American Journal of Applied Sciences 5 (7): 788-797, 2008
ISSN 1546-9239

Configurable Multirate Filter Banks

Ali Al-Haj
Department of Computer Engineering, School of Electrical Engineering,
Princess Sumaya University for Technology, PO Box 1928, Al-Jubeiha, 11941 Amman, Jordan

Abstract: Multimedia communications require efficient and real-time implementations of multirate digital signal processing systems. The backbone structures of multirate systems are digital multirate filter banks. Therefore, efficient multimedia communications rely, in the first place, on real-time implementations of multirate filter banks. In this paper, we describe a Field Programmable Gate Array (FPGA) implementation of the analysis and synthesis filter banks which are the fundamental components of multirate systems. The implementation utilizes the parallel form of the distributed arithmetic technique which enables maximum exploitation of the parallelism inherent in the multirate filtering operation. Performance results demonstrate the effectiveness of the implementation and suggest that the FPGA platform is indeed attractive for implementing multirate filter banks..

Key words: Multirate Filter Banks, Filed Programmable Gate Arrays (FPGAs), xilinx virtex devices, parallel distributed arithmetic, efficient parallel implementation

INTRODUCTION

Multimedia digital signal processing applications require sampling audio and video signals using different sampling rates[1]. This requirement has given rise to the design, development and application of multirate systems in communications, image and video processing, speech coding, spectrum analysis, radar and antenna systems[2]. The sampling rate conversion process in these systems has been traditionally done by passing the multimedia signal through a digital to analog converter, and then re-sampling the output analog signal at the required rate. However, this method introduces distortion to the signal because of the quantization effects inherent in the analog to digital conversion process[3].

Single-rate digital filters have been used to perform the sampling rate conversion process to overcome limitations of the analog to digital conversion approach. However, single-rate filters proved to be slow in terms of processing time due to the many filtering taps that must be used. Ultimately, multirate filters were developed to offer relatively low sampling rate, thereby resulting in fewer filtering taps compared to single-rate filters[4]. These filters convert a set of input samples into another set that represent the same signals sampled at the required frequency. Multirate filters are commonly employed in systems requiring real-time performance, and therefore they are still receiving considerable attention in modern research.

In this paper, a description of a parallel and high speed, single-chip implementation of the fundamental multirate filter banks is presented. The hardware implementation platform is based on Virtex field programmable gate arrays (FPGAs)[5]. The fine grained parallelism found in Virtex FPGAs is well-matched to the high-sample rates and distributed computation often found in multirate digital signal processing applications[6]. Furthermore, the reconfigurable lookup-table architecture of Virtex FPGAs makes it possible to modify filter coefficients to suit different applications. We make maximal utilization of Virtex FPGA resources by implementing the computation of the fundamental multirate filter banks in accordance with the parallel distributed arithmetic technique[7]. This technique rearranges the filter operation of the fundamental banks in such a way so as to match the architecture of Virtex FPGA, resulting in large performance gains.

This paper is organized as follows. Section 2 gives an overview of the XSV-300 FPGA prototyping board. Section 3 introduces the two fundamental multirate filter banks; the analysis filter bank and the synthesis filter bank. Section 4 presents direct, serial, and parallel distributed arithmetic implementations of FIR filters. The FPGA implementation of the analysis filter bank is

Corresponding Author: Ali Al-Haj, Department of Computer Engineering, School of Electrical Engineering,
Princess Sumaya University for Technology, PO Box 1928, Al-Jubeiha, 11941 Amman, Jordan

described in section 5, and the FPGA implementation of the synthesis filter bank is described in section 6. Finally, section 7 discusses the performance results and section eight concludes the paper.

FIELD PROGRAMMABLE GATE ARRAYS

A filed programmable gate array (FPGA) is an integrated circuit that contains many identical logic cells interconnected by a matrix of wires and programmable switch, as shown in Figure 1. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnect form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit [8].

For the particular implementation reported in this paper, we have used a prototyping board called the XSV-300 FPGA Board, developed by XESS[9]. The board, shown in Fig. 2, employs a Xilinx XCV300 FPGA with 300,000 gates [10]. It can accept video with up to 9-bits of resolution and output video images through a 110 MHz, 24-bit RAMDAC. It can also process stereo audio signals with up to 20 bits of resolution. Two independent banks of 512K x 16 SRAM are provided for local buffering of signals and data. The onboard Virtex FPGA is programmed using Verilog HDL; a popular hardware description language[11]. The language has capabilities to simulate and verify a design model using a Verilog simulator. As a programming development environment, Xilinx ISE Foundation Series tools have been used.

