Proceedings of International Conference
on Computer Architecture

Parallel and Pipelined VLSI Architecture Designs for Distributed Arithmetic-Based Recursive Digital Filters[1]

Yin-Tsung Hwang and Ching-Long Su

Department of Electronic Engineering
National Yunlin Institute of Technology
Touliu, Yunlin, Taiwan, R.O.C.
E-mail: hwangyt@cad.el.yuntech.edu.tw

Abstract

In this paper we first propose a new design technique based on Distributed Arithmetic (DA) scheme to enhance high speed recursive digital filtering. The proposed design exploits parallelism down to the bit-level and can outperform the conventional bit parallel design in both speed and hardware complexity. The scheme is further improved with the introduction of algorithm look-ahead transform and design tactics such as structure pipelining and block processing. The resultant design features an initiation interval as small as the delay for computing one data bit.

1. Introduction and related work

In real time DSP applications, systems are required to process continuous data stream promptly so that no arriving data will be lost. Initiation interval [1], which indicates the separation period between the processing of two successive data, is therefore considered as an important index to measure the real time performance. While conventional software programming on general purpose DSP processors fails to meet the speed requirements, dedicated VLSI design has been considered as an effective method to realize the DSP algorithms with intensive computations. To enhance the computing speed, various design techniques such as parallel processing and pipelinig have been widely employed to shorten the initiation interval. These design tactics, however, cannot be applied arbitrarily to the recursive computing systems where a minimum initiation interval must be maintained to observe the data dependence constraints. In the conventional bit-parallel design scheme, all bits in a data word are processed atomically. Therefore, the minimum initiation interval can at best be the delay of the largest bit-parallel operation in one recursion. In most cases, this corresponds to a bit-parallel multiplication. In this paper, we propose a distributed arithmetic (DA) based design scheme to exploit the parallelism in recursive DSP computing down to the bit level. Distributed arithmetic [2] is basically a bit-serial word-parallel approach. Instead of waiting for the completion of entire data word, we can initiate the following recusrion's computation as soon as partial data bits of the current recursion are derived. The initiation interval can thus be greatly reduced with minimum hardware overhead. Various research on digital recursive filters have been proposed. In bit-parallel design approach, the linear systolic array in [3] can achieve an initiation interval two times the total delay of a parallel multiplier plus two carry propagation adders. The best known result is given in [4] which has an initiation interval equal to the delay of two parallel multipliers. In [5], scattered look-ahead transform is applied first to create more algorithm parallelism. Computing concurrency is then implemented via design techniques such as pipelining or block processing. In

[1]This research is supported by NSC grant 86-2213-E-224-017

bit-serial design approach, C.-W. Wu et al. [6] employ look-ahead transform as well and derive a bit-level systolic array architecture for a block pipelined second-order IIR filter. Based on a new bit-level inner product computing scheme, C.-L. Wang et al. [7] also propose several bit-level systolic array structures for FIR and IIR filters.

2. ARMA filter and the bit-parallel design scheme

ARMA (Auto Regressive Moving Average) filter is a typical recursive DSP algorithm widely used in applications like spectrum estimation. Throughout this paper, we will use it as an example to demonstrate our design scheme. The transfer function of an ARMA filter can be described as

$$H(z) = \frac{\sum_{k=1}^{q} b_k z^{-k}}{1 - \sum_{k=1}^{p} a_k z^{-k}}. \qquad (1)$$

It consists of a non-recursive MA filtering and a recursive AR filtering. The MA filter has the difference equation as

$$u(j) = \sum_{k=1}^{q} b_k x(j-k) \qquad (2)$$

The AR (or IIR) filter has the difference equation as

$$y(j) = \sum_{k=1}^{p} a_k y(j-k) + u(j) \qquad (3)$$

The resultant difference equation for the ARMA filter becomes

$$y(n) = \sum_{k=1}^{p} a_k y(n-k) + \sum_{k=1}^{q} b_k x(n-k). \qquad (4)$$

The realization of the ARMA filter can be achieved by cascading an AR filter with an MA filter. Note that both AR and MA sections can be characterized by an inner product operation.

