Sharing Memory in some LIFO buffers

Charles M. Rader, *Fellow, IEEE*

***Abstract*—There are some signal processing applications in which an indefinitely long sequence of data arrives one sample at a time, but where we must segment the incoming data stream into consecutive non-overlapping blocks of some constant length, and reverse the sample sequence within each block. For example we need this to realize a recursive digital filter which has poles outside the unit circle of the z-plane.**

We can accomplish this time-reversal of samples by using a last-in-first-out (LIFO) buffer, a memory which can store up the entire block of samples, so that then those samples can be read out in reverse order. However, we would need two LIFO buffers, one for the odd numbered blocks and another for the even numbered blocks. We describe a way in which we can share one small memory and address register between both LIFO buffers.

I. Introduction

Consider an application in which a very long, perhaps indefinitely long, sequence of data x_n arrives one sample at a time, for which we need to segment the samples into consecutive non-overlapping blocks, each having K samples, and we want to produce another set of samples, one at a time in non-overlapping blocks, with each block having its samples in a time-reversed order. We can illustrate this with $K = 4$. This K is much smaller than any likely real application, but it makes it easier to show examples.

When the signal processing is done by a general purpose computer, the time-reversal of blocks of samples is not needed because the signal would reside in the computer memory and can be accessed in any order. This note is motivated by signal processing using an application specfic integrated circuit (ASIC) or a field programmable gate array (FPGA). For such cases, the logic gates are devoted to specific functions, many different functions happen at the same time, and most logic elements are busy almost all the time.

The given input sequence and the desired output sequence, using $K = 4$, would be

$$input : [x_1, x_2, x_3, x_4], [x_5, x_6, x_7, x_8], [x_9, x_{10}, x_{11}, x_{12}]$$

$$output : [x_4, x_3, x_2, x_1], [x_8, x_7, x_6, x_5], [x_{12}, x_{11}, x_{10}, x_9]$$

The μ^{th} segment, ending in μK, and its reversal, are

$$\begin{array}{cccc} x_{\mu K-3}, & x_{\mu K-2}, & x_{\mu K-1}, & x_{\mu K} \\ x_{\mu K}, & x_{\mu K-1}, & x_{\mu K-2}, & x_{\mu K-3} \end{array}$$

There is no way that x_n can appear in the output sequence earlier than it appears in the input sequence. So each output sequence block must appear with a delay, and the least possible delay is as shown below, where $x_{\mu K}$ appears at the same time in the input and the output.

Charles M. Rader was with the MIT Lincoln Laboratory. Lexington, MA 02173. He is now retired.

$$\begin{array}{l} x_{\mu K-3}, x_{\mu K-2}, x_{\mu K-1}, \; x_{\mu K} \\ \qquad\qquad\qquad\qquad x_{\mu K}, x_{\mu K-1}, x_{\mu K-2}, x_{\mu K-3} \end{array}$$

The μ^{th} output block, besides being time-reversed, is delayed in time by at least $K - 1$ clocks.

A normal LIFO buffer used to time-reverse block μ must have at least $K-1$ memory registers. The first $K-1$ samples of block μ of the input will be written into the first $K - 1$ registers of the memory, so the memory address will increase by 1 on each clock. But on clocks $K, 2K, \ldots$ the memory is not used and the input sample goes directly to the output stream. On the subsequent $K - 1$ clocks, the memory address is decreased by 1 on each clock as the successive contents are read out of the memory and fed to the output.

The following table shows, in detail, how each memory word is used on each clock for the normal LIFO buffer. This is shown for the first block but other blocks would all be treated in the same manner. The notation **a**→**b**→ means "The quantity **b** is read from the memory word and passed to the output, then the quantity **a** is written into the same memory word." When a or b is a ⊔ it is a *don't care* value.

clk	in	$word\ 1$	$word\ 2$	$word\ 3$	out
1	x_1	$x_1 \rightarrow \sqcup \rightarrow$			
2	x_2		$x_2 \rightarrow \sqcup \rightarrow$		
3	x_3			$x_3 \rightarrow \sqcup \rightarrow$	
$*K$	x_K				x_K
5				$\sqcup \rightarrow x_3 \rightarrow$	x_3
6			$\sqcup \rightarrow x_2 \rightarrow$		x_2
7		$\sqcup \rightarrow x_1 \rightarrow$			x_1

In the *clk* column, the * next to K is to remind us that for this clock, the sample is passed directly to the output and the memory is not used.

It takes $2K-1$ clocks to put a block into the LIFO and then read it to the output. But there are only K clocks before the next block appears. So we cannot use the same LIFO buffer for every block, but we can time-reverse any number of successive blocks using two LIFO buffers. Since each output block has the same K-sample length as an input block, the output blocks can fit together in a consecutive non-overlapping manner:

However, the $K - 1$ samples that use the memory require only $K-1$ reads and $K-1$ writes. So it seems worthwhile to see if one $K - 1$ word memory could be used for both LIFO buffers, for the odd numbered and even numbered blocks. In the table above, the ⊔s are really *don't care* values because they are not taken from the input sequence and not put into the output sequence. That means that they are available for some other use.