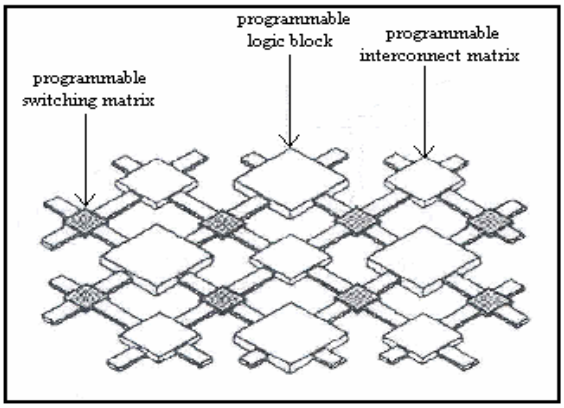

Fig. 1: General FPGA architecture

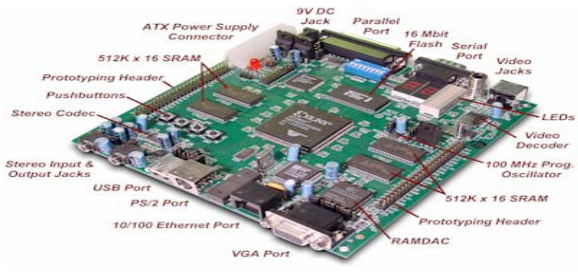

Fig. 2: The XSV-300 FPGA board

FUNDAMENTAL MULTIRATE FILTER BANKS

The fundamental multirate filter banks are the analysis filter bank and the synthesis filter bank [12]. The analysis filter bank is shown in Figure 3a. It consists of two decimators connected in parallel; the upper decimator is a low pass, $H_0(z)$, followed by a down-sampler, and the lower decimator is a high pass filter, $H_1(z)$, followed by a down-sampler. Each down-sampler operates by taking a filtered sequence x[n] and generating an output sequence y[n] according to the relation y[n] = x[2n]. All filtered elements in the subsequence x[2n+1] are discarded. On the other hand, the synthesis filter bank is shown in Fig. 3b. It consists of two interpolators connected in parallel; the upper is a low pass filter, $G_0(z)$, proceeded by an up-sampler, and the lower is a high pass filter,$G_1(z)$, proceeded by an up-sampler [12]. Each up-sampler inserts an equidistant

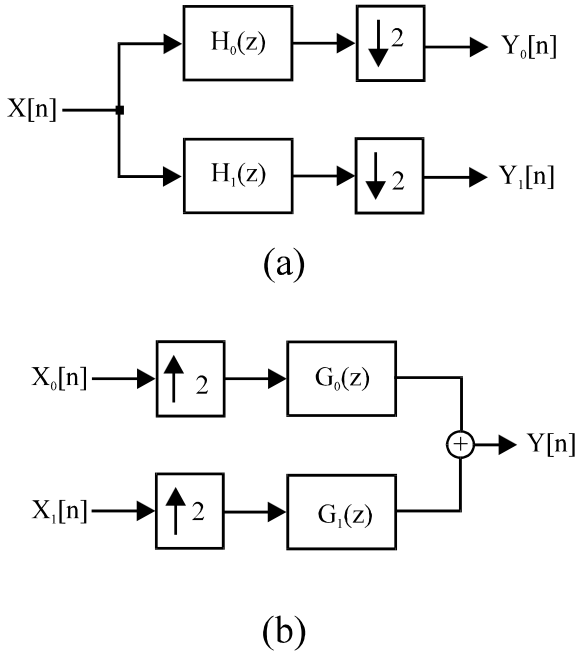

Fig. 3: Fundamental multirate filter banks
(a) Analysis filter bank and (b) Synthesis filter bank

zero-valued sample between every two consecutive samples on the input sequence x[n] . An output sequence y[n] is developed such that y[n] = x[n/2] for even indices of n, and 0 otherwise. This makes the sampling rate of the output sequence y[n] twice as large as the sampling rate of the original sequence x[n].

FIR FILTER IMPLEMENTATION

The effectiveness of any implementation of the two fundamental multirate filter banks depends, to a great extent, on how efficient FIR filters are implemented. This is due to the fact that FIR filters, by virtue of their stability, are the most commonly used filters in multirate systems. In this section we describe three possible implementations of FIR filters; a direct implementation, a serial distributed arithmetic implementation, and a parallel distributed arithmetic implementation.

Direct implementation: An FIR filter of length M is characterized by the transfer function H(z):

$$H(z) = \sum_{k=0}^{M-1} h[k] z^{-k} \quad (1)$$

As shown in Fig. 4, each filter tap consists of a delay element, an adder, and a multiplier [13]. However, a major drawback of this implementation is that filter throughput is inversely proportional to the number of filter taps. That is, as filter length is increased, the filter throughput is proportionately decreased. Nonetheless, we will use this direct structure to implement the fundamental filter banks for the sake of performance comparison.

Serial distributed arithmetic implementation: Distributed arithmetic is an efficient method of inner product computation which constitutes the core of the FIR filter operation[14]. It uses lookup tables and addition in place of multiplication. Compared to lumped arithmetic-based architectures, distributed arithmetic architectures are completive in both speed and hardware requirements. In addition they are extremely regular, which makes them most suitable for programmable logic realization[15].

A simple derivation of the distributed arithmetic methods is as follows[16]. Let the variable Y hold the result of an inner product operation between a data vector x and a coefficient vector a. The distributed arithmetic representation the inner product operation is given as follows:

$$Y = \sum_{j=1}^{B-1}\left[\sum_{i=1}^{N} x_{ij}\alpha_i\right]2^{-j} + \sum_{i-1}^{N}\alpha_i(-x_{10}) = \sum_{j=1}^{B-1} F_j 2^{-j} - F_0 \quad (2)$$

where the input data words x_i have been represented by the 2's complement number presentation in order to bound number growth under multiplication. The variable x_{ij} is the j^{th} bit of the x_i word which is Boolean, B is the number of bits of each input data word and x_{0i} is the sign bit. Distributed arithmetic is based on the observation that the function F_j can only take 2^N different values that can be pre-computed offline and stored in a look-up table. Bit j of each data x_{ij} is then used to address this look-up table. Equation (2) clearly shows that the only three different operations required for calculating the inner product. First, a look-up to obtain the value of F_j, then addition or subtraction, and finally a division by two that can be realized by a shift.

