An Area-Efficient Systolic Architecture for Real-Time VLSI Finite Impulse Response Filters

V. Visvanathan Nibedita Mohanty S. Ramanathan

Computer Aided Design Laboratory
Indian Institute of Science
Bangalore-560012.

Abstract

An area-efficient systolic architecture for real-time, programmable-coefficient finite impulse response (FIR) filters is presented. A technique called pipelined clustering is introduced to derive the architecture in which a number of filter tap computations are multiplexed in an appropriately pipelined processor. This multiplexing is made possible by the fact that the processor is clocked at the highest possible frequency under the given technology and design constraints. Reduction in hardware proportional to the ratio of data arrival period and clock period is achieved. The proposed systolic architecture is 100% efficient and has the same throughput and latency and approximately the same power dissipation as an unclustered array. The architecture is completely specified, including a description of the multiplexers and synchronisation delays that are required.

1 Introduction

Real time filters are characterised by the feature that the rate of arrival of data is fixed and hence the filter has to deliver a certain computational throughput. When both the data rate and filter order are high, this requires enormous hardware resources. In fixed coefficient filters, both the area and the latency of each filter tap is reduced through the use of canonical signed digit representation whereby each filter coefficient is hard-wired in the corresponding tap [1]. Such an approach is however not possible in programmable-coefficient finite impulse response (FIR) filters which is the domain of our interest.

For programmable-coefficient filters, the area can be reduced by a maximal reuse of hardware. In this paper, we introduce a technique called pipelined clustering which is used to derive a systolic FIR filter architecture that uses minimal hardware to sustain the required data rate under the given technology and design constraints. The technology and design constraints are encapsulated as a lower bound on the period of the system clock: T_{min}. Thus given T_{min} and the data period T_{data}, $\lfloor \frac{T_{data}}{T_{min}} \rfloor$ filter tap computations are multiplexed on a physical multiply-add unit which is appropriately pipelined. Further, the resulting systolic architecture that is derived using pipelined clustering is 100% efficient and has the same throughput, latency and approximately the same power dissipation as an unclustered full-sized array.

The technique of pipelined clustering that is used to derive the area efficient systolic array for FIR filtering is based on the principles of retiming, slowdown and holdup (RSH) transformations [2], [3] and clustering [4], [5], [6], [7]. It therefore appears that it is equally applicable to other regular algorithms. It is superior to other existing techniques [4], [6] in that it is a synthesis methodology that results in an architecture that is optimal for the given design and technology constraints. Further, the resulting architecture is precisely specified and includes a complete description of the multiplexers and synchronisation delays that are required.

The paper is organised as follows. In Section 2, we begin with a brief discussion on systolic carry-save FIR filters. In Section 3, the concept of abstract processor arrays for slow and fast data rates are introduced. Pipelined clustering is presented in Section 4. The features of this new methodology are also discussed in the same section. In Section 5, we summarise our work.

2 Systolic FIR Filters

The input, output relationship for an FIR filter can be represented by the following difference equation [8]:

$$y_n = \sum_{i=0}^{N-1} b_i x_{n-i}$$

where x_n, y_n are the input and output of the filter at time n respectively, b_i's are the coefficients of the filter and N is the order of the filter [1].

Our emphasis in this paper will be on systolic architectures for the realisation of coefficient programmable carry-save FIR filters. Various methodologies have been reported in the literature to arrive at different systolic realisations of FIR filters [2], [9], [10], [11]. A simple observation is that RSH transformations when applied to the direct-form architecture of FIR filter can lead to all the existing systolic realisations [12].

The carry-save methodology is widely used in FIR filters [1]. The basic idea is to postpone the time con-

0-8186-3180-5/92 $3.00

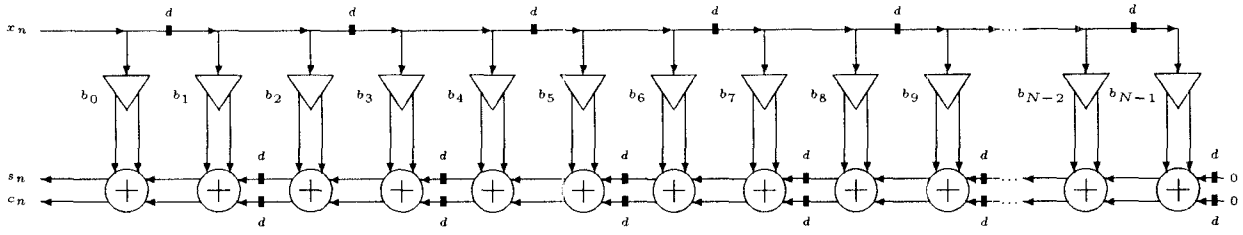

Figure 1: Systolic carry-save FIR filter architecture for order N.