3. Distributed arithmetic based ARMA filter designs

Distributed arithmetic structure has been found useful in the cases such as vector analysis, inner product and DCT (Discrete Cosine Transform) [8]. Contrary to the lumped computing in bit-parallel design, each data word is now accessed bit by bit, from the LSB to the MSB. Equal-weighted bits of multiple data words, however, are processed and accumulated simultaneously to achieve the computing concurrency. Multiple partial products, one from each data bit, are now to be accumulated. This differs from the bit-parallel designs which produce no result at all until the last accumulation cycle is done. In DA's approach, the initiation interval can be greatly reduced in that the next recursion's computing can begin as soon as the current recursion finishes one bit's computing. We will next use a four-tap MA filter to illustrate the design scheme. The design for the AR part can be similarly derived. By assuming the word length n of $x(i)$ equal to 8, Eq. (2) can be rewritten as

$$y(j) = \sum_{i=0}^{7} [b_1 \cdot x_{j-1}^{i} + b_2 \cdot x_{j-2}^{i} + b_3 \cdot x_{j-3}^{i} + b_4 \cdot x_{j-4}^{i}] \cdot 2^{i} \qquad (5)$$

The inner product is obtained by accumulating n shifted partial sums. Each partial sum is the accumulation of partial products from the bit i of all data words. Four data words are now processed concurrently while the data bits within the same word are accessed sequentially. Contrary to the conventional ROM implementation, we use a multi-operand carry save adder (MCSA) network for faster generation of partial sum. Multi-operand adder is used to reduce the levels of summation. The carry save structure eliminates the carry propagation process in each accumulation and defer it to the final stage. The scheme can be graphically illustrated by Figure 1. To avoid the carry propagation in each accumulation step, the intermediate result is split into three bit vectors (one for sum and two for carries). Together with the four partial products, there are seven operands to be added in each bit column and a 3-bit binary is generated. This leads to a 7-to-3 MCSA. If the data width of the filter coefficient b_k's is equal to m, a total of m 7-to-3 MCSAs plus three m-bit latches are required for computing one iteration of

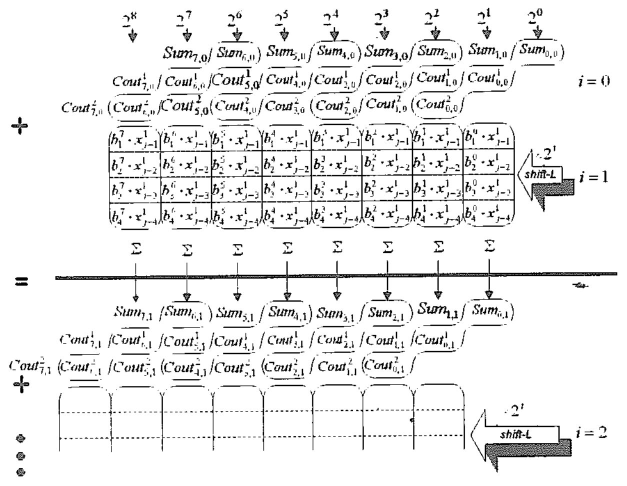
Figure 1: Carry save addition for DA based inner product operation

$y(j) = \sum b_i \cdot x(j-k)$. This computing scheme can be realized by a bi-directional systolic array consisting of m MCSAs followed by a rear processor as shown in Figure 2. The m MCSAs are reused n times to complete one inner product operation and generate a temporary result in three binary numbers. The final result is obtained by combining three binaries into one by the rear processor. Since all data are accessed bit serially in the DA scheme, there is no need to perform bit-parallel summation and the rear processor is simply a 5-to-3 MCSA. The critical path of the design lies in the MCSA with a delay of three cascaded full adders.

