INTEGER SEQUENCE WINDOW BASED RECONFIGURABLE FIR FILTERS

Arulalan Rajan[1], H S Jamadagni[1], Ashok Rao[2]

[1]Centre for Electronics Design and Technology, Indian Institute of Science, India
(mrarul,hsjam)@cedt.iisc.ernet.in

[2]JSS College of Arts, Commerce and Science, Mysore, India
ashokrao.mys@gmail.com

Abstract

Conventional hardware implementation techniques for FIR filters require the computation of filter coefficients in software and have them stored in memory. This approach is static in the sense that any further fine tuning of the filter requires computation of new coefficients in software. In this paper, we propose an alternate technique for implementing FIR filters in hardware. We store a considerably large number of impulse response coefficients of the ideal filter (having box type frequency response) in memory. We then do the windowing process, on these coefficients, in hardware using integer sequences as window functions. The integer sequences are also generated in hardware. This approach offers the flexibility in fine tuning the filter, like varying the transition bandwidth around a particular cutoff frequency.

Keywords: FIR filter, windowing technique, integer sequences, reconfigurable architecture

1. INTRODUCTION

Hardware implementation of FIR filters requires the filter to be designed using some computer-aided-engineering tools such as MATLAB etc. Once the design is complete, the coefficients are available; the FIR filter can be then realized on hardware using one of the following structures.

- Direct form or transversal structure
- Transposed direct form or broadcast structure
- Linear phase structure
- Complementary FIR structure
- Lattice structures

[1] [2] [3] discuss these architectures in detail. These structures have multipliers, adders and delays as building blocks, conforming to the fact that the output of the FIR filter is obtained by convolving the filter's impulse response coefficients with the input samples. A totally different FIR architecture based on distributed arithmetic has also been reported in [4]. However, in all these implementations, we need to obtain the filter coefficients using software and then store them in memory. The disadvantage with this method is that whenever there is a need for fine tuning the filter, new filter coefficients have to be computed again in software and stored back in memory. For example even a slight variation in transition bandwidth will require change of filter length or change of window function and hence compute a new set of coefficients. Thus the traditional approach is in a way static.

An alternate approach to overcome this problem is proposed in this paper. Here, we obtain the impulse response coefficients of an ideal filter, (having box type response), with a certain cutoff frequency, using software tools and store a large number of them in memory. We then employ windowing technique on these coefficients to obtain the impulse response coefficients of an FIR filter. Taking into consideration, the simplicity and ease of generating integer sequences on hardware, we use integer sequences as windowing functions instead of conventional classical windows. Once we obtain the FIR filter coefficients, we can use any of the available hardware realizations for implementing FIR filters. The advantage of this approach is the following. Whenever there is a need for varying the transition bandwidth or any other parameter, we generate new window coefficients (here, it can be either increasing or decreasing the length of the integer sequence and obtaining the elements of the sequence or obtaining a totally new integer sequence itself) and use them to obtain a finite set of FIR filter coefficients. This is easier because, we have already stored the impulse response coefficients of the ideal filter in memory. We multiply these values with the hardware generated window coefficients. Thus, fine tuning of the filter can be done very efficiently and easily.

The paper is organized as follows: section 2 deals with the existing technique for implementing FIR filters on hardware. Section 3 elaborates our approach. We present our results in section 4 and conclude in section 5.

2. EXISTING TECHNIQUES FOR HARDWARE IMPLEMENTATION OF FIR FILTERS

Hardware implementation of FIR filters, in the first place, requires the computation of the filter coefficients. These coefficients are generated using software tools such as MATLAB. The Filter Design and Analysis (FDA) tool in MATLAB takes the following inputs and outputs the desired FIR filter coefficients.

- Filter type – low pass, high pass or others
- Design technique for FIR filter
- Attenuation in stopband
- Frequency specifications like passband, transition band and stopband specifications etc.

Applying one of the known FIR filter design techniques [5] and giving the other parameters as listed above, we obtain the filter coefficients, which are finite in number. The process of designing the FIR filter completes with obtaining the filter coefficients.

