fed back when it reaches the K/2 column; either by broadcasting if it arrives at cycle k = K+1 or through some delay elements if it arrives at cycle k<K+1.

Split-and-shift: This rule is used to gain more efficient implementation, where each partial product, which is a 2n-bit number, is divided into least significant n-bit number, LSD and most significant n-bit number, MSD. To carry out the radix-2^n computation, it is necessary that either the LSD or the MSD are moved from cell (k,j) to cell (k-1, j) or (k+1, j), respectively, where k denotes rows and j denotes columns.

It is worth mentioning that the first six rules bring efficiency at the array level, while the seventh rule brings the efficiency at the BC level.

More efficient inherent S/S algorithms: Table 7 can be synthesized into two S/S classes of architectures, namely unidirectional and bidirectional depending on the accumulation direction. It is clear that the synthesized architectures will not be efficient as the accumulation path (critical path) is too long (spanning the whole row). However, Shift time forward/backward rules can be applied Table 7 to generate another two algorithms, which can be synthesized to more efficient S/S classes of architectures. The reason is that the accumulation path will be localized and reduced to only one column at a time.

Table 9 is the result of applying Shift time forward rule and represents a family of fully systolic unidirectional algorithms. On the other side, Table 10 is the result of applying Shift time backward rule and represents a family of semi systolic bidirectional algorithms which can be made fully systolic by applying Systolise rule.

Table 9: Systolic unidirectional inherent S/S algorithm

T	0	1	2	3	P
0	$u_0 v_0$				
1	$u_0v_1+u_1v_0$				
2	$u_0v_2+u_2v_0$				
3	$u_0v_3+u_3v_0$	u_1v_1			p_0
4		$u_1v_2+u_2v_1$			p_1
5		$u_1v_3+u_3v_1$			p_2
6			u_2v_2		p_3
7			$u_2v_3+u_3v_2$		p_4
8					p_5
9				u_3v_3	p_6
10					p_7

Table 10: Semi systolic bidirectional inherent S/S algorithm

T	P	0	1	2	3
0	p_0	$u_0\ v_0$			
1	p_1	$u_0v_1+u_1v_0$	u_1v_1		
2	p_2	$u_0v_2+u_2v_0$	$u_1v_2+u_2v_1$	u_2v_2	
3	p_3	$u_0v_3+u_3v_0$	$u_1v_3+u_3v_1$	$u_2v_3+u_3v_2$	u_3v_3
4	p_4				
5	p_5				
6	p_6				
7	p_7				

Table 11: General twin pipe inherent S/S algorithm

T	P_1	P_2	0	1	2	3	P_1	P_2
0	p_0		$u_0\ v_0$				p_0	
1	p_1		$u_0v_1+u_1v_0$				p_1	
2	p_2		$u_0v_2+u_2v_0$	u_1v_1			p_2	
3	p_3		$u_0v_3+u_3v_0$	$u_1v_2+u_2v_1$			p_3	
4	p_0	p_4	$u_0\ v_0$	$u_1v_3+u_3v_1$	u_2v_2		p_0	p_4
5	p_1	p_5	$u_0v_1+u_1v_0$		$u_2v_3+u_3v_2$		p_1	p_5
6	p_2	p_6	$u_0v_2+u_2v_0$	u_1v_1		u_3v_3	p_2	p_6
7	p_3	p_7	$u_0v_3+u_3v_0$	$u_1v_2+u_2v_1$			p_3	p_7
8		p_4		$u_1v_3+u_3v_1$	u_2v_2			p_4
9		p_5			$u_2v_3+u_3v_2$			p_5
10		p_6				u_3v_3		p_6
11		p_7						p_7