suming carry propagation and hence reducing the critical path delay through the computational block [13]. The clock length of the carry-save FIR filter architecture shown in Figure 1 is given by $T_{FIR} = T_{CSM} + 2\times T_{CSA}$, where T_{CSM} and T_{CSA} are the combinational delays through the carry-save multiplier and adder respectively. Out of all the existing systolic architectures, the carry-save FIR filter architecture shown in Figure 1 [2] has the best overall performance metric and is particularly well suited for clustering [12].

The processing element (PE) of the FIR filter architecture is shown in Figure 2. The implementation of this PE uses two carry-save multipliers followed by required number of 6→2 compressor slices[1] to compress six inputs to two outputs which are in carry and save form [14]. Truncation can be incorporated in the design of the 6→2 compressor itself to restrict the output word-length [13], thereby making all the PE's constituting the full size systolic FIR filter array identical. The function performed by the PE is $s_{out} + c_{out} = s_{in} + c_{in} + x_{in}\times b_{2i} + x_{out}\times b_{2i+1}$.

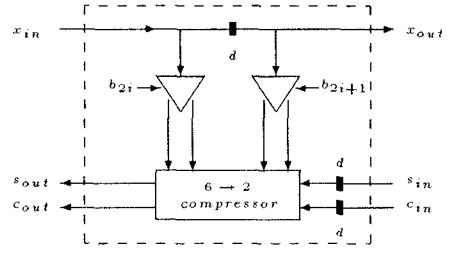

Figure 2: Processor Element PE_i of the carry-save FIR filter architecture.

3 Abstract Processor Array

In this section, we introduce the notion of Abstract Processor Array (APA) which is characterised by the fact that the clock of the array is matched to the incoming data rate, i.e., $T_{clock} = T_{data}$, where T_{data} and T_{clock} are the input data and clock period respectively. As per the definition of systolic arrays [4], [15] the fundamental period δ of the systolic full sized array is equal to the input data period, i.e., $\delta = T_{data}$. The processing element of the APA is denoted as Abstract Processing Element (APE). Let C denote the combinational delay through the APE and $\mathcal{L}$(APA) and $\mathcal{T}$(APA) denote the latency and throughput of the APA respectively.

The APA is different depending on the relative values of T_{data} and C. The salient features of the APA for the slow ($T_{data} \geq C$) and fast ($T_{data} < C$) data rate cases are as follows:

3.1 Slow Data Rate

If $T_{data} \geq C$, then the resulting APA is same as the full size processor array of Figure 1. The features of this APA are summarised as follows:

- For an FIR filter of order N, the corresponding APA requires $\lceil \frac{N}{2} \rceil$ APE's as shown in Figure 3.
- Latency of this APA is equal to δ, i.e., $\mathcal{L}$(APA) $= \delta$.
- The throughput rate of the APA is equal to f_δ, i.e., $\mathcal{T}$(APA) $= f_\delta$, where $f_\delta = 1/\delta$.

3.2 Fast Data Rate

If $T_{data} < C$, then we introduce $(l-1)$ holdup registers through the input of the full size processor array of Figure 1, where $l = \lceil \frac{C}{T_{data}} \rceil$. These holdup registers are retimed to pipeline the processing elements which results in the corresponding APA for the fast data rate as shown in Figure 4. The $(l-1)$ delays shown at the input of each multiplier in the APE (refer Figure 4) indicates that the APE is pipelined into l equal stages. The features of this APA are summarised as follows:

- For an FIR filter of order N, the corresponding APA require $\lceil \frac{N}{2} \rceil$ APE's as shown in Figure 4.
- The latency of the APA is equal to l fundamental periods, i.e. $\mathcal{L}$(APA) $= l \times \delta$.
- The throughput rate is equal to f_δ, i.e. $\mathcal{T}$(APA) $= f_\delta$, where $f_\delta = 1/\delta$.

