VLSI Design, 2002 Vol. 14 (4), pp. 363–372

Taylor & Francis
Taylor & Francis Group

Word-serial Architectures for Filtering and Variable Rate Decimation

EUGENE GRAYVER* and BABAK DANESHRAD†

Wireless Integrated Systems Laboratory, Electrical Engineering Department, 56-425B Eng. IV Bldg., UCLA Los Angeles, CA 90095-1594, USA

(Received 1 May 2000; Revised 2 January 2001)

A new flexible architecture is proposed for word-serial filtering and variable rate decimation/interpolation. The architecture is targeted for low power applications requiring medium to low data rate and is ideally suited for implementation on either an ASIC or an FPGA. It combines the small size and low power of an ASIC with the programmability and flexibility of a DSP. An efficient memory addressing scheme eliminates the need for power hungry shift registers and allows full reconfiguration. The decimation ratio, filter length and filter coefficients can all be changed in real time. The architecture takes advantage of coefficient symmetries in linear phase filters and in polyphase components.

Keywords: FIR filter; Decimator; Multi-rate processing; Word-serial architecture; Low power DSP

INTRODUCTION

Digital filters in general and decimation/interpolation‡ filters in particular are probably the most ubiquitous DSP components. Almost no signal processing systems exist without at least one filter. FIR filters are used most widely because of their simplicity and robustness. A very large body of research has been devoted to designing and implementing FIR filters for a variety of applications. However, research has been mostly devoted to one of two areas: namely, very high speed fully parallel implementations and implementations based on general purpose DSP chips for low data rates. A rather large range of medium data rate applications has been ignored. Low power systems have traditionally used parallel implementations (Fig. 1a), wasting a lot of silicon area. Designs that require flexibility and small area have usually been based on a programmable DSP, wasting a considerable amount of power. This paper proposes a flexible architecture that combines the small size and low power of an ASIC with the programmability and flexibility of a DSP (Fig. 1b).

The proposed architecture is designed to implement a decimator with a variable integer decimation rate D, which reduces to a regular filter if $D = 1$ [9]. Variable interpolators and decimators (VID) are needed to provide different data rates for communications applications, variable compression ratios for sound and images, subband coding, and many other applications. At this time, most VIDs fall into one of two categories: inflexible halfband power-of-2 [1], and power hungry continuously variable [2,3]. The fact is that most of the variable data rates required in any one application are an integer (but not necessarily 2^k) multiple of some minimal data rate. Thus, a VID that provides integer rate changes would satisfy a great number of applications.

The proposed architecture allows the designer to specify the subset of integer decimation ratios to implement, and allows for different filter to be used for each rate, or even different filters for the same rate. Some existing designs provide variable decimation rates [1–3], other designs allow changes in filter length and coefficients [4], but no currently available designs provide both features. The architecture's capabilities are similar to those of a DSP in that a single ROM can be used to specify the filter length, coefficients, and rate change, essentially constituting a "micro-program." The architecture also takes advantage of the often-overlooked coefficient symmetry in the separate polyphase components. As a special case, the coefficient symmetry in regular linear phase FIR filters is also exploited. This flexibility is achieved using minimal overhead by implementing a very efficient programmable memory addressing scheme. In fact, the memory addressing scheme described in this paper can also be used in some generic DSPs to reduce the number of memory accesses while increasing efficiency.

*Tel.: +1-310-472-7632/395-9559.
†Corresponding author. Tel.: +1-310-395-9559. Fax: +1-310-206-8495. E-mail: babak@ee.ucla.edu
‡Only decimation will be considered for the rest of the paper, but the results are easily modified for an interpolator.

ISSN 1065-514X print/ISSN 1563-5171 online © 2002 Taylor & Francis Ltd
DOI: 10.1080/10655140290011177

The remainder of this paper is organized as follows: First, the existing filtering/decimation implementations are described and their inadequacies are pointed out. A brief discussion of coefficient symmetries in regular and polyphase linear phase filters is presented in "Word-serial section". We then describe an architecture and memory addressing scheme for a regular FIR filter that supports both linear and non-linear phase filters. The proposed architecture is extended to support a general polyphase filter, with separate sections devoted for methods of exploiting different types of symmetry. The configuration mechanism that allows changing the decimation rate, filter length, and filter coefficients is discussed in "Variable decimation section". The implementation of the MAC unit based on a modified Canonical Signed Digit (CSD) multiplier is given in "Multiplier unit section". Some final remarks are given in the Conclusion.

Present Art

As mentioned above, most currently used digital filter designs fall into one of two groups: parallel architectures implemented on an ASIC, and serial architectures implemented on a generic DSP. The rapid progress in ASIC technology has made parallel architectures wasteful for all but the very high-speed designs. Most parallel ASIC designs are also quite inflexible. On the other hand, ASICs allow great control over the implementation details, including data precision and arithmetic algorithms. This control is needed to design low power systems. In general, DSPs are not suitable for low power design. They are, by nature, large and complex devices with fixed (and often large) bus widths. The arithmetic operations are performed using fast but power hungry blocks. Few DSPs have sufficient on-chip memory resources to implement a large filter and thus require power draining board-level memory accesses. Moreover, most DSPs cannot compute both the filter output and memory addressing for a non-trivial addressing scheme in a single cycle. Additional cycles require faster clock rates and burn yet more power. For all of these reasons, only an ASIC based solution will be considered in this paper.

FIR filter architectures fall into one of two general groups: namely, fully parallel and word-serial. A fully parallel implementation dedicates a MAC and a register for every filter tap and runs at the data rate (Fig. 1a). A word-serial implementation shares a single MAC for all the taps and runs at N times the data rate (Fig. 1b). A parallel implementation is large but fast and usually power efficient. A word-serial implementation takes up very little area but is usually slower and consumes more power. The major problem with most realizations of a word-serial architecture is the need to shift data and coefficients to the MAC, requiring a total of N^2 data transfers per output and consuming a lot of power. An alternative approach is to store both the data and the coefficients in RAM and ROM[¶], respectively, and simulate the shifting of data using an efficient memory addressing scheme [5]. However, most of the reported RAM based implementations do not take advantage of the coefficient symmetry [5]. One of the reasons for this oversight is the requirement of a rather complex addressing scheme.