Table 12: General Area efficient inherent S/S algorithm

T	P	0	1	P
0	p_0	$u_0\ v_0$		p_0
1	p_1	$u_0v_1+u_1v_0$		p_1
2	p_2	$u_0v_2+u_2v_0$	u_1v_1	p_2
3	p_3	$u_0v_3+u_3v_0$	$u_1v_2+u_2v_1$	p_3
4	p_4	u_2v_2	$u_1v_3+u_3v_1$	p_4
5	p_5	$u_2v_3+u_3v_2$		p_5
6	p_6		u_3v_3	p_6
7	p_7			p_7

In addition, it is clear from Table 7 that the area utilization of the columns decreased linearly as we move from cell 0 to cell K-1. In the 2nd K cycles the grey cells are used, only, in propagating the results (in bidirectional algorithms) or staying idle (in the unidirectional ones).

To maximize the efficiency of these algorithms, two techniques, namely the twin piping and the cell re-mapping (area efficiency) can be derived systematically from the inherent S/S algorithm (Table 7). These two techniques are derived by applying Cell Duplicate and Slide back rules.

Table 11, which is a generic twin pipe algorithm, is the result of applying Cell Duplicate rule. Furthermore, Table 11can be modified in the same systematic way described previously, by applying the Shift time forward/ backward rules, to yield systolic unidirectional and semi systolic bidirectional twin pipe algorithms respectively. Furthermore, the semi systolic algorithms can be made fully systolic by applying the Systolise rule.

Table 13: Systolic unidirectional non-inherent S/S algorithm

T	0	1	P	New u	New v
0	$u_0\ v_0$			u_0	v_0
1	$u_0v_1+u_1v_0$		p_0	u_1	v_1
2	$u_0v_2+u_1v_1$		p_1	-	v_2
3	$u_0\ v_3+u_1v_2$	$u_2\ v_0$	p_2	u_2	v_3
4	u_1v_3	$u_2v_1+u_3v_0$	p_3	u_3	-
5		$u_2v_2+u_3v_1$	p_4	-	-
6		$u_2v_3+u_3v_2$	p_5	-	-
7		u_3v_3	p_6	-	-
8			p_7	-	-

The second technique, cell re-mapping, can be achieved by applying Slide back rule. It can be seen from Table 7 that the conditions of Slide back are satisfied; so this rule can be applied to Table 7 to generate a generic area efficient algorithm (Table 12).

As mentioned early, for this technique to work, the K/2 most significant digits need to be made available to the algorithm again at the last K cycles. Furthermore, instead of modifying the interface in an ad hoc manner (by using shift registers at the front end of the structures derived from this algorithm, which is the case in the non scalable area efficient structures reported in the open literature (Aggoun *et al*., 2004) the Data-feedback rule can be applied. It is worth mentioning that area efficient algorithms derived using the proposed TM have three merits. Firstly, they are developed in a systematic way; secondly they preserve the true scalability of the original algorithm as they avoid the use of word length dependent elements; and finally they ended up using less silicon compared with area efficient algorithms which have been designed in an ad hoc manner. The area efficient algorithm, Table 12, can then be modified in the same systematic way described previously, by applying the Shift time forward/backward rules, to yield systolic unidirectional and semi systolic bidirectional area efficient algorithms.

Furthermore, the semi systolic algorithms can be made fully systolic by applying the Systolise rule.

More efficient non-inherent S/S algorithms: Table 8, can be synthesized into two S/S classes of architectures (unidirectional and bidirectional) depending on the accumulation direction. It should be noted that the synthesized architectures will not be efficient as the accumulation path is too long (spanning the whole row).

However, Shift time forward/backward rules, can be applied to Table 8 to generate more efficient S/S algorithms. The reason is that the accumulation path will be localized and reduced to only one column at a time. Table 13 is the result of applying shift time forward rule and represents a family of fully systolic unidirectional algorithms. On the other side, Table 14 is the result of applying Shift time backward rule and represents a family of semi systolic bidirectional algorithms which can be made fully systolic by applying the Systolise rule.