Since each APE does a valid computation in each clock period, the Hardware Utilisation Efficiency (HUE) (see [4], [6], [15] for a formal definition) is 100%. This would appear to imply that a further reduction in the array size is not possible which maintains the same throughput and latency. The above statement is true for the APA since T_{clock} is set as T_{data} as is typical in many current real-time systolic systems. However, as will be made clear in the following sections, by increasing the clock frequency to the

[1] 6→2 compressor is the tree implementation of an adder with outputs in carry and save form.

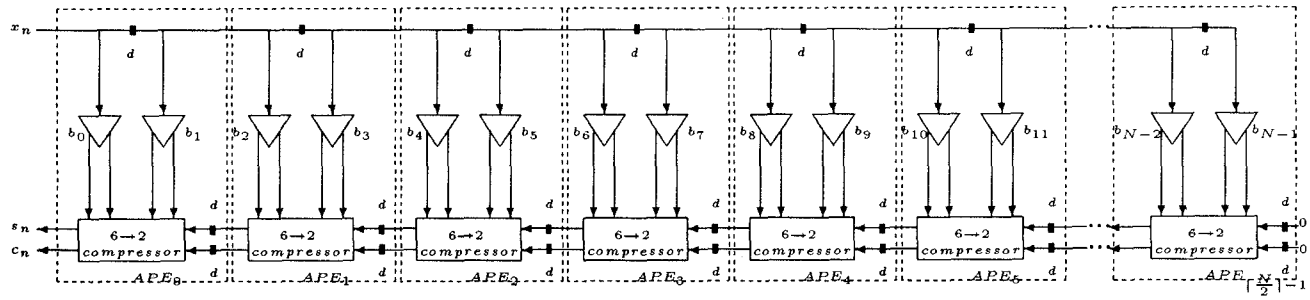

Figure 3: The Abstract Processor Array for the FIR filter for order N for slow data rate.

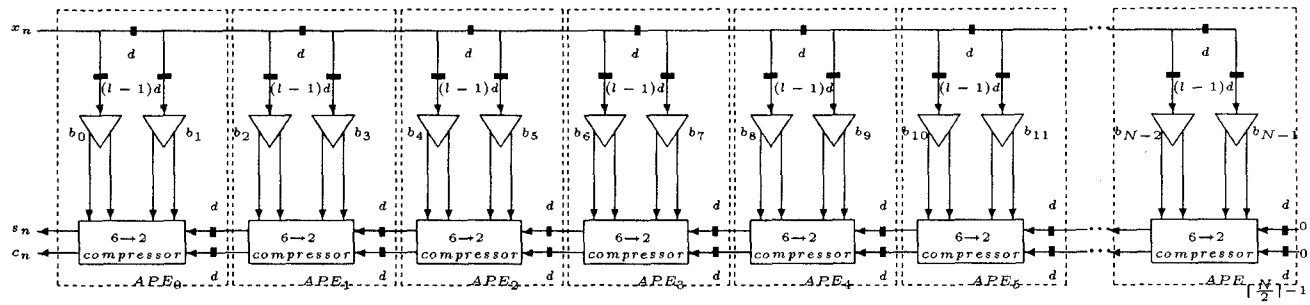

Figure 4: The Abstract Processor Array for the FIR filter for order N for fast data rate.

maximum value possible under the given design and technology constraints, enormous reduction in hardware is possible.

4 Pipelined Clustering

The technique of clustering [4] is essential to reduce the number of processor elements in the final physical processor array implementation. We propose a new methodology of Pipelined Clustering which is used to arrive at an efficient and reduced size processor array implementation from the Abstract Processor Array (APA). The method as applied to FIR filtering can be viewed as an extension of the passive and active clustering of [4]. The resulting reduced size processor array obtained by clustering is referred to as Physical Processor Array (PPA) and the processor element of this PPA is denoted by Physical Processor Element (PPE). The following are some of the important features of the PPA:

- The PPA is 100% efficient and the number of PPE's is minimum.
- the PPA is systolic.
- $\mathcal{L}(\text{PPA}) = \mathcal{L}(\text{APA})$.
- $\mathcal{T}(\text{PPA}) = \mathcal{T}(\text{APA})$.