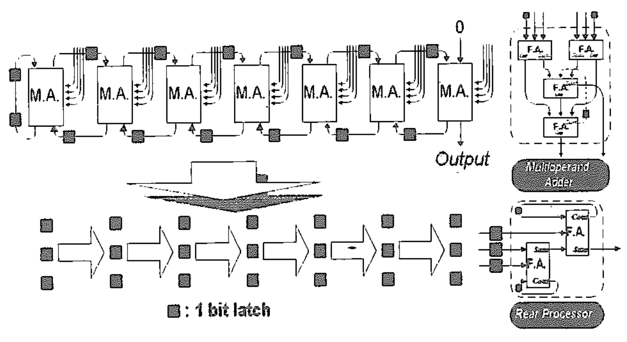

Figure 2. Basic DA computing module for MA filter

The derived systolic array is considered as a basic DA module which performs pipelined inner product operation and can be readily adapted to MA or AR filtering. Assume each time step is equal to the delay of an MCSA. For each computing recursion, it takes n time steps to finish the partial product accumulation in the bi-directional array and another n time steps to obtain all the bits of the final result from the rear processor. If only one such DA module is employed, we may obtain a two-stage macro-pipelining by overlapping the computations of two successive recursions between the bi-directional array and the rear processor. The initiation interval for both MA and AR filters is thus equal to n times the delay of an MCSA. Combining MA and AR filters, the resultant ARMA filter design is shown in Figure 3.

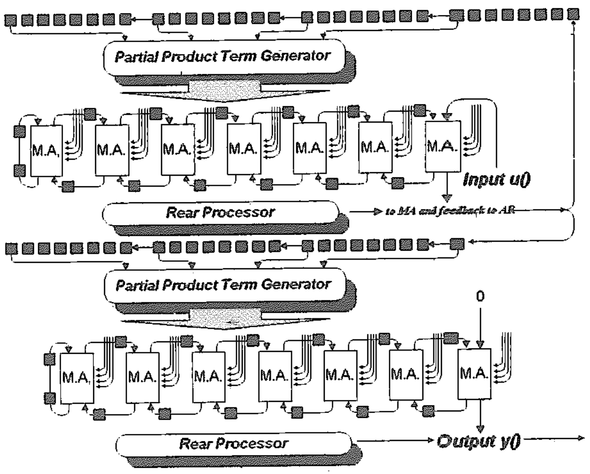

Figure 3. ARMA filter design with single DA computing module

To achieve higher computing rate, multiple DA modules can be employed to handle multiple output data recursions concurrently. The computations of successive output data words are thus overlapped and separated by a delay equal to the initiation interval. As mentioned, the next recursion's computation can be initiated as soon as the first bit of the current recursion becomes available. The minimum initiation interval is equal to an MCSA delay provided n DA modules are used. Clearly, there is a trade-off between the initiation interval and the hardware complexity. In the ARMA filter case, three factors will affect the final design, i.e.

the word length of input data, the word length of filter coefficients and the tap order. Tap order m determines the size of the MCSA. The input word length n determines how many rows of MCSA array are needed. The coefficient word length r will affect the width of each MCSA array. Figure 4 shows a filter design with parameter $(n,r,m) = (3,8,4)$.

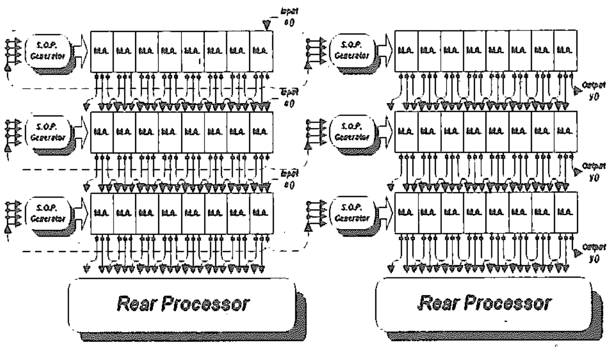

Figure 4: Word-overlap DA ARMA filter design

In this design, each row of the MCSA array is responsible for computing one particular bit for all computing recursions. Final result is obtained from the rear processor in the form of n-bit binary per time step.