Table 14: Semi systolic bidirectional non-inherent S/S algorithm

T	P	0	1	New u	New v
0	p_0	$u_0 v_0$		u_0	v_0
1	p_1	$u_0v_1+u_1v_0$	$u_2 v_0$	$u_1 u_2$	v_1
2	p_2	$u_0v_2+u_1v_1$	$u_2v_1+u_3v_0$	u_3	v_2
3	p_3	$u_0 v_3+u_1v_2$	$u_2v_2+u_3v_1$	-	v_3
4	p_4	u_1v_3	$u_2v_3+u_3v_2$	-	-
5	p_5		u_3v_3	-	-
6	p_6			-	-
7	p_7			-	-

It should be noted that, for the bidirectional semi systolic algorithm (Table 14) two operands from U are required at some cycles; so for this algorithm to work in a S/S fashion, two operands of U have to be fed at each cycle. It is worth pointing out that Table 14 is the essence of the bit serial-parallel algorithm in (Ait-Boudaoud *et al*., 1991) and the digit serial-parallel algorithm of (Ashur *et al*., 1996). This demonstrates clearly the power of the TM, which proves that there is no need to make all the bits/digits of the multiplicand, U, available (by parallel loading) at each cycle, thus noticeable saving in I/O pins, area and energy will be gained.

Moreover, following the same procedure described early, Table 8 can be made more efficient by applying cell duplicate rule to produce a twin pipe algorithm. This generic twin pipe algorithm can be modified in the same systematic way described early by applying Shift time forward/backward rules, to yield systolic unidirectional and semi systolic bidirectional twin pipe algorithms. Furthermore, the semi systolic bidirectional twin pipe algorithms can be made fully systolic by applying the Systolise rule.

It should be mentioned that area efficient algorithms cannot be generated from non-inherent S/S algorithms as they do not satisfy the conditions of the Slide back rule. The reason is that the number of columns is already halved by the merging process and no more scope for any further remapping. This is another merit of the TM where, at the table level and before any synthesis, it shows the maximum level of efficiency that can be gained from structures generated from a specific algorithm. This will definitely save time and efforts and shorten the time to market period.

As a final remark, all algorithms can be made more efficient, at the BC level, by applying the Split-and-Shift rule.

RESULTS

Detailed case study: New high performance scalable non-inherent unidirectional radix-2^n SSMs: The TM is utilized, here, in designing a new unidirectional radix-2^n SSM., Table 15 is the result of applying the Split-and-Shift rule to the systolic unidirectional algorithm (Table 13).

Table 15: Systolic uni-non-inherent radix-2^n S/S algorithm

T	BC 0	BC 1	P
0	$(u_0 v_0)_L$		
1	$(u_0 v_0)_H + (u_0v_1+u_1v_0)_L$		
2	$(u_0v_1+u_1v_0)_H + (u_0v_2+u_1v_1)_L$		
3	$(u_0v_2+u_1v_1)_H + (u_0 v_3+u_1v_2)_L$	$(u_2 v_0)_L$	p_0
4	$(u_0 v_3+u_1v_2)_H + (u_1v_3)_L$	$(u_2 v_0)_H + (u_2v_1+u_3v_0)_L$	p_1
5	$(u_1v_3)_H$	$(u_2v_1+u_3v_0)_H + (u_2v_2+u_3v_1)_L$	p_2
6		$(u_2v_2+u_3v_1)_H + (u_2v_3+u_3v_2)_L$	p_3
7		$(u_2v_3+u_3v_2)_H + (u_3v_3)_L$	p_4
8		$(u_3v_3)_H$	p_5
9			p_6
10			p_7

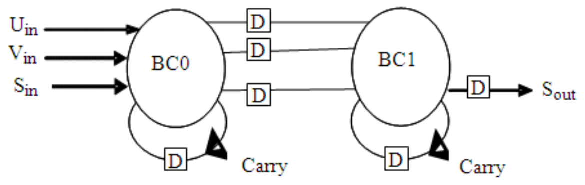

Fig. 2: Non-inherent systolic unidirectional radix-2^n S/S architecture

Figure 2 shows the unidirectional digit SSM synthesized from Table 15. It consists of K/2 BCs, which corresponds to the mapping of all cells (i, j) with i≥j onto a single BC j, where j = 0, 1, 2,…,(K/2-1).

