RECONFIGURABLE HARDWARE FOR EFFICIENT IMPLEMENTATION OF PROGRAMMABLE FIR FILTERS

Tracy C. Denk, Chris J. Nicol, Patrik Larsson, Kamran Azadet

Lucent Technologies
Bell Laboratories
Holmdel, NJ 07733, USA

ABSTRACT

We present the architecture of a programmable FIR filter for use in DSP and communication applications. A filter with this architecture is capable of running a wide variety of single-rate and multirate filtering algorithms with low latency. Flexibility is achieved by distributed register files that store input data and filter coefficients. The functionality of the filter is programmed by a set of pipelined control signals that are independent of the filter length. We demonstrate how to generate these control signals for a variety of configurations. In addition to its flexibility, the architecture is scalable, modular, and has no broadcast signals, making it ideally suited for VLSI implementations.

1. INTRODUCTION

Finite impulse response (FIR) filters are important blocks in digital signal processing (DSP) and communications systems. These filters are used to perform a wide variety of tasks such as spectral shaping, matched filtering, noise rejection, channel equalization, wavelet decompositions/reconstructions, subband decompositions/reconstructions, etc.

Reconfigurability can be obtained in programmable FIR filters by using variable-length delay lines [1] and register files [2]. Using register files results in reduced data movement in the filter compared to variable-length delay lines, resulting in lower power consumption. Whereas the architecture in [2] only supports FIR filters followed by decimation, this paper describes an architecture that can reconfigured to implement a wide variety of single-rate and multirate programmable FIR filters, and we demonstrate how to derive the control signals for the architecture. The architecture has the speed/area/power advantages of a custom filter implementation, and can be used in applications where latency is critical, such as adaptive filters. The proposed architecture also has properties which are desirable for VLSI implementation, namely, it is scalable, modular, and has only local interconnection of control and data signals.

2. ARCHITECTURE DESCRIPTION

This section provides a top-down description of the hierarchical FIR filter architecture.

At the highest level, the filter consists of I identical basic filter (BF) blocks cascaded together as shown in Figure 1(a). The BF blocks are controlled by the signals DW (Data Write), DA (Data Address), and CA (Coefficient Address). These signals originate from a single source and are distributed to the BF blocks in a pipelined manner. At the output of the architecture is an accumulator which sums its inputs for a number of consecutive clock cycles which is controlled by the signal AC (Accumulator Control).

Each BF block consists of J identical multiply-add (MA) units cascaded together as shown in Figure 1(b). In particular, Figure 1(b) shows the basic filter block BF_i which uses the control signals DW_i, DA_i, and CA_i.

Each MA unit consists of two register files, a multiplier, and an adder, as shown in Figure 1(c). In addition, each MA unit is controlled by the five signals CRA (Coefficient Read Address), DLT (Data Look-Through), DWE (Data Write Enable), DRA (Data Read Address), and DWA (Data Write Address).

The two register files labeled RF_C and RF_D in Figure 1(c) are used to store filter coefficients and data samples, respectively [2]. The register file RF_D has a write port (W), a read port (R), and four control signals, namely, write enable (WE), read address (RA), write address (WA), and look-through (LT). For any given clock cycle, if $WE = 1$, the data at the write port is written to the location specified by the write address WA. The data at the read port is read from the location specified by the read address RA. The special case when $WE = 1$ and $RA = WA$ is governed by the look-through signal, LT. If $LT = 1$ and the read address is the same as the write address, the value at the write port is passed directly to the read port (look-through); if $LT = 0$, the value at the read port is the value that was previously stored in the location specified by the read address (master-slave action). Parts (b) and (c) of Figure 1 show that $LT = 1$ for the data register file RF_D in MA_1 in each BF block, and $LT = 0$ for RF_D in $MA_2, MA_3, \ldots, MA_J$ in each BF block.

The coefficient register file RF_C shown in Figure 1(c) has a read port (R) and a read address (RA) control signal. Because the coefficients can be loaded into this register file off-line, the ports used to write data to the register file, namely WE, WA, and W are not shown. The coefficient register files do not use look-through.

With a slight modification of the control signals, we can reduce the latency of the architecture. This modification requires separate control for the signals DWE, DWA, and DRA in multiply-add unit MA_1 in basic filter block BF_1. These separate control signals are denoted as $FDWE$, $FDWA$, and $FDRA$, and are shown in Figure 2. The examples in Section 3.2 use this feature of the architecture.

