FIR Filter Implementation for High-Performance Application in a High-End FPGA

Stefan Pijetlović, *Student Member, IEEE,* Miloš Subotić, *Student Member, IEEE*, Vladimir Marinković, *Member, IEEE*, and Nebojša Pjevalica, *Member, IEEE*

Abstract — **In this paper a high-performance application which uses multiple 48k tap FIR filters is presented. Due to its size, complexity and restrictions such as real-time, small latency and large memory bandwidth, the filter was implemented in UltraScale+, a high-end FPGA from Xilinx. The system was verified using a gold reference model written in C (high-level algorithm verification) and an analytical model calculated manually. The system was also tested using a development board and SystemVerilog (for register-transfer level and timing verification). The obtained results show a perfect match between the reference models and the actual output. The main novelty of the paper is the implementation of such an immense real-time signal processing system based on FIR filters consisting of over a million taps all together in a single design spread out across a chip containing three dies. Details about the resources allocated within the FPGA are also given in a table in the results chapter.**

Keywords — **FIR filter, FPGA, real-time.**

I. Introduction

THE traditional approach to Finite Impulse Response (FIR) filter implementation for high-performance applications usually involves Application-Specific Integrated Circuits (ASICs) or dedicated Digital Signal Processing chips (DSPs). However, in recent years the market pressure as well as advances in technology have made Field-Programmable Gate Arrays (FPGAs) a viable alternative in certain cases. The high cost development of ASICs as well as long time-to-market periods has made this solution less attractive for some applications, especially when there is fierce competition involved. DSPs have the issue of reduced flexibility due to their sequential-execution architecture which can prevent them from achieving the desired performance in certain cases, such as when the bandwidth they provide is not sufficient. On the other hand, FPGAs offer a balance of costs, time-to-market, performance and flexibility which the traditional approaches lack [1] while still retaining the possibility to cover real-time use cases which was one of the critical parts of this application.

FIR filters in FPGA is not a new concept. There are many published papers regarding this subject [2]-[4]. However, the length of the filters or the number of taps is relatively small compared to the scale of this application, even for newer published works [5], [6]. Even when the difference between the number of taps was not several orders of magnitude, these applications were used in systems whose impulse response lasted several seconds or more. One of the main features and restrictions with this system is the real-time aspect – the overall latency of the system has to be below 2 microseconds. This further contributes to the novelty of the paper alongside the space required on the chip due to the system's size.

One of the issues associated with large chips is a higher chance of errors during the manufacturing process. Larger designs require larger areas of silicon which in turn have a greater probability to be faulty. The overall trend present in the semiconductor industry is increasing the chip complexity because certain features are being implemented in ever smaller areas. Because the defect density does not always decrease over successive technology generations and the complexity of the chips increases, the probability of having faults within a chip is rising [7]. All of this contributes to the increase of production cost. To combat this issue a solution emerged in a form of having multiple chips working in conjunction in the same package, creating the so called System in Package (SiP). SiP is similar to System on Chip (SoC), but the circuits are more loosely integrated and not on a single semiconductor die.

The challenge with designing these kinds of systems is how to distribute the components of the system and not lose out on performance when the two or more chips have to communicate between each other. In this particular case, a highly intensive application is requested which includes multiple adaptive FIR filters. Each filter has 48k coefficients, working at a 48 kHz sampling rate and yielding a 1 second impulse response with some of the coefficients being shared between the filters. The entire process is linked to the Personal Computer (PC) where all the pre and post-

Paper received March 29, 2019; revised May 20, 2019; accepted May 22, 2019. Date of publication July 31, 2019. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Branimir Reljin.

This paper is revised and expanded version of the paper presented at the 26th Telecommunications Forum TELFOR 2018 [13].

This project was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia under the grant TR32029, year 2018.

Stefan Pijetlović - RT-RK Institute for Computer Based Systems, Radnička 30a, 21000 Novi Sad, Serbia (e-mail: stefan.pijetlovic@rt-rk.com).