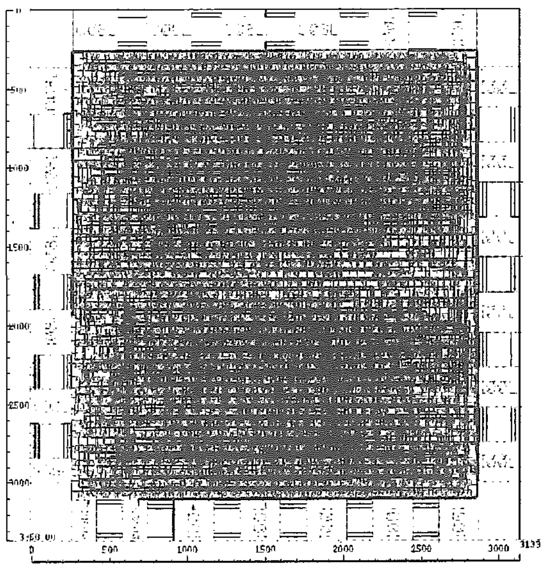

Figure 5: Word-overlap DA ARMA filter layout view

Each data bit, nonetheless, corresponds to the results of different computing recursions. The readers are referred to [9] for more circuit details. To evaluate the effectiveness of the proposed scheme, the proposed ARMA filter is implemented in 0.8 μm single poly double metal (SPDM) CMOS technology. We choose the design parameters $(n,r,m) = (8,8,4)$ and employ the maximum number of DA modules for the highest throughput rate operation. The chip layout using a standard cell design is shown in Figure 5. The die size is 3404.7 by 3097 um^2 and the HDL simulation with delay parameters extracted from the layout shows the design can operate at a speed as high as 142.8 MHz.

4. Algorithm look-ahead transformation

The proposed DA scheme can be further improved if look ahead transform is incorporated. Look-ahead transform creates computing parallelism by increasing the dependence distance between two recurrent computings. Consider a 2-tap DA ARMA filter with difference equation as follows:

$$y(n) = \sum_{k=1}^{2} a_k y(n-k) + \sum_{k=1}^{2} b_k x(n-k) \tag{6}$$

By applying the look-ahead transform twice, i.e. 2-step clustered look-ahead transform, we have

$$\begin{cases} y(n) = (a_1^3 + 2a_1a_2)y(n-3) + (a_1^2a_2 + a_2^2)y(n-4) \\ \qquad + u(n) & \text{(7a)} \\ u(n) = b_1x(n-1) + (a_1b_1 + b_2)x(n-2) + (a_1^2b_1 + a_1b_2 + \\ \qquad a_2b_1)\cdot x(n-3) + (a_1^2b_2 + a_2b_2)x(n-4) & \text{(7b)} \end{cases}$$

The dependence distance in the AR section is increased from 1 to 3 after the look ahead transform. Consequently, three independent computing threads can be achieved. Note that the number of product terms in the AR part (Eq 7a) remains the same after look-ahead transform. This means the effective initiation interval can now be reduced by a factor of 3. The number of product terms in the MA part (Eq 7b), nonetheless, will increase linearly with the look-ahead steps. Since they are not part of the recursive loop, they can always be speeded up at the cost of extra hardware complexity. After look-ahead transform, two design techniques can be applied, i.e., *structure pipelining* and *block processing* [10].

5. Structure pipelining and block processing for DA ARMA filter

In structure pipelining, increased dependence distance after look ahead transform means a finer grained pipelining. Different computing threads are then processed in a pipelined interleaving manner. Assume the dependence distance is 3 after look ahead transform. Refer to Figure 6, since MCSA is the most time consuming processing element in the design, we may apply the retiming procedure and divide each MCSA into 3 stages. The clock period, as well as the initiation interval, can now be reduced to about one third. The speed up is almost equal to three if we neglect the setup and hold time delays of the latches. The incurred hardware overhead in this case is relatively small. Note that the MA part is enlarged from 5-to-3 MCSAs to the 7-to-3 MCSAs. The final timing diagram of the word-overlap DA is given in Figure 7. In block processing, duplicated hardware and the number of the computing multiple computing threads are processed separately in threads is referred as the block size. For a size L block processing, a block of L input samples can be processed concurrently and generate a block of L output samples. Consider the same 2-tap 2-step look-ahead ARMA filter example, thr differenvr equation in Eq (6) can be rewritten as

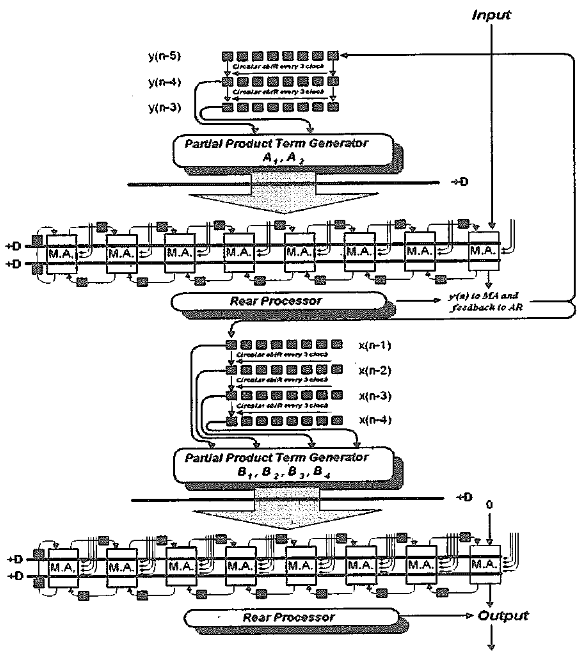
Input
y(n-5)
y(n-4)
y(n-3)
Partial Product Term Generator A_1, A_2
+D
M.A.
Rear Processor
y(n) to MA and feedback to AR
x(n-1)
x(n-2)
x(n-3)
x(n-4)
Partial Product Term Generator B_1, B_2, B_3, B_4
0
Output