Pipelined Clustering is a 4-step process which consists of slowdown, holdup and retiming followed by mapping of PE's of the Abstract Processor Array onto Physical Processor Element (PPE) and scheduling of computations on the clustered array.

With the given T_{data} and a lower bound on the clock period T_{min}, if $T_{data} < T_{min}$, then a physical realisation of the filter is not possible. When $T_{data} \geq T_{min}$, with the application of pipelined clustering the number of PPE's in the PPA can be reduced by a factor of 'p' as compared to the number of APE's in the APA, where $p = \lfloor \frac{T_{data}}{T_{min}} \rfloor \geq 1$. The clock that is used in the PPA is given by $T_{clock} = \frac{T_{data}}{p}$, thus $T_{clock} \geq T_{min}$. The number of stages by which the PPE has to be pipelined is n, where $n = \lceil \frac{C}{T_{clock}} \rceil \geq 1$.

The resulting PPA for the case $p \geq n$ is the clustering of APA corresponding to slow data rate. Similarly the resulting PPA for the case $p < n$ is the clustering of APA corresponding to the fast data rate. Note that $p < n$ implies $n > 1$. Independent of whether the data rate is slow or fast, the following four steps involved in the method of pipelined clustering are applied to the APA shown in Figure 3.

Step1: Slowdown the entire APA by a factor of p, where $p = \lfloor \frac{T_{data}}{T_{min}} \rfloor$.

Step2: Introduce $(n-1)$ holdup registers through the input of the APA, where $n = \lceil \frac{C}{T_{clock}} \rceil$ and $T_{clock} = \frac{T_{data}}{p}$. Retime these registers to pipeline the APE's into n equal stages.

Step3: 'p' number of locally-interconnected APE's are mapped onto one PPE with appropriate control overhead circuitry and synchronisation delays. An APE_i of the APA gets mapped onto PPE_j according to the relation $j = \lfloor \frac{i}{p} \rfloor$.

Step4: The computations of these mapped APE's are started serially on a PPA according to

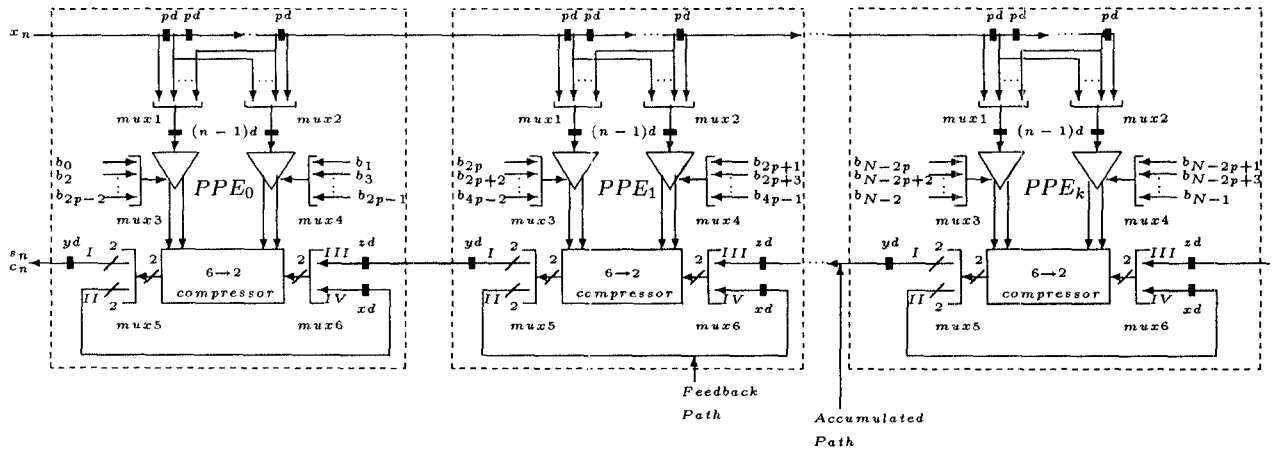

Figure 5: Physical Processor Array for a general case of Pipelined Clustering where $k = \lceil \frac{\lceil N/2 \rceil}{p} \rceil - 1$.

the same spatial order in which APE's are present in the APA. For example, on PPE_0, the computation of APE_0 is started in the first clock cycle, that of APE_1 in the second clock cycle, and so on.