Miloš Subotić - RT-RK Institute for Computer Based Systems, Radnička 30a, 21000 Novi Sad, Serbia (e-mail: milos.subotic@rt-rk.com).

Vladimir Marinković - RT-RK Institute for Computer Based Systems, Radnička 30a, 21000 Novi Sad, Serbia (e-mail: vladimir.marinkovic@rt-rk.com).

Nebojša Pjevalica - University of Novi Sad, Faculty of Technical Sciences, Computing and Control Engineering Department, Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia (e-mail: pjeva@uns.ac.rs).

processing is managed. The amount of processing power needed for the application is not a critical part when compared to the necessary memory bandwidth due to required parallelism. This is the main reason why the Central Processing Unit (CPU) could not be used for such an application due to its limits when working with memory intensive problems. Even though the CPU could work in conjunction with the cache memory, there is simply far too much work which needs to be done in parallel and most caches do not have enough ports to enable sharing data between many processing units. One possible solution which could provide sufficient computation power and memory bandwidth is the Graphical Processing Unit (GPU). However, GPUs were designed to provide immense throughput and parallelism in computing. As such, even the high-end GPUs introduce latency which is greater than specified for the application and so this approach was discarded. In the end, a high-end FPGA UltraScale+ from Xilinx was used along with the VCU118 development board [8].

II. Proposed architecture

The main reason why this large high-end FPGA was chosen was the achievable memory throughput. Smaller FPGAs have enough processing power but are limited by their Block Random Access Memories (BRAMs) which are agile (accessible every clock cycle) but with limited capacity. If the amount of memory provided by the BRAMs is insufficient for the application, then Dynamic Random Access Memory (DRAM) has to be used. This type of memory does not enable access per every clock cycle which in turn reduces the overall speed and throughput of the system. The FPGA used in this project has Ultra Random Access Memory (URAM) which increases the size of the internal BRAM up to 6 times [9]. Another requirement which contributed to the complexity of the design was the floating-point representation of the samples, so that the PC could manage them. Most FIR filters and DSPs use fixed-point arithmetic so a special conversion block was needed which impacted the responsiveness and latency of the system.

The data flow (shown in Fig. 1) is the following: first, the audio data is sampled by an analogue-digital converter (ADC) at a sampling rate of 48 kHz. The audio comes from multiple channels which all need to be filtered in parallel.

When sampled, the audio signal is transmitted via I2S protocol to the FPGA from where it is forwarded to the PC using the Peripheral Component Interconnect Express (PCIe) bus. Initially on the PC, the input samples are pre-processed and sent back to the FPGA where the filtering is done. Afterwards, the filtered signal is sent back to the PC via PCIe for the post-processing. In the end, the finally processed signal is sent to the digital-analogue converter (DAC) and finally out through the speakers. A dedicated Linux driver for handling the PCIe protocol was written for this purpose. The pre and post-processing are simple and low demanding operations which can be achieved in real-time on the PC such as volume control. These are customer specific blocks not vital to the proper functioning of the FIR filter.

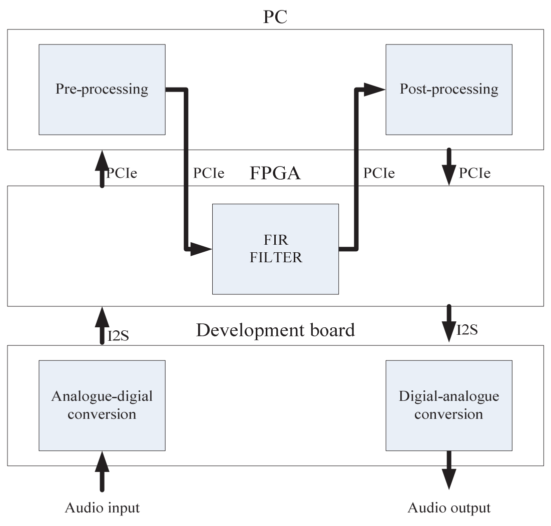

Fig. 1. Data flow.