As for the implementation of the FIR filters is concerned, we develop one of the many architectural realizations on the hardware. These realizations are typically based on sum-of-products, where the products are obtained by multiplying the incoming signal sample and the filter coefficients obtained from the above mentioned design technique.

A totally different approach is followed in Distributed Arithmetic (DA) based FIR filter architecture [4]. In contrast to a conventional sum-of-products architecture, in DA we always compute the sum of products of a specific bit *b* over *all* coefficients in one step. The computation is carried out using a small table and an accumulator with a shifter. The DA implementation of FIR filter is very attractive for low-order cases due to lookup table (LUT) address space limitations [6].

In both approaches listed above, we need to precompute the filter coefficients in software and then use them in hardware. We also observe that the ideal filter coefficients (infinite in number) for a particular cutoff frequency ω_c remain the same. Any change in transition band around ω_c forces the FIR filter coefficients to be computed again. Thus we need to alternate between hardware and software, when the filter transition band varies. Any changes in attenuation in stopband or ripples in passband and stopband also require regeneration of new filter coefficients in software.

It would be an efficient approach if we can generate these FIR filter coefficients on hardware, rather than alternating between the two domains. Our approach precisely does the same by exploiting the fact that the ideal impulse response coefficients of a filter designed for a particular ω_c remains the same. We describe our approach in section 3.

3. PROPOSED APPROACH

In this section, we present our approach to design and implement FIR filters on hardware. We discuss the filter design strategy first, followed by the implementation aspects.

3.1. *FIR Filter design strategy*

Conventionally FIR filters were designed using one of the available techniques such as windowing technique, frequency sampling technique etc. Of the two, windowing technique is the simplest technique, which involves the truncation of impulse response coefficients (infinite in number) of an ideal filter (having box type frequency response) by making use of special class of finite duration functions called window functions. With the use of window functions, the ringing effects at the passband edge due to abrupt truncation of the infinite sequence are reduced. An important point to be noted here is that, the truncated impulse response decays slowly to zero. Moreover, the window functions also introduce a transition in the frequency response of the FIR filter, the width of the transition being determined by the width of the window's mainlobe. This width is then proportional to $(1/N)$ where N is the filter length. The sidelobes produce ripples in passband and stopband.

A quick look at the classical window functions [7], tell us that the most commonly used ones are real valued and have cosine-like response. In addition to these real-valued functions, we find integer sequences being used for windowing purposes [8]. Fig.1a and fig.1b gives the comparison of integer sequences as window functions with the classical windows.

The advantage of using these integer windows is that they have comparable or better performance than the classical window functions. Another advantage is that these sequences can be generated very easily on the hardware.

We exploit the fact that these integer sequence window coefficients can be easily generated on hardware compared to the hardware generation of the classical window coefficients.