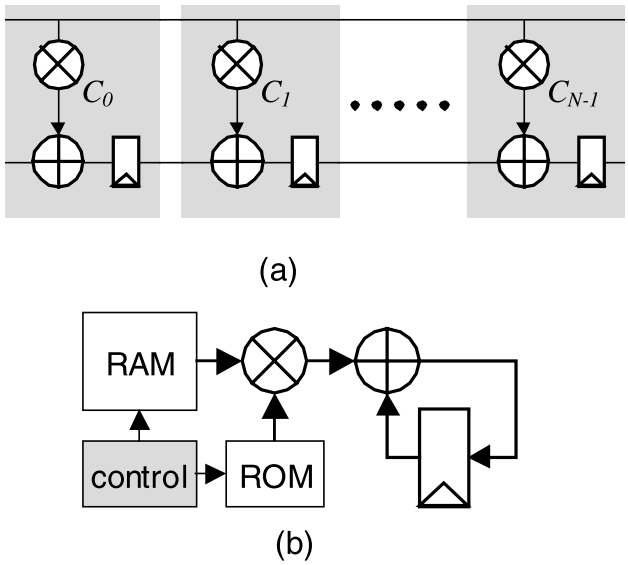

FIGURE 1 Comparing a fully parallel (a) vs. a word-serial, (b) implementation of $\sum_{k=0}^{N-1} x(n-k)C_k$.

Filter Background

An FIR filter or order N is defined by a set of coefficients $\{C_k\}$ has a transfer function, H, given in Eq. (1). If the coefficients are symmetric about the middle (Eq. (2)), $H(z)$ has linear phase. Linear phase is very desirable in a large number of applications. This property can be exploited to reduce the number of multiplications per input sample by a factor of two by combining the symmetric coefficients (3). If N is odd, the middle coefficient does not have a symmetric counterpart, and this special case must be taken into account in Eq. (3).

$$H(z) = \sum_{k=0}^{N-1} C_k z^{-k} \tag{1}$$

$$C_k = C_{N-k-1} \quad \text{where } 0 \le k \le \lfloor \frac{N}{2} \rfloor \tag{2}$$

$$H(z) = \sum_{k=0}^{\lfloor \frac{N}{2} \rfloor - 1} C_k \left(z^{-k} + z^{N-k-1} \right) + \left\{ C_{\lfloor \frac{N}{2} \rfloor} z^{\lceil \frac{-N}{2} \rceil} \right. \tag{3}$$

when N is odd

An important feature of multirate processing (decimation) is the polyphase representation, since it leads to

[¶]Here and hereafter: The term ROM will be used to refer to a memory whose contents does not change every sample. It may be implemented as a true ROM, or as a regular RAM that is loaded with appropriate data at the start of operation. See "Variable decimation rates section" for further discussion.

$$H \equiv \{\ C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15},$$
$$C_{15}, C_{14}, C_{13}, C_{12}, C_{11}, C_{10}, C_9, C_8, C_7, C_6, C_5, C_4, C_3, C_2, C_1\ \}$$

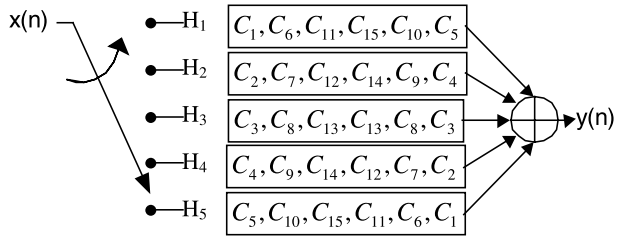

FIGURE 2 A polyphase 30-tap decimate-by-5 filter.

computationally efficient implementations of decimation filters [7]. Polyphase decomposition splits the original N coefficients into D groups of $M = N/D$ taps each, where D is the decimation ratio (4).

$$H(z) = \sum_{k=0}^{N-1} C_k z^{-k} = \sum_{m=0}^{D} z^{-l} H_m(z^D)$$
$$\text{where } H_m(z) = \sum_{n=0}^{M-1} C_{nD+m} z^{-n} = \sum_{n=0}^{M-1} C_n^m z^{-n} \tag{4}$$

The original coefficient symmetry is lost after the decomposition. However, a new form of coefficient symmetry can be observed in the polyphase component filters, H_m [7]. This result has not been widely used, and is usually not exploited except in the degenerate case of $D = 2$ [5,6]. The reasons for this are twofold: first, the derivation is non-trivial and not widely known, second, the symmetry is less obvious and is more difficult to exploit. Let us consider polyphase decomposition of a 30-tap decimate-by-5 filter linear phase filter ($N = 30$, $D = 5$, $M = 6$). This filter, shown in Fig. 2, will be used as an example.

The polyphase component filters can have one of two types of symmetry as demonstrated for this filter.

1. *Type I* symmetry. One filter component can be symmetric to another filter component. E.g. $C_k^1 = C_{M-k-1}^5$, $C_k^2 = C_{M-k-1}^4$.
2. *Type II* symmetry. The filter component can be internally symmetric, as a regular linear phase FIR. E.g. $C_k^3 = C_{M-k-1}^3$..

A particular polyphase filter can have one or both of these symmetries, depending on the length of the original filter and the decimation ratio. Thus, a general filtering and decimation architecture must be able to exploit both symmetries. While it is quite possible to determine the polyphase symmetry structure based on just N and D [7], a significant amount of hardware would be required. Instead, the proposed architecture implements memory addressing for any decimation filter based on a microprogram stored in a small ROM. For each polyphase component ($1 \ldots D$), the ROM stores three values listed below, elaborated later in this paper, and illustrated in Table I for the filter in Fig. 2.

