RECONFIGURABLE FIR FILTER IN FPGA

Igor Dzukleski and Cesar Ortega-Sanchez

Electrical and Computer Engineering Department
Curtin University of Technology
GPO Box U1987
Perth, Western Australia 6102
E-mail: c.ortega@curtin.edu.au

ABSTRACT

Finite Impulse Response (FIR) filters are widely used in Digital Signal Processing (DSP) systems. The throughput of real-time, FIR filters is limited by the processing capability of its implementation. Software implementations are very flexible but the inherent sequential execution of programs makes them slow. The fastest FIR filters are those implemented in dedicated hardware, but these systems are usually fixed and modifications to the filter are not possible. This paper presents the design and implementation of a Field-Programmable Gate-Array (FPGA)-based, FIR filter. By exploiting the field-programmability of FPGAs it is possible to create different filters with fixed functionality, or even create adaptive filters whose parameters are adjusted in real-time by some intelligent algorithm.

1. INTRODUCTION

The Finite Impulse Response (FIR) filter is used in many Digital Signal Processing (DSP) systems to perform signal preconditioning, anti-aliasing, band selection, decimation/interpolation and low-pass filtering functions. In high-performance applications, digital filters need to operate in real-time, i.e. data have to be processed immediately after they have been acquired. The throughput of real-time FIR filters is often limited by the processing capability of the system. Providing a system with a high processing speed and flexibility is therefore a crucial factor [1].
The FIR-filter computes an output from a set of input-samples. These input samples are multiplied by a set of coefficients and then added together to produce the output, hence the filter behaviour is determined by the set of coefficients. A general FIR filter is described by the following equation:

$$Y(n)=k_0x(n)+k_1x(n-1)+k_2x(n-2)+\ldots+k_mx(n-m) \quad (1)$$

Where k_i is the i-th coefficient, x(n) is one sample of the input signal, and Y(n) is one point of the output signal. m is the number of filter coefficients called taps, and n is the number of input samples. Figure 1 shows the logical structure of an FIR Filter.

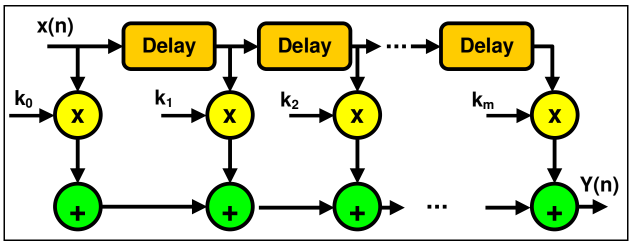


Figure 1. Logical Structure of an FIR filter.

High throughput is often a crucial requirement in real-time systems. Also, there is a need to have flexible systems that can be changed according to new specifications.

FIR-filters can be implemented in either hardware or software. Systems based on software are flexible, but due to the sequential nature of program-execution, they often suffer from insufficient processing capability. Figure 2 shows the software implementation of a 256-tap FIR filter.

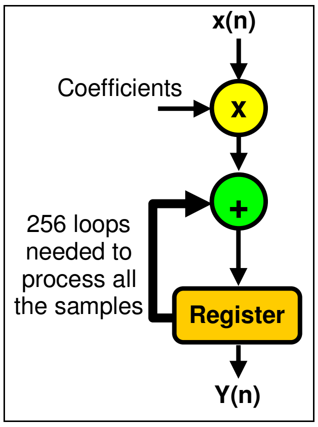


Figure 2. Sequential implementation of FIR filter

Dedicated hardware, on the other hand, can provide the highest processing performance, but is less flexible for changes [2]. Figure 3 illustrates the parallel implementation of a 256-tap FIR filter.

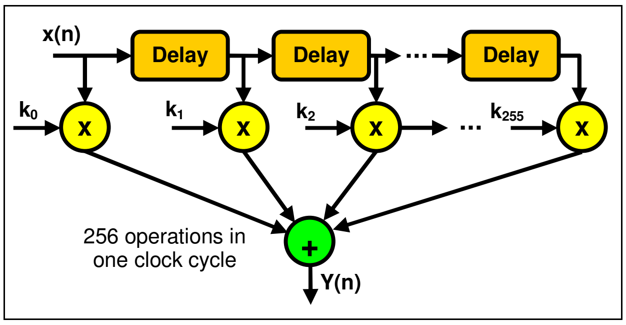


Figure 3. Parallel implementation of FIR filter.