Due to the chosen mapping, it can be seen that the carry digit, C_{out}, generated by BC, j, is delayed then fed back to the same BC. The multiplicand U pass through K holding latches, where each one is devoted to a specific significance by the use of specific control signals, Con. However, the multiplier, V, propagate through a set of delay elements. A BC, j, start by computing the product $u_{2j}v_{i-3j}$ during cycle i = 3j and then the terms $[u_{2j}v_{i-3j}+u_{2j+1}v_{i-3j-1}]$ are computed during the next K cycles. The terms computed in BC j will be added to an accumulating result from the BC, j-1 and then propagated in the next cycle to the BC, j+1. Following the above discussion, a BC, j, should contain two n-bit multipliers, accumulators and latches controlled by a control signal, Con, to allow storage of the input data u_{2j} and u_{2j+1}. The BC is shown in Fig. 3 and the multiplier structure is depicted in Fig. 4. Carry save arithmetic implemented using n-bit 4-2 compressors is used for the accumulation of the partial results to reduce the hardware cost (Almiladi and Ibrahim, 2009). The BC is subdivided into two stages working in a pipeline manner. The first stage deals with the multiplication process of the relevant data, while the second stage performs the addition of the result of the first stage with the shifted result from the neighbor BC on the left.

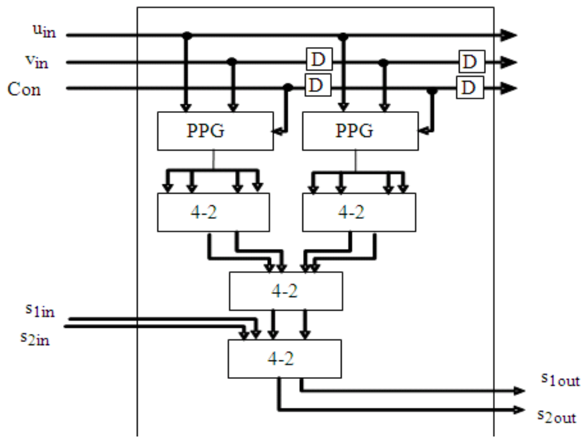

Fig. 3: BC of the non-inherent unidirectional radix-2^n SSM

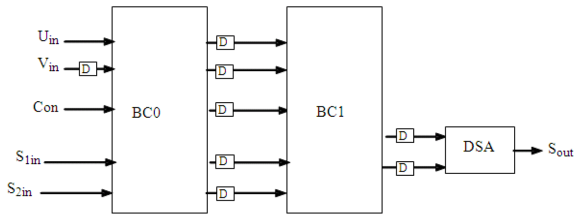

Fig. 4: A non-inherent systolic unidirectional radix-2^n SSM

The first part of the first stage of the BC is implemented using two PPGs and $\log_2(n)$ rows of n-bit 4-2 compressors arranged in a tree type structure as shown in Fig. 3 for n = 4 and can be generalized for any digit-size. The carry digit, C_{out}, consist of the MSDs of the two partial products, $u_{2j}v_{i-3j}$ and $u_{2j+1}v_{i-3j-1}$ and all the carry bits generated by the n-bit 4-2 compressors. The carry digit, C_{out}, is delayed and then fed back into the same BC as shown in Fig. 2. The delay and feed back to the same BC, j, of the MSDs of all the partial products, $u_{2j}v_{i-3j}$ and $u_{2j+1}v_{i-3j-1}$, is carried out by delaying these MSDs and adding them to the LSDs of the same partial products generated in the following cycle. This is achieved by delaying the input data, v_{i--3j} during cycles i = 3j and v_{i--3j} and v_{i-3j-1}, (during the next K cycles) prior to their multiplication with the stored data u_{2j} and u_{2j+1} (Almiladi and Ibrahim, 2009). This results in the reduction of the number of delay elements from n(n-1)/2 to (n-1) per PPG. This is a total of K (n-1)(n-2) reduction in the total number of delay elements. The partial product generated by the first stage is shifted to the second stage, which consists of an n-bit 4-2 compressor. The second stage is used to add the carry and sum word from the first stage to the partial sums, S_{1in} and S_{2in} transferred from the neighboring BC on the left. The output of the second stage is shifted to the next BC on the right and the process is repeated. As shown in Fig. 4, a final adder is needed to sum the two digits, S_{1out} and S_{2out} produced by the last BC on the right hand side of the multiplier to obtain the correct result.