TABLE I Control ROM for filter in Fig. 2

	Polyphase component				
	H_1	H_2	H_3	H_4	H_5
Symm type	1	1	0	1	1
Rrow	0	1	2	1	0
Trow	1	1	1	0	0

1. *Symm type*. "1" for type I symmetry, "0" for type II
2. *Trow*. Type I: "1" for the first component in the pair, "0" for the second. Type II: always "1".
3. *Rrow*. Logical row number in RAM1, RAM2 corresponding to a polyphase component.

PROPOSED ARCHITECTURE

The basic advantage of coefficient symmetry is that two input samples can be processed in one cycle using just one MAC. Since two new values are needed for each cycle, two RAMs (or one 2-read/1-write RAM) must be used. The control logic generates appropriate addresses for the two RAMs and the coefficient ROM and writes the input data to the appropriate RAM for every new input sample. The contents of the accumulator is dumped and reset for every new output sample (see Fig. 3). The system operates at $M/2$ times the input data rate since the MAC has to process M filter taps for every input sample (factor of two savings are a result of coefficient symmetry). The memory addressing scheme implemented in the control logic is the key to this architecture and will be discussed in the next three sections. Although a number of different addressing schemes can be developed, they are usually expensive to implement in hardware and are frequently inefficient. An effective addressing scheme must map easily onto simple hardware. It must not waste any cycles just on memory access, insuring that read and write operations occur in the

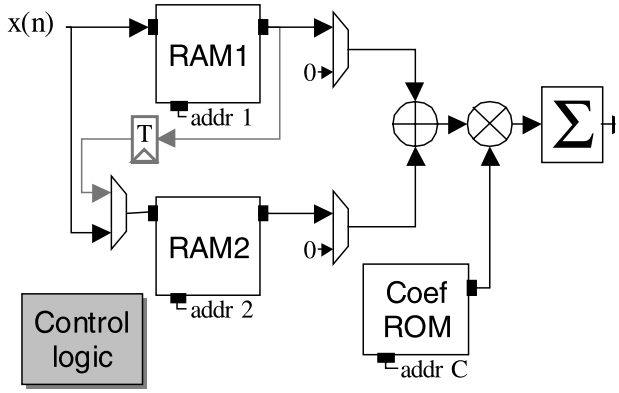

FIGURE 3 Block diagram of the proposed architecture.

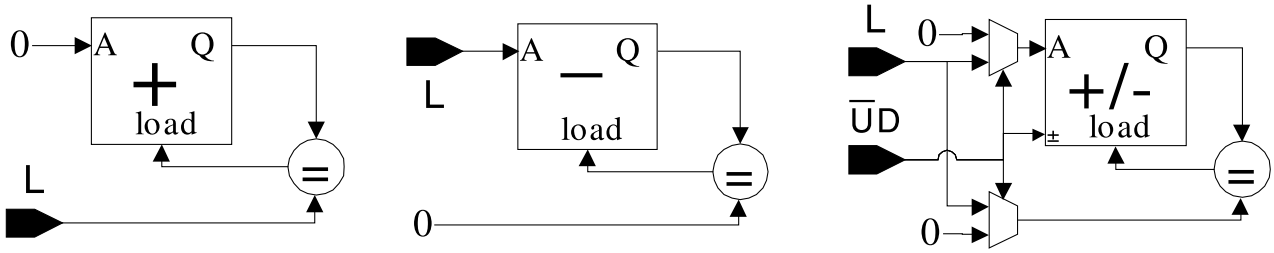

FIGURE 4 Implementing up, down, and up/down counters modulo *L*.

order needed for the computational unit. It must be flexible, but strive to use the same hardware in all modes of operation.

One of the key points of this addressing scheme is the fact that all arithmetic operations needed for address generation are performed modulo L, some integer. In general, operations modulo L are difficult and expensive to implement in hardware. The two notable exceptions are for the case of $L =$ power of two, and if the operation is an up/down counter. The desired flexibility of the proposed architecture violates the power-of-two restriction. Once an up-counter reaches L, it is reset to 0, thereby implementing the modulus operation. Likewise, once a down-counter reaches 0, it is reset to $L - 1$, as shown in Fig. 4. Thus, all operations are implemented using simple counters, as shown in Fig. 6.

Word-serial FIR

Let us first design control logic for a regular FIR. For completeness, both linear (LP) and non-linear (NLP) phase cases must be supported, as well as both even and odd filter lengths. The memory addressing scheme discussed in this section will then be extended to the case of a polyphase decimation filter.

We start by observing that the memory address and write signals must be generated such that the newly arriving sample replaces the oldest sample already in memory. Further, the addresses to the two RAMs and to the coefficient ROM must be synchronized such that the outputs of each correspond to a term in Eq. (3). If the filter is LP, the coefficient symmetry can be exploited by folding the delay chain as shown in Fig. 5a. If the filter is NLP, the delay line can still be folded, but the top and bottom rows must be computed separately (Fig. 5b). The top row of delay elements is mapped to RAM1 and the bottom row to RAM2. For each incoming sample, the oldest sample in RAM1 is stored to a temporary register and the previous oldest sample is written to RAM2, as shown in Fig. 3. The new sample is written to RAM1. The input samples "move" in opposite directions in the two RAMs. This behavior can be implemented by circular windows in each RAM (length $\lceil \frac{N}{2} \rceil$ in RAM1 and length $\lfloor \frac{N}{2} \rfloor$ in RAM2), shifting in opposite directions after every input sample. This idea is illustrated in Table II for the case of $N = 7$. As can be seen from this example, the starting address is decremented for RAM1 and incremented for RAM2 after every sample. A total of $\lceil \frac{N}{2} \rceil$ cycles are required to compute a single output of a LP filter, and a total of $\lceil \frac{N}{2} \rceil + \lfloor \frac{N}{2} \rfloor = N$ cycles are needed for the NLP case.