The filters shown in figures 2 and 3 perform the same function. The implementation shown in Figure 3 will do 256 operations in one clock cycle, whereas the system in figure2 will require at least 256 clock cycles to calculate one point of the output. However, the improvement in performance obtained with the parallel implementation usually requires expensive dedicated hardware.

Reconfigurable hardware devices offer both the flexibility of computer software, and the ability to construct custom high performance computing circuits. Thus, in many cases they achieve a good compromise between software and hardware solutions. The flexible nature of these devices opens up a new range of circuits that exploits their reconfigurability. A large variety of real-world applications exist for such hardware [3].

Another advantage of FPGAs is the fact that they can accept last minute design modifications as well as future design iterations without making extensive software or hardware changes, saving both time and money. Also, designing the FPGA using a hardware description language such as VHDL, makes the design portable and easy to change and test. Apart from being design-flexible, FPGAs provide optimal device utilisation through conservation of board space and system power which are important advantages not available with many stand-alone DSP chips [1].

2. FPGA-BASED FIR FILTER

The objective of the work presented in this paper is to implement a reconfigurable FIR filter in an FPGA. First, a basic cell to implement one tap of the FIR filter is designed and then the basic cell is replicated to form the required FIR filter. The coefficients and some routing resources will be downloaded on-line, so that the characteristic equation of the filter can be altered on-the-fly.

2.1 The Basic Cell

The basic cell is programmed onto the FPGA and then used to implement reconfigurable FIR filters. Basic cells are interconnected depending on the type and length of the filter to be implemented. Note to be successful in cascading in directions other then linear, a Router will need to be used between the basic cells. This is perhaps future work. Figure 3 shows the basic cell and the configuration bits that are required to configure multiple cells into desired digital filters.

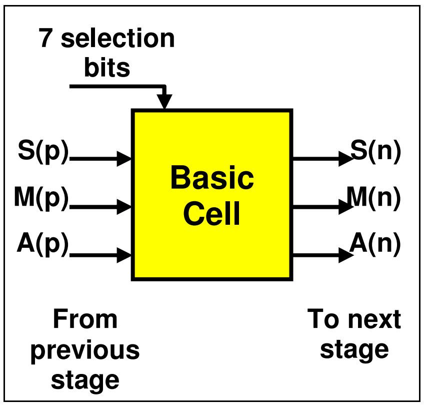


Figure 3. Block diagram of FIR basic cell.

The main components of the basic cell are Registers (delays), Multipliers and Adders. Using these basic components, a cell with the ability to be configured to use all or some of these resources has been implemented. Figure 4 shows the internal architecture of the basic cell.