Figure 6 : The design of look-ahead DA ARMA filter after cut-set retiming

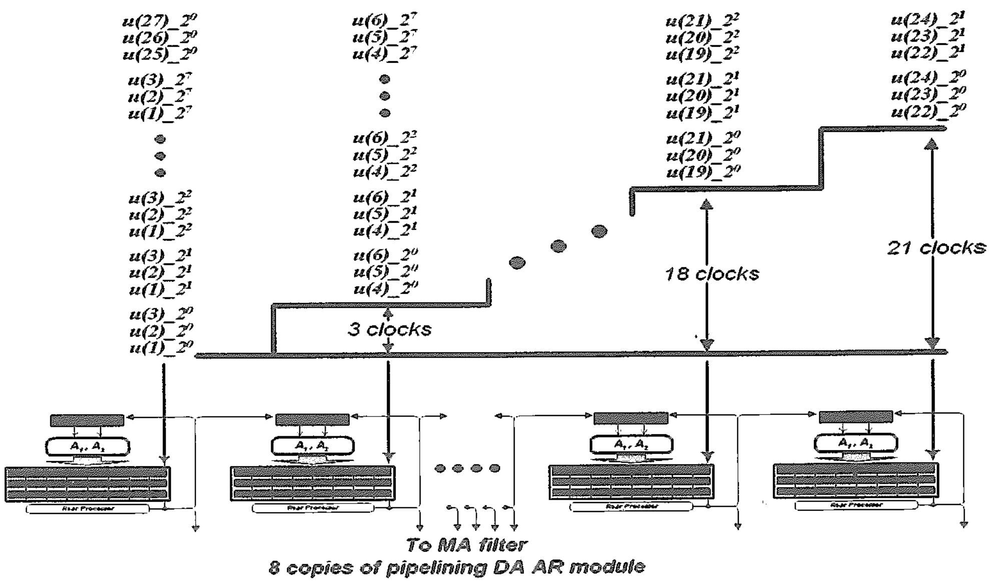
$u(27)\_2^0$ $u(26)\_2^0$ $u(25)\_2^0$
$u(3)\_2^7$ $u(2)\_2^7$ $u(1)\_2^7$
$u(3)\_2^2$ $u(2)\_2^2$ $u(1)\_2^2$
$u(3)\_2^1$ $u(2)\_2^1$ $u(1)\_2^1$
$u(3)\_2^0$ $u(2)\_2^0$ $u(1)\_2^0$
$u(6)\_2^7$ $u(5)\_2^7$ $u(4)\_2^7$
$u(6)\_2^2$ $u(5)\_2^2$ $u(4)\_2^2$
$u(6)\_2^1$ $u(5)\_2^1$ $u(4)\_2^1$
$u(6)\_2^0$ $u(5)\_2^0$ $u(4)\_2^0$
$u(21)\_2^2$ $u(20)\_2^2$ $u(19)\_2^2$
$u(21)\_2^1$ $u(20)\_2^1$ $u(19)\_2^1$
$u(21)\_2^0$ $u(20)\_2^0$ $u(19)\_2^0$
$u(24)\_2^1$ $u(23)\_2^1$ $u(22)\_2^1$
$u(24)\_2^0$ $u(23)\_2^0$ $u(22)\_2^0$
3 clocks
18 clocks
21 clocks
A_1, A_2
To MA filter
8 copies of pipelining DA AR module

