KIET IJCE KIET International Journal of Communications & Electronics
Volume. No. 6, Issue No. 1, Jan-June 2018, ISSN: 2320 – 8996

FPGA Implementation of Adaptive Filter

Neelesh Ranjan Srivastava[#1], Satya Prakash Singh[*2]
Department of Electronic & Communication Engineering, KIET Group of Institutions, Ghaziabad, UP, INDIA,
1nr.srivastava@kiet.edu,
2satya.singh@kiet.edu

Abstract

Filtering data in real-time requires dedicated hardware to meet demanding time requirements. If the statistics of the signal are not known, then adaptive filtering algorithms can be implemented to estimate the signals statistics iteratively. Modern Field Programmable Gate Arrays (FPGAs) include the resources needed to design efficient filtering structures. Furthermore, some manufacturers now include complete microprocessors within the FPGA fabric. This mix of hardware and embedded software on a single chip is ideal for fast filter structures with arithmetic intensive adaptive algorithms.
This paper aims to combine efficient filter structures with optimized code to create a system-on-chip (SoC) solution for various adaptive filtering problems. Several different adaptive algorithms have been coded in VHDL as well as in C for the Power PC 405-microprocessor. The designs are evaluated in terms of design time, filter throughput, hardware resources and power consumption.
.

Key Word: **-**Filters, Adaptive filters, Algorithm, FPGA, Arrays-Programming and System-on-chip.

INTRODUCTION

On systems that perform real-time processing of data, performance is often limited by the processing capability of the system. Therefore, evaluation of different architectures to determine the most efficient architecture is an important task. This topic discusses the purpose of the paper, and presents an overview and the direction.
The purpose of this paper is to explore the use of embedded System-on-Chip (SoC) solutions that modern Field Programmable Gate Arrays (FPGAs) offer. Specifically, it will investigate their use in efficiently implementing adaptive filtering applications. Different architectures for the filter will be compared. In addition, the Power PC embedded microprocessor will be employed for the various training algorithms. This will be compared to training algorithms implemented in the FPGA fabric only, to determine the optimal system architecture.
Digital Signal Processing (DSP) has revolutionized the manner in which we manipulate data. The DSP approach clearly has many advantages over traditional methods, and furthermore, the devices used are inherently reconfigurable, leading to many possibilities.
Modern computational power has given us the ability to process tremendous amounts of data in real-time. DSP is found in a wide variety of applications, such as: filtering, speech recognition, image enhancement, data compression, neural networks; as well as functions that are unpractical for analog implementation, such as linear-phase filters. Signals from the real world are naturally analog in form, and therefore must first be discretely sampled for a digital computer to understand and manipulate. The signals are discretely sampled and quantized, and the data is represented in binary format so that the noise margin is overcome. This makes DSP algorithms insensitive to thermal noise. Further, DSP algorithms are predictable and repeatable to the exact bits given the same inputs. This has the advantage of easy simulation and short design time. Additionally, if a prototype is shown to function correctly, then subsequent devices will also.
There are many advantages to hardware that can be reconfigured with different programming files. Dedicated hardware can provide the highest processing performance, but

is inflexible for changes. Reconfigurable hardware devices offer both the flexibility of computer software, and the ability to construct custom high performance computing circuits. The hardware can swap out configurations based on the task at hand, effectively multiplying the amount of physical hardware available. In space applications, it may be necessary to install new functionality into a system, which may have been unforeseen. For example, satellite applications need to be able to adjust to changing operation requirements. With a reconfigurable chip, functionality that was not predicted at the outset can be uploaded to the satellite when needed.

2 Programmable Logic Devices

Programmable logic is loosely defined as a device with configurable logic and flip-flops linked together with programmable interconnects. The first programmable device was the programmable array logic (PAL) developed by Monolithic Memories Inc. (MMI) in 1975. Considering that any Boolean function can be realized as a sum-of-products or equivalently as a product-of-sums by utilizing De Morgan's law, the PAL structure is rather intuitive. It generally consists of inputs with inverters leading into a series of AND gates whose outputs lead into a series of OR gates. This makes the products of any combination of the inputs and their complements available to the OR gates for the sum. A similar device, the programmable logic array (PLA), reverses the order of the AND and OR gates, which led to greater functionality. The reason is that the product terms can be shared across the OR gates at the outputs, effectively giving the chip more logic width. The structure in Figure 1 is a usual PLA before programming, with all possible connections are pre-wired typically by fuses. To implement a custom design, a programmer is used to blow the fuses with high current and break the unwanted connections.

