Filter Design Methods for FPGAs

An optimal FPGA filter design methodology should support the rapid creation of initial results using pre-configured IP blocks, as well as an easy migration path replacing these blocks with application-tailored, user-defined architectures.

AccelChip, Inc.
1900 McCarthy Blvd.
Suite 204
Milpitas, CA 95035
(408) 943-0700

www.accelchip.com

Overview

FPGAs have been praised for their ability to implement filters since the introduction of DSP savvy architectures such as the Xilinx Virtex II™ and Altera Stratix II™. At the heart of the filter algorithm is the multiply-accumulate operation, which can be efficiently realized using dedicated DSP resources on these devices. More than 500 dedicated multiply-accumulate blocks are now available, making them exceptionally well suited for high-performance, high-order filtering applications that benefit from a parallel, non-resource shared hardware architecture.

The process of designing a filter in an FPGA involves two distinct steps: designing the filter response and designing the filter implementation. Well over 100 different filter design tools, including toolbox functions available in MATLAB®, are available for designing filter response and generating coefficient tables, each with varying levels of sophistication. Graphical filter design tools provide easy-to-use selections for specifying passband, filter order, and design methods, as well as provide plots of the response of the filter to various standard forms of inputs. One of the most popular is the FDAtool from The MathWorks shown below, which generates a behavioral MATLAB model and coefficient tables.

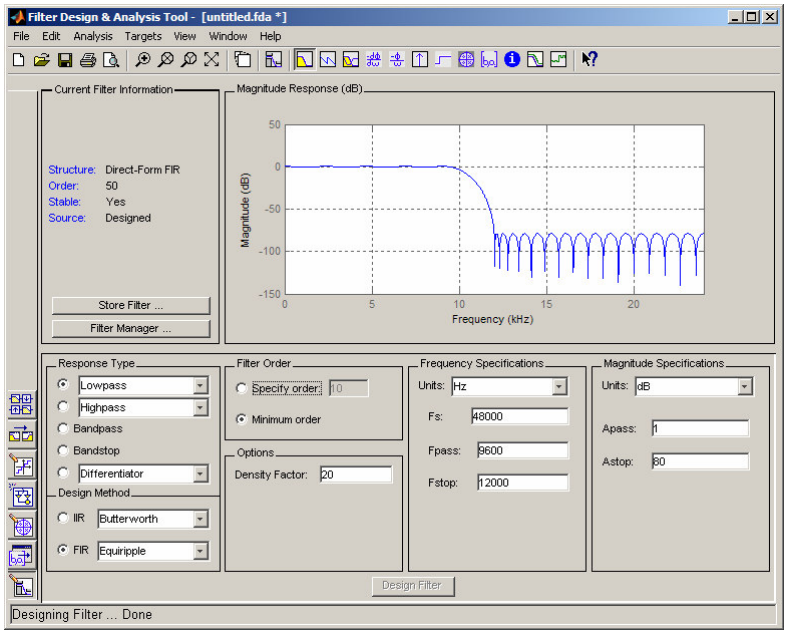

Figure 1 – MathWorks Filter Design & Analysis Tool

The magnitude and phase plots provide an estimate of how the filter will perform; however, to determine the true response, the filter must be simulated in a system model using either calculated or recorded input data. The creation and analysis of representative data can be a complex task with the majority of designers preferring to work initially in the MATLAB environment. MATLAB combines the ease of working in a high-level, mathematical language with an extensive set of pre-defined functions to assist in the creation and analysis of filter data.

Once a correct filter response has been determined and a coefficient table has been generated, the second step is to design the hardware architecture. The hardware designer must choose between area, performance, quantization, architecture, and response. Implementation options include low-level hardware descriptions, such as hand-coded RTL or low-level Simulink® schematics; pre-configured IP blocks from Xilinx, Altera, AccelChip or The MathWorks; and the use of a high-level synthesis tool such as AccelChip® DSP Synthesis. This paper will explore the advantages and disadvantages of each of these hardware design approaches for specific applications.