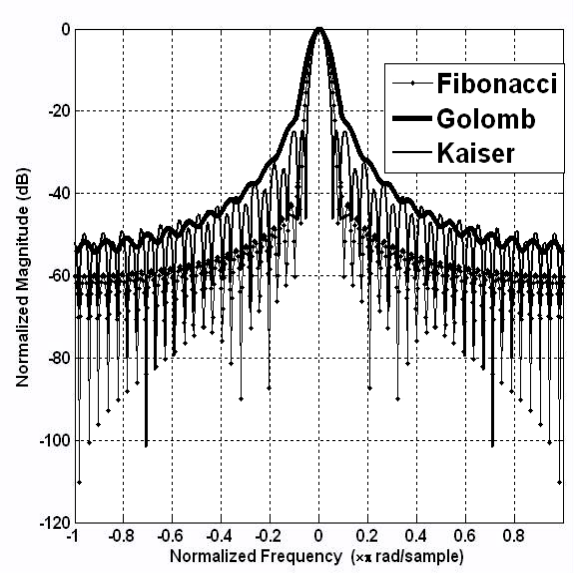

Fig. 1a

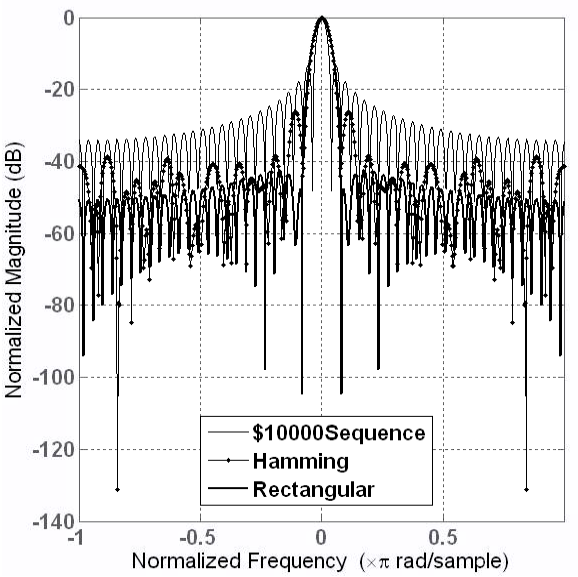

Fig. 1b

Normalized Frequency Response of integer sequence window and classical windows [8]

In our approach, we obtain the FIR filter coefficients through windowing technique, using integer sequences as windowing functions.

3.2 *Hardware implementation of Integer Sequence generators*

In our approach to implementing FIR filters on hardware, we first look at the hardware generation of integer sequences, since we use integer sequences as window functions for obtaining the FIR filter coefficients. Of all the integer sequences listed in [9], we look at the Golomb sequence, whose generating function is given by

$a[n] = 1 + a[n - a[a[n - 1]]]$, $a[1]= 1$, $a[2] = 2$.

The first few elements of the sequence are 1,2,2,3,3,4,4,4,5,5,5… $a[n]$ is the number of times n occurs. More elements of this sequence can be found in [9] and some applications in [10]. Implementing the generating function directly on hardware is quite cumbersome as it involves a number of memory accesses to output one value of the sequence. However, on observing the sequence, we find that there is a pattern embedded in the sequence. The sequence has no elements missing from the natural counting order. Every element in the natural counting order, other than 1, repeats at least twice. For example, if we look at the first eleven elements of the sequence, we find only five numbers from the natural counting order occurring. We make use of the repetitions to obtain a very compact architecture for generating Golomb sequence in hardware, as shown in fig.2.

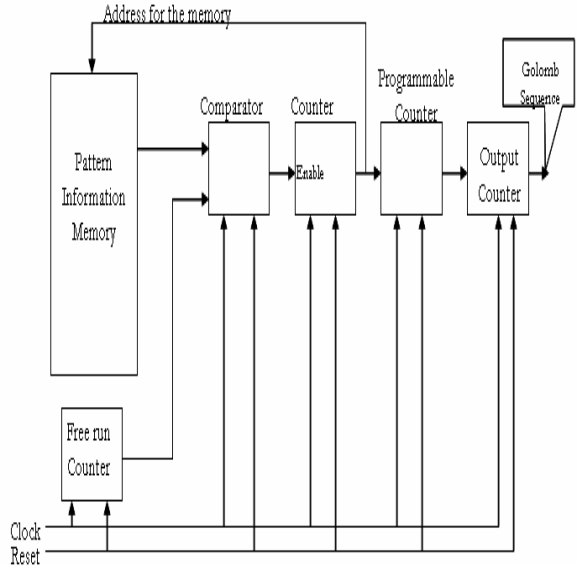

Fig. 2 Golomb Sequence Generator

The pattern information memory in fig.2 contains the position, in the sequence, at which a given repeat value changes. For example in the Golomb sequence, 1 occurs only once. Therefore a count corresponding to 1 is stored in the first location of the memory. In location 2, we store the number 5, which corresponds to the position in the sequence at which the number of repetitions changes from 2 to 3. This means that in the natural counting order, 2 and 3 are repeated twice. The third location contains number 11. The difference between third and second location in the memory corresponds to 6 positions in the sequence having a repetition of 3. These 6 values correspond to the numbers 4 and 5 each repeating thrice. Location 4