Distributed arithmetic implementation of an FIR filter consists of a look-up table (LUT), a cascade of shift registers and a scaling accumulator[17]. The LUT stores all possible partial products over the FIR filter coefficients, as shown in Fig. 5. Input samples are presented to the input parallel-to serial shift register at the input signal sample rate. As the input sample is serialized, the bit-wide output is presented to the bit-serial shift register cascade, 1-bit at a time. The cascade stores the input sample history in a bit-serial format and is used in forming the required inner-product computation. The bit outputs of the shift register cascade are used as address inputs to the look-up table. Partial results from the look-up table are summed by the scaling accumulator to form a final result at the filter output port.

Since the LUT size in a distributed arithmetic implementation increases exponentially with the number of coefficients, the LUT access time can be a bottleneck for the speed of the whole system when the LUT size becomes large. Hence, we decomposed the 8-bit LUT into two 4-bit LUTs, and added their outputs using a two-input accumulator. The partitioned-LUT FIR filter architecture is shown in Fig. 6. The total size of storage is now reduced since the accumulator is less costly than the larger 8-bit LUT. Furthermore,

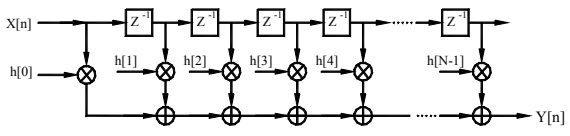

Fig. 4: Direct FIR filter structure

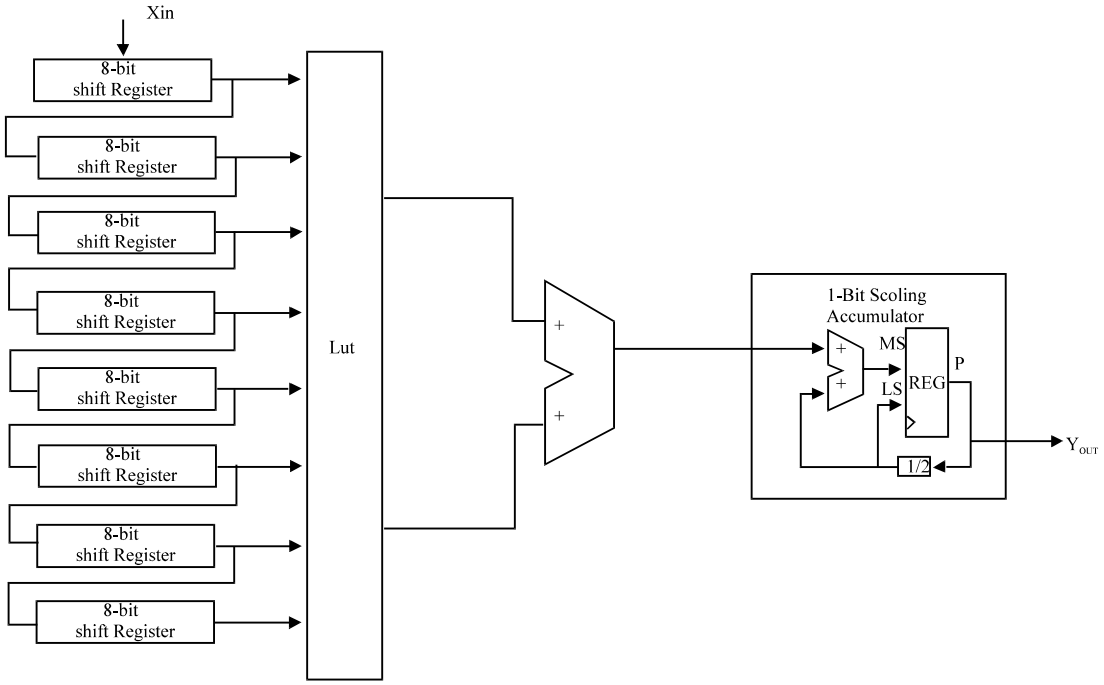

Fig. 5: Serial distributed arithmetic implementation of the FIR filter

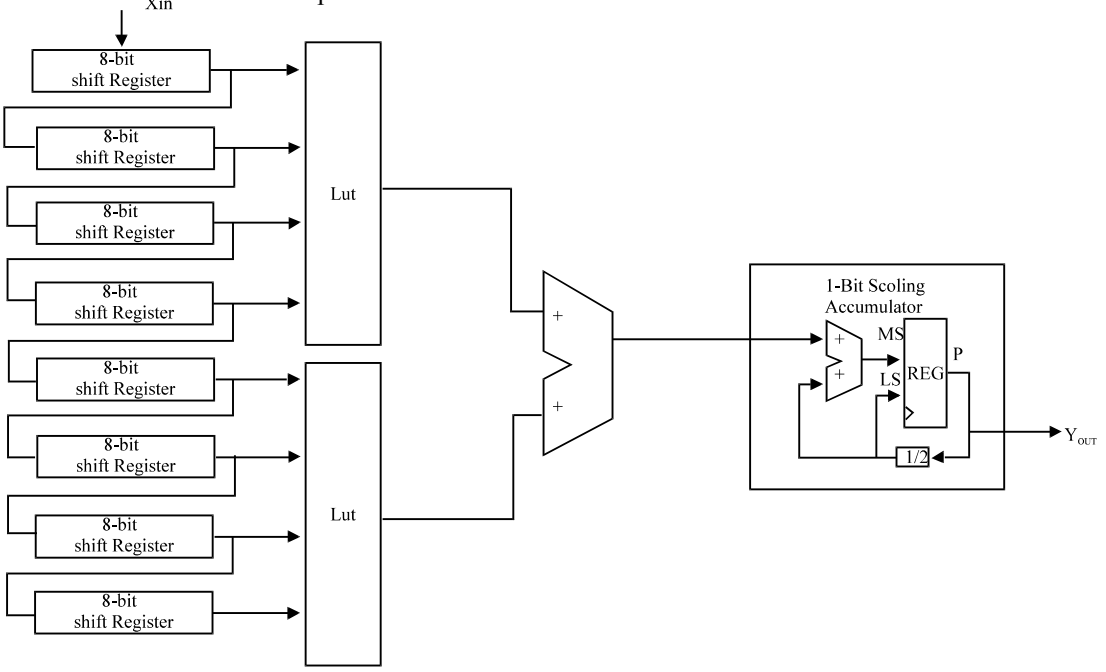

Fig. 6: Partitioned serial distributed arithmetic implementation of the FIR filter

partitioning the larger LUT into two smaller LUTs accessed in parallel reduces access time. In addition, throughput of the filter is maintained regardless of the length of the FIR filter.

Parallel distributed arithmetic fir filter: As with most hardware applications, we can obtain more performance by using more hardware. In this case, more than one bit sum can be computed at a time by

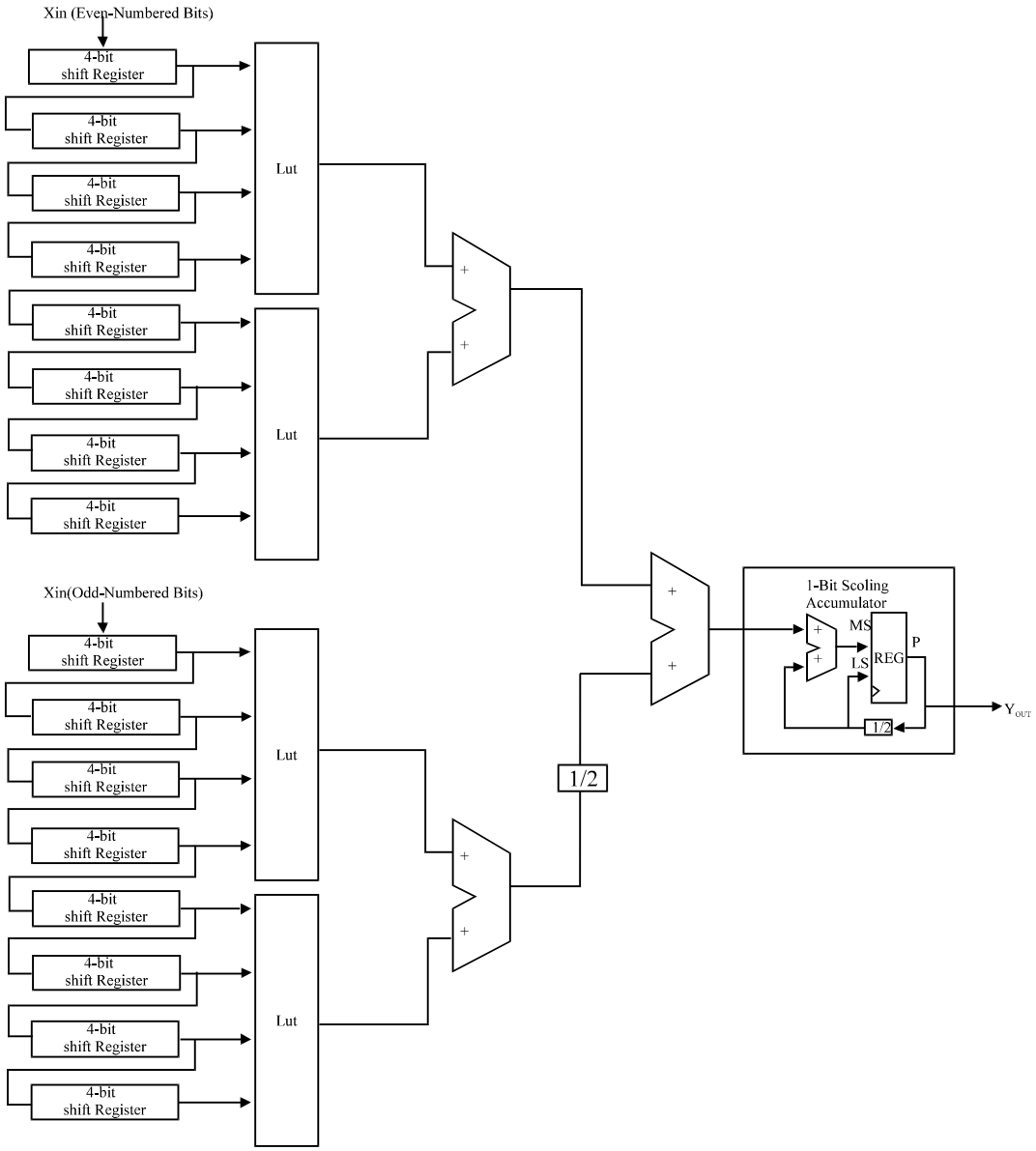

Fig. 7: Two-bit parallel distributed arithmetic implementation of the FIR filter

duplicating the LUT and adder tree. A parallel implementation of the inherently Serial Distributed Arithmetic (SDA) FIR filter, shown in Fig. 6, corresponds to partitioning the input sample into M sub-samples and processing these sub-samples in parallel. Such a parallel implementation requires M-times as many memory look-up tables and so comes at a cost of increased logic requirements. We describe below the implementation of our PDA FIR filter at two different degrees of parallelism; a 2-bit PDA FIR filter and a fully parallel 8-bit PDA FIR filter.

