Configurable Logic for Digital Signal Processing

April 28,1999

Chris Dick, Bob Turney
Xilinx Inc.
2100 Logic Drive
San Jose
CA 95124

Ali M. Reza
Dept. Electrical Engineering and Computer Science
University of Wisconsin
Milwaukee

INTRODUCTION

The software programmable digital signal processor (DSP) has been the cornerstone of commercial configurable signal processing hardware for approximately 16 years. In comparatively recent times, approximately the last 5 years, a new option has been available for constructing programmable high-performance arithmetic datapaths: field programmable gate arrays (FPGAs). Recently, FPGA technology has undergone remarkable advances in density, performance and power. These advances have opened new dimensions for the signal processing engineer, allowing the construction of DSP systems that maintain the flexibility of software based solutions but providing performance closer to application specific integrated circuits (ASICs).
This paper provides an overview of FPGA DSP from both an applications and implementation perspective. Examples of FPGA DSP in image processing and digital communications will be used to illustrate the utility of FPGAs for high-performance DSP. An overview of the discrete wavelet transform (DWT) will be provided, and considerations for its efficient FPGA implementation will be discussed. Anovel technique for efficiently implementing FPGA-based digital filters will be presented. The design of several digital receiver functions as well as the use of run-time re-configuration will be covered.

WAVLET BASED SIGNAL PROCESSING

The discrete wavelet transform (DWT) is one of the most useful and efficient tools used to analyze digital signals in various signal processing areas. Some of the common application areas for the DWT are: seismic exploration, speech analysis and synthesis, image analysis and compression, signal detection, communications, filter banks, time varying spectral estimation, noise filtering and signal or image restoration, feature detection for general pattern recognition algorithms, adaptive systems, and biomedical signal processing. In signal analysis, the DWT has some inherent generic advantages and are near optimal for wide class of problems. As a decomposition tool the DWT separates components of a signal in a way that is superior to most other methods for analysis, processing, or compression. This powerful and flexible decomposition tool also offers new nonlinear processing option for signal and image processing, detection, filtering, and compression.

One major concern, in signal and image processing as well as the communication communitie is the effective implementation of the wavelet transform and advanced tools for designing wavelet systems. When there are limitations in processing time and/or system size, implementation of the DWT becomes an engineering issue. In this situation, FPGA's offer a framework which not only addresses the issue of speed for real-time processing but also offers flexibility in terms of system size.

WAVELET TRANSFORM THEORY

A wavelet system consists of *building blocks* to construct or represent a function. This system, which is not unique, is identified by its two-dimensional basis, $\psi_{j,k}(t)$ for $i, j = 1, 2, \ldots$, in an orthogonal expansion and by $\psi_{j,k}(t)$ and $\tilde{\psi}_{j,k}(t)$ for $i, j = 1, 2, \ldots$, in a biorthogonal expansion **[5]**. The main property of this expansion is localization of the function (or signal) in the time-frequency or space-frequency domain. In other words, a wavelet representation will provide information about the signal in both time and frequency (or space and frequency). The basis for a wavelet expansion system is generated from a single scaling function or wavelet by simple *scaling* and *translation.* The generating wavelet or mother wavelet, represented by $\psi(t)$, results in the following two-dimensional parameterization of $\psi_{j,k}(t)$'s .

$$\psi_{j,k}(t) = 2^{j/2}\psi(2^j t - k) \; ; \quad \text{for } j,k \in \mathbf{Z} \tag{1}$$

where $\mathbf{Z}$ is the set of all integers and the $2^{j/2}$ factor normalizes each wavelet to maintain a constant norm independent of scale j. All useful wavelet systems satisfy the *multiresolution* conditions. In this case, the lower resolution coefficients can be calculated from the higher resolution coefficients by a tree-structured algorithm called a *filter-bank* **[7]**. This allows a very efficient calculation of the DWT coefficients. The multiresolution idea is better understood by using a function represented by $\varphi(t)$ and referred to as a scaling function. A two-dimensional family of functions is generated, similar to Eq. (1), from the basic scaling function by

$$\varphi_{j,k}(t) = 2^{j/2}\varphi(2^j t - k) \; ; \quad \text{for } j,k \in \mathbf{Z} \tag{2}$$