Figure 7 : Timing diagram of a 3-stage structure pipelining word-overlap AR filter

$$\begin{cases} y(3k) = A_1 y(3k-3) + A_2 y(3k-4) + u(3k) \\ u(3k) = B_1 x(3k-1) + B_2 x(3k-2) + B_3 x(3k-3) + B_4 x(3k-4) \end{cases}$$
$$\begin{cases} y(3k+1) = A_1 y(3k-2) + A_2 y(3k-3) + u(3k+1) \\ u(3k+1) = B_1 x(3k) + B_2 x(3k-1) + B_3 x(3k-2) + B_4 x(3k-3) \end{cases}$$
$$\begin{cases} y(3k+2) = A_1 y(3k-1) + A_2 y(3k-2) + u(3k+2) \\ u(3k+2) = B_1 x(3k+1) + B_2 x(3k) + B_3 x(3k-1) + B_4 x(3k-2) \end{cases} \tag{10}$$

where A_i and B_i are precomputed coefficients. Three basic DA ARMA filter modules are employed to compute three systems of difference equations in parallel. The data dependence graph (DG) for a block size 3 AR filter is shown in Figure 8.

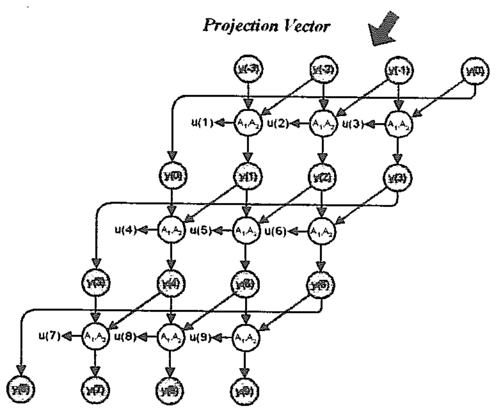

Figure 8 : DG of the block size 3 AR filter

Note that each shaded circle represents a data *word* processed bit serially. The white circle represents DA inner product computation. Similarly, the dependence graph of the MA filter is given in Figure 9.

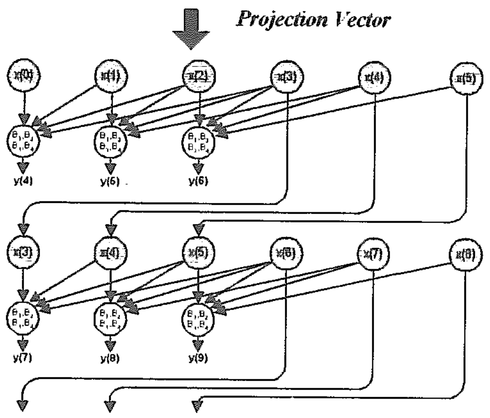

Figure 9 : DG of the block size 3 MA filter

Mapping along the indicated projection directions leads a systolic array design as shown in Figure 10. Compared with the structured pipelining design where only one 3-stage pipelined DA ARMA module is employed, three modules are needed in the block processing case. For the word-overlap design given in section 3, similar speed up can be obtained as well. The corresponding timing diagram is given in Figure 11. Apparently, the speed up in structure pipelining design is mainly attributed to deeper pipelining while the speed up in block processing design is mostly due to parallel processing. The choice between these two techniques depends on granularity of basic computing module.