Let's see what happens if we put more input blocks directly after the input block we have shown:

clk	in	$word\ 1$	$word\ 2$	$word\ 3$	out
1	x_1	$x_1 \rightarrow 0 \rightarrow$			
2	x_2		$x_2 \rightarrow 0 \rightarrow$		
3	x_3			$x_3 \rightarrow 0 \rightarrow$	
$*K$	x_K				x_K
5	x_5			$x_5 \rightarrow x_3 \rightarrow$	x_3
6	x_6		$x_6 \rightarrow x_2 \rightarrow$		x_2
7	x_7	$x_7 \rightarrow x_1 \rightarrow$			x_1
$*2K$	x_{2K}				x_{2K}
9	x_9	$x_9 \rightarrow x_7 \rightarrow$			x_7
10	x_{10}		$x_{10} \rightarrow x_6 \rightarrow$		x_6
11	x_{11}			$x_{11} \rightarrow x_5 \rightarrow$	x_5
$*3K$	x_{3K}				x_{3K}
13	x_{13}			$x_{13} \rightarrow x_{11} \rightarrow$	x_{11}
14	x_{14}		$x_{14} \rightarrow x_{10} \rightarrow$		x_{10}
15	x_{15}	$x_{15} \rightarrow\ x_7 \rightarrow$			x_7
$*4K$	x_{4K}				x_{4K}
17	x_{17}	$x_{17} \rightarrow x_{15} \rightarrow$			x_{15}
18	x_{18}		$x_{18} \rightarrow x_{14} \rightarrow$		x_{14}

This shows that we can completely share the memory and the address generating logic between the two LIFO buffers. The trick is that for odd-numbered blocks the memory addresses first increase, and then decrease, but for the even-numbered blocks the memory addresses first decrease and then increase. But this is precisely how the addresses would have behaved if we used only even-numbered blocks in one LIFO buffer, so we get the hardware for the other LIFO buffer, meant for odd-numbered blocks, totally free and the resulting time-reversed output blocks follow one another in perfect sequence.

The memory addressing uses an address register A containing $A(n)$ at clocktime n. We have shown this register as changing like

$$A(n+1) \leftarrow \begin{cases} A(n) & n = \mu K \\ A(n)+1 & n \neq \mu K, \quad \text{odd numbered blocks} \\ A(n)-1 & n \neq \mu K, \quad \text{even numbered blocks} \end{cases}$$

A pulse every K^{th} clock can signal when the address register updating mode toggles.

Instead of using an up-down counter, the address register A could be a maximal length shift register[1] running alternately forward and backward, which can use slightly fewer gates than an up-down counter.

II. Recursive Filtering

Recursive digital filtering is capable of realizing many linear filtering computations very efficiently. Every recursive digital filter transfer function is characterized by its poles and zeros. If one or more of its poles has magnitude greater than 1, the recursion that implements the filter cannot be run in the normal time direction because it is unstable.

It is possible, however, to implement the filter as a cascade of two subfilters, one with poles all inside the unit circle, and the other having only poles outside the unit circle.[2], [3] The subfilter with poles inside the unit circle subfilter can operate directly on the normally ordered and unsegmented input data. The other subfilter's input is the first subfilter's output, segmented into consecutive non-overlapping blocks which are time-reversed as shown in the previous section. The second subfilter uses a recursion which is stable when run in the reversed time direction.

Filtering a time-reversed block of K samples produces an output block somewhat longer than K samples. It is not infinitely long because after the subfilter has processed the last sample in the input block, subsequent input samples are all 0 so that the subfilter output decays exponentially. Thus we can truncate the output after some L samples beyond the K samples which make up the input block.

Each block's subfilter output can be segmented into two parts, a *body* of length K and a *tail* of length L. We would typically choose K greater than L. The filtered blocks are then combined by using overlap-add, e.g. the tail gets added to the previous body. So each body, after that addition, is still of length K, the same length as the original input block, and the output blocks follow one another as consecutive non-overlapping samples. They are time-reversed, so they will need to be time-reversed again to restore normal sample order. A shared memory LIFO buffer pair identical to the one used in the previous section can accomplish this.

The hardware that implements the recursion to produce the body of a block can be used immediately afterward to produce the body of the subsequent block - as long as when the block ends the filter's initial conditions are reset. At the same time, the state of registers in the filter must be passed to a second instantiation of the filter which produces the tails. But since this second instantiation has an all-zero input, it needs slightly less hardware and, of course, it doesn't need to function at all for the last $K - L$ clocks of the block.

References

[1] Golomb, S. W., "Shift Register Sequences", *Holden-Day, Inc., San Francisco, CA 1967*

[2] S. R. Powell and P.M. Chau, "Time reversed filtering in real-time," *IEEE Int. Sympos. on Circuits and Systems*, May 1990, vol. 2, pp. 1239-43

[3] C. M. Rader and L. B. Jackson,"Approximating Noncausal IIR Digital Filters having Arbitrary Poles, including New Hilbert Transformer Designs, via Forward/Backward Block Recursion," *IEEE Trans. CAS -I*, Dec. 2006, vol. 52, pp. 2779-87