Using MATLAB for Filter Design

MATLAB with the Filter Design Toolbox provides the industry's most comprehensive environment for designing and verifying a filter algorithm. The FDATool, which is part of this toolbox, provides a quick and easy way to generate coefficients for an extensive range of filter styles. Also included is extensive support for modeling adaptive filters including Least Mean Square (LMS), Recursive Mean Square (RMS), Affine Projection (AP), and FIR Adaptive Filters (FD), along with a complete set of discrete IIR and FIR filter objects. Each filter type comes complete with examples and documentation, making quick work of the learning curve.

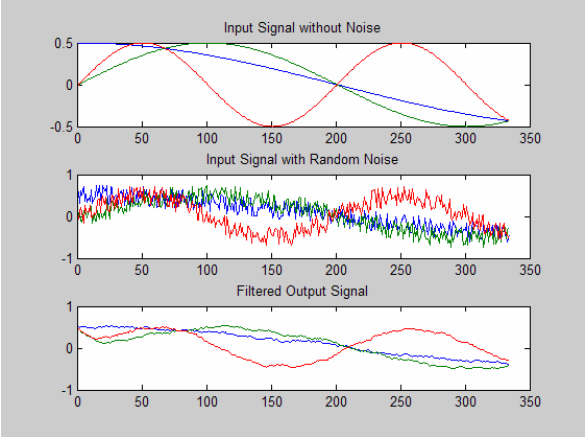

Figure 2 – MATLAB Plot of Adaptive FIR Filter using Direct Matrix Inversion

Often the task of analyzing a filter in a representative system model is more complex then designing the filter itself. For this, MATLAB is eminently suitable. Wide-ranging functionality is available to generate input stimulus to the filter, including impulse or step response, sinusoidal and random number generators, plus an extensive set of application-specific noise generator functions for testing communication, image processing, audio, and general purpose filters. To assist in filter analysis, MATLAB provides more than 30 separate "Methods for Analyzing Filters" functions.

The MATLAB environment coupled with the Filter Design Toolbox offers the most powerful, flexible, and extensive filter design and analysis environment in the industry. The challenge

FPGA filter designers will face when using this rich environment will be insuring that the hardware implementation matches the MATLAB simulations. This is discussed later.

Selecting a Hardware Architecture

Once the filter response has been finalized, the next step is to design the hardware. Deciding on an optimal architecture for a particular application involves tradeoffs between area, performance, and response. The abundance of dedicated MAC (Multiply-Accumulate) blocks in the Xilinx Virtex 4 and Altera Stratix II devices present the option for implementing either an area efficient "serial" or a high-performance "parallel" FIR filter or something in between. Serial architectures share a single MAC resource, making it very area efficient but at the expense of performance, because one clock-cycle is required for each tap delay register. This architecture is an excellent choice for area sensitive, low-performance applications.

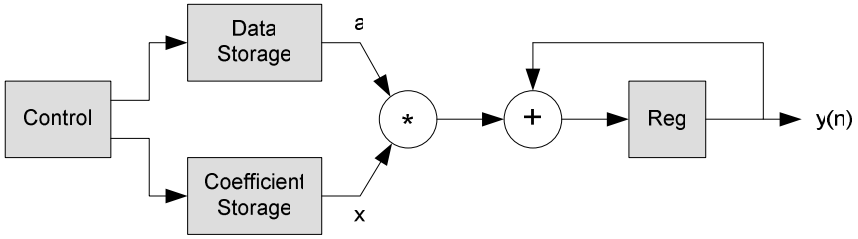

Figure 3 – Resource Shared Direct Form FIR Filter Architecture

Although commonly implemented on FPGAs, a serial MAC FIR filter does not exploit the large number of dedicated hardware MAC units available on the newer devices. High-performance, high-order filtering applications, that are able to exploit dedicated multiplier or DSP blocks, often turn to FPGAs for a solution. By replicating the multiply-adder logic once for each tap delay register, the entire filtering calculation can be performed in a single clock cycle. With more than 500 DSP blocks available, even very large order filters can sustain input sampling rates over 400 MHz! This type of performance far exceeds even the fastest DSP processors by orders of magnitude. Figure 4 shows the block diagram for an "unrolled" implementation of a direct form FIR filter.