If the filter is LP, the outputs of the two RAMs are first summed and then multiplied by the appropriate coefficient. However, for N odd, the middle coefficient is multiplied by just the output of RAM1. If the filter is NLP, coefficients $C_{0 \ldots \lfloor \frac{N}{2} \rfloor}$ and $C_{\lceil \frac{N}{2} \rceil + 1 \ldots N-1}$ are multiplied by just the outputs of RAM1 and RAM2, respectively. This behavior is implemented by selecting *zero* in the multiplexer at the output of the "unused" RAM, as shown in Fig. 3. The zeroed outputs are highlighted in Tables II and III. Memory addressing for a NLP filter with $N = 7$ is shown in Table

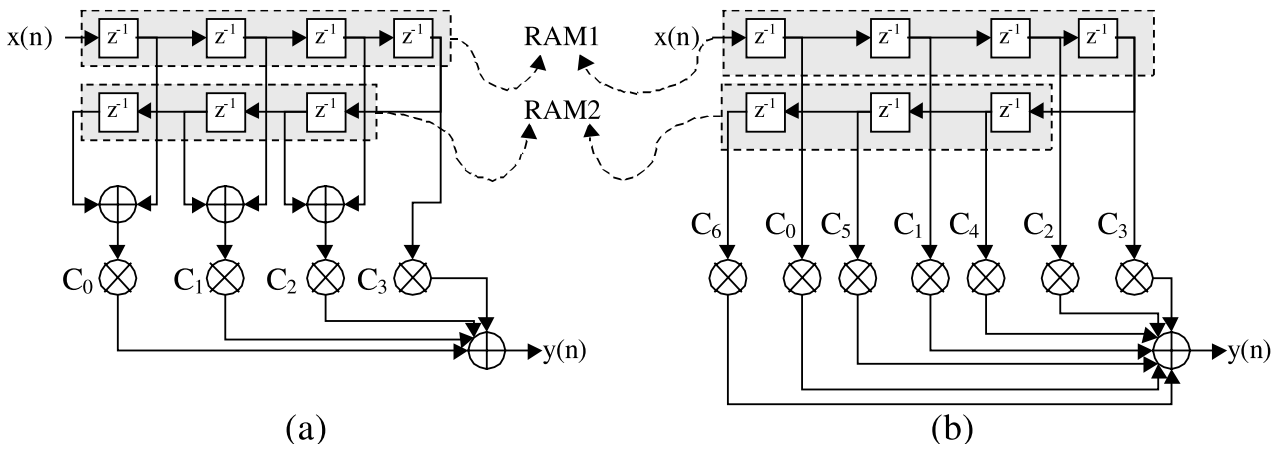

FIGURE 5 Delay line folding for LP (a) and NLP (b) FIR.

TABLE II Addressing for an $N = 7$ LP FIR*

Sample	RAM1				RAM2				ROM			
0	0^w	1^r	2	3	0	1	2	0^w	0	1	2	3
1	3^w	0^r	1	2	1	2	0	1^w				
2	2^w	3^r	0	1	2	0	1	2^w				
3	1^w	2^r	3	0	0	1	2	0^w				

* Here and hereafter: the superscript *w* indicates a write occurring at that time, superscript *r* indicates that the temporary is loaded with the output of RAM1 at that time.

III. Note that the ROM addresses are counted backwards for RAM2 outputs (corresponding to the bottom row of the folded delay chain). Also, note that the "unused" RAM could be disabled to save power. This memory addressing scheme can be implemented using just up/down counters as shown in the simple circuit in Fig. 6.

Word-serial Polyphase Decimator

Having described a memory addressing scheme and architecture for a simple FIR filter, we can turn our attention to extending it to a polyphase decimation filter. For simplicity we will assume that the original FIR filter was LP, and that $M = N/D$ is even. The results presented below can be easily extended to a more general case by following the reasoning presented above.

The polyphase filter, discussed in "Filter background section", can be though of as a bank of semi-independent FIR filters. The decimation operation implies that only one polyphase component is needed to process an input sample at any one time. However, type I symmetric component pairs must be considered jointly to take advantage of the coefficient symmetry.

The filter in Fig. 2 is mapped onto the proposed architecture in Fig. 3 by logically partitioning the RAMs into $D/2$ rows with M words in each row. Each row is used to hold the samples for a corresponding polyphase component. Data for the first member of each symmetric pair is stored in a particular row in RAM1 and for the second member in the same row in RAM2. For example, data for H_1 and H_5 are stored in RAM1[0] and RAM2[0], respectively. Likewise, the data for type II symmetric components is split between RAM1 and RAM2, as will be discussed in "Addressing for type II symmetry section".

Addressing for Type I Symmetry

Let us first consider memory addressing for type I coefficient symmetry. The decimation filter in Fig. 2 will again be used as an example. The coefficients in the first component, H_1, are a mirror image of those in H_5. The first sample in H_1 is multiplied by the same coefficient as the last sample in H_5, which simplifies to $[x_1(0) + x_5(5)]^* C_1$. The main difference between this scenario and the regular FIR described above is the fact that the top and bottom rows receive independent inputs. Thus, the input samples are fed to both RAM1 and RAM2 using the multiplexer at RAM2 input, as shown in Fig. 3. In a fully parallel implementation, this coefficient symmetry would be exploited as shown in Fig. 7.

Let us consider the contents of *row* 0, corresponding to H_1 in RAM1 and to H_5 in RAM2 after 26 samples. The values in Table IV correspond to the time the input sample was written to the RAM cell. The values are stored in increasing order in RAM1 and in decreasing order in RAM2. As new samples are written, they overwrite the oldest sample already in the RAM. The last column of the table contains the addresses into the RAMs. The value written on the last cycle is **bold**. As can be seen from Fig. 2, a sample is written to H_5 four samples after H_1.