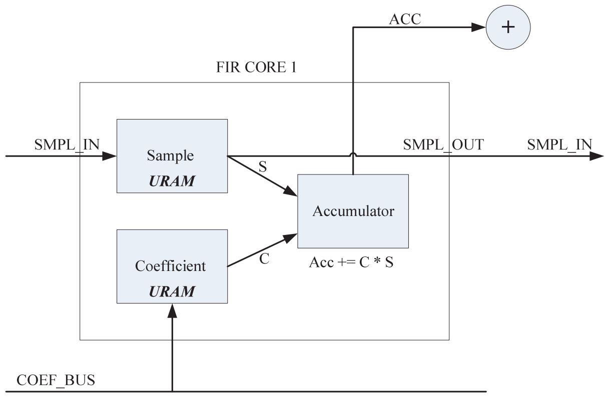

Fig. 2. FIR Core.

There are several ways to obtain a specific number of taps for a single filter. The requirement for the system stated that each filter will have 48k taps, which equals 1 second of the impulse response time and that there will be 72 of these filters. Since the nature of the problem was filtering several channels at once, it made sense to have a dedicated processing unit for each channel. The input signal was audio with a sampling rate of 48 kHz. This means that 48k taps could be obtained by having 48k processing cores working in parallel at the same frequency as the sampling one. However, this is way too resource intensive even for high-end FPGAs and would greatly underutilize the maximum working frequency of the FPGA. The opposite extreme would be to have a single core working at 2.3 GHz, provided that this core could handle the memory throughput as well. This clock frequency is too high for FPGAs, so this approach was also discarded. The goal was to find a solution somewhere in between, which would result in a compromise between attainable frequency and available resources.

The approach taken was to use the highest possible working frequency while not breaching the resource limit. Even though the FPGA itself could operate on 800 MHz, the floating-point unit created a bottle neck which reduced the working clock frequency down to 200 MHz. This resulted in 12 processing units working in parallel, each having a filter with 4k taps. These processing units were named FIR Cores and the structure of one is show in Fig. 2.

The Core consists of two buffers, one for the samples and one for coefficients, and an accumulator which multiplies the two and accumulates the result. Each core receives an input sample from the previous core, except the first one which receives the sample directly from the PC. The FIR coefficients are transferred by the coefficient bus. Results from all 12 cores are added together and this forms the output of a single FIR Unit. There is a total of 72 FIR Units within the system.

III. Implementation

One of the most important questions when implementing this kind of system is how to split all the components across multiple dies. If the split is not done correctly, there is a high chance that the communication between the parts will cause a performance drop large enough for the system to not meet the timing restrictions.

Firstly, we introduce the term pblock. A pblock is an abstract container which holds primitives such as lookup tables, DSP slices, flip-flops, etc. and certain functionalities can be assigned to each pblock. Pblocks operating on the same die cause no issues. However, if two pblocks that are not on the same die have to communicate, they do so using the special channels called microvias which are miniature holes in the silicon area drilled by lasers. The path which the signal takes when travelling through the microvias is much longer compared to that between two blocks on the same die. This also has to be taken into consideration as they can greatly impact the timing constraints of the system, especially if high frequencies are involved.

The system layout is presented in Figure 3. Our FPGA at hand consists of three dies so there are 6 pblocks available. With the FIR filter being both the biggest and most crucial part of the system, it was decomposed first. Each die is divided into two independent pblocks. This was achieved using the Vivado tool from Xilinx using the concept of hierarchical design [10], [11].

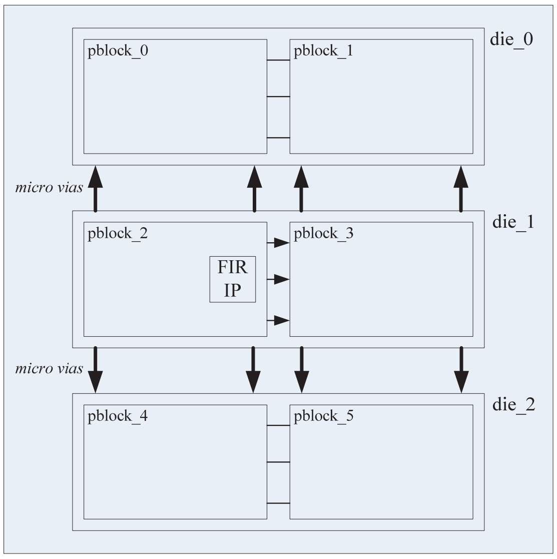

Fig. 3. System layout.