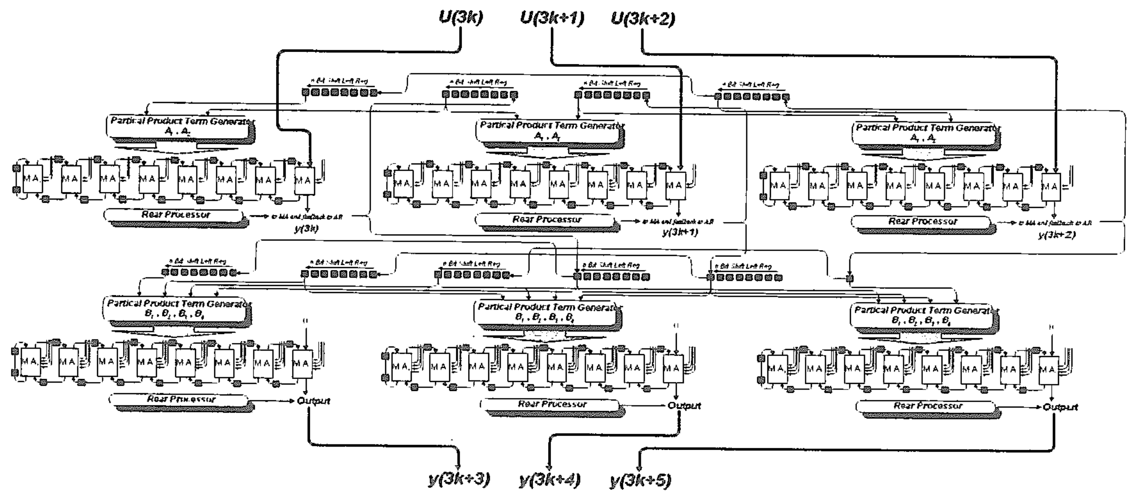

Figure 10 : Full blown version of a block size 3 two-tap DA ARMA filter

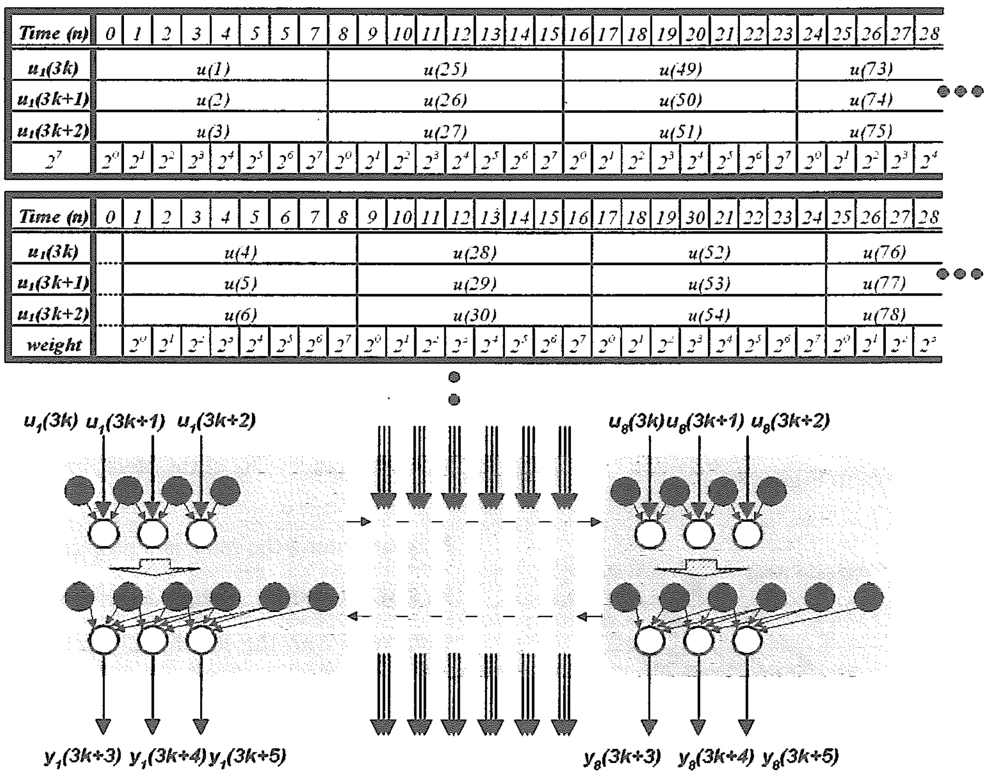
Time (n)
$u_1(3k)$
$u_1(3k+1)$
$u_1(3k+2)$
weight
$u_1(3k)$ $u_1(3k+1)$ $u_1(3k+2)$
$u_8(3k)$ $u_8(3k+1)$ $u_8(3k+2)$
$y_1(3k+3)$ $y_1(3k+4)$ $y_1(3k+5)$
$y_8(3k+3)$ $y_8(3k+4)$ $y_8(3k+5)$