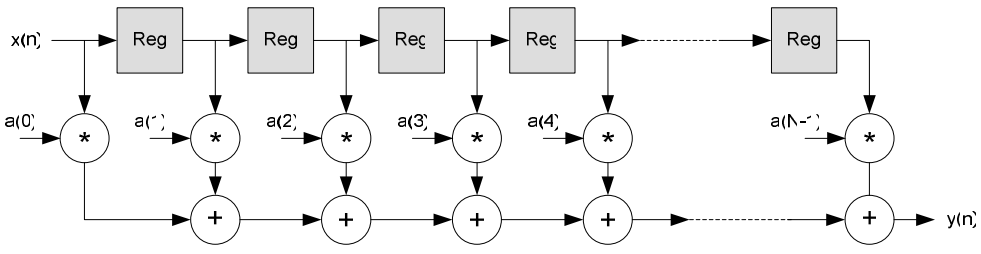

Figure 4 – 'Unrolled" Direct Form FIR Filter Architecture

When targeting an FPGA device with dedicated DSP blocks capable of supporting cascaded "multiply-add" operations such as the Xilinx Virtex 4 or Altera Stratix II, highest performance is achieved using a "transposed" architecture. Utilizing the same resources as a "direct" form FIR filter, data samples are applied in parallel to all tap multipliers through pipeline registers. The products are then applied to a cascaded chain of registered adders, combining the effect of accumulators and registers.

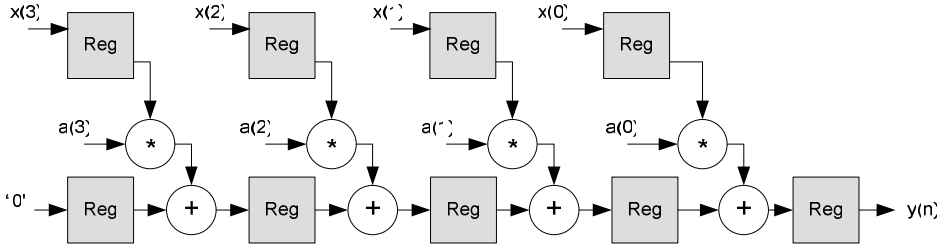

Figure 5 – "Transposed" Direct Form FIR Filter Architecture

Both direct and transposed form FIR filters have trade-offs and limitations. It is up to the designer to choose the style that is most appropriate to the application. In general, a smaller filter benefits from the direct approach, while the larger filter benefits from the transposed form. Devices that include dedicated DSP blocks that support a "multiply-add" and "multiply-accumulate" operation will benefit, in almost every case, by using the direct form for area-sensitive designs and the transposed form for high-performance applications.

A fourth FIR filter architecture worth considering for FIR implementation in FPGAs is called "distributed arithmetic" (DA). A simplified view of a DA FIR is shown in Figure 6. In its most basic form, DA-based computations are bit-serial in nature – serial distributed arithmetic (SDA) FIR. These computations can be parallelized using multiple sets of LUTs to achieve concurrency. This architecture is referred to as parallel distributed arithmetic (PDA).