The critical path of the architecture depends on the parameter J. In each BF block, there is a combinational path through a multiplier the adders in $MA_J, MA_{J-1}, \ldots, MA_1$. Assuming that these J adders are arranged as a tree as in [2], the propagation delay through the BF block is $T_M + T_A \lceil \log_2(J+1) \rceil$, where T_M

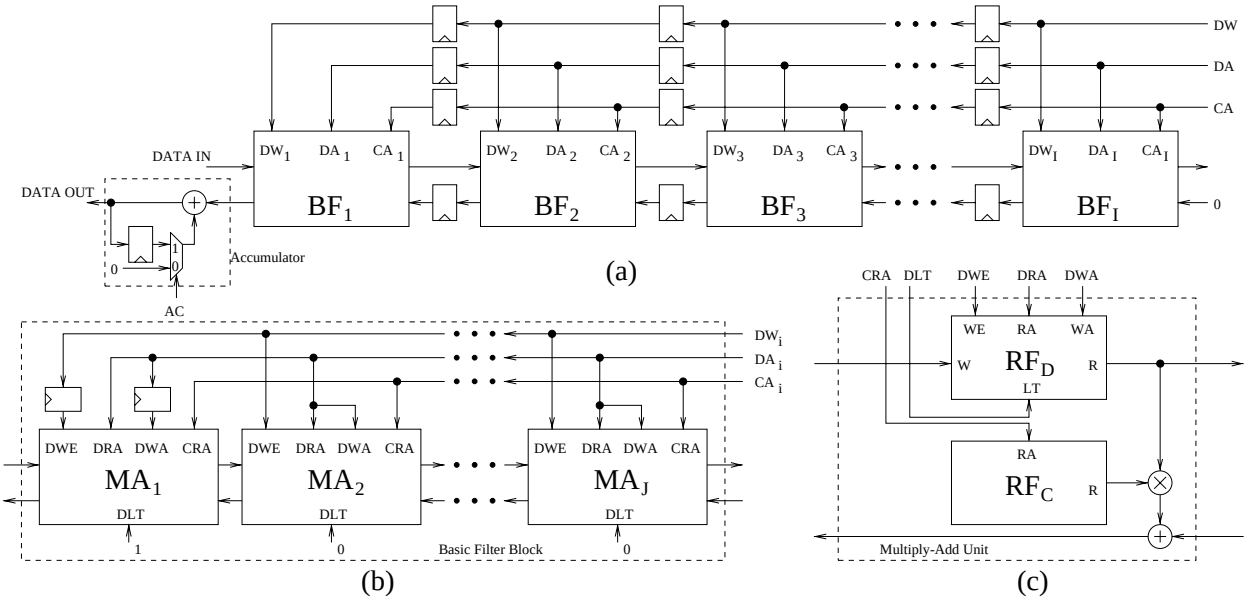

Figure 1: (a) The proposed architecture consists of I cascaded basic filter blocks. (b) The basic filter block BF_i consists of J cascaded multiply-add units. (c) A multiply-add unit consists of a data register file, a coefficient register file, a multiplier, and an adder.

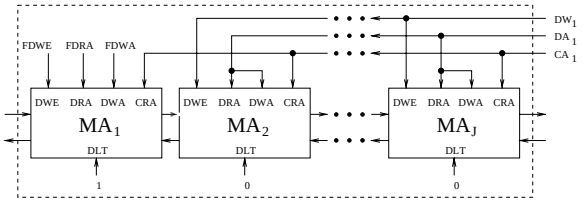

Figure 2: The basic filter block BF_1 showing the separate control signals $FDWE$, $FDWA$, and $FDRA$ which are used to reduce the latency of performing FIR filtering.

and T_A are the delays through a multiplier and adder, respectively, and $\lceil x \rceil$ is the ceiling of x, which is the smallest integer greater than or equal to x. The output of the BF block BF_1 passes through the accumulator (see Figure 1(a)), so the critical path of the architecture is $T_M + T_A(\log_2(J+1)+1)$. The parameter J can be varied to adjust the critical path given the sample rate, data and coefficient wordlengths, process technology, and the number of coefficients time-multiplexed to each multiplier.

3. DERIVATION OF CONTROL SIGNALS

The proposed architecture requires the control signals DW, DA, CA, AC, $FDWE$, $FDWA$, and $FDRA$ for each desired filtering algorithm. This section describes a methodology for determining these control signals followed by some examples.