A 2-bit Parallel Distributed Arithmetic (PDA) FIR filter implementation is shown in Fig. 7. It corresponds to feeding the odd bits of the input sample to an SDA LUT adder tree, while feeding the even bits, simultaneously, to an identical tree. Compared to the serial DA filter, shown is Fig. 6, the shift registers are each replaced with two similar shift registers at half the bit size. The odd bit partials are left shifted to properly weight the result and added to the even partials before accumulating the aggregate by a 1-bit scaling adder. Finally, since two bits are taken at a time, the scaling accumulator is changed from 1-to-2-bit shift (1/4) for scaling.

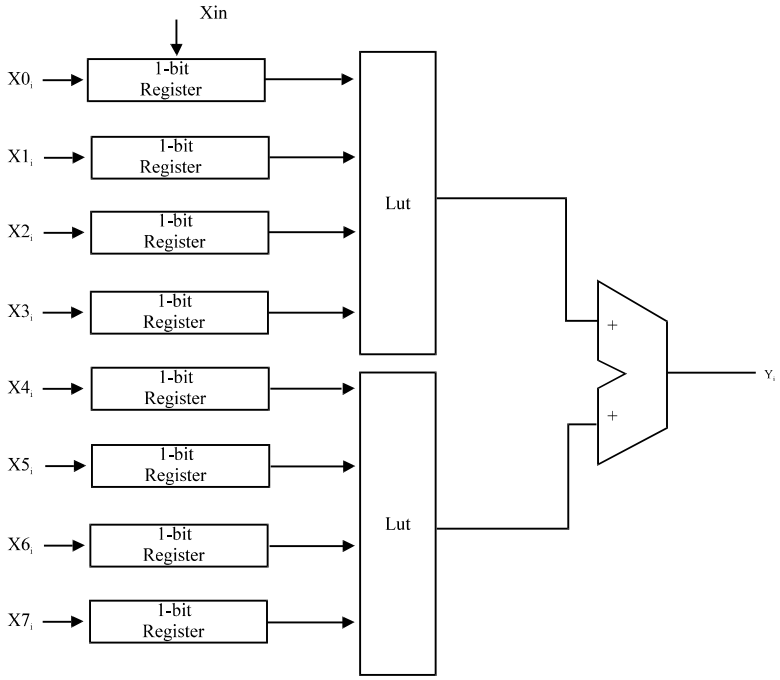

Fig. 8: Single-bit parallel distributed arithmetic implementation of the FIR filter

As for the fully parallel 8-bit PDA FIR filter implementation, the 8-bit input sample is partitioned into eight 1-bit sub-samples so as to achieve maximum speed. Fig. 8 and 9 show the ultimate fully parallel PDA FIR filter, where all 8 input bits are computed in parallel and then summed by a binary-tree like adder network. The lower input to each adder is scaled down by a factor of 2. No scaling accumulator is needed in this case, since the output from the adder tree is the entire sum of products.

ANALYSIS FILTER BANK IMPLEMENTATION

In this section, we describe an FPGA-based implementation of the analysis multirate filter banks. We first describe the architecture of the filter bank as it will be implemented on the Viretx FPGA. Next, we present a simulation waveform which verifies the functionality of the analysis filter bank implementation. Finally we present performance figures. The implementation has been physically realized on the XSV-300 FPGA prototyping board described earlier in this paper.

Architecture: We implemented the analysis filter bank as shown in Fig. 10. For the sake of performance comparison, the FIR filter block diagram shown in the figure was implemented in three structural forms; direct, serial and parallel distributed arithmetic structures. In the figure, an active-high output control pin, labeled DATA RDY, has been implemented in the FIR filter and connected directly to the CLK input of a 1-bit counter. The input port of the FIR filter is connected to the input samples source, whereas the output port is connected to a parallel-load register. The register loads its input bits in parallel upon receiving a high signal on its CLK input from the counter, and blocks its input otherwise.

Assuming unsigned 8-bit input samples, each of the two parallel decimators operates as follows. When the DATA RDY signal is activated, the FIR filter triggers the counter to advance to the next state every time it completes a filtering operation. If the new state is 1, the parallel-load register is activated, and it stores the data received at its input from the FIR filter. If the new state is 0, the register is disabled, and consequently the FIR output is blocked from entering the register, and ultimately discarded. The above procedure repeats, so that when the counter has 1 on its output, the FIR data is stored , and when it has a 0 on its output, the FIR data is discarded. Fig. 10: FPGA-based implementation of the analysis filter bank.