There are a total of 72 FIR filters, so each pblock contains 12 of them. The remaining parts of the system such as I2S blocks, I2C blocks, AXI interconnect etc. were less resource demanding so they were scattered across the pblocks as well.

The FIR IP supports the full AXI protocol with all channels (write address, write data, write response, read address and read data). Even though, inside the FIR IP itself AXI protocol could have been used, there was a need to unload the protocol in order to reduce the resource demand. The protocol was simplified by removing the ready signal from the communication handshake. This could be achieved due to the fact that both sides are synchronized in producing and consuming data. This gives us unidirectional bus channels which help implement timing relaxations easier. Buffering registers were added which introduced delay but kept the latency below the threshold. If unidirectional bus channels were not used, then in order to use the full AXI protocol we would have to include repeaters which would use up more valuable resources.

The ADC and DAC are similar chips with the majority of resolution specs being the same: 24 bits for data, 192 kHz sample rate, 114 dB dynamic range and using the Delta-Sigma architecture.

The software architecture is divided into three core parts: the application, the communication library and the kernel driver. The purpose of the application is to manage the pre and post processing as well as updating the coefficients. The library exposes an API from the driver to the application. The kernel driver handles the communication with the DMA such as initializing transfers of both samples and coefficients.

To achieve best performance we used scatter-gather DMA mode. Unlike traditional DMA transfers which need a setup for each transaction, scatter-gather enables the user to create a list of transactions. Once all the transactions are setup and arranged in a list, the transfer is initiated and the interrupt which informs that the transfer is done occurs only when all transactions are finished. This reduces the overhead generated from the traditional approach which would make an interrupt after each transfer, thus reducing the overall delay time and increasing performance.

Xilinx offers an IP called XDMA [12] which has this mode supported. Another added benefit which XDMA brings is that scatter-gather lists are located in the DDR-RAM on the PC which greatly offloads the FPGA. This is achieved because the transfers from the CPU cache are larger chunks of 512 bits. An alternative would be to have these data be stored on the FPGA memory which could only be transferred in chunks of 64 bit due to the width of the PCIe bus and is also the maximum uncached data access size from the CPU.

The architecture promotes a pipeline structure. The data is streaming from the ADC through the PC to the FPGA, then back to the PC and finally through the DAC into the real world. When a single block of data was processed and sent to the next stage, the block which did the processing can read new data and start processing it. In order to achieve this, between several stages the so called ping-pong buffers were added. Even though each IP has its own working clock, the I2S IP dictates the tempo of the entire process. Every IP has its dedicated FIFO buffers which if full or empty would block the DMA. That being said, the application would have to wait for all the data to be read from the buffers or written to them before it can start a new transaction. What these ping-pong buffers enable is to have reading and writing done in parallel. The application writes to the ping buffer while the DMA is transferring the pong buffer. Once the transfer is complete, a swap occurs and the application starts writing to the pong buffer. Meanwhile, the IPs can read from the ping buffer. These ping-pong buffers are effectively handled with the scatter-gather.

There are also specific hardware supported ping-pong buffers which are used for the coefficients. These buffers are available to the user who can update them in real-time. The algorithm is the same as with the buffers already mentioned. These updates represent the training of the system which tries to achieve the desired impulse response.

IV. Verification and results

As far as verification and testing goes, there were several issues which needed to be addressed. The entire design was synthesized as an AXI IP and as such could theoretically be simulated as one (illustrated in Fig. 4).