Any continuous function, $x(t)$, can be represented, at a given resolution or scale j_0, by a sequence of coefficients given in $x(t) = \sum_k a_k \varphi_{j_0,k}(t)$ expansion. Higher values of j correspond to higher resolution **[6]**. The lower resolution function, $\varphi(t)$, can be expressed by a weighted sum of shifted version of the same scaling function at the next higher resolution, $\varphi(2t)$, by

$$\varphi(t) = \sum_k h(k)\sqrt{2}\varphi(2t - k), \qquad k \in \mathbf{Z} \tag{3}$$

where the coefficients $h(k)$ are called the scaling functions. This recursive equation is fundamental to the theory of the scaling function and is referred to by the refinement equation, the multiresolution analysis equation, or the dilation equation. Design of a wavelet system is the choosing of the set of coefficients represented by the $h(k)$'s, in Eq. (3).

Important features of a signal can be described better by defining a slightly different set of functions that span the difference between the spaces spanned by various scales of the scaling function. Since these wavelets reside in the space spanned by the next narrower scaling function, they can be represented by a weighted sum of shifted scaling function $\varphi(2t)$ as

$$\psi(t) = \sum_{k} h'(k)\sqrt{2}\varphi(2t-k), \qquad k \in \mathbf{Z} \tag{4}$$

for some set of coefficients $h'(k)$. Wavelet coefficients, $h(k)$, and scaling function coefficients, $h'(k)$, can be related to each other by some imposing conditions. For example, by forcing the orthogonality condition on the integer translates of the wavelets, and based on the requirement that wavelets should span the "difference" spaces **[1]** or by requiring mirror image **[4]**, it is shown that

$$h'(k) = \pm(-1)^k h(1-k) \qquad \text{or} \qquad h'(k) = \pm(-1)^k h(k) \tag{5}$$

The function generated by (4) gives the prototype or mother wavelet $\psi(t)$ for a class of expansion functions of the form of Equation (1) in which 2^j is the scaling of t, $2^{-j}k$ is the translation in t, and $2^{j/2}$ maintains the constant norm of the wavelet at different scales. It is shown **[1]** that any continuous function can be represented by the following expansion, defined in terms of a given scaling function and its wavelet derivatives:

$$x(t) = \sum_{k=-\infty}^{\infty} c_{j_0}(k)\varphi_{j_0,k}(t) + \sum_{j=j_0}^{\infty}\sum_{k=-\infty}^{\infty} d_j(k)\psi_{j,k}(t) \tag{6}$$

In this expansion, the first summation gives a function that is a low resolution or coarse approximation of $x(t)$ at scale j_0. For each increasing j in the second summation, a higher or finer resolution function is added. The choice of j_0 sets the coarsest scale whose space is spanned by $\varphi_{j_0,k}(t)$. The set of coefficients in the wavelet expansion represented by (6) is called the *discrete wavelet transform* of $x(t)$.

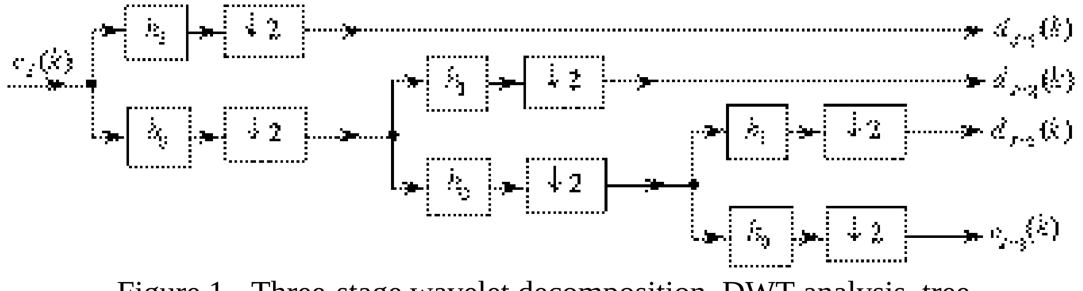


Figure 1 - Three-stage wavelet decomposition, DWT analysis, tree.