Functional simulation: The analysis filter bank was modeled and verified using Verilog's functional simulator. The corresponding simulation waveform is

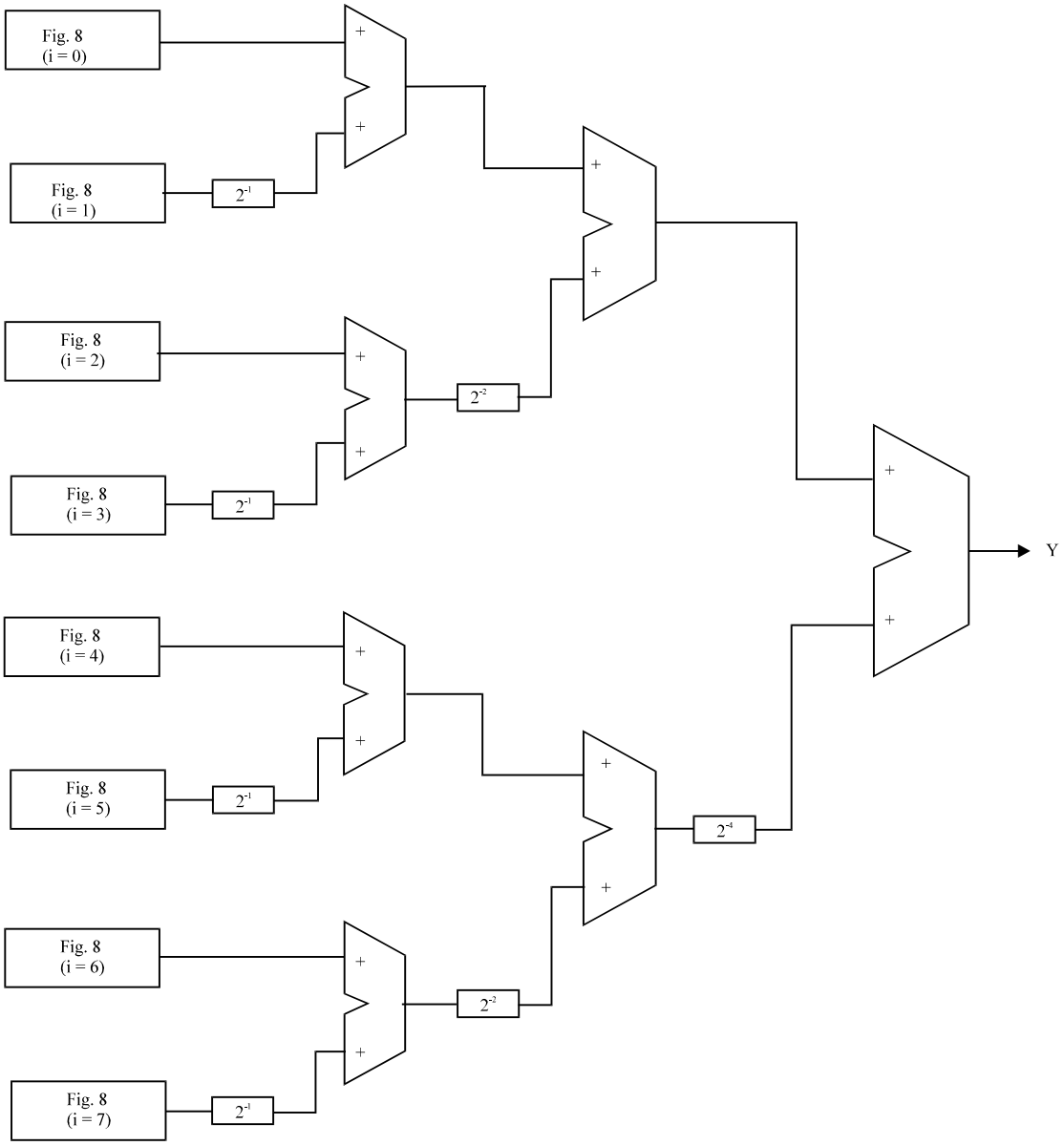

Fig. 9: Eight-bit parallel distributed arithmetic implementation of the FIR filter

shown in Fig. 11. As shown in the figure, an input sample, X, has a rate of 1sample/1 clocks and the two output samples, Y_0 and Y_1, have a rate of 1sample/ 2clocks. The output sampling rate of the output is clearly half the input sampling rate. We also maintained sufficient precision of the analysis filter bank outputs, Y_0 and Y_1, where their values have been represented using 20 bits. Allocating sufficient bits to the intermediate and output coefficients has been a necessary step to keep the perfect synthesis capabilities of the synthesis filter bank.

Performance Evaluation: The analysis filter bank has been implemented using four different structures of the FIR filter; direct structure, serial distributed arithmetic structure, 2-bit parallel distributed arithmetic structure, and 8-bit parallel structure. The throughput (MHz) and hardware utilization (slices) of the four implementation are listed in table 1. It is noted from these results that the 8-bit parallel distributed arithmetic implementation outperforms the other implementations with regard to the throughput performance. However, this parallel implementation requires the most hardware resources,

