EFFICIENT MULTI-CHANNEL FIR FILTERS IN FPGA

Péter SZÁNTÓ, János LAZÁNYI and Béla FEHÉR
Department of Measurement and Information Systems,
Budapest University of Technology and Economics
Budapest, Hungary,
{szanto, lazanyi, feher}@mit.bme.hu

Abstract: This paper presents an FIR filter architecture suitable for embedded FPGA based applications. With the limited resource requirements and its high performance, the architecture is suitable to implement real time, multi-channel filtering structures even in smaller FPGAs.

Keywords: Digital Signal Processing, FIR, FPGA

1 Introduction

High quality filtering is often required in a variety of embedded designs, such as multi-channel audio, imaging, or measurement applications. While today's DSPs (Digital Signal Processor) are very efficient in implementing filter algorithms, an FPGA (Field Programmable Gate Array) based solution has clear advantages in high performance, multi-channel applications, where the performance of comparably priced DSPs is not enough. On the one hand, new FPGA families offer a large number of high performance, embedded hardware multipliers to implement MAC (Multiply and ACcumulate) units, while on the other hand hard- or soft core microprocessors are well suited to implement control functions, and communication with the environment.

High quality audio and measurement applications often require more than 100 kHz sampling frequency and more than 16 bit resolution as a minimum. For large size filters, like a 100 tap FIR filter, processing of one channel requires 10 million multiply and accumulate in a second. Adding more channels linearly increases this requirement – the sequential approach of DSPs may make real time processing impossible very soon. In contrast, FPGAs allow high level of parallelism, where adding more channels is only limited by the available resources.

1.1 FIR Filter in FPGAs

FPGAs offer many different implementation possibilities for FIR filters. The basic FIR filter structures consist of a delay line and a computational block composed of

multipliers and adders (Figure 1). Direct hardware implementations of the architecture can employ registers in the delay line and different type of arithmetic functional modules. The convolution sum of the FIR filter is basically an inner product operation of two vectors, the input signal samples and the coefficients. Frequently the coefficients are fixed values and will not be changed during the operation. In this case very efficient inner product processors can be used, like in [1], and [2], based on the so called distributed arithmetic. The name origins from the property, that there are no independent multiplier blocks, all of them are immersed into the computational array. In spite of the bit serial architectural style, these kinds of solutions can provide competitive execution speed, especially in case of large number of taps, because the sample rate is almost independent from the filter order.

In case of adaptive or configurable filters the usual, memory and MAC based solutions are easier to use [3] [4]. Here the maximum sample rate is reverse proportional with the filter order, if one use a single MAC unit only, so large filters or filter banks needs parallel architectures to maintain the speed requirements. Parallelization can be introduced on filter level, or sub-filter level, but in the later case additional adder tree is required to form the final output result. Filter bank implementations can use independent sample and coefficient memory, and can be controlled in a synchronized way.

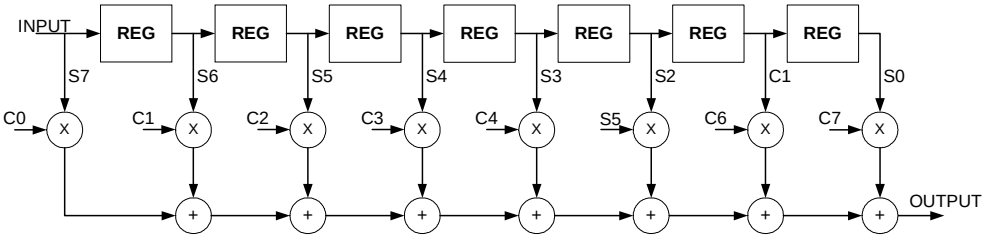

Figure 1. Basic FIR filter structure

1.2 Multiplier implementations

Direct hardware implementation of FIR filters were implemented by memory based look-up table multipliers or true distributed arithmetic data flow inner product processors. These solutions were very flexible in input sample and coefficient size, but required a lot of general purpose resources of the FPGA [4].