DISCUSSION

The case study presented in the previous section showed the power of the TM in designing new efficient serial multipliers in an easy way. Before the TM tables where used to prove functionality but not designing architectures. Now tables are used for the full design cycle, from the algorithm to the detailed architecture along with all the detailed enhancements (twin pipe, area efficient and so on) and the fine control needed for those architecture as proved in the previous section. The merit of TM over other methodologies comes from the fact that tables carry more information. In addition, the information contained in the tables are explicit and leads to straightforward algorithms derivation and architectures implementation.

CONCLUSION

In this study a new design methodology for the design of radix-2^n SSMs is presented. The methodology is a table based one, where the radix-2n arithmetic, as well as two primary rules, is used to construct the main multiplication table. Another two primary rules are then applied to the main table to derive the main two families of multiplication algorithms namely, inherent S/S algorithms and non-inherent S/S algorithms. The resulting algorithms are processed further by applying other seven secondary rules to yield various more efficient S/S multiplication algorithms. Also, a detailed case study of using the TM in designing new highly regular, modular and scalable non-inherent digit SSMs is presented. This design proves the power of TM in designating efficient multipliers in an easy and systematic way.

REFERENCES

Aggoun, A., M.K. Ibrahim and A. Ashur, 1998a. Design methodology for sub-digit pipelined digit-serial IIR filters. J. Sig. Process., 68: 73-86. DOI: 10.1016/S0165-1684(98)00058-9

Aggoun, A., M.K. Ibrahim and A. Ashur, 1998b. Bit-level pipelined digit-serial array processors. IEEE Trans. Circ. Syst., 45: 857-868. DOI: 10.1109/82.700933

Aggoun, A., A.F. Farwan, M.K. Ibrahim and A. Ashur, 2004. Radix-2^n serial-serial multipliers. IEE Proc. Circ. Devices Syst., 151: 503-509. DOI: 10.1049/ip-cds:20040412

Ait-Boudaoud, D., M.K. Ibrahim and B.R. Hayesgill, 1991. Novel cell architecture for bit level systolic arrays multiplication. IEE Proc. E, 138: 21-26.

Almiladi, A. and M.K. Ibrahim, 2009. High performance scalable radix-2^n Gf(2n) serial-serial multipliers. J. Circ. Syst. Comput., 18: 11-30. DOI: 10.1142/S0218126609004892

Ashur, A., A. Aggoun and M.K. Ibrahim, 1996. Systolic digit-serial multiplier. IEE Proc. Circ. Devi. Syst., 143: 14-20.

Kung, S.Y., 1988. VLSI Array Processors. Prentice-Hall, ISBN: 13: 978-0139427497, pp: 600.

Smith, S., M. McGregor and P. Denyer, 1987. Techniques to increase the computational throughput of bit-serial architectures. IEEE Int. Conf. Acoust. Speech Sign. Process., 12: 543-546.

Wu, C.W. and P.R. Cappello, 1989. Block multipliers unify bit-level cellular multiplications. Int. J. Comp. Aid. VLSI Des., 1: 113-125.