The schematic circuit diagram of a PPA for a general case of pipelined clustering is shown in Figure 5. Note that the clustering is locally sequential and globally parallel [16]. The switching function of the various muxes that are present in the PPE's are discussed in the section on control overhead circuitry. The expressions for the number of delays required in the accumulated and feedback path will be derived using timing analysis in the section on synchronisation delays.

4.1 Control Overhead Circuitry

Since the clock of the PPA implementation is $T_{clock} = \frac{T_{data}}{p}$, every data cycle is divided into p clock cycles. For any value of p and n the control overhead circuitry consists of six muxes.

- Mux1, mux2 and mux3, mux4 (refer Figure 5) are used as input and coefficient selectors to the multipliers in the PPE. Mux1, mux2, mux3 and mux4 are of size p to 1 and select a different input combination for each clock cycle as per step 4 of pipelined clustering. Mux5 and mux6 are output and input selectors for the PPE respectively and are of size 2 to 1 for all values of p and n.

- Position of mux5 for various possible values of p and n are described as follows:

  - **Slow data rate** ($p \geq n$): Mux5 is connected to position II for all the clock cycles of a data period except the nth clock cycle in which it is connected to position I.
  - **Fast data rate case** ($p < n$): Mux5 is in position II for all the clock cycles of a data period except in the kth clock cycle (where $k = p$ if $n \bmod p = 0$, otherwise $k = n \bmod p$) in which it is connected to position I.

- Position of mux6 for various possible values of p and n are described as follows:

  - **Slow data rate with no pipelining of PPE** ($p \geq n$ **and** $n = 1$): Mux6 is connected to position IV for all the clock cycles of a data period except the pth clock cycle in which it is connected to position III.
  - **Slow data rate with pipelining of PPE** ($p \geq n$ **and** $n > 1$): Mux6 is connected to position IV for all the clock cycles of a data period except the $(n-1)$th clock cycle in which it is connected to position III.
  - **Fast data rate case** ($p < n$): Mux6 is connected to position IV for all the clock cycles of a data period except kth clock cycle (where $k = p$ if $(n-1) \bmod p = 0$, otherwise $k = (n-1) \bmod p$) in which it is connected to the position III.

4.2 Synchronisation Delays

On a time axis normalised to T_{clock}, let $0, p, 2p, \ldots$ denote the data arrival instants. Therefore, $[0,p], [p,2p], \ldots, [kp,(k+1)p], \ldots$ are the data periods. In any data period, at any PPE_i, the computations of APE_{ip}, APE_{ip+1}, APE_{ip+2}, ..., APE_{ip+p-1} are scheduled serially in the p clock cycles. With $T_{clock} = \frac{T_{data}}{p}$, where $p = \lfloor \frac{T_{data}}{T_{min}} \rfloor$ and $n = \lceil \frac{C}{T_{clock}} \rceil$, we will derive the expression for synchronisation delays that are required in the accumulated and feedback path by using the following timing analysis.

Accumulated Path

Let us consider the computations of processing elements PPE_0 and PPE_1 of the PPA in any general data period $[kp,(k+1)p]$, where $k \in \mathcal{Z}^+$. When the computation of APE_{p-1} starts at PPE_0 at time instant $(k+1)p-1$ in the data period $[kp,(k+1)p]$, in

its last pipelining stage at time instant $(k+1)p+n-2$, it requires the result of APE_p computed at PPE_1 in the previous data period $[(k-1)p, kp]$ which will be available at time instant $(k-1)p+n$. Hence, there is an interval of $2p-2$ clock cycles between the time instant at which the computation of APE_p is ready and the time instant at which APE_{p-1} requires the result from APE_p. This implies that $2p-1$ number of delays are required in the accumulated path.

Feedback Path

Let us consider the computations of APE_0 and APE_1 which are mapped on to PPE_0 in any general data period $[kp, (k+1)p]$, where $k \in \mathcal{Z}^+$. When the computation of APE_0 starts at PPE_0 at time instant kp in the data period $[kp, (k+1)p]$, in its last pipelining stage at time instant $kp+n-1$, it requires the result of APE_1 computed at PPE_0 in the previous data period $[(k-1)p, kp]$ which will be available at time instant $(k-1)p+n+1$ (computation of APE_1 in data period $[(k-1)p, kp]$ started at time instant $(k-1)p+1$). Hence, there is an interval of $p-2$ clock cycles between the time instant at which the result of APE_1 is ready and the time instant at which APE_0 requires the result from APE_1. This implies that $p-1$ number of delays are required in the feedback path.