Recent FPGAs have dedicated hardware multipliers [6]. Some vendors have fixed width signed multiplier (the typical width is 18 bit), while others allow some kind of configurability, but in all cases this is much more limited in flexibility, than LUT based solutions. At the same time performance of the dedicated multipliers is much greater: older architectures can reach about 150 MHz, while the newest families work over 400 MHz.

2 Architecture

The presented architecture – which allows 32 bit input samples and 32 bit coefficients to be used – is implemented in a Xilinx Virtex2-Pro FPGA[5]. The basic block diagram is shown on Figure 2.

Besides having one multiplier and accumulator, the MAC units contain relatively large sample and coefficient memory. The size of these memories is large enough to store

more input sample and coefficient arrays, therefore cascaded filters with different coefficients can be used to implement, for example, input compensation.

The control part of the MAC units is centralized, and implemented as a fast state machine. All control signals of the MACs, are generated by the control logic.

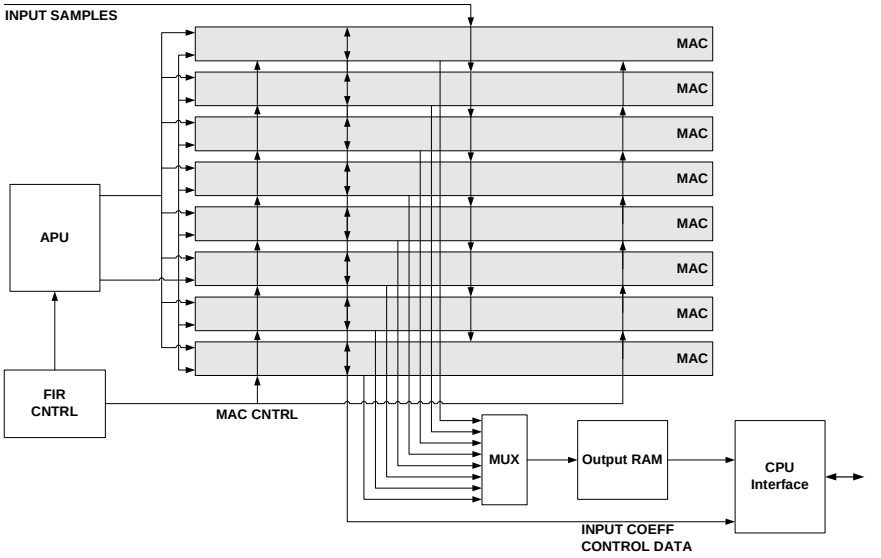

Figure 2. Basic block diagram

2.1 MAC unit

The MAC unit is a serial implementation with one 18 bit multiplier, as Figure 3 shows. The multiplier is fed by the sample and coefficient memories, which are 18 bit wide. The output of the multiplier is connected to one input of the accumulator. The other input of the accumulator can be the output of the accumulator, or the output of the accumulator scaled down by 16 bit.

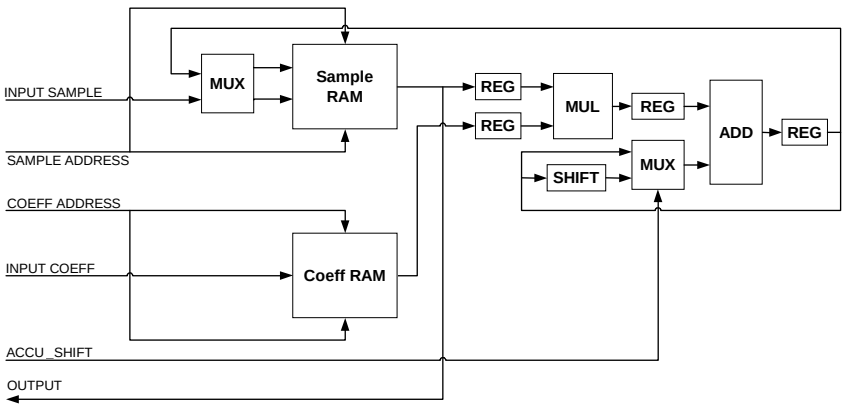

Figure 3. MAC unit

The accumulator output can be written back into the sample memory. Both this and the coefficient memory have external data and address inputs, which are controlled by the Address Processing Unit. Memories have 18 bit word width to suit the multiplier, both the 32 bit input data and the 32 bit coefficient are separated into two 18 bit words in the appropriate memory. As memories are dual-ported, write operations can happen in one clock cycle using both ports.