We immediately observe that the combined output of the $\{H_5, H_1\}$ pair cannot be computed after 26 samples. All the necessary data is available for H_1 in RAM1[0], but the last needed value for H_5 has not yet been stored and will not arrive for four more samples. There are two options at this point: the system could wait for the missing value and then process the entire row, or it can process the available data only. Note that the system cannot process data in other rows since that would require additional read/write ports on the RAMs. Since addition is distributive, there is no problem with breaking up the computation of the outputs of H_1 and H_5 into two steps. Columns 3–5 are computed at the 26th cycle and columns 0–2 are computed at the 30th, thus avoiding any idle time. Note that the columns are processed in such an order that the new sample is only required on the last cycle, allowing relaxed timing

TABLE III Addressing for an $N = 7$ NLP FIR*

Sample	RAM1				RAM2				ROM			
0	0^w	1	2	3	0	1	2	0	0	1	2	3
	0	1^r	2	3	0	1	2	0^w	6	5	4	3
1	3^w	0	1	2	1	2	0	1	0	1	2	3
	3	0^r	1	2	1	2	0	1^w	6	5	4	3

* Here and hereafter: the superscript *w* indicates a write occurring at that time, superscript *r* indicates that the temporary is loaded with the output of RAM1 at that time.

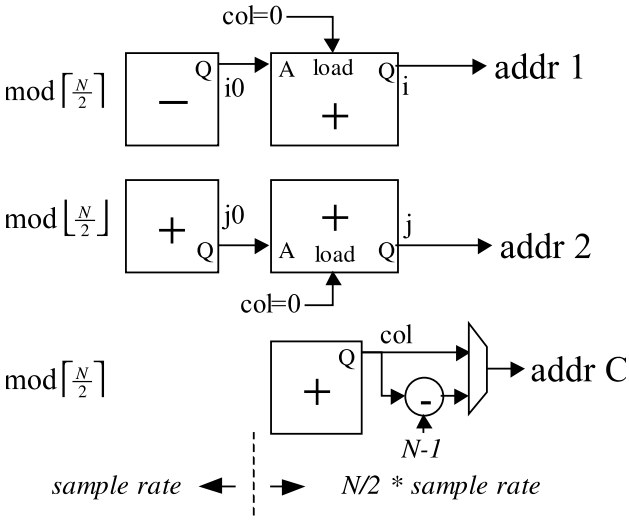

FIGURE 6 Control logic for FIR architecture in Fig. 3.

constraints on that signal. We can now formulate the memory addressing scheme for type I symmetric rows for the general case.

Define counters: {i0,j0,i,j: mod M}, {row: mod D}, {col: mod M/2}

If Trow *then* i = i0; j = j0; *else* i = i0 − 1; j = j0 − 1;

For col = 0 *to* M/2 − 1

Y = Y + (RAM1[Rrow][i] + RAM2[Rrow][j])*ROM[Rrow][col + Trow*M/2]

If Trow *then* i++; j++; *else* i−−; j−−;
If Trow *then* RAM1[i] = $x(n)$ *else* RAM2[j] = $x(n)$ row++
If row = 0 *then*

Output and reset Y

i0++; j0−−;

Addressing for Type II Symmetry

Type II coefficient symmetry is present in all linear phase FIR filters, and in some polyphase components of decimation filters with *D* odd, as shown below for the case of H_3 in Fig. 2. The memory addressing scheme and architecture are therefore very similar to those discussed in "Word-serial FIR section". Using the same approach as used for type I symmetry, the top row of delay elements is mapped to RAM1 and the bottom row to RAM2, as shown in Fig. 8.

This memory addressing scheme is also very similar to that used for type I symmetry. This similarity is desirable since it does not require any additional hardware, reduces the switching activity and circuit delays. This convenience is made possible by using *M* memory locations for each row instead of the minimum of *M*/2. However, this "wasted" memory may not be wasted at all. Memories are generated as rectangular arrays, and it is highly likely that a RAM size would be a multiple of *M*, thereby implicitly making each row a full *M* elements. An exception to this case would be the case of decimate by 1—a regular linear phase filter,

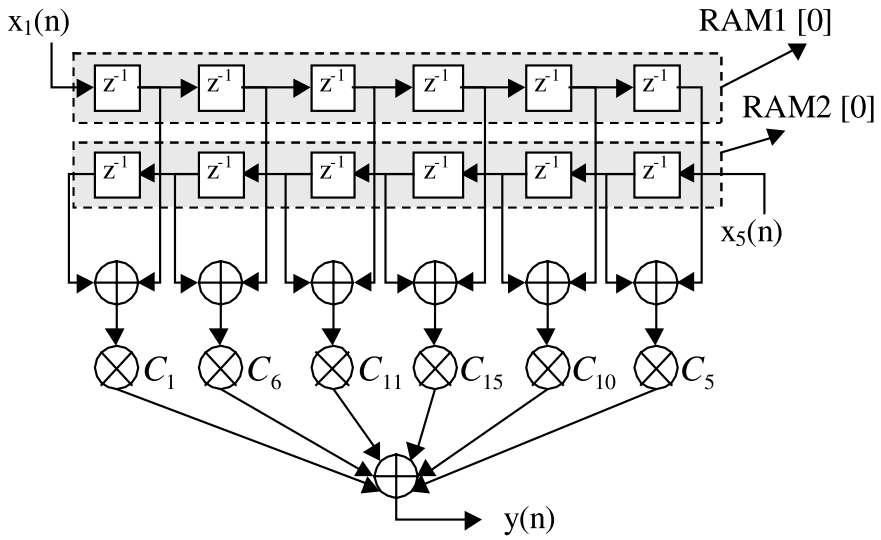

FIGURE 7 Exploiting type I coefficient symmetry.

TABLE IV Contents of the first row in RAM1 and RAM2

Sample	Memory	Data						Address		
26	RAM1[0]	1	6	11	16	21	**26**	3	4	5^{w}
	RAM2[0]	X	25	20	15	10	5	3	4	5
30	RAM1[0]	1	6	11	16	21	26	2	1	0
	RAM2[0]	**30**	25	20	15	10	5	2	1	0^{w}

which was discussed in "Word-serial FIR section". An example of this scheme for the filter in Fig. 2 is given in Table V. This scheme is implemented using the same circuit as for type I symmetry (Fig. 6) and is described by the code below.