3.1. Methodology

The steps for determining the control signals for an FIR filter are as follows:

1. Re-organize the filter computations using polyphase structures [3] and associativity.
2. Retime [4] and schedule [5] the algorithm. The primary goal of this step is to choose a retiming and scheduling solution which maps the algorithm to the proposed architecture. The secondary goal is to maximize the hardware utilization of the architecture.
3. Fold the architecture [6], [7].
4. Map the registers and multiplexers in the folded architecture to the register files in the architecture described in Section 2 so the pipelined control structure in Figure 1 can be used.

3.2. Examples

This section provides examples of deriving the control signals for the proposed architecture for $I = 2$ and $J = 4$. We assume that the data register files RF_D can each store four data samples, and the coefficient register files RF_C can each store four filter coefficients. Functions that can be implemented on this hardware are listed in Table 1. The values of K, L, and M in this table are defined in Figure 3, where $y'[n] = \sum_{k=0}^{K-1} h[k]x'[n-k]$ and the multirate blocks are described in [3].

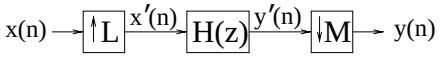

Figure 3: An FIR filter with an expander and decimator. The block $H(z)$ computes $y'[n] = \sum_{k=0}^{K-1} h[k]x'[n-k]$.

In the first example, the architecture is configured to implement filter F8 in Table 1. Since the hardware is clocked at four times the input sample rate, four coefficients are time-multiplexed to a single multiply-add unit.

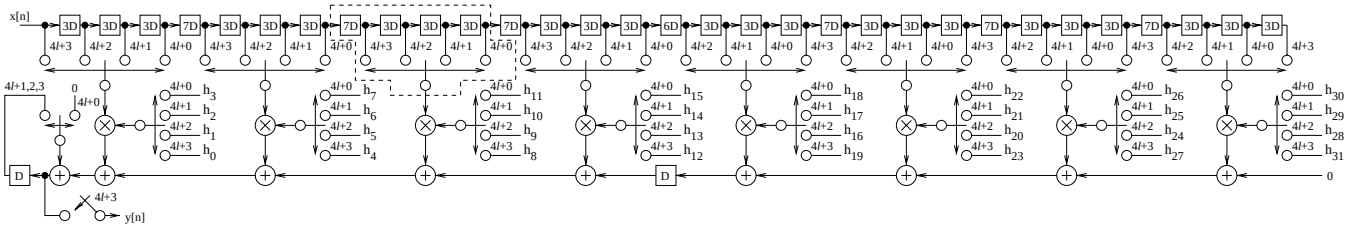

Figure 5: The folded architecture which results from folding the DFG in Figure 4.

Table 1: Filters that are implemented using the parameters assumed in Section 3.2. The values K, L, and M are defined in Figure 3. The architecture uses clock period T.

Label	K	L	M	Input Period	Output Period
F1	32	4	1	$4T$	T
F2	24	3	1	$3T$	T
F3	32	2	1	$4T$	$2T$
F4	16	2	1	$2T$	T
F5	8	1	1	T	T
F6	16	1	1	$2T$	$2T$
F7	24	1	1	$3T$	$3T$
F8	32	1	1	$4T$	$4T$
F9	16	1	2	T	$2T$
F10	32	1	2	$2T$	$4T$
F11	24	1	3	T	$3T$
F12	32	1	4	T	$4T$

A data-flow graph of the algorithm after reorganization, retiming, and scheduling is shown in Figure 4. The folding set cardinality for this DFG is $N = 4$, i.e., the folded architecture executes each node in this DFG exactly once every four clock cycles. Folding this DFG using the techniques described in [6] results in the folded architecture in Figure 5. Note that this folded architecture has the same form as the proposed architecture in Section 2 except that registers and multiplexers are used instead of register files.