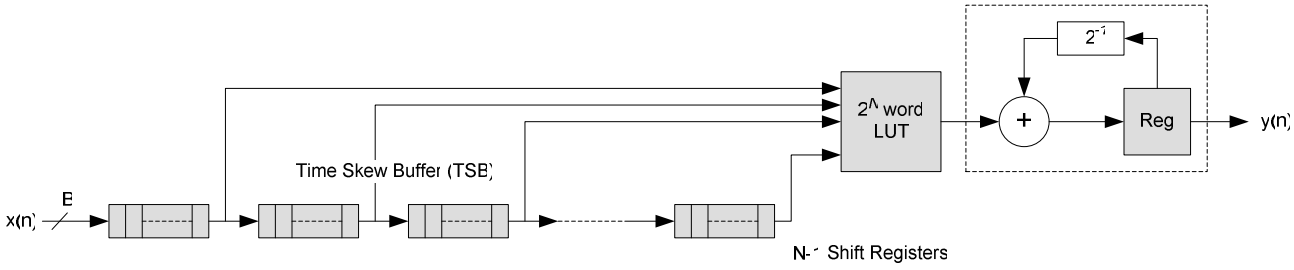

Figure 6 – Serial Distributed Arithmetic FIR Filter Architecture

The advantage of the distributed arithmetic approach is its efficiency. The basic operations required are a sequence of table look-ups, additions, subtractions, and shifts of the input data sequence. These functions map efficiently into LUT-based FPGAs, such as Xilinx Virtex II,

Virtex 4™ and Altera Stratix™ and Stratix II. Distributed arithmetic architectures are an excellent choice for high-performance filter applications targeting FPGA devices that do not include dedicated multipliers or DSP48 blocks or when these resources are not available. A unique characteristic of the DA architecture is that the sample rate is decoupled from the filter length, making the DA architecture appealing for high-order filters. The trade off introduced here is one of silicon area (FPGA slices) for performance. As the filter length is increased in a DA FIR filter, more logic resources are consumed, but throughput and performance are maintained. The table below summarizes the benefits of each architecture.

Filter Architecture	Pros	Cons
Direct	• Area efficient when implemented using shared multiply-accumulate operations • Optimal architecture for devices that offer a dedicated synchronous multiply block such as the Xilinx Virtex-II and Virtex-II Pro	• Performance decreases on large order filters • Architecture does not allow for maximum efficiency of DSP blocks in the Virtex-4 architecture
Transposed	• High-performance when mapped to DSP blocks that support multiply-add and multiply-accumulate operations such as the Xilinx Virtex-4 and the Altera Stratix-II	• Not as area efficient as the direct form for small order filters
Distributed Arithmetic	• Filter sample rate does not decrease with an increase in filter order • Offers highest performance in FPGAs when dedicated DSP or synchronous multiplier blocks are not available	• Performance diminishes with increase in bit-widths • Consumes more area than direct or transposed forms

Although this discussion on filter hardware architectures has been limited to constant coefficient, single-rate, single-channel FIR filters, the concepts will apply for programmable coefficient, multi-rate, and multi-channel variations. The adaptive filters referenced in the overview, however, have distinct architectural characteristics that depart significantly from FIR filters and are beyond the scope of this paper.

FIR Filter Implementation – IP Generators

When implementing the actual filter on an FPGA, the designer is presented with a fundamental choice of whether to use an IP core or design a custom implementation. Both options have merits and limitations. FIR filter IP cores are readily available from multiple sources with the most common forms being technology-specific netlists, synthesizable RTL, and synthesizable MATLAB. All these generators can construct a filter using a pre-defined coefficient table.

Xilinx and Altera both offer high-performance but device-specific filter IP in the form of technology mapped netlists. This form of IP is typically the most cost effective and offers the best results but provides the fewest options. Often each core is hand-crafted to leverage device-specific hardware resources, such as Xilinx Block RAM, and includes placement information which helps maintain performance as filter size grows. The format of the IP, however, precludes user modification of the source or retargeting to a different device.

The MathWorks offers device-independent filter IP; however, they don't provide the ability to exploit the high-performing resources of the FPGA. This form of IP tends to be a bit more expensive but offers the user the additional options of retargetability and modification of the RTL source. The quality of the results will vary with the quality of the RTL model and the RTL synthesis tool used for implementation. In general, however, some area and performance is sacrificed to obtain general purpose, retargetable RTL models.