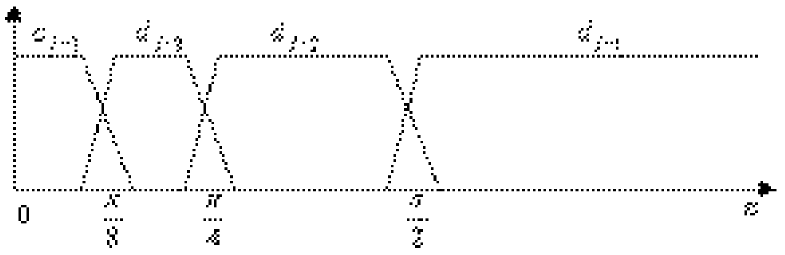


Figure 2 - Logarithmic division of the frequency band between different DWT sequences.

WAVELET TRANSFORM SYSTEMS

Researchers in related engineering and applied mathematics areas have developed many different wavelet transform systems each with specific properties. The difference between these wavelet transforms is mainly in their scaling functions and the way that they are developed. There are two major classes of wavelet transform systems. One class consists of orthogonal wavelets and the other one consists of biorthogonal wavelets. Other wavelet transform systems, not included in the two main categories, have generally limited applications.

A set of functions is called a basis set for a given function space if there is a unique set of coefficients, for any particular function in that space, $g(t)$, that satisfy the following equation **[3]**

$$g(t) = \underset{l}{\ } a_l f_l(t) \qquad (7)$$

The required condition for a set of functions to construct a basis is that they should be independent. The set is called an orthogonal basis if $\langle f_m(t), f_n(t)\rangle = 0$ for all $m\ ?\ n$. The set is an orthonormal basis if $\langle f_m(t), f_n(t)\rangle = \delta(m-n)$. The basis functions are normalized to unity norm: $\|f_l(t)\| = 1$ for all l's. In the case of orthonormal basis, any function in the function space can be expressed as

$$g(t) = \underset{l}{\ } \langle g(t), f_l(t)\rangle f_l(t) \qquad (8)$$

not orthogonal to each other, but they are orthogonal to the corresponding elements of the expansion set $\langle f_m(t), \tilde{f}_n(t)\rangle = \delta(m-n)$. Since this kind of orthogonality requires two sets of basis functions, the expansion set and the dual set, the system is called biorthogonal. Any function in the function space can be expressed by

$$g(t) = \underset{l}{\ } \langle g(t), \tilde{f}_l(t)\rangle f_l(t) \qquad (9)$$

A biorthogonal system is more general and allows a larger class of expansions. If a set of functions are dependent and yet does allow the expansion described in (8), then the set is called a frame. Frames are over-complete versions of a basis set, and tight frames are over-complete versions of an orthogonal basis set. If one is using a frame that is neither a basis nor a tight frame, a dual frame can be specified so that analysis and synthesis can be done for a non-orthogonal basis.

Requiring orthogonality uses up a large number of the degrees of freedom, results in complicated design equations, prevents linear phase analysis and synthesis filter banks, and prevents asymmetric analysis and synthesis systems. By allowing nonorthogonal basis and dual basis, in biorthogonal wavelet systems, we can attain greater flexibility in achieving other goals. In this case, by relaxing the orthogonality constraint, a pair of scaling functions and their corresponding scaling filters are introduced **[2][4]**. If $h(k)$ and $\tilde{h}(k)$ represent the pair of scaling filters, then, they should satisfy

$$\sum_{k} \tilde{h}(k)h(k-2l)=\delta(l)$$

In the orthogonal case, (25), $h(k)$ is orthogonal to even translation of itself while in the biorthogonal case, $\tilde{h}(k)$ is orthogonal to even translation of $h(k)$. In the orthogonal case, for finite duration $h(k)$, the length of the filter $h(k)$ has to be even. In the biorthogonal case, the difference between the lengths of $h(k)$ and $\tilde{h}(k)$ must be even. Thus, their length must be both either even or odd **[4]**. In this situation, it is possible to design linear phase filters, the property that is not possible to attain in orthogonal wavelet systems.