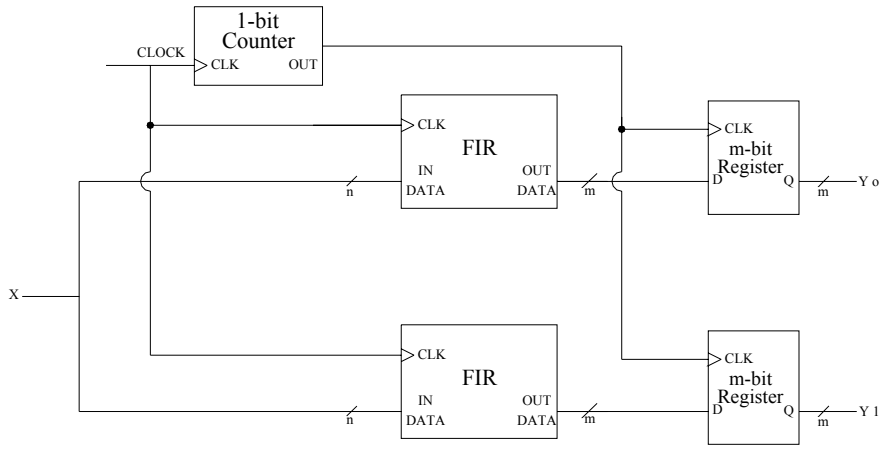

Fig. 10: FPGA-based implementation of the analysis filter bank

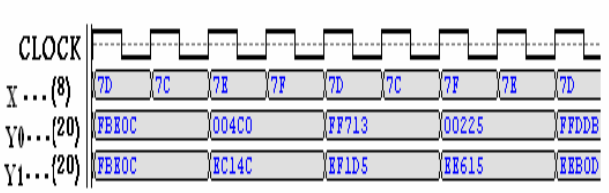

Fig. 11: Simplified functional Verilog simulation of the analysis filter

and this is expected since more parallelism exploitation requires more hardware resources.

SYNTHESIS FILTER BANK IMPLEMENTATION

In this section, we describe an FPGA-based implementation of the synthesis multirate filter banks. We repeat what we have done with the analysis filter bank. First, we describe the FPGA-based architecture of the synthesis filter bank. Next, we present the simulation waveform which verifies the functionality of the synthesis filter bank implementation, and finally we present performance figures. The implementation described has been physically realized on the XSV-300 FPGA prototyping board described earlier in this paper.

Architecture: We implemented the synthesis filter bank as shown in Fig. 12. The input port of the FIR filter is connected to the output port of a parallel-load register; whereas the input port of the register is connected directly to the input samples source. The operation of the register depends on the signal received on its active-high CLR (clear) input from the most significant output bit of a 4-bit counter.

Assuming the input samples source sends out successive samples separated by 16 clock periods, the interpolator operates as follows. Let an input sample be transferred, through the parallel-load register, to the FIR filter. The transfer process takes place during the first eight counts of the 4-bit counter in which the counter's MSB remains 0, thus enabling the register to transfer its input data to its output port. During the next eight counts, the MSB of the count becomes 1, and thus clearing the register and consequently transferring zeros to its output. The zero output is maintained until the last count (FFFF H) is reached. The above procedure repeats so that an input sample enters the FIR filter during the first eight clocks, followed by a zero during the next eight clocks, and so on.

Functional simulation: The synthesis filter bank operation was modeled and verified using Verilog's functional simulator. The simulation waveform is displayed in Fig. 13. As shown in the figure, the synthesis filter bank up-samples each of the two input samples arriving at a rate of 1 sample/2 clocks and sends out their filtered summation at a rate of 1sample/1clock.

Performance Evaluation: Similar to the analysis filter bank, the synthesis filter bank has been implemented using different FIR structural forms; direct structure, the serial distributed arithmetic structure, 2-bit parallel distributed arithmetic structure, and 8-bit parallel distributed arithmetic structure. The throughput and hardware utilization of the four implementation are listed in Table 2, for the sake of comparison. The results demonstrate again the superiority of the parallel distributed arithmetic implementation over the other

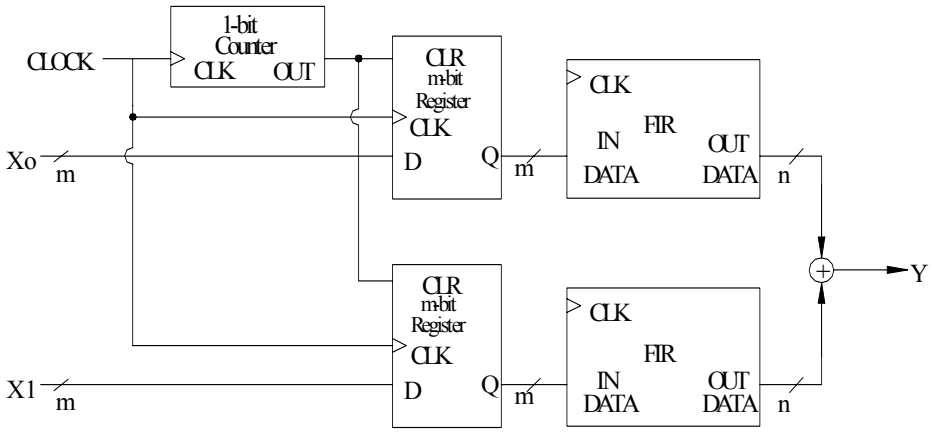

Fig. 12: FPGA-based implementation of the synthesis filter bank

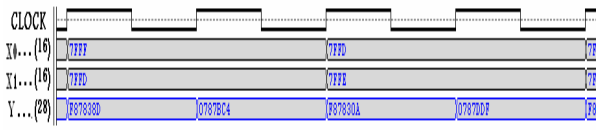

Fig. 13: Simplified functional Verilog simulation of the analysis filter

Table 1: Performance of different implementations of the analysis filter bank.