contains 23. This means that there are 12 positions which have four repetitions, corresponding to the four repetitions of numbers 6, 7 and 8. This information is read from the pattern information memory and is compared with the count of a free running counter. When the comparator inputs are equal, *Counter2* is enabled. The output of this counter serves as the address to the pattern information memory and also as the terminal count for the *Programmable down counter*. When this counter reaches zero, the *Output counter* is enabled. The output of the *Output counter* yields the required sequence. The architecture shown in fig.2 is used for generating Golomb like sequences. This implementation requires as less as 10 locations for obtaining as large as 512 elements. Thus, there is a huge reduction in the memory required.

3.3 ***Implementation of FIR filter***

In section 2, we made an observation that for a particular cutoff frequency $\omega_{c,}$ the impulse response coefficients of an ideal filter is fixed and these are infinite in number. We store a considerably large number of these coefficients in memory. For example, we store about 8192 coefficients in memory. These 8192 coefficients are then windowed using an integer sequence of finite length *N*, to obtain *N* FIR filter coefficients, $N << 8192$. These coefficients are stored in memory and any realization of FIR filter can be used for implementation. The advantage of doing so is that, around a given cutoff frequency $\omega_{c,}$ we can have a transition band of any width, the width being determined by the length of the sequence. However, this scheme becomes complex when the classical windows are to be used for windowing purpose. The reason is that the classical windows can be generated in hardware by using CORDIC based architecture or by developing architecture for Taylor's series.

The architecture in fig.2 results in generating one element of an integer sequence per clock cycle. This is not the case with conventional windows, where a conventional CORDIC architecture [11], outputs one partial coefficient every 16 clock cycles. Further processing of this partial coefficient yields the final window coefficient. Thus one window coefficient is generated every 20 clock cycles.

Another interesting thing to note with our approach is that, whenever there is a change in *N*, say from N_1 to N_2, the generation of integer sequences is very easy than the generation of classical window coefficients. The reason is that for any $N_2 > N_1$, we need to generate only a few more elements than recalculating all the values again. For example, say we had a Fibonacci sequence of length 10 and we need to increase the length to 20, the first 10 elements will remain the same, while we need to compute the other 10 elements alone. This is not the case with classical windows, because all the coefficients depend on *N*.

Since we do not alternate between software domain and hardware domain for obtaining the FIR filter coefficients, we need to look for techniques to modify the FIR filter coefficients in hardware. This can be done by generating different integer sequences on the hardware and then using them for windowing the already stored impulse response coefficients of the ideal filter (with box type response). It is not a wise approach to have multiple integer sequence generators on hardware to generate different sequences. Thus we need to look at a reconfigurable universal integer sequence generator. This will generate different sequences as required and will make the FIR filter also a reconfigurable structure. One such universal integer sequence generator is shown in fig.4.

The memory for pattern information in fig.3 is similar to that in fig. 2, except that this memory is segmented to accommodate the pattern information of various sequences. The *Win-Sel* multiplexer allows the selection of a particular integer sequence. The count *source counter* gives the address for the *pattern information memory* (PIM) and also the elements of a particular sequence. The output of the *comparator* enables the *source counter*. The inputs to the *comparator* are the pattern information stored in the PIM and an *upcounter*. The desired sequence is obtained through a *2:1multiplexer*. This multiplexer is used when we need to generate sequences which do not have all the numbers in natural counting order, like Fibonacci sequence.