The multiresolution definition for biorthogonal wavelet systems is given as follows:

$$\varphi(t)=\sum_{k} h(k)\sqrt{2}\varphi(2t-k), \qquad k\in \mathbf{Z} \;;\; \tilde{\varphi}(t)=\sum_{k} \tilde{h}(k)\sqrt{2}\tilde{\varphi}(2t-k), \qquad k\in \mathbf{Z} \tag{11}$$

The biorthogonal wavelets, $\psi(t)$ and $\tilde{\psi}(t)$, are defined by

$$\psi(t)=\sum_{k}(-1)^{k}\tilde{h}(1-k)\sqrt{2}\varphi(2t-k)=\sum_{k} g_{1}(k)\sqrt{2}\varphi(2t-k), \qquad k\in \mathbf{Z} \tag{12}$$

$$\tilde{\psi}(t)=\sum_{k}(-1)^{k}h(1-k)\sqrt{2}\tilde{\varphi}(2t-k)=\sum_{k} h_{1}(k)\sqrt{2}\tilde{\varphi}(2t-k), \qquad k\in \mathbf{Z} \tag{13}$$

Under some technical conditions, we can expand functions using the wavelets, $\psi(t)$, and reconstruct them using their dual functions, $\tilde{\psi}(t)$.

The classical wavelet system results in a logarithmic frequency resolution shown in **Figure 2**. The low frequencies have narrow bandwidth and the high frequencies have wide bandwidth. This is called "constant-Q" filtering and is appropriate for some applications but not all. The *wavelet packet* system allows a finer and adjustable resolution of frequencies at higher bandwidth. It also gives a rich structure that allows adaptation to particular signals or signal classes. This can be achieved by allowing the highpass wavelet branch as well as the lowpass wavelet branch to split and down sampled, in an iterative fashion. The resulting three-stage analysis tree is shown in **Figure 3**. In this case, the resulting filter bank structure is like a full binary tree.

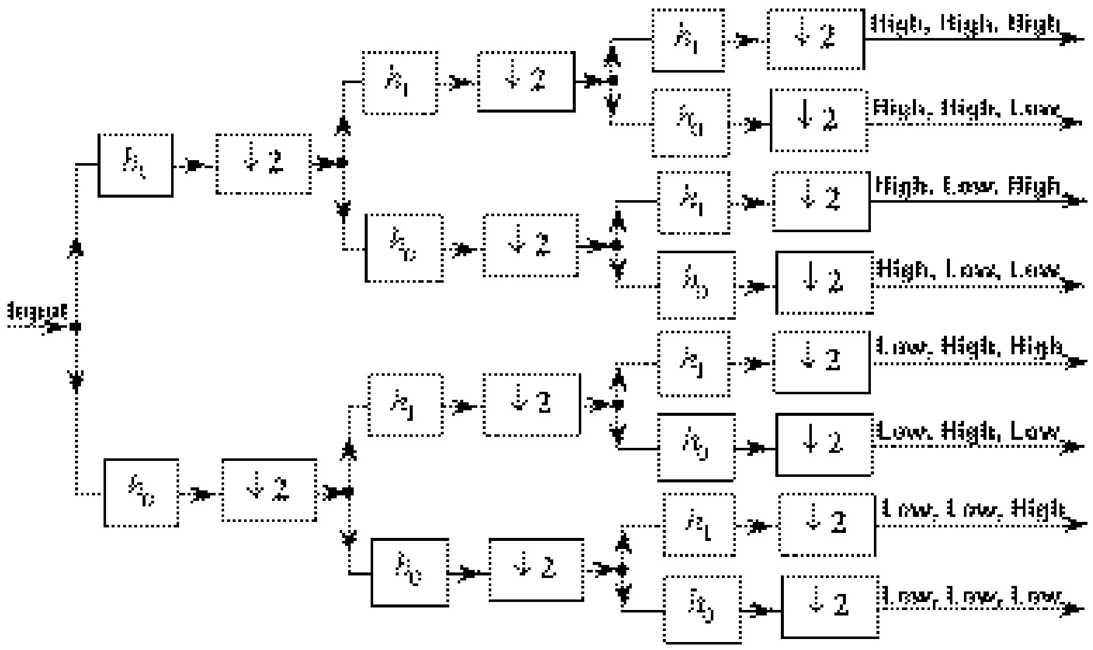


Figure 3 - The full binary tree for the three-scale wavelet packet transform

WAVELET REALIZATION

The filter bank realizations of the discrete wavelet transform and its inverse are based on two basic building blocks, shown in Figure 4. In the forward transform, the two decimating FIR filters are $h_0(k)$ and $h_1(k)$, and in the inverse transform, the two interpolating filters are $g_0(k)$ and $g_1(k)$.