Implementation	Throughput (MHz)	Utilization (slice)
Direct	42.33	186 (6%)
Serial Distributed Arithmetic	78	125 (4%)
2-bit Parallel Distributed Arithmetic	144.3	277 (9%)
8-bit Parallel Distributed Arithmetic	178.2	556 (18%)

Table 2: Performance of different implementations of the synthesis filter bank

Implementation	Throughput (MHz)	Utilization (slice)
Direct	34.8	206 (7%)
Serial Distributed Arithmetic	71.1	154 (5%)
2-bit Parallel Distributed Arithmetic	139.5	338 (11%)
8-bit Parallel Distributed Arithmetic	169.5	676 (22%)

implementations. Its also worthwhile comparing the performance of the analysis and synthesis filter banks. Referring back to Tables 1 and 2, its obvious that the throughput of the synthesis filter bank is lower than the throughput of the analysis filter bank. This is due to the fact that, the synthesis filter bank is a little more computation-intensive than the analysis filter bank by virtue of the up-sampling operation which inserts a zero sample between every two successive input samples. Moreover, the synthesis filter bank architecture employs a summation node at the end, as shown in Figure 3b, to reconstruct two incoming signal. This is reflected in a slight increase in the number of Virtex slices compared to the slices needed by the analysis filter bank.

RESULTS AND DISCUSSION

We carried out four different implementations of the fundamental multirate filter banks as follows. The first was a conventional implementation in which the direct FIR structure was used. The second, third and fourth implementations were based on serial and parallel distributed arithmetic.

Referring back to the results given in Tables 1 and 2, its noted that the throughput of any of the three distributed arithmetic implementations is higher than the throughput of the direct conventional implementation. This is due to the fact that, distributed arithmetic replaces the time-consuming conventional multiply accumulate operations with fast look-up tables and shift operations. Furthermore, partial products of all multiply accumulate operations were pre-computed offline and stored in the LUTs, thus saving a great a mount of real-time computation. As for Virtex slice utilization, the serial distributed arithmetic implementation uses less hardware resources than the direct implementation which uses conventional arithmetic. This is because the conventional arithmetic multiplier requires more logic resources than the distributed arithmetic multiplier which requires small LUTs, simples adders and shift registers.

Comparing the performance figures of the serial and parallel distributed arithmetic implementations, we note that there is a considerable throughput increase for

a reasonable increase in slice count between the serial and the 8-bit parallel distributed arithmetic implementation. The results clearly demonstrate the speed/cost scalability of the distributed arithmetic algorithm, and suggest that in between the serial distributed arithmetic and fully parallel distributed arithmetic there exist opportunities to increase performance by a factor of two or more, with corresponding reasonable increase in logic requirements.

Finally, its worth mentioning that we carried out two software implementation of the fundamental filter banks. The first was on an advanced digital signal processor [18], and the other on a Pentium III processor. The performance figures we obtained were much inferior to all FPGA implementations described in this paper.

CONCLUSIONS

Multirate digital signal processing systems employ filter banks to enable sampling rate conversion of digital signals. Efficient and high speed implementations of fundamental multirate filter banks are required. In this paper, we presented several implementations of the fundamental filter banks, analysis and synthesis filter banks, and showed that the one based on parallel distributed arithmetic gave the best performance results. The implementation was carried out on an FPGA-based, reconfigurable hardware platform, which is well-suited for the implementation of distributed arithmetic lookup tables. Performance results demonstrated clearly the effectiveness of parallel distributed arithmetic on FPGA-based platforms.

REFERENCES

1. Jovanovic-Dolecek G., 2002. Multirate Systems: Design and Application. Idea Group Publishers,
2. Harris F., 2004. Multirate Signal Processing for Comm. Systems. Prentice Hall.
3. Proakis J. and D. Manolakis, 1995. Digital Signal Processing. Prentice Hall.
4. Vaidyanathan P., 1993. Multirate Systems and Filter Banks. Prentice Hall.
5. Wolf W., 2004, FPGA-Based System Design. New Jersey: Prentice Hall.
6. Xilinx Corporation, 2000. Virtex Data Sheet.
7. White S., 1989. Applications of Distributed Arithmeti to Digital Signal Processing: A Tutorial, in IEEE ASSP Magazine, pp: 4-19.
8. Seals, R. and G. Whapshott, 1997. Programmable Logic: PLDs and FPGAs. UK: Macmillan.
9. Xess Corporation. www.xess.com <http://www.xess.com>.2002.
10. Xilinx Corporation, 1998. Xilinx breaks one million-gate barrier with delivery of new Virtex series.
11. Ciletti M., 1999. Modeling, Synthesis, and Rapid Prototyping with the Verilog HDL, New Jersey: Prentice Hall.
12. Vaidyanathan P., 1990. Multirate Digital Filters, Filter Banks, Polyphase Networks, and Applications: A Tutorial, in Proceedings of the IEEE, vol. 78, no.1, pp: 56-93.
13. Mitra S., 1998. Digital Signal Processing. McGraw Hill.
14. Koren I., 1993. Computer Arithmetic Algorithms, Prentice Hall, New Jersey.
15. New B., 1995. A Distributed Arithmetic Approach to designing scalable DSP Chips, EDN Magazine.
16. Peled A., A New Hardware Realization of Digital Filters, IEEE Trans. On Acoustics, Speech and Signal Processing., Vol. 22, No. 5, pp: 456-462, December 1973.
17. Taylor F., An Analysis of the Distributed Arithmetic Digital Filter, IEEE Trans. On Acoustics, Speech and Signal Processing, Vol. 34, No. 5, pp: 1165-1170, May 1986.
18. Texas Instruments Corporation,2000. TMS 320C6711 Data Sheet.