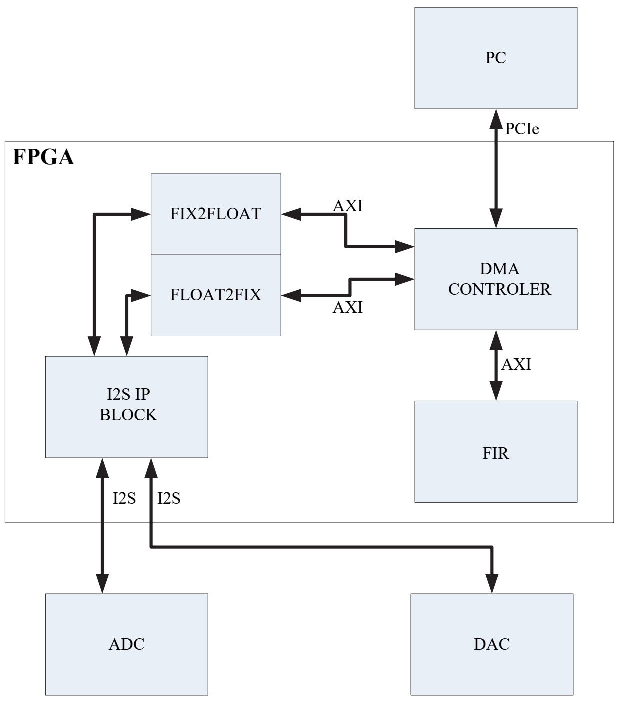

Fig. 4. System protocols.

The main reason the simulation is important is to remove any bugs which can occur in the system and with the elements linked to our system. Another key feature of having a simulation is the unparalleled visibility per each cycle which is impossible to recreate in a real world test scenario. However, due to the complexity of the design it was not possible to simulate the full design with the available workstations. Even if the available resources were not insufficient, the simulation would simply last too long to be useful for debugging. A single real time second in the simulation could take hours or even days, which would easily put the project to a halt.

Instead, a shrunk version of the system was simulated for debugging purposes and to check the structural integrity, primarily to verify the Advanced Extensible Interface (AXI) protocol was working as intended. This version used a single FIR Unit with a shortened FIR Core. Prior to testing, we needed to make sure that there were no errors in the design because the synthesis process lasted over 8 hours which made design changes extremely demanding and in final instance quite expensive. Once the simulations showed the system was error free, the next step was the real world test on the development board with both the shrunken and full version of the design.

A gold reference model for the shrunken version of the design was generated using a script written in C. The inputs used for this gold model were fed into the system and the results were compared to those generated previously with an exact match. Verifying the full version design brought new challenges. Even generating a meaningful reference model took too long – about a minute for every one second of the actual signal. For this purpose, simple input signals were chosen such as the delta impulse and a saw tooth whose response could be analytically calculated. Finally, this simple input signal was fed to the system which outputted the same results as the analytical model predicted with a perfect match.

The results presented in this chapter show the amount of FPGA resources required to meet all the restrictions placed on the application. The number of logic elements, DSP slices, memory elements as well as the entire % of resource usage is given in Table 1 shown below. Lookup Tables (LUT) represent units of combinatorial logic, Flip-flops are used for individual registers, where as Block RAM (BRAM) and Ultra RAM (URAM) are used for larger quantities of memory. As shown in Fig. 2. URAM was used for the samples and coefficients, while BRAM had a more supportive role in buffers which were used to help with the communication, mainly with the AXI protocol. Lastly, the number of DSP slices is used for support regarding floating-point operations for fast multiply and accumulate operations.

Table 1: UltraScale+ resource utilization.

Resource	Utilized	Available	Utilization [%]
LUT	973358	1182240	67.11
FF	1182432	2364480	50.01
BRAM	540	2160	25.00
URAM	936	960	97.50
DSP	3600	6840	52.63

The values in the table indicate that the memory was in fact one of, if not the biggest obstacle which needed to be overcome. The % of URAM usage nearly reached 100%, while the % of LUTs and DSPs which indicate processing power was somewhere in the middle.