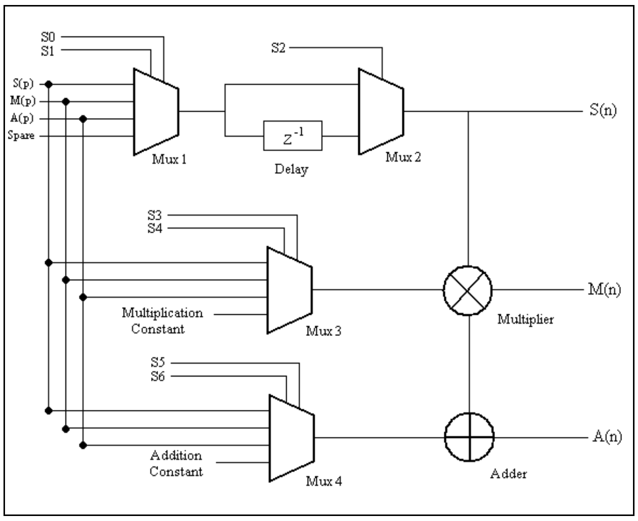


Figure 4. Structure of the FIR reconfigurable basic cell

As mentioned above several basic cells can be configured to form different filters, filters of different lengths, and filters with different coefficients. For example if a multiplication is not needed then the multiplication constant can be set to 1; or if no addition is required then the addition constant can be set to 0. In addition to the canonical implementation of FIR filters (one basic cell after another forming a line), other topologies can be explored. A close inspection of Figure 4 reveals that output S(n) can propagate either the S(p) input without delay, or the S(p) input after one clock cycle (canonical FIR filter implementation), or the result from the multiplication or addition in the previous cell. In a similar way, the inputs to the multiplier and adder in every basic cell can be selected. Note that one of the operands for the multiplier and adder can be a constant. Constants going into the multiplier are the coefficients of the FIR filter.

2.2 Reconfigurable FIR filter

The design of digital circuits using FPGAs requires the use of sophisticated software tools. In this work, Xilinx's ISE 7.1 has been used to write, synthesise and simulate VHDL descriptions of the basic cell and FIR filters. The verified design of the basic cell can be stored in a library for use in the implementation of different FIR filters.

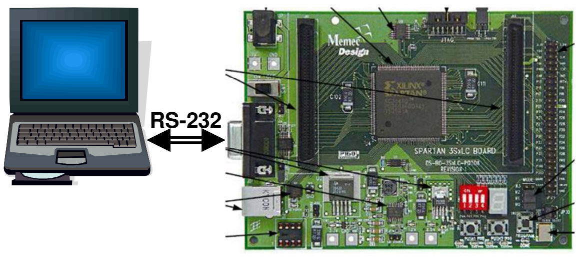
The actual design of an FIR filter requires determining the number of taps and the coefficients associated to each tap. This process can be done using another program, for example MatLab. Once the FIR filter has been designed and tested by simulation, implementing it in an FPGA only requires connecting as many basic cells as needed to meet the specification.

The implementation of FIR filters using the basic cell is simple because it requires adjacent cells to be joined in a linear fashion, connecting the outputs of one cell to the inputs of the following cell. This topology is translated by ISE software into a configuration file that is downloaded to the FPGA that will physically implement the filter.

Once the topology for the FIR filter has been downloaded to the FPGA, two reconfigurations can be applied. The first is to set the value of the coefficients in each tap. By changing the coefficients, the FIR filter will have a different characteristic equation. The other possible change is the topology of the filter. This is done by changing the configuration bits that control the multiplexers inside each basic cell. These bits will determine, among other things, the number of taps in a filter. The number of taps in a filter determines the quality of the results. Long filters have better characteristics than shorter ones, but they require more resources to be implemented. Hence, the maximum size for reconfigurable FIR filters will be determined by the FPGA's available resources.

Reconfiguration of the FPGA-based FIR filter must be accomplished without having to reset the FPGA and load all the basic cells over again for every new configuration. Instead, it must be done seamlessly just by overriding the previous configuration bits and coefficients.

In this work, configuration bits and coefficients are downloaded to the FPGA as a binary file using a conventional PC RS-232 serial port and the Hyper Terminal program. Figure 5 shows the setting used in this project to implement reconfigurable FIR filters in an FPGA.



Programs in the PC
ISE: Design of basic cell and reconfigurable FIR filter.
MatLab: Design of FIR filter.
Hyper Terminal: Download coefficients and configuration bits.

Figure 5. Setting used to test the system

The procedure to implement a reconfigurable FIR filter is as follows:

1. Using Xilinx Load the array of basic cells on the FPGA.
2. Once the Basic Cells are on the FPGA use Hyper Terminal to load the configurations of these cells and the data to be filtered.
3. Load a new configuration by loading a new file.

It is important to note that the configuration files will need to be monitored closely so that the correct bits are used at the correct places on the FPGA, should one bit be out of place then the whole configuration is invalid.

3. APPLICATIONS

Reconfigurable digital filters can be used in adaptive or learning systems.

In adaptive filters new coefficients have to be constantly generated as a function of the noise presented to the system. For example, filters change their characteristic response to filter-out intermittent noise.