Summary

From the above analysis, it follows that both for slow and fast data rate cases, $2p-1$ number of delays are required in the accumulated path and $p-1$ number of delays are required in the feedback path.

4.3 Features of Pipelined-Clustered FIR Filters

Following are the features of the PPA arrived at by using pipelined clustering.

- The Physical Processor Array is systolic and the fundamental period of the systolic PPA is p clock cycles which is same as one data period, i.e., $\delta = p \times T_{clock} = T_{data}$. The number of delays that are required in the feedback path of the PPE of the PPA is equal to $p-1$, i.e., $x = p-1$ (refer Figure 5). In order to maintain the systolic nature of the PPA, $2p-1$ number of delays that are present in the accumulated path of the PPE have to be split into y and z number of delays and placed in each PPE as shown in Figure 5. The splitting is accomplished as follows:
  - **Slow data rate:** y number of delays that are required at position I of the PPE is equal to $p-n$ and z number of delays that are required at position III of the PPE is equal to $p+n-1$.
  - **Fast data rate:** y number of delays that are required at position I of the PPE is equal to $p-n \bmod p$ and z number of delays that are required at position III of the PPE is equal to $p+(n \bmod p)-1$.
- When $p=1$ and $n=1$, the PPA is same as the APA corresponding to the slow data rate and when $p=1$ and $n>1$, the PPA is the same as the APA corresponding to the fast data rate.
- **Throughput:** Due to the systolic nature of the PPA, the terminal processor places a valid output at the end of every data period. Since there is an output in each data period, the throughput of this PPA is f_{data} which is same as f_δ, so $\mathcal{T}(PPA) = \mathcal{T}(APA)$.
- **Latency:** The latency of the PPA is the same as the latency of the corresponding APA. In the slow data rate case, the computation of APE_0 which is mapped onto PPE_0 goes through n pipelining stages and the output gets latched at the end of the same data cycle. So in this case latency is one data cycle which is the same as δ of the APA. For the fast data rate case, it takes more than one data period for the output of APE_0 to be ready since $n>p$. In this case the latency is $\lceil \frac{n}{p} \rceil \delta$ which is the same as that of the corresponding APA. Hence $\mathcal{L}(PPA) = \mathcal{L}(APA)$.
- **I/O Bandwidth:** The number of inputs and outputs and the I/O bandwidth of a PPE is the same as that of the corresponding APE.
- **Pipelinability:** Pipelining of the APE can be achieved up to a 6→2 compressor delay. Further pipelining below the 6→2 compressor delay can be achieved by introducing additional holdup registers and then retiming them. However this will require modification in the APE configuration and will also add to more input-output latency.
- **Power Dissipation:** The dynamic power dissipation of the PPA is the same as that of the corresponding APA. This is because in the PPA $1/p$ as many processors are being clocked at p times the frequency as compared to the APA. The above analysis ignores the power dissipation in the overhead circuitry.
- The control overhead which mainly consists of muxes is constant and the size of these muxes depends upon the clustering factor p. The number of synchronisation delays in each PPE is also dependent upon the clustering factor p.

4.4 Mapping to Fixed Size Array

The above technique is directly applicable to the problem of mapping an APA onto a fixed size PPA for non real-time applications. The objective is to maximise the throughput subject to the constraint that the number of PPE's N_0 in the PPA is fixed and the lower bound on the clock of the system is T_{min}.

If the abstract processor array has N APE's, then the mapping factor p is given by $p = \lceil \frac{N}{N_0} \rceil$. As discussed in the features of pipelined clustering, the throughput of the PPA is $f_\delta = 1/\delta$, where δ is the fundamental period of the systolic PPA. Since the fundamental period δ of the PPA is p clock cycles, hence

throughput = $\frac{1}{T_{min} \times p}$. So one can use the T_{min} as the clock period for the PPA to get the maximum throughput of $\frac{1}{T_{min} \times p}$.