The next step is to map the delays and multiplexers in the folded architecture to register files. This is demonstrated for the portion of the folded architecture in Figure 5 that is enclosed by the dotted lines. This portion is drawn in Figure 6(a). Figure 6(b) uses a life-time chart [8] to show how data moves through Figure 6(a) from time unit 0 to time unit 31. Each horizontal line represents one of these 32 time units, and the vertical lines represent the variables that must be stored. The number of live variables is given for each time step. Notice that the maximum number of live variables for any time step is 4, meaning a register file with four storage locations is sufficient. At the top of each vertical line, the register location (chosen from $\{R0, R1, R2, R3\}$) is indicated for that variable. The time units during which the variables are output to OUT_1 and OUT_2 are denoted using 'o' and 'x', respectively, on the vertical lines. The write address, the read address for OUT_1 (labeled "read$_1$ address"), and the read address for OUT_2 (labeled "read$_2$ address") are shown for each time step. During time steps when reads are performed for OUT_1 *and* OUT_2, the same read address is used for both of the reads, meaning they can be implemented using a single read port, which agrees with the architecture shown in Figure 1. The last three columns in Figure 6(b) provide sufficient

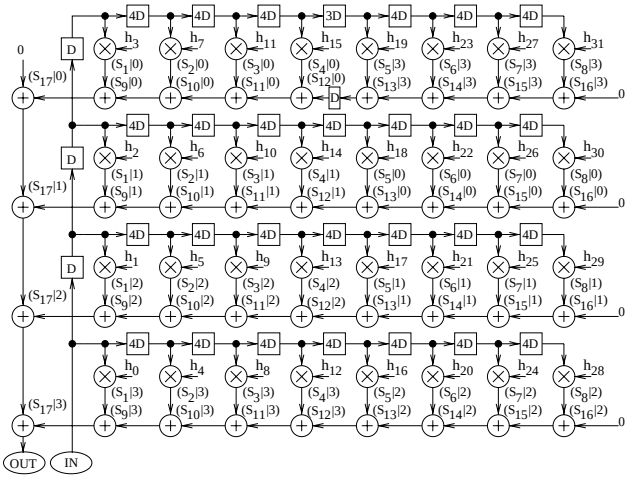

Figure 4: The DFG for Example 1 after reorganization, retiming, and scheduling. The notation $(S_n|m)$ is from [6] and indicates scheduling information which is used during folding.

information for addressing the register file that replaces the portion of the architecture shown in Figure 6(a).

The control signals for the architecture can be determined after mapping all of the delays and multiplexers to register files. The control signals for this example are given in Table 2. Since the control signals are periodic with period 16, the control signals for any time step can be determined by substituting the appropriate value of l into the table. The i-th instance of RF_C in the architecture stores the coefficients $h(4i+3)$, $h(4i+2)$, $h(4i+1)$, and $h(4i+0)$ in locations 0, 1, 2, and 3, respectively, for $i = 0, 1, \ldots, 7$.

The second example is filter F12 in Table 1. The control signals for the proposed architecture to implement this filter are given in Table 3.

The third example is filter F1 in Table 1. The control signals for the proposed architecture to implement this filter are given in Table 4.

4. CONCLUSIONS

A programmable FIR filter architecture has been presented. By supplying appropriate control signals, a wide variety of single-rate and multirate FIR filters can be implemented using the same hardware. Because it is scalable, modular, and has no global data or control signals, the architecture is well-suited for VLSI implementation. The critical path of the architecture can be adjusted by changing the number of multiply-add modules in a basic filter

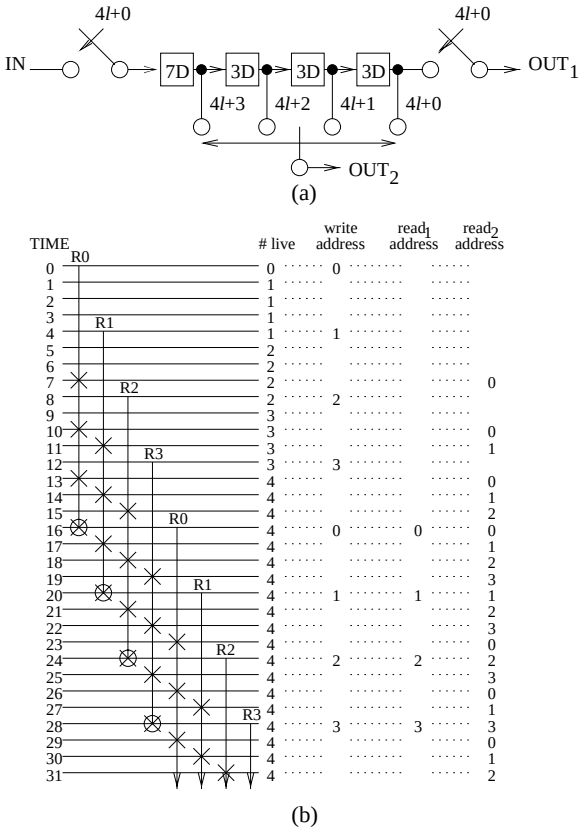

Figure 6: (a) The portion of Figure 5 enclosed in the dotted polygon, and (b) its lifetime chart and register mapping.

block. We also demonstrated how to generate control signals for this architecture.