Define counters: {i0,j0,i,j: mod M}, {row: mod D}, {col: mod M/2}

i = i0; j = j0;

RAM2[Rrow][j] = T;

T = RAM1[Trow][i];

For col = 0 *to* M/2 − 1

Y = Y + (RAM1[Rrow][i] +
RAM2[Rrow][j])*ROM[Rrow][col]

i++; j++;
RAM1[Rrow][i] = $x(n)$
row++
If row = 0 *then*

Output and reset Y

i0++; j0−−;

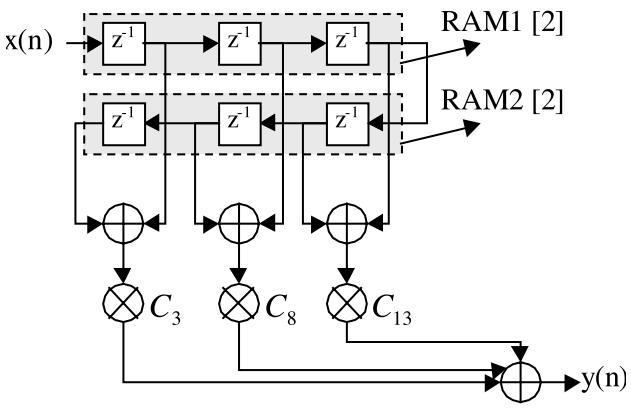

FIGURE 8 Delay line folding for type II symmetry.

Control Logic for Type I and II Symmetry

The polyphase decimator requires slightly more complex control logic than a regular FIR filter (Fig. 6). The main difference is that we must be able to address multiple filters in the same RAMs and ROM, with each filter stored in a logical row (see "Word-serial polyphase decimator section"). Since a RAM must be addressed linearly (i.e. only a single index is possible), the two dimensional matrix notation used in the previous two sections must be translated into a single dimension as: $RAM[row] \times [col] \rightarrow RAM[row*M + col]$. Consequently, a multiplier and three adders are needed to support the extended addressing into RAM1, RAM2, ROM. Further, we need one additional counter to keep track of the polyphase component (*row*). The resultant circuit is shown in Fig. 9.

Variable Decimation Rates

The memory addressing scheme discussed above is very flexible and can accommodate any combination of filter lengths, *N*, and decimation ratios, *D*. The envisioned use of this architecture is as a flexible synthesizable core. A designer needs only specify the maximum *M*, *D* to be supported by the core to determine the size of generated counters. The architecture can be operated at any clock rate above the minimum of *M*/2 times the input data rate. The required internal clocks and an output clock for a decimation filter are generated by the architecture itself. It is unlikely that all possible combinations of *N*, *D* are needed in any particular design. The architecture then allows the designer to specify a subset of *M*, *D* pairs without making any changes to the core itself. We can envision two different scenarios for the proposed decimator: (1) a stand-alone entity with minimal external control [10], and (2) an entity tightly coupled with a higher level processing intelligence [11].

In the first scenario, the programmability and rate variation are achieved using three ROMs. One ROM is needed to store the filter coefficients for all the desired *M*, *D* combinations. The second ROM is used to describe the polyphase symmetry for each desired filter as discussed in "Filter background section". The third ROM contains the offsets into the first two ROMs to access the starting location corresponding to the currently selected *M*, *D* pair. These offsets are combined with offsets generated by the control logic to access the appropriate coefficient and row information for every cycle. Clearly, the ROMs may be quite large if a large number of different decimation ratios

TABLE V Contents of the third row in RAM1 and RAM2

Sample	Memory	Data						Address			T
28	RAM1[2]				18	23	**28**	3^{r}	4	5^{w}	13
	RAM2[2]				**13**	8	3	3^{w}	4	5	
33	RAM1[2]	**33**			18	23	28	4^{r}	5	0^{w}	18
	RAM2[2]			**18**	13	8	3	2^{w}	3	4	
38	RAM1[2]	33	**38**		18	23	28	5^{r}	0	1^{w}	23
	RAM2[2]		**23**	18	13	8	3	1^{w}	2	3	
43	RAM1[2]	33	38	**43**	18	23	28	0^{r}	1	2^{w}	28
	RAM2[2]	**28**	23	18	13	8	3	0^{w}	1	2	
48	RAM1[2]	33	38	43	**48**	23	28	1^{r}	2	3^{w}	33
	RAM2[2]	28	23	18	13	8	**33**	5^{w}	0	1	

are required. The arrangement is shown schematically in Fig. 10 for three different filters of increasing length. In the second scenario, an external controller can load the desired filter coefficients and specifications when necessary. Only the first two ROMs are needed in this case, and they need only be as large as the longest required filter. There may be a third scenario as well, when higher level processing is available, but the reconfiguration must be instantaneous. In that case, the arrangement used for the first scenario can be simplified to allow just two different filters at any given time. A new filter can then be loaded into the ROMs (through an additional port, or during idle time), and then switched to by simply changing the control to the offset ROM.

Multiplier Unit

The MAC unit in the decimator consumes most of the power during filtering operation and thus warrants careful design. A modified form of the Canonical Signed Digit (CSD), known as CSD+, arithmetic is used in the proposed architecture [8]. Horner's rule is used to transform the shifted partial products into a nested configuration [5]. This results in smaller average and maximum shifts allowing for simpler hardware while achieving superior quantization noise performance. The SD multiplication encodes the constant coefficient in a tertiary number system, $\{-1, 0, +1\}$. When implemented in hardware, the "0" digits can be ignored. The hardware is designed to support a maximum of L non-zero digits. Each coefficient is coded in terms of the L digits, and the hardware consists of L add/shift stages (shaded portion of Fig. 11, $L = 3$). Each digit is stored as a record of three values: a *shift* that takes into account the Horner's rule, a *sign* to represent the $\{-1, +1\}$ digits, and a *zero* to represent the $\{0\}$ digit to allow for coefficients with fewer than L non-zero digits. Multiple SD representations can exist for the same binary number. It is thus possible to select SD representations for the coefficients to minimize the required number of shifts at