To generate one result, three multiplications are required:

$$RES = ((C_{LSW} * S_{MSW} + C_{MSW} * S_{LSW}) >> 16) + C_{MSW} * S_{MSW}$$

Where C and S denote coefficient and sample, while MSW is the upper 18 bit word and LSW is the lower 18 bit word, respectively. Our analysis showed that the fourth partial result (LSW*LSW) has hardly any effect on the output result, as it is masked by the quantization noise. When filtering, the above three partial multiplication results are computed for all tap number samples, starting with LSW*MSW types in order not to lose precision. The sum of the first two partial results should be scaled down before adding it to the third partial result.

2.2 Control Logic

As mentioned earlier, the control signals and the memory addresses of the MAC units are generated externally, respectively by the FIR Control Logic (FCL) and the Address Processing Unit (APU). Figure 4 shows the details of these units.

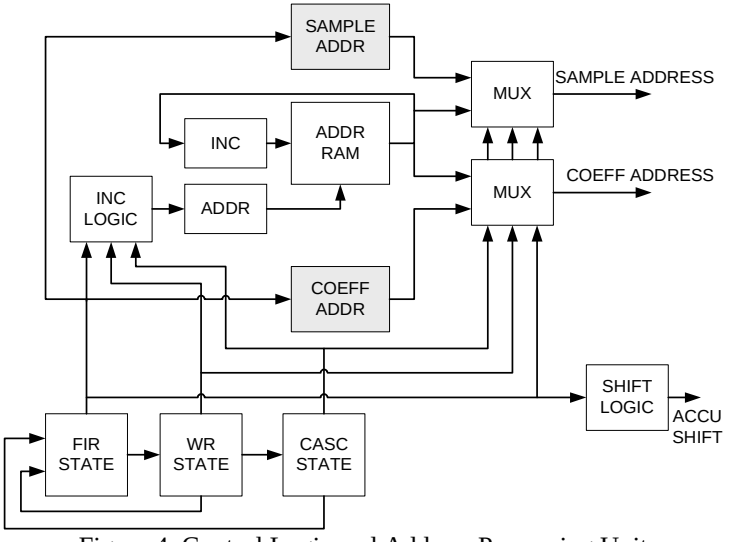

Figure 4. Control Logic and Address Processing Unit

In the current implementation, both the sample and coefficient memory can store four blocks of data. This allows four different filters to be applied to an input, and also makes decimation possible with the appropriate addressing scheme. Some words of the first block are reserved as special purpose storage space. The 32 bit data are stored in two consecutive

words: the 18 bit LSW is stored at even addresses, while the MSW is stored at odd addresses.

The control unit is mainly composed of three state machines. FIR STATE defines the three multiplication states during filtering. WR STATE is responsible for managing result write back to sample memory and sample write address management. CASC STATE defines the states when multiple filters are cascaded together and applied to the input samples.

The address generator has three main parts: a counter for sample addresses during filtering (shown gray), a counter for coefficient addresses for filtering (shown gray) and a more general address memory.

The unit starts processing after the new input samples have arrived; the new samples are written into the sample memories at the appropriate address ("sample write address 0"), which is defined by a word in the address memory. After this, the unit gets into filtering stage, and applies the filter stored in the first memory block of the coefficient memory. During this stage, sample address and coefficient address registers are used to address the memories; just like with any other addresses, the mentioned registers identifies a two-word block in the memory, the least significant address bit is generated according to the state registers. The filtering itself takes three times the tap number clock cycles due to the implementation of the wide multiplier. At the end of this stage, the result is written back into the sample memory to two spaces: to the reserved output space, and to the appropriate address of the next block. This appropriate address is also stored in the address memory. During the write back stage, the write address of the input sample ("sample write address 0") is incremented, so that the next input sample will be written to the correct address. Further, cascaded filter stages work exactly the same; just the samples are read from and written to the appropriate blocks in the sample memory. The number of filter stages to do is defined in a mode register, and CASC STATE is responsible for applying the correct filters.