5 Conclusions

In sum, we have presented an area efficient systolic architecture for real-time, programmable-coefficient FIR filters. The technique of pipelined clustering has been introduced and used to derive the architecture in which $p = \frac{T_{data}}{T_{clock}}$ number of APE's are mapped onto one PPE. The precise multiplexing and synchronisation delays that are required have also been derived. It has been shown that the reduced size PPA has the same latency, throughput and power dissipation (ignoring the power dissipated in the overhead circuitry) as the full sized array.

While the technique of pipelined clustering has been introduced in this paper in the specific context of synthesising area efficient systolic FIR filters, it can quite clearly be used for any systolic array. We have for example, successfully used it to develop a high clock speed reduced size array for IIR filtering.

The approach developed in this paper does not explicitly consider the problem of minimising the number of synchronisation delays. In this context it is interesting to note that if one clusters the APE's in the reverse direction, i.e., schedules the computations of the APE's on the PPE in the order which is the reverse of that given in Section 4, then the resulting clustered array is multirate systolic [15] and has fewer synchronisation delays but more input-output latency. For more complex regular algorithms, the identification of the optimal cluster and the minimisation of the number of synchronisation delays will be considerably more difficult than for the case of FIR filters. Techniques available in [16] and [17] may be useful in solving these problems.

Acknowledgements

This work was funded in part by grants from the Department of Electronics, Government of India.

References

[1] R. Jain, et.al., "FIRGEN: A computer-aided design system for high performance FIR filter integrated circuits," *IEEE Transactions on Signal Processing*, Vol. 39, No. 7, pp. 1655-1668, Jul 1991.

[2] L. Glasser and D. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Reading, MA : Addison-Wesley, pp. 403-408, 1985.

[3] C. E. Leiserson, et.al., "Optimizing synchronous circuitry by retiming," *Proc. 3rd Caltech Conf. on Very Large Scale Integration*, pp. 87-116, 1983.

[4] Jichun Bu, "Systematic design of regular VLSI processor arrays," Ph.D Dissertation, Delft University of Technology, May 1990.

[5] J. Bu, E. F Depreterre and P. Dewilde, "A design methodology for fixed-size systolic arrays," *Proc. International Conference on Application Specific Array Processing, Princeton, New Jersey, IEEE Computer Society*, pp. 591-602, Sep 1990.

[6] K. K. Parhi, C. Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal of Solid-State Circuits*, Vol. 27, No. 1, pp. 29-43, Jan 1992.

[7] S. K. Rao and T. Kailath, "Regular iterative algorithms and their implementation on processor arrays," *Proc. IEEE*, pp.259-269, Mar 1988.

[8] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1989.

[9] C. Y. Roger Chen and Michael Z. Moricz, "A delay distribution methodology for the optimal systolic synthesis of linear recurrence algorithms," *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 6, pp. 685-697, Jun 1991.

[10] G. J. Li and B. W. Wah, "The design of optimal systolic arrays," *IEEE Transactions on Computers*, Vol. C-34, No. 1, pp. 66-77, Jan 1985.

[11] M. Sheeran, "The design and verification of regular synchronous circuits," *IEE Proc.*, Vol. 133, Pt.E, No. 5, pp. 295-304, Sep 1986.

[12] Nibedita Mohanty, "Architectural synthesis of systolic VLSI digital filters," M.E Thesis, ECE Dept., Indian Institute of Science, June 1992.

[13] T. G. Noll, "Carry save arithmetic for high-speed digital signal processing," *Proc. IEEE ISCAS*, Vol. 2, pp. 982-986, 1990.

[14] P. J. Song and G. D. Michelli, "Circuit and architecture trade-offs for high-speed multiplication," *IEEE Journal of Solid-State Circuits*, Vol. 26, No. 9, pp. 1184-1198, Sep 1991.

[15] Sailesh. K. Rao, "Regular iterative algorithms and their implementation on processor arrays," Ph.D Dissertation, Stanford University, Oct 1985.

[16] A. Darte, "Regular partitioning for synthesizing fixed-size systolic arrays," *Integration*, Vol. 12, No. 3, pp. 293-304, Dec 1991.

[17] H. V. Jagdish and T. Kailath, "Obtaining schedules for digital systems," *IEEE Transactions on Signal Processing*, Vol. 39, No. 10, pp. 2296-2316, Oct 1991.