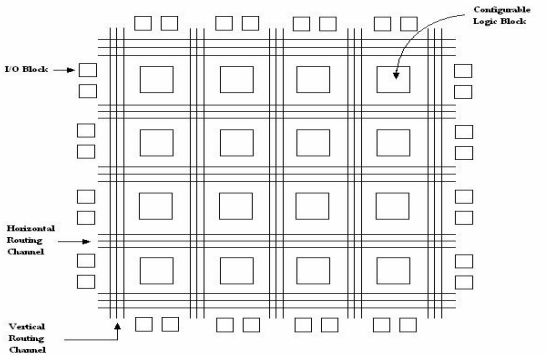

Fig. 1 PLA Structure

An improvement from PAL and PLAs came with the introduction of the complex programmable logic device (CPLD), which allows for more complex logic circuits. A CPLD consists of multiple PAL-like blocks connected by programmable interconnects. While PALs are programmed with a programmer, a CPLD is programmed in-system with the manufacturers' proprietary method or with a JTAG cable connected to a computer. CPLDs are well suited to complex, high-performance state machines. An alternative type of PLD developed more recently is the field programmable gate array (FPGA). Xilinx introduced the FPGA in 1984. These devices have a more flexible, gate-array-like structure with a hierarchical interconnect arrangement. The fundamental part of the FPGA is the look-up table (LUT), which acts as a function generator, or can alternatively be configured as ROM or RAM. They also include fast carry logic to adjacent cells making them suitable for arithmetic functions and further DSP applications.

3 FPGA Architecture

The majorities of FPGAs are SRAM-based and can therefore be programmed as easily as standard SRAM. The SRAM bits are coupled to configuration points in the FPGA (Figure 2 left) and controls whether or not a connection is made. This is normally accomplished by a passgate structure (Figure 2

right) that turns the connection on or off depending on the logic value (True or False) supplied by the SRAM. Because they are SRAM based, FPGAs are volatile. As such, they must be programmed each time power is applied. This is normally accomplished with another part of the circuit that reloads the configuration bit stream, such as a PROMThe configuration bit stream stored in the SRAM controls the connections made and also the data to be stored in the Look-up tables (LUTs). The LUTs areessentially small memories that can compute arbitrary logic functions. Each manufacturer has a distinct name for their basic block, but the fundamental unit is the LUT. Altera call theirs a Logic Element (LE) while Xilinx's FPGAs have configurable logic blocks (CLBs) organized in an array. The configurable logic blocks of an FPGA are generally placed in an island style arrangement. Each logic block in the array is connected to routing resources controlled by a interconnect switch matrix.

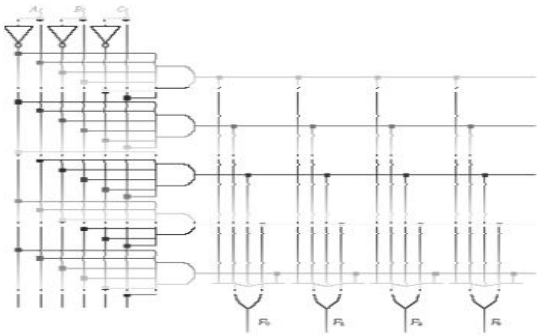

Fig. 3 FPGA cell Structure

With this layout, a very large range of connections can be made between resources. A downside to this flexible routing structure is that unlike the CPLD, signal paths are not fixed beforehand, which can lead to unpredictable timing. However, the tradeoff is the FPGA's increased logic complexity and flexibility. Each CLB in a Xilinx FPGA encompasses four logic slices, which in turn contain two 4-input function generators, carry logic, arithmetic logic gates, wide function multiplexers and two storage elements. The LUT is capable of implementing any arbitrary defined Boolean function of four inputs and the propagation delay is therefore constant regardless of the function. Each slice also contains flip-flops and a fast carry chain. The dedicated fast carry logic allows the FPGA to realize very fast arithmetic circuits.

Manually defining the routing connections in a programmable device may have been feasible with the early PALs but is nearly impossible considering the density of modern FPGAs. Configuring these programmable devices can be achieved in several ways, such as schematic design entry, the use of hardware description languages (HDLs), and the use of high-level language compilers. These methods are listed in increasing levels of abstraction, with schematic design entry being the lowest level.

4 Overview of Adaptive Filters

In practice, signals of interest often become contaminated by noise or other signals occupying the same band of frequency. When the signal of interest and the noise reside in separate frequency bands, conventional linear filters are able to extract the desired signal. However, when there is spectral overlap between the signal and noise, or the signal or interfering signal's statistics change with time, fixed coefficient filters are inappropriate Figure 4 shows an example of a wideband signal whose Fourier spectrum overlaps a narrowband interference signal.