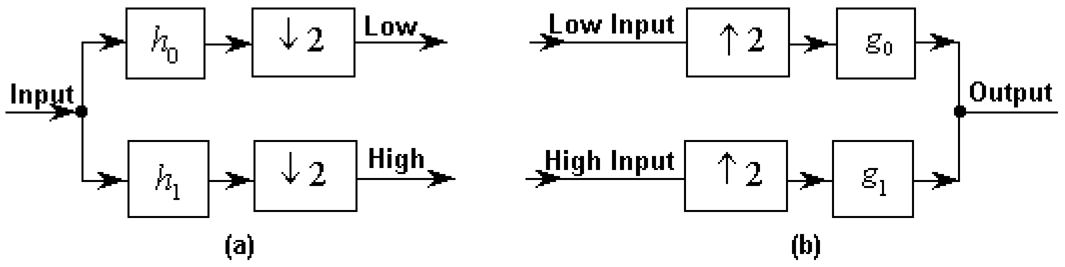


Figure 4: - Basic Blocks for (a) analysis and (b) synthesis filter banks

Perfect reconstruction requires that the output of the synthesis block be the same as a delayed and scaled version of the input of the analysis block. Based on this desired condition, the analysis and synthesis filters should always satisfy the following two relations[4]:

$$\begin{aligned} h_1(k) &= (-1)^k g_0(k) \\ g_1(k) &= -(-1)^k h_0(k) \end{aligned} \tag{14}$$

Direct realization of the two building blocks shown in Figure 5 require that all computations be carried out at the higher input rate for the analysis filters and at the higher output rate for the synthesis filters. Realization of the analysis and synthesis filters can be accomplished at the lower rate by taking advantage of polyphase representation of these filters[7]. The synthesis and analysis building blocks can be utilized in the form shown in Figure 5. Both analysis and synthesis building blocks are realized by only the knowledge of analysis and synthesis lowpass filters. In this configuration, all the filtering operations are carried out in the lower rate.

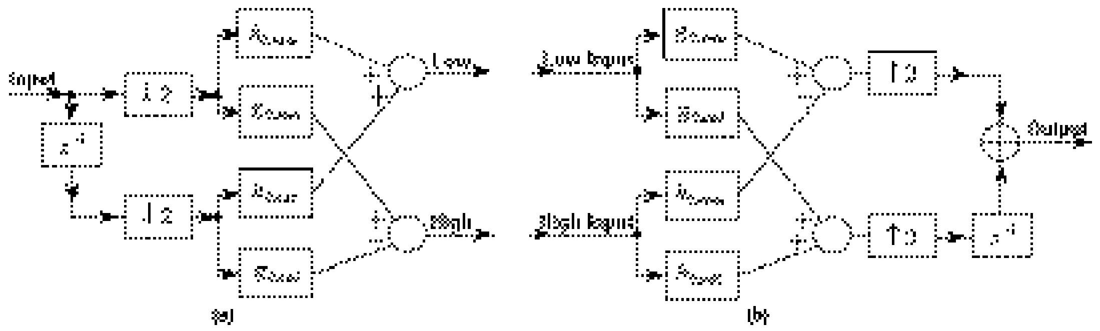


Figure 5 - Polyphase realization of the basic (a) analysis and (b) synthesis blocks

In biorthogonal realization, the assumption is that the desired wavelet transform is a particular biorthogonal system whose scaling filters, $h_0(k)$ with K coefficients and $g_0(k)$ with $\tilde{K}$ coefficients, are known. Both of these filters should be known with highpass filters obtained from perfect reconstruction conditions given in (11). Unlike the orthogonal wavelet filters, which are always even, biorthogonal filters can both be selected to be odd or even.