Figure 13 : The timing diagram of word-overlap block processing DA ARMA filter

6. Performance comparison and conclusion

In this section, a performance comparison between various DA designs is given. The bit-parallel design proposed in [4] is used as the comparison basis to other DA designs. It has the shortest initiation interval and the least hardware complexity among other known designs using the bit-parallel approach. The comparison is based on the following four criteria, 1) critical path delay or clock period 2) initiation interval 3) hardware cost and 4) design cost, which is the product of initiation interval and hardware cost. All the figures derived in the comparison are subject to the delay or circuit complexity of a full adder. We also assume both input data and filter coefficients are 8-bit wide and neglect the delay and circuit overheads of latches. The comparison result is shown in Table 1, where LA, BP, SP, WO represent look ahead, block processing, structure pipelining and word overlap schemes, respectively. Compared with the delay in the bit parallel design, the clock period for the basic DA module is only equal to $3\,t_{FA}$. Since output data is bit-serial, 8 clocks are required to obtain an output word, which makes the equivalent initiation interval equal to 8 clock periods, i.e. $24\,t_{FA}$. Even through its speed up is not prominent when compared with the bit parallel approach, the hardware cost, however, is much smaller. The word overlap DA design version basically achieves speedup at the cost of extra hardware (i.e. using multiple basic DA modules). Four versions of look-ahead transform designs are included in the comparison. It is obvious that structure pipelining is more hardware efficient than block processing design provided the resultant pipelining stage is reasonably large to ignore the latch overhead. Overall, the best speed performance design in this comparison is the look-ahead DA version with block processing. The lowest hardware cost design is the basic DA ARMA filter. We also conduct a speed comparison with other commercial machines such as TI DSP processor, Sun Sparc workstation and Pentium PC. The result is given in Table 2. The speed of the basic 4 tap DA ARMA filter design (the slowest VLSI version) is derived from HDL simulation with delay parameters extracted from the standard cell layout. In DSP processor case, the assembly program is hand coded and optimized. The codes for both Sparc and Pentium machines are written in C and compiled with optimization. The numbers in the table are the average of 10 runs. Again, the speed performance is one to two order of magnitude faster than the commercial machines, which proves the effectiveness of our designs.

platforms	basic DA VLSI	TI 320 C40-50	Sparc 20 model 51	Pentium 133
initiation interval	41ns	500ns	2522ns	4024ns

Table 2: Speed comparison with commercial machines

Design	bit-parallel design	basic DA	WO DA	LA+SP DA	LA+BP DA	LA+SP WO DA	LA+BP WO DA
critical path	14	3	3	1	3	1	3
initiation interval	28	24	3	8	8	1	1
hardware cost	256	68	544	94	281	748	2244
design cost	7168	1632	1632	748	2244	748	2244

Table 1 : Performance Analysis of proposed DA and bit-parallel designs

References

[1] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constrains," *IEEE Trans. Circuits and Systems*, vol. CAS-28, pp. 196~202, March 1981.

[2] A. Peled and B. Lin, "A new approach to the realization of nonrecursive digital filters," *IEEE Trans. Audio and Electroacoustics*, vol. AU-21, No. 6, pp. 477~485, Dec. 1973.

[3] D.J. Evans, M.Gusev, "New linear systolic arrays for digital filters and convolution," *Parallel Computing*, vol. 20, pp. 29~61, 1994.

[4] C.L. Su, Y.T. Hwang, et. al. "The design & VLSI implementation of a high speed low chip area ARMA filter," in *Proc. The 6th VLSI Design/CAD Symp*, pp. 229~232, Aug. 1995.

[5] K.K. Parhi, D.G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital Filters- -Part I: Pipelining using scattered look-ahead and decomposition," *IEEE Trans. ASSP*, vol. 37, pp. 1099-1117, July 1989.

[6] C.-W. Wu and P.R. Cappello, "Application-specific CAD of VLSI second-order sections," *IEEE Trans. ASSP*, vol. 36, pp. 813-825, May 1988.

[7] C.-L. Wang, C.-H. Wei, and S.-H. Chen, "Efficient bit-level systolic implementation of FIR and IIR filters," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 484-493, Apr. 1988.

[8] S.A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE ASSP*, vol. 16, pp. 4~19, July 1989.

[9] Y.-T. Hwang and C.-L. Su, "A New Design Approach and VLSI Implementations of Recursive Digital Filters," 1996 *IEEE Int'l Symp. on Circuits and Systems*, vol. IV, pp. 304-307, May 1996.

[10] S.K. Mitra and R. Gnanasekaran, "Block implementation of recursive digital filters - New structures and properties," *IEEE Trans. on Circuits and Systems*, Vol. 27, pp. 667-672, Aug. 1980.