Table 1 Resource Utilization Summary

Window Sequence	CLB Slices	Max. Operating Frequency MHz
CORDIC based Hamming window generator	180	72
Fibonacci (FSM)	70	152
Hofstadter Conway (FSM)	110	120
Golomb(FSM)	130	90
Recursive Triang (FSM)	80	130
Golomb (alternate Architecture)	24	130

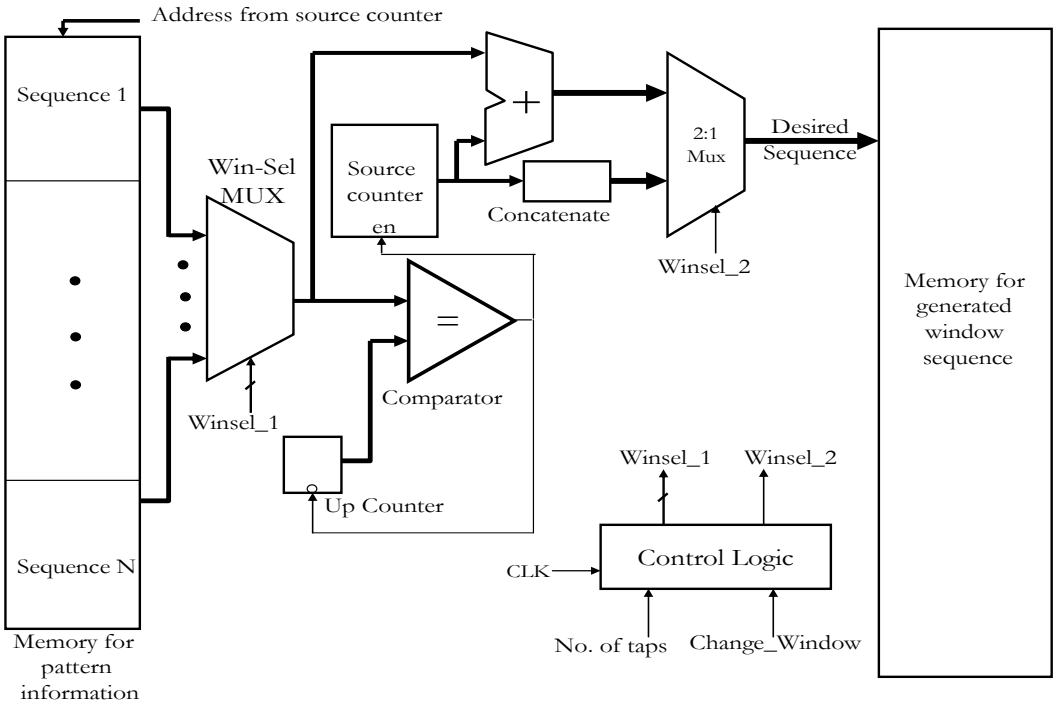

Fig. 3 Universal integer sequence window generator

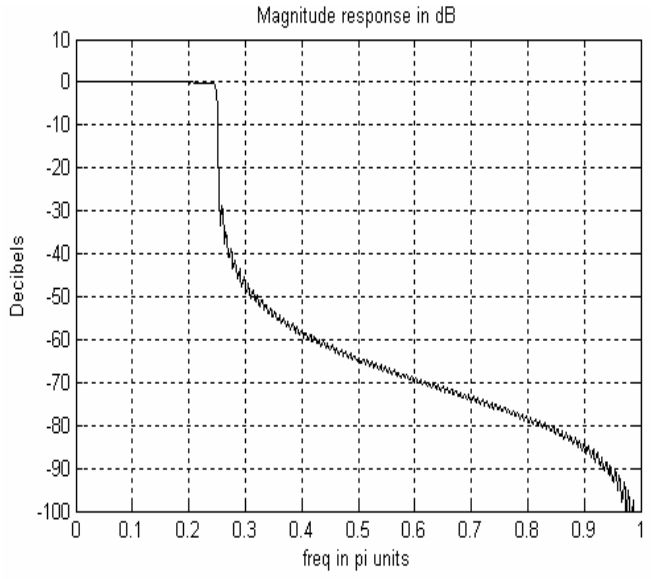

Fig. 4 LPF response with Golomb sequence as window function

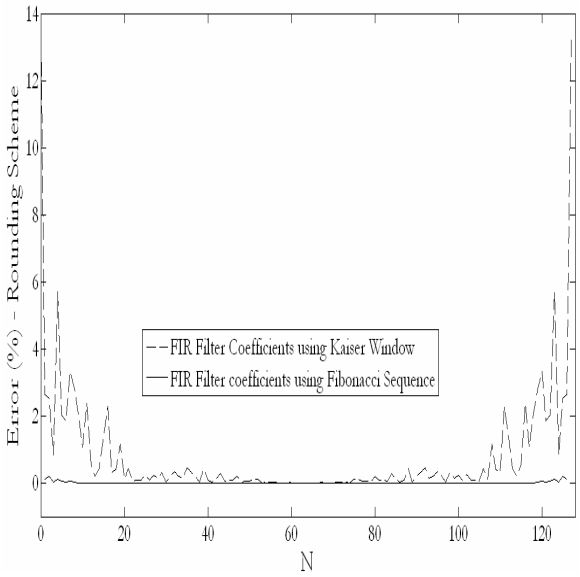

Fig.5 Comparison of error % in FIR filter obtained using Kaiser window and Fibonacci sequence

4. RESULTS

Architecture shown in fig.2 was implemented on Xilinx XC3s200-ft256 FPGA. The resource utilization summary is given in Table.1. We find that the architecture fig.2 takes one-sixth of the total resources required for the finite state machine implementation of Golomb sequence and operates at a higher frequency. The implementation of CORDIC based Hamming window generator is found to consume more FPGA resources, while operating at very low frequency.

Fig. 4 shows the magnitude response of a 512 tap FIR filter designed using Golomb sequence as the window function. The low pass filter was designed to have a cutoff frequency of 0.25 radians. We find from fig.4 that there is a sharp cutoff at 0.25 rads, though the attenuation is about 30dB.

Fig. 5 gives the comparison of quantization error in the FIR filter coefficients designed using Kaiser window and Fibonacci sequence as window. We find that the quantization error is very less when integer sequences are used.

6. CONCLUSION