The look-through gives a slight speed penalty compared to [2], but the low-power characteristics remain the same. Therefore, we have shown that a high degree of flexibility can be achieved with negligible speed/power/area penalty compared to a custom application-specific filter implementation. A VLSI implementation of the proposed architecture is well-suited as a hardware accelerator on a general-purpose DSP.

5. REFERENCES

[1] M. M. Cai, D. A. Luthi, P. A. Ruetz, and P. H. Ang, "A 40 MHz programmable and reconfigurable filter processor," in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp. 13.2.1–13.2.4, 1990.

[2] C. J. Nicol, P. Larsson, K. Azadet, and J. H. O'Neill, "A low-power 128-tap digital adaptive equalizer for broadband modems," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 11, November 1997.

[3] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[4] C. Leiserson, F. Rose, and J. Saxe, "Optimizing synchronous circuitry by retiming," *Third Caltech Conference on VLSI*, pp. 87–116, 1983.

Table 2: The control signals and timing for implementing filter F8 in Table 1. The control signals are listed in the order $DW - DA - CA - FDWE - FDWA - FDRA - AC$.

TIME	CONTROL	IN	OUT
$16l+0$	0 1 1 0 x 1 0	-	-
$16l+1$	0 2 2 0 x 2 1	-	-
$16l+2$	0 3 3 0 x 3 1	-	-
$16l+3$	1 1 0 1 0 0 1	$x(4l+0)$	$y(4l+0)$
$16l+4$	0 2 1 0 x 2 0	-	-
$16l+5$	0 3 2 0 x 3 1	-	-
$16l+6$	0 0 3 0 x 0 1	-	-
$16l+7$	1 2 0 1 1 1 1	$x(4l+1)$	$y(4l+1)$
$16l+8$	0 3 1 0 x 3 0	-	-
$16l+9$	0 0 2 0 x 0 1	-	-
$16l+10$	0 1 3 0 x 1 1	-	-
$16l+11$	1 3 0 1 2 2 1	$x(4l+2)$	$y(4l+2)$
$16l+12$	0 0 1 0 x 0 0	-	-
$16l+13$	0 1 2 0 x 1 1	-	-
$16l+14$	0 2 3 0 x 2 1	-	-
$16l+15$	1 0 0 1 3 3 1	$x(4l+3)$	$y(4l+3)$

Table 3: The control signals and timing for implementing filter F12 in Table 1. The control signals are listed in the order $DW - DA - CA - FDWE - FDWA - FDRA - AC$.

TIME	CONTROL	IN	OUT
$4l+0$	1 1 1 1 0 0 0	$x(4l-3)$	-
$4l+1$	1 2 2 1 0 0 1	$x(4l-2)$	-
$4l+2$	1 3 3 1 0 0 1	$x(4l-1)$	-
$4l+3$	1 0 0 1 0 0 1	$x(4l-0)$	$y(l)$

[5] T. C. Denk and K. K. Parhi, "A unified framework for characterizing retiming and scheduling solutions," in *Proceedings of IEEE ISCAS*, vol. 4, (Atlanta, GA), pp. 568–571, May 1996.

[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, January 1992.

[7] T. C. Denk an d K. K. Parhi, "Systematic design of architectures for M-ary tree-structured filter banks," in *VLSI Signal Processing, VIII* (T. Nishitani and K. Parhi, eds.), pp. 157–166, IEEE Press, October 1995.

[8] K. K. Parhi, "Systematic synthesis of DSP data format converters using life-time analysis and forward-backward register allocation," *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, vol. 39, no. 7, pp. 423–440, July 1992.

Table 4: The control signals and timing for implementing filter F1 in Table 1. The control signals are listed in the order $DW - DA - CA - FDWE - FDWA - FDRA - AC$.

TIME	CONTROL	IN	OUT
$4l+0$	0 0 1 1 0 0 0	$x(l)$	$y(4l+0)$
$4l+1$	0 0 2 0 x 0 0	-	$y(4l+1)$
$4l+2$	1 0 3 0 x 0 0	-	$y(4l+2)$
$4l+3$	0 0 0 0 x 0 0	-	$y(4l+3)$