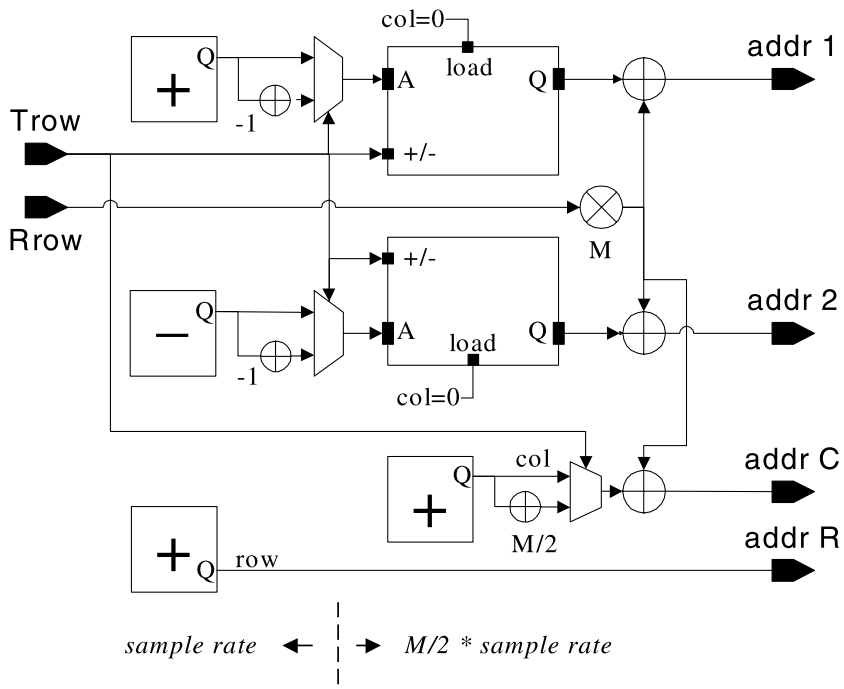

FIGURE 9 Control logic for decimator architecture in Fig. 3.

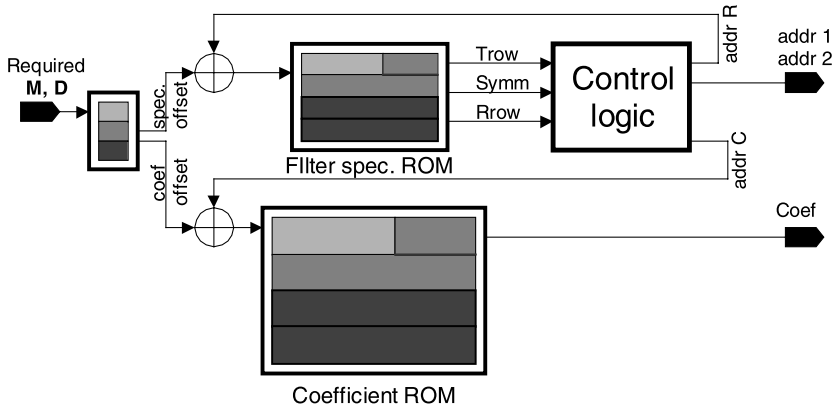

FIGURE 10 ROM configuration for programmability.

each stage. The SD multiplier in Ref. [10] uses $\{1, 1, 2\}$ bits to encode shifts for stages 1–3 respectively, and achieves 40 dB attenuation in the filter stopband. Thus, a full SD coefficient requires just 10 bits. These constraints prevent us from representing a few binary numbers, and do not define a *Canonical* SD set. However, the effect on the achievable filter frequency responses is minimal.

Let us consider an example to elucidate the CSD+ arithmetic. A number $0.2734375 = 2^{-2} + 2^{-5} - 2^{-7} = 2^{-2}(1 + 2^{-3}(1 - 2^{-2}))$ is encoded in Table VI. Let us consider the product $0.2734375*7 = 1.914$. The multiplication using CSD + encoding is accomplished as shown in Table VII. The last aspect of CSD + arithmetic we will consider is 2's complement quantization bias. It is well known that truncating a positive 2's complement number decreases its magnitude while the same operation increases the magnitude of a negative number. This results in a constant negative bias in the output data stream. This bias becomes significantly more pronounced if the truncation occurs in conjunction with right-shifts, as is the case for CSD + multiplication. Consider two products implemented in 4-bit precision: $1/4 \times 1/4$ and $-1/4 \times 1/4$, which is equivalent to $0.010 \gg 2 = 0.000 = 0$ and $1.110 \gg 2 = 1.111 = -1/8$. Clearly, right-shifting a negative N-bit 2's complement number will always result in a magnitude of at least 2^{-N+1}. We can reduce this bias in one of three ways: use rounding instead of truncation, increase the bus width/precision, redefine the right-shift operation to take this effect into account. The last solution requires the least hardware and is the most power efficient.

function Y = right_shift(X: signed(N − 1…0), S: integer)

Y = X ≫ S; *If* Y(N − 1…1) = "1" *then* Y = 0;