Decimation does not really change the working mechanism, just the address generators increments slower: for example, the write address of the second memory block only increments once every two input samples. Therefore, two filtered results generated from the samples in the first memory block are written to the same address in the second memory block, the second one being the real decimated output. As there are four blocks in the sample and coefficient memories, eight times decimation can be achieved this way

3 Results

A sample application was implemented using a Xilinx Virtex2-Pro FPGA. The filter structure achieved over 160 MHz clock frequency.

Table 1 summarizes the required logic resources. As can be seen, the solution is really resource efficient with minimum number of additional components.

Table 1. Resource usage of the filterbank

	LUT	FF	MUL	BRAM
MAC	~100	~80	1	2
CNTRL + APU	~90	~80	-	-

The timing parameters of the filter structures were tuned by design constrains. The design goal was above 100 MHz clock rate to maintain the minimum 100 kHz input signal frequency. The critical paths were reduced by simple pipeline stages, so careful balancing

of the internal logic delays made possible to achieve a 160 MHz clock rate, which is much better, than the minimum requirement.

The filter design was realized by the Matlab Filter Design Tool. For the required parameters, larger than 100 dB selectivity above the Nyquist frequency, about a 100 tap linear phase FIR filter was necessary. The special implementation of the 32 bit multiplication do not violated the large stop band attenuation; the quantization noise remained below the specification. The pass band ripple is less than 0.01 dB, which means a very flat transfer characteristic for the filter. The transfer function of one designed filter is shown on Figure 5.

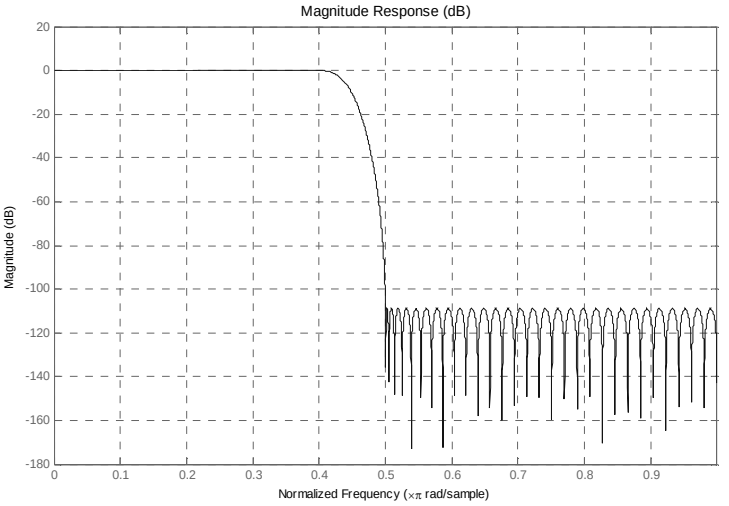

Figure 5. The transfer function of one designed FIR filter

References

[1] FEHÉR, B. 1983. Efficient Synthesis of Distributed Vector Multipliers. In Microprocessing nad Microprogramming Volume 38, Numbers 1-5. ISSN 0165-6074 pp. 345-350.

[2] FEHÉR, B. 1984. Coefficient dependent logic synthesis of FIR digital filters Microelectronics Journal, vol. 25. 1994. pp.228-235.

[3] VALLS, J, PEIRÓ, M, SANSALONI, T, BOEMO, E. 1998. A Study about FPGA-based Digital Filters, IEEE SIPS

[4] VALLS, J, BOEMO, E, Efficient fpga-implementation of two's complement digit-serial/parallel multipliers, 2003, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Volume: 50 Issue: 6 , Page(s): 317 -322

[5] Virtex-II Pro Data Sheet, Xilinx Inc. http://www.xilinx.com.

[6] XtremeDSP Design Considerations User Guide, Xilinx Inc. http://www.xilinx.com