WAVELET IMPLEMENTATION

VLSI implementation of the DWT or wavelet packet has been addressed in journal articles [8][9][10]. In addressing the implementation of the DWT analysis tree, it is useful to define the basic wavelet engine as shown in Figure 4a. Referring to Figure 1 we recognize that each stage has processing elements which are half the rate of the previous stage. By using the polyphase realization we can implement a DWT analysis tree with a single engine and proper scheduling of that resource. This feedback method is particularly attractive in terms of IO cycles off chip to and from memory. We can essentially design a 1D DWT by time sharing a single DWT engine.

VLSI implementation of wavelet packet (Figure 3) requires more processing due to the full binary tree structure. Since each stage is further processed a DWT engine being shared down the levels of the tree structure is not desirable. By observing the amount of processing performed at each stage we find that a more desirable VLSI structure is one of a DWT engine per stage. Through proper scheduling within each stage a stream in / stream out structure is designed which handles the decimation properly and fully utilizes the DWT engine. Other approaches can be taken which design DWT engines which are smaller and slower but do add to the complexity of the VLSI design. In both the DWT and wavelet packet the inverse operations have dual arguments for processing.

FPGA DIGITAL FILTERS

There have been many techniques for implementing digital filters using FPGAs reported in the open literature. One of the more successful approaches has been the implementation of Peled and Liu's distributed arithmetic (DA) filter realization [11] reported in [12]. This section extends the DA approach and describes an efficient approach for implementing narrowband FPGA filters using an *interpolated* finite impulse response (FIR) filter.

INTERPOLATED FIR – *IFIR*

Linear phase filters are conventionally implemented using a standard tapped delay-line or transversal filter structure. However, many other realization options present themselves if the filter designer is prepared to take advantage of the certain features that may be characteristics of the filter requirements themselves, or that become available when the filter is taken in context with adjacent processing blocks. Filters whose bandwidth is considerably smaller than the system sample rate, or *narrow-band filters,* occur in many practical applications. Several options other than the standard FIR realization can be used to produce efficient narrow-band filter implementations.

The cascade filter structure shown in Figure 6 is a two-stage *interpolated FIR (IFIR)* filter.

$$\begin{tikzcd} x(n) \arrow[r] & M(z^k) \arrow[r] & I(z) \arrow[r] & y(n) \end{tikzcd}$$

Figure 6: Interpolated FIR (IFIR) architecture.

The sample rate for both stages is the same as that at the input and output. So although the name alludes to multirate techniques, strictly speaking this is not a multirate filter, since the sampling frequency at all stages is the same. With IFIRs the multirate aspect of the system is embodied in the filter impulse response.

The transfer function $M(z^k)$ of the first stage is a function of z^k.

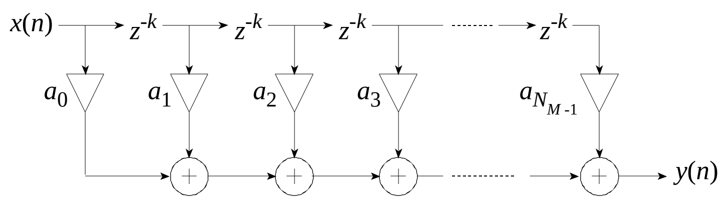


Figure 7: Upsampled FIR architecture $M(z^k)$.

It can be implemented by replacing the unit delay operator z^{-1} in a transversal filter $M(z^k)$ by z^{-k} as shown in Figure 7. This means simply replacing each delay element by

k units of delay. This is equivalent to inserting $k-1$ zeros between the original impulse response values of $M(z)$. Now recall that a k-fold rate expansion of a time series causes $k-1$ spectral replicates to fold back into the frequency interval $[0\ \text{a}\ \pi]$. The same observation is obviously true for $M(z^k)$, since a filter impulse response is just a time-series. The situation is shown in the frequency domain plot of Figure 8 for a rate expansion k=4.