In this paper, we have seen a new strategy to implement FIR filters on hardware using integer sequences as window functions. The window functions are also generated in hardware, which is totally different from the existing approaches to implement FIR filters. We find that, given the impulse response coefficients of ideal filter for a particular cutoff frequency, using our approach of hardware based windowing technique; we can generate FIR filter coefficients on the fly to fine tune the performance of filter. We also observe that the need for alternating between software and hardware to design and implement FIR filters is not as frequent as the existing approaches. Thus, we find that around a particular cutoff frequency, the filter response can be changed on the fly using the proposed approach.

REFERENCES

[1] Lars Wanhammar, *"DSP Integrated Circuits"*, Academic Press, 1999.

[2] E.C.Ifeachor, B W Jervis, "*Digital Signal Processing: A Practical Approach*", Addison-Wesley, 1993.

[3] L.R.Rabiner, Ben Gold, "*Theory and Applications of Digital Signal Processing"*, Prentice-Hall, Englewood Cliffs,NJ,1975

[4] Xilinx Inc, "*Distributed Arithmetic FIR filter"*,URL www.xilinx.com/ipcenter

[5] Andreas Antoniou, *"Digital Filters: Analysis, Design and applications"*, McGraw-Hill, 1999, New York.

[6] U. Meyer-Baese, *"Digital Signal Processing with Field Programmable Gate Arrays"* Springer, 2001

[7] Fredric J. Harris,"*On the Use of Windows for Harmonic Analysis with Discrete Fourier Transform,* Proc. of the IEEE, Vol.66, No.1, Jan 1978, pp 51-83.

[8] Arulalan Rajan, H S Jamadagni, Ashok Rao, "Novel Window Functions for Digital Filtering", *Proc. of the 5th IEEE International Conference on Information Technology: New Generations*, Las Vegas, April 2008, pp.1184-1185.

[9] Manfred.R.Schroeder,*"Number Theory in Science and Communication"*, Springer, 4th Ed (2006).

[10] http://www.research.att.com/~njas/sequences/

[11] Ray Andraka, "A Survey of CORDIC Algorithms for FPGA based Computers", *Proc. Of the 6th International Symposium on FPGAs*, Feb 1998, Montery, CA, pp 191-200.