Reconfigurable digital filters can be combined with an evolutionary strategy, such as genetic algorithms, to evolve rather than mathematically calculate the coefficients [2]. It is possible to conceive a system where the required frequency response of the filter is well characterised by means of, for example, the frequency spectrum of the desired output. In this case, the Fast Fourier Transform (FFT), of the filter's output could be used to evaluate the quality of the filter. If a genetic algorithm were to be used to evolve the required filter, then the following procedure would need to be followed:

1. Generate a population of filters. Every individual of the population would be a set of parameters that define the response of the filter.
2. Evaluate the fitness of every individual in the population by loading the parameters into the filter and comparing its output against the FFT of the desired response.
3. Select the fitter individuals to generate the next generation of coefficient sets.
4. Add variation into the new population by randomly changing the value of some parameters (mutation).
5. Repeat steps 2 to 4 until the response of the filter is close enough to the desired response.

Figure 6 shows a block diagram of the evolvable digital filter.

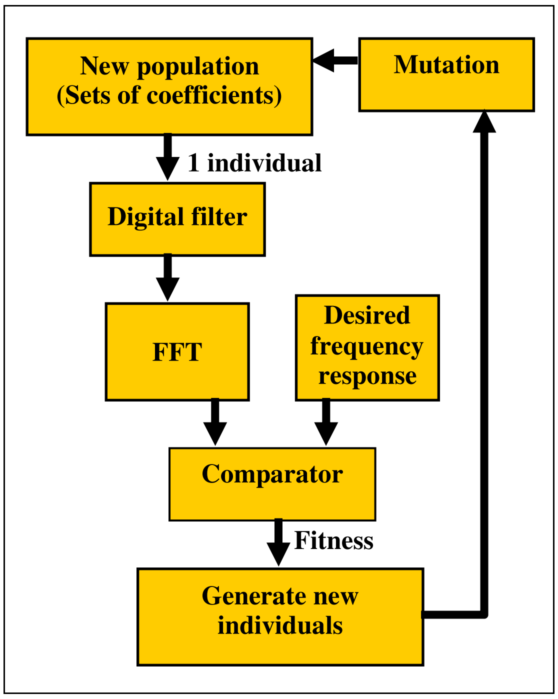


Figure 6. Evolution of digital filter using genetic algorithms

If all the blocks shown in figure 6 could be implemented in the same FPGA, then a very fast and reliable way of creating digital, adaptable filters would be attained.

CONCLUSIONS AND FUTURE WORK

Reconfigurable hardware devices offer both the flexibility of software, and the ability to construct custom, high-performance, computing circuits. This flexibility allows FIR-filters implemented in FPGAs to be used in real-time, high-throughput applications.

New generations of FPGAs contain vast amounts of resources including not only logic, but also integrated peripherals such as memory blocks, hardware multipliers, microcontrollers, and high-speed input output ports. These devices provide the platform needed to implement complete DSP, embedded solutions, without the need to design an ASIC.

By combining reconfigurable, FPGA-based, DSP components with evolutionary techniques, interesting intelligent, evolvable systems could be created. These devices could adapt to changes in their environment and learn about the world as they explore it [6, 7]. Interesting times lay ahead. Indeed.

REFERENCES

[1] Poonawala M. *FIR Filter Design using FPGA*, MSc Thesis, School of Electronic and Electrical Engineering, University of Leeds, UK, 2003.

[2] Vinger K. and Torresen J. "Implementing Evolution of FIR filters efficiently in an FPGA", Proc. of 2003 NASA/DoD Conference on Evolvable Hardware (EH-2003), July, 2003, Chicago, Illinois, USA.

[3] Villasenor J. and Mangione-Smith W. "Configurable Computing", Scientific American, June 1997.

[4] Wang Y. "Implementation of digital filter by using FPGA", BEng Thesis, Electrical and Computer Engineering Department, Curtin University of Technology, 2005.

[5] Stettbacher J. "DSP Functions on FPGA's", MATLAB Digest - January 2004, http://www.mathworks.com/company/newsletters/digest/jan04/dsp_fpga.html.

[6] Bentley P. *Digital Biology*, Simon and Schuster, New York, 2001.

[7] Sipper M. *Machine Nature*, McGraw-Hill, 2002.