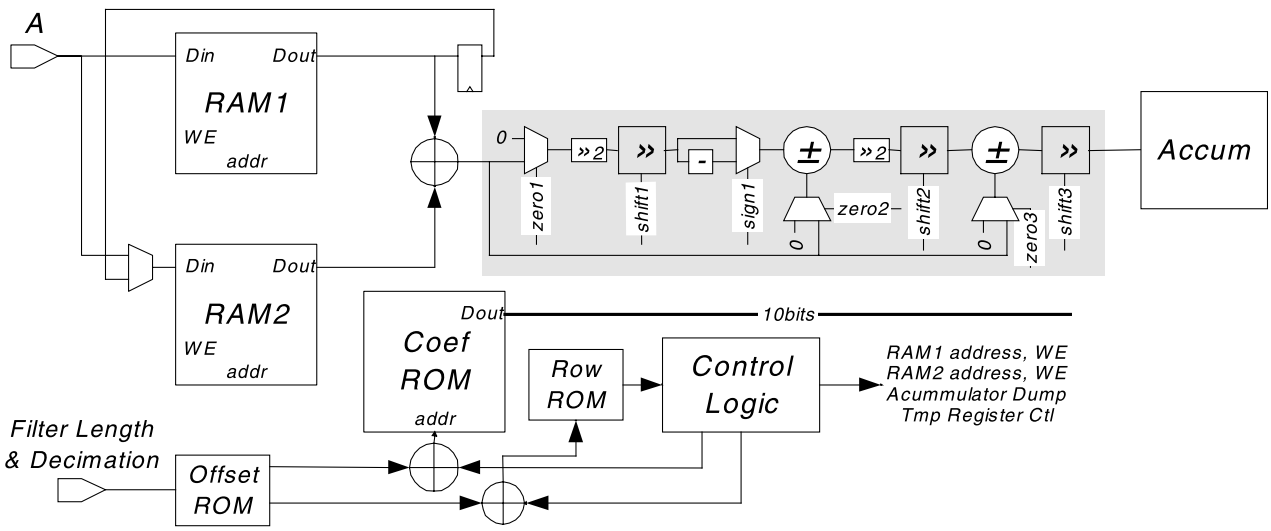

FIGURE 11 Variable rate decimator with the nested SD multiplier.

TABLE VI CSD and CSD + encoding of "0.2734375"

Stage #	CSD			CSD +		
	Shift	Sign	Zero	Shift	Sign	Zero
1	2	+	0	2	−	0
2	5	+	0	3	+	0
3	7	−	0	2	+	0

TABLE VII Using CSD + to implement "0.2734375∗7"

Stage#	Operation	Result
1	$-7/2^2$	−1.75
2	$(-1.75+7)/2^3$	0.6525
3	$(0.6525+7)/2^2$	1.914

CONCLUSION

This paper makes two closely related but independent contributions. First, an efficient and highly flexible memory addressing scheme was introduced to reduce the number of memory accesses required for decimation and filtering. This form of data storage and addressing results in considerable power savings as compared to the traditional shift-register approach. The addressing scheme can be easily implemented in hardware using a few simple counters, and thus has low overhead. Second, a hardware architecture was proposed to utilize the addressing scheme in a configurable variable rate decimator. The architecture uses a hierarchical ROM based programming method that allows the designer to implement just the subset of filter length and decimation ratio combinations required for each application. Additional combinations can be provided without changing the control logic by simply modifying the program ROMs. The architecture therefore combines the best features of an ASIC with those of a DSP.

This simple and elegant addressing scheme is ideally suited for implementation on either an ASIC or an FPGA with on-chip RAM. An HDL macro based on this architecture can allow a designer to quickly generate a decimator or a filter with the desired data and coefficient precision. In the future, the memory addressing scheme may be extended to support not just a single filter/decimator but a cascade of filters using the same hardware.

References

[1] Samueli, H., Lin, T., Hawley, R. and Olafson, S. (1992) "VLSI architectures for a high-speed tunable digital modulator/demodulator/bandpass-filter chip set", *Proceedings of IEEE International Symposium on Circuits and Systems* **May**, 1065.

[2] Erup, L., Gardner, R.M. and Harris, R.A. (1993) "Interpolation in digital modems—Part II: Implementation and performance", *IEEE Transactions on Communications* **41**, 998–1008.

[3] Duan, Ji-Ning, Ko, LiJen and Daneshrad, Babak (1999) Versatile Beamforming ASIC Architecture for Broadband Fixed Wireless Access Proceedings of IEEE CICC '99.

[4] Yoon, Sung Hyun and Sunwoo, M.H. (1998) "An efficient variable-length tap FIR filter chip", *Proceedings of the ASP-DAC '98*, 157–161.

[5] Brandt, B. (1994) "A low-power area efficient digital filter for decimation and interpolation", *IEEE JSSC* **29**(6).

[6] Willson, Jr., Alan N., Kuo, Tzu-Chieh and Kwentus, Alan (1998) "Programmable interpolation filter for digital communications applications", *Proceedings of ISCAS 98*.

[7] Vaidyanathan, P.P. (1992) Multirate Systems and Filter Banks (Prentice Hall, Englewood Cliffs, NJ).

[8] Oh, W.J. and Lee, Y.H. (1995) "Implementation of programmable multiplierless FIR filter with powers-of-two coefficients", *IEEE Transactions on CAS II: Analog and Digital Signal Processing* **42**, 553–556.

[9] Grayver, E. and Daneshrad, B. "Low Power, Area Efficient Programmable Filter and Variable Rate Decimator", *Proceedings of IEEE ISCAS '00*, Geneva, Switzerland, May 2000, pp. 341–344.

[10] Grayver, E. and Daneshrad, B. (2001) "A low-power all-digital FSK receiver for space applications", *IEEE Transactions on Communications* **49**, 911–921.

[11] Grayver, E. and M'Closkey, R. "Automagic Gain Control ASIC for MEMS Gyro Applications", *Proceedings of American Control Conference*, *Invited Session*, June 2001.

Authors' Biographies

Eugene Grayver is currently working on low power solutions for 3G mobile receivers at InnovICs Corp. He received a BS degree in electrical engineering from the California Institute of Technology in 1996, and MS and PhD degrees from the University of California, Los Angeles in 1998 and 2000. His research interests include reconfigurable implementations of digital signal processing algorithms, low power VLSI circuits for communications, and system design of wireless data communication systems.

Babak Daneshrad is an assistant professor with the Electrical Engineering Department at the University of California, Los Angeles (UCLA). His research interests include the design of systems and VLSI ASICs for wireless data communications. He obtained the BEng and MEng Degrees with emphasis in Communications from McGill University in 1986 and 1988, respectively, and the PhD degree from UCLA in 1993 with emphasis in Integrated Circuits and Systems. While at UCLA he investigated systems and VLSI circuits for HDSL and ADSL applications.