One other thing worth mentioning is the power consumption. The total power needed for the system is just below 50 W. This is several times less than that needed for a strong GPU which could provide the sufficient throughput and processing power (disregarding the latency issues) which usually operates on a level of 150-200 W with spikes going over these values. This is another benefit of using an FPGA, especially for prolonged usage over several years.

Even though most of the issues during this project were handled, some remain unresolved and unexplained, the main one being efficiency of the PCIe bus. It was observed that by increasing the number of lanes to 4 times, 8 or even 16 did not yield almost any increase in performance. Even then a higher speed was used (5 GT/s instead of 2 GT/s) the acceleration was basically immeasurable. Several different motherboards were tried but neither one gave significantly better results than the previous one. Caution is advised for future solutions which intend to increase the performance by spending more resources on the PCIe bus.

V. Conclusion

A high-performance, very long response time FIR filter implemented in a high-end FPGA is presented. This is an extended version of the same titled paper, presented earlier at the Telfor conference [13], which goes much deeper into the details of the implementation. The majority of design decisions were the direct result of customers specifications. The results exposed adequate resource utilization along with accomplishment of the strict criteria such as low latency and real-time. These performances could not have been achieved by traditional approaches due to high memory throughput demand alongside the before mentioned restrictions.

References

[1] P. Longa and A. Miri, “Area-Efficient FIR Filter Design on FPGAs using Distributed Arithmetic,” *2006 IEEE International Symposium on Signal Processing and Information Technology*, Vancouver, BC, 2006, pp. 248-252.

[2] J. B. Evans, “Efficient FIR filter architectures suitable for FPGA implementation,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, no. 7, pp. 490-493, July 1994.

[3] T. T. Do, H. Kropp, C. Reuter, and P. Pirsch, “A flexible implementation of high-performance FIR filters on Xilinx FPGAs,” *In International Workshop on Field Programmable Logic and Applications*, pp. 441-445, August 1998, Springer, Berlin, Heidelberg.

[4] J. Valls, M. M. Peiro, T. Sansaloni and E. Boemo, “A study about FPGA-based digital filters,” *1998 IEEE Workshop on Signal Processing Systems. SIPS 98. Design and Implementation (Cat. No.98TH8374)*, Cambridge, MA, USA, 1998, pp. 192-201.

[5] A. Ryou and J. Simon, “Active cancellation of acoustical resonances with an FPGA FIR filter,” *Review of Scientific Instruments*, vol. 88, no. 1, pp. 013101, 2017.

[6] S. Gannot and M. Moonen, “Subspace methods for multimicrophone speech dereverberation,” *EURASIP Journal on Advances in Signal Processing*, vol. 2003, no. 11, pp. 1074-1090, 2003.

[7] M. Palesi, S. Kumar and V. Catania, “Leveraging Partially Faulty Links Usage for Enhancing Yield and Performance in Networks-on-Chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 3, pp. 426-440, March 2010.

[8] VCU118 Evaluation Board https://www.xilinx.com/support/documentation/boards_and_kits/vcu118/ug1224-vcu118-eval-bd.pdf, loaded on 27th March 2019..

[9] UltraRAM: Breakthrough Embedded Memory Integration on UltraScale+ Devices https://www.xilinx.com/support/documentation/white_papers/wp477-ultraram.pdf, loaded on loaded on 27th March 2019..

[10] Vivado Design Suite User Guide, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_1/ug905-vivado-hierarchical-design.pdf, loaded on 27th March 2019.

[11] Vivado Design Suite Tutorial https://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_1/ug946-vivado-hierarchical-design-tutorial.pdf, loaded

[12] XDMA IP product guide https://www.xilinx.com/support/documentation/ip_documentation/xdma/v4_0/pg195-pcie-dma.pdf, loaded on 27th March 2019.

[13] S. Piietlovic, M. Subotić, V. Marinkovic and N. Pjevalica, “FIR Filter Implementation for High-Performance Application in a High-End FPGA,” *2018 26th Telecommunications Forum (TELFOR)*, Belgrade, 2018, pp. 1-4.