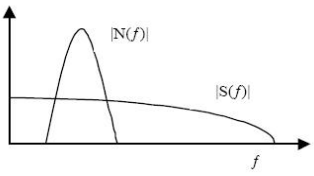

Fig. 4 Narrowband interference N (f) in a wideband signal S (f)

This situation can occur frequently when there are various modulation technologies operating in the same range of frequencies. In fact, in mobile radio systems co-channel interference is often the limiting factor rather than thermal or

other noise sources. It may also be the result of intentional signal jamming, a scenario that regularly arises in military operations when competing sides intentionally broadcast signals to disrupt their enemies' communications.
Furthermore, if the statistics of the noise are not known a priori, or change over time, the coefficients of the filter cannot be specified in advance. In these situations, adaptive algorithms are needed in order to continuously update the filter coefficients.

Adaptive Algorithms

There are numerous methods for the performing weight update of an adaptive filter. There is the Wiener filter, which is the optimum linear filter in the terms of mean squared error, and several algorithms that attempt to approximate it, such as the method of steepest descent. There is also least-mean-square algorithm, developed by Widrow and Hoff originally for use in artificial neural networks. Finally, there are other techniques such as the recursive-least-squares algorithm and the Kalman filter. The choice of algorithm is highly dependent on the signals of interest and the operating environment, as well as the convergence time required and computation power available.

Wiener Filters

The Wiener filter, so named after its inventor, was developed in 1949. It is the optimum linear filter in the sense that the output signal is as close to the desired signal as possible. Although not often implemented in practice due to computational complexity, the Wiener filter is studied as a frame of reference for the linear filtering of stochastic signals to which other algorithms can be compared.

Least-Mean-Square Algorithms

The least-mean-square (LMS) algorithmis similar to the method of steepest-descent in that it adapts the weights by iteratively approaching the MSE minimum. Widrow and Hoff invented this technique in 1960 for use in training neural networks. The key is that instead of calculating the gradient at every time step, the LMS algorithm uses a rough approximation to the gradient.

Recursive Least Squares Algorithm

The recursive-least-squares (RLS) algorithm is based on the well-known least squares method. The least-squares method is a mathematical procedure for finding the best fitting curve to a given set of data points. This is done by minimizing the sum of the squares of the offsets of the points from the curve. The RLS algorithm recursively solves the least squares problem.

FPGA Implementation

Field programmable gate arrays are ideally suited for the implementation of adaptive filters. However, there are several issues that need to be addressed. When performing software simulations of adaptive filters, calculations are normally carried out with floating point precision. Unfortunately, the resources required of an FPGA to perform floating point arithmetic is normally too large to be justified, and measures must be taken to account for this. Another concern is the filter tap itself. Numerous techniques have been devised to efficiently calculate the convolution operation when the filter's coefficients are fixed in advance. For an adaptive filter whose coefficients change over time, these methods will not work or need to be modified significantly.
First, the issues involved in transitioning to a fixed-point algorithm will be detailed. Next, the design of the filter tap will be considered. The reconfigurable filter tap is the most important issue for high performance adaptive filter architecture, and as such it will be discussed at length. Finally, the integration of the embedded

processor for the coefficient update will be discussed.

Embedded Microprocessors

The current trend in programmable logic is the inclusion of embedded DSP blocks and microprocessors. The Virtex-II Pro FPGA form Xilinx contains an embedded PowerPC 405 microprocessor, and numerous soft IP cores. To design for this environment the Embedded Development Kit must be used.

IBM Power PC 405

The IBM PowerPC 405 is a 32-bit RISC microprocessor embedded in Xilinx's Virtex-II Pro FPGA. The core occupies a small die area and consumes minimal power making it ideal for system-on-chip (SoC) embedded applications. It can run at a clock speed of over 400 MHz to produce over 600 Dhrystone MIPS. A memory management unit (MMU), a 64-entry unified Translation Look-aside Buffers (TLB), debug support, and watchdog timers enable an embedded operating system to function for no additional logic cost.

RESULTS

Several different implementations were tested, including hardware only designs as well as combined hardware/software embedded systems. This topic gives an overview of the hardware verification method, presents the implementation results, and compares them to the results from Matlab trials.