AccelChip offers both high-performance and device-independent IP available in AccelWare® IP Toolkits. AccelWare IP cores are provided as synthesizable MATLAB models that offer several advantages. First, the abstraction level of the IP allows for the greatest range of options. Describing a filter in MATLAB is much easier than creating a technology-specific placed netlist or RTL. This allows AccelChip to build a wider range of hardware architecture options into the model, providing the user the greatest number of choices. Secondly, the architecture of the final hardware can be further modified during the DSP synthesis process with AccelChip through the use of "directives." This provides an efficient means of leveraging the architectural features of the specific FPGA device, such as dedicated DSP blocks, RAMs, ROMs, and shift registers, to achieve high quality of results in the target device. The table below summarizes the synthesis directives available in AccelChip.

AccelChip Synthesis Directive	Effect on Results
Rolling/unrolling of For loops	Improves sustainable data throughput rate by reducing the number of cycles for processing per input sample
Expansion of vector and matrix additions and multiplications	Improves sustainable data throughput by reducing the number of cycles for processing per input sample
RAM/ROM memory mapping of 1D and 2D arrays	Improves FPGA utilization by mapping 1D and 2D arrays into dedicated FPGA RAM resources
Pipeline insertion	Improves sustainable data throughput rate by improving clock frequency performance
Shift register mapping.	Improves FPGA utilization by mapping data buffers to shift register logic

IP generators provide an excellent choice when time-to-market or ease-of-use demands prevails. The general purpose nature of these programmable IP generators, however, seldom yields the optimal hardware for a specific application. For this, a user-defined architecture should be considered.

FIR Filter Implementation – User Defined

FIR filter implementations with aggressive area or performance targets may require the use of a hand-crafted implementation over a pre-defined IP block. Doing so allows the designer to exploit application-specific characteristics, such as unique impulse response, coefficient symmetry, ones in the coefficient table, negative coefficients, coefficient length, or knowledge about the dynamic range of the input data to the filter. Most filter IP generators include some simple coefficient symmetry optimization but do not account for every possible situation.

User-defined implementations have traditionally been achieved using hand-coded RTL. This approach offers the designer a high degree of control over the implementation process but is decoupled from the MATLAB simulation environment and can be time consuming. A typical 32 tap FIR filter Verilog model can range between 300 – 500 lines of hand-created VHDL or Verilog code and will often require the instantiation of FPGA-specific RAMs or DSP blocks to achieve optimal results.

The quantization must be determined for each signal and register in the design, which typically requires a re-write of the original floating-point model in language that supports fixed-point modeling and filter response analysis. Once complete, the final RTL code must be verified back to the original filter model to insure functional correctness. A recent survey conducted by AccelChip found that designers were evenly split between these major DSP design tasks when asked to identify their most significant bottlenecks.

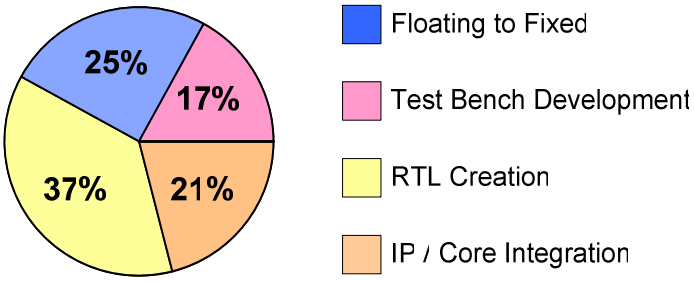

Figure 7 – Bottlenecks in the DSP Design Process

Adopting a high-level synthesis flow based on MATLAB provides designers the control over implementation necessary to exploit the unique symmetry and data conditions of the filter application while maintaining a high degree of abstraction. Shown below is a synthesizable model for a direct form FIR filter in its most basic form:

```
function outdatabuf=fir(indatabuf)

coeff = load('coefficients.txt');

persistent tap_delay;
if isempty(tap_delay)
  tap_delay = zeros(1,length(coeff));
end
```

```
outdatabuf = tap_delay * coeff(end:-1:1)';
tap_delay = [tap_delay(2:length(coeff)) indatabuf];
```

AccelChip synthesis directives can be used to map the coefficients to on-chip RAM, unroll the vector multiply of the tap_delay and coefficients to improve the sample rate, and to map the tap_delay line into on-chip shift registers.