Full Precision Analysis

The application tested was adaptive noise cancellation, for reasons discussed in a previous topic. A sine wave is the desired signal, but is corrupted by a higher frequency sinusoid and random Gaussian noise with a signal to noise ratio of 5.865 dB. A direct form FIR filter of length 16 is used to filter the input signal. The adaptive filter is trained with the LMS algorithm with a learning rate $\mu = 0.05$. The filter is also trained with the RLS algorithm with the parameters $\delta=1$ and $\lambda=0.99$.

The floating-point precision results are presented. It appears that the filter trained with the LMS algorithm has learned the signals statistics and is filtering acceptable within 200 - 250 iterations. When trained with the RLS algorithm, the filters weights are near optimal within 50 training iterations, almost an order of magnitude faster, as expected.

Fixed Point Analysis

The above algorithms were converted so that all internal calculations would be done with a fixed-point number representation. This is necessary, as the embedded PowerPC has no floating-point unit (FPU), and FPGA's don't natively support floating-point either. Although a FPU could be designed in an FPGA, they are resource intensive, and therefore can feasibly only support sequential operations. Doing so however would fail to take full advantage of the FPGA's major strength, which is high parallelization.

The LMS and RLS algorithms were modified as detailed in Chapter 4, and a transposed representation of the LMS was also implemented. A scale of 256 with 16-bit precision was found to be suitable. The results of the fixed-point LMS algorithm were comparable to the full precision representation of the same algorithm. The RLS though, was considerably worse. The 16-bit fixed-point results are presented.

For a 16-bit fixed representation, the RLS algorithm displayed a significant degradation as compared to the algorithm represented with floating-point accuracy. It appears that the RLS algorithm is very sensitive to the internal precision used, and therefore its superiority over the LMS algorithm is diminished when a fixed representation is needed. In fact, considering the extra computation needed, the RLS algorithm is barely better, yet requires significantly more development time due to its complexity.

Because the error at the output of a transposed-form FIR filter is due to the

accumulation of past inputs and weight, it converges much differently than the direct form FIR. The transposed LMS algorithm takes much longer to converge and never converges close enough. However, it may be suitable when absolute data throughput is necessary. The output of a fixed-point transposed form LMS algorithm after approximately 2500 iterations.

Power Consumption

When choosing between different designs for embedded applications, the power consumed by the device is an important issue. The power consumed by the different architectures. Designs utilizing the
PowerPC microprocessor averaged 920 mW of power use, while designs using FPGA logic only averaged 757 mw. On average, the inclusion of the PowerPC uses 164 more mW than design without it.

CONCLUSIONS

A significant amount of design time was spent working out minor bugs and intricacies within the EDK software. In fact, the extra time spent debugging cancelled out the amount of time saved by coding in a high-level language. Hopefully, these bugs will be addressed in future software releases. In addition, reference designs provided by Avnet were used for the memory transfer to the SRAM through the PCI bus. Since the PCI bus and the PPC405 share the SRAM, a control bit and machine interrupt are used. This bit needed to be set manually during testing. It is unclear whether this is due to software or hardware errors. Until an appropriate fixed-point structure is found for the Recursive
Least-Squares algorithm, the Least Mean-Square algorithm was found to be the most efficient training algorithm for FPGA based adaptive filters. The issue of whether to train in hardware or software is based on bandwidth needed and power specifications, and is dependent on the complete system being designed. While the extra power consumed would make the PowerPC seem unattractive, as part of a larger embedded system this could be practical. If many processes can share the PowerPC then the extra power would be mitigated by the creation of extra hardware that it has avoided. Furthermore, an area greatly simplified by the inclusion of a microprocessor is memory transfers. These are sequential in nature and within the EDK there are many memory IP modules. With no microprocessor, a finite state machine for timing, as well as a memory interface is needed, and these will consume more power, although still less than the PowerPC. Lastly, the microprocessor can be used to easily swap out software training algorithms for application testing and evaluation. Embedded microprocessors within FPGA's are opening up many new possibilities for hardware engineers, although it requires a new design process. The future of embedded Systems-on-Chip design will involve more precisely determining the optimal hardware and software tradeoffs for the functionality needed.

REFERENCES

[1]K.A. Vinger, J. Torresen, "Implementing evolution of FIR-filters efficiently in an FPGA." Proceeding,. NASA/DoD Conference on Evolvable Hardware, 9-11 July 2003. Pages: 26 – 29.
[2] E. C. Ifeachor and B. W. Jervis, Digital Signal Procesing, A Practical Approach, Prentice Hall, 2002.
[3] Project Veripage, retrieved from: http://www.angelfire.com/ca/verilog/history.html.