A common architectural variation is to exploit an even number of symmetric filter coefficients, which can reduce the hardware resources significantly. The example below shows the MATLAB for this variation:

```
function outdatabuf=fir(indatabuf)
coeff = load('coefficients.txt');

persistent tap_delay;
if isempty(tap_delay)
  tap_delay = zeros(1,length(coeff));
end

% sum of samples
sample_sum = tap_delay(1:length(coeff)/2) + tap_delay(end:-
1:(length(coeff)/2)+1);

% array multiply using only half of coefficient values
outdatabuf = sample_sum * coeff((length(coeff)/2):-1:1);
tap_delay = [tap_delay(2:length(coeff)) indatabuf];
```

When implemented in hardware, the architecture that exploits coefficient symmetry delivers twice the performance with 30% fewer gates for a fully parallel architecture.

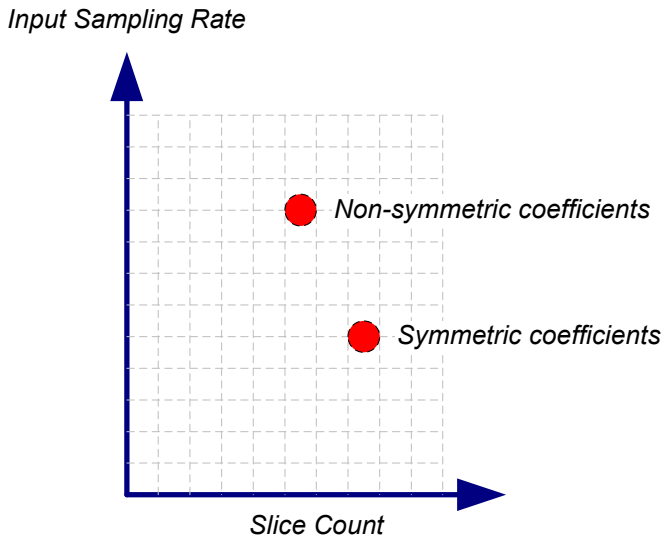

Figure 8 – Symmetric vs. Non-Symmetric Coefficients

Another common variation is a filter with complex inputs “I” and “Q” data typical of a QAM system. Again, this modification can be easily described in a synthesizable MATLAB model as shown below:

```
function [Ioutdata,Qoutdata]=IQ_filter(Iindata,Qindata)
coeff = load('coefficients.txt');

persistent Itap_delay Qtap_delay
if isempty(Itap_delay)
  Itap_delay = zeros(1,length(coeff)); % I data
  Qtap_delay = zeros(1,length(coeff)); % Q data
end

% shows extending simple filter to handle 'complex' I/Q input with real
coefficients
Ioutdata = Itap_delay * coeff(end:-1:1);
Qoutdata = Qtap_delay * coeff(end:-1:1);
Itap_delay = [Itap_delay(2:length(coeff)) Iindata];
Qtap_delay = [Qtap_delay(2:length(coeff)) Qindata];
```

These simple examples highlight the efficiency of MATLAB for describing user-defined filter architectures and the ease with which they can be taken to hardware using AccelChip DSP Synthesis. The hardware architecture can be further refined, from within AccelChip, to employ block RAMs, shift registers, and improve sample rates without modification to the MATLAB source by using synthesis directives.

Summary

The use of FPGAs for filter implementation will continue to grow as more design teams learn to exploit the unique architectural advantages these devices have to offer for high-performance filter applications. An optimal FPGA filter design methodology should support the rapid creation of initial results using pre-configured IP blocks as well as an easy migration path replacing these blocks with application-tailored, user-defined architectures. MATLAB provides the definitive filter modeling environment, and, together with AccelChip, makes possible a design flow that is familiar for algorithm developers used to DSP processors while offering the functionality required by hardware developers driving performance.