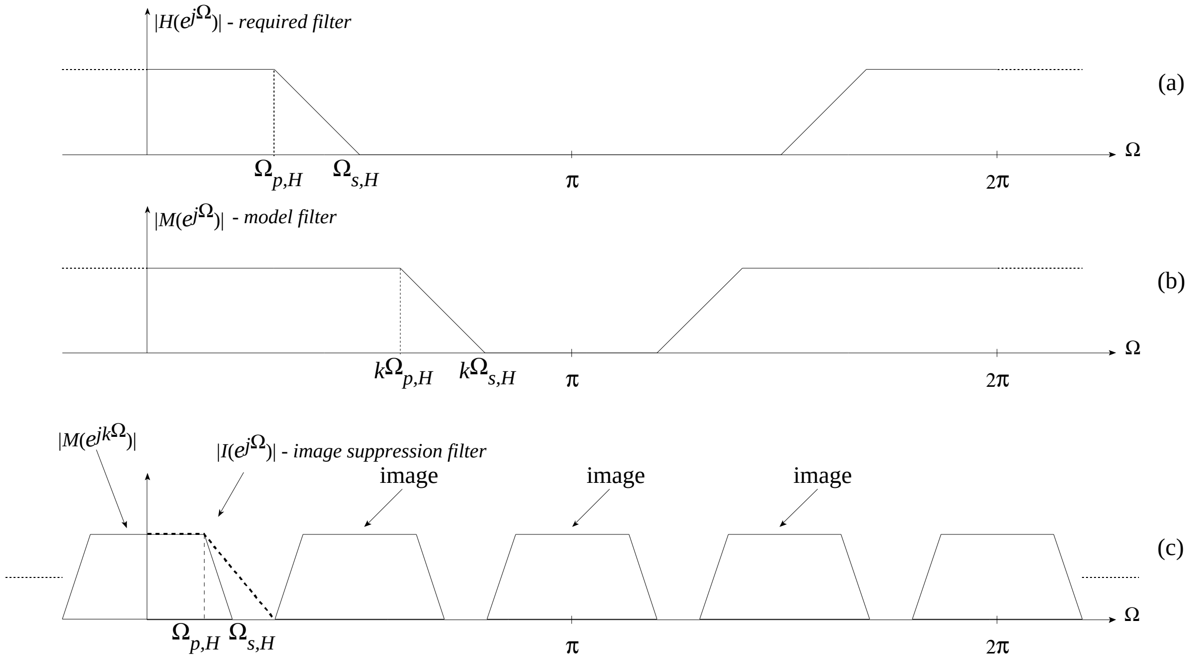


Figure 8: IFIR - frequency domain illustration.

In going from $M(e^{j\Omega})$ to $M(e^{jk\Omega})$ the frequency domain response is compressed by the factor k. The critical frequencies in the baseband spectrum of $M(e^{jk\Omega})$ are k times narrower than those of $M(e^{j\Omega})$.

So far we have seen that rate expanding a filter's impulse response has the desired effect of decreasing the baseband filter transition-band without introducing any additional arithmetic. The narrow-band response comes at only the expense of some additional memory to accommodate the increased storage requirements for the input signal time history. However, the spectral replicates or *images* are a problem and must be dealt with. This is the function of the *image rejection filter* $I(z)$ in Figure 6 and Figure 8(c). The resultant transfer function is $H(z) = M(z^k)I(z)$

The compressed response reduces the computational complexity by a factor of approximately k. The complexity for constructing $I(z)$ must also be considered. In many situations a suitable choice of k will result in a shallow transition slope for the image suppressor, so that this filter will have substantially fewer filter coefficients than the desired filter $H(z)$ and only contribute modestly to the computation load.

IFIR DESIGN

The IFIR approach is suitable for any narrow-band filter process-lowpass, bandpass or highpass. The design approach will be described using a lowpass methodology.

Consider the synthesis of a lowpass filter $H(z)$ with a passband edge frequency $\Omega_{p,H}$, stopband edge frequency $\Omega_{p,S}$ and with $\delta_{p,H}$ and $\delta_{p,S}$ passband and stopband ripple respectively. First a model filter $M(z)$ is defined having a passband edge frequency $\Omega_{p,M} = k\Omega_{p,H}$ and stopband edge frequency $\Omega_{s,M} = k\Omega_{s,H}$. The passband edge frequency of $I(z)$ must be the same as the required filter $H(z)$, so $\Omega_{p,I} = \Omega_{p,H.}$ The stopband edge frequency for the image suppression filter must be chosen so that its magnitude response provides the specified attenuation at the first intercept point with the first image of $M(z^k)$ above baseband. Some simple arithmetic results in this design frequency being defined as $\Omega_{s,I} = 2\pi k - \Omega_{s,H.}$ The passband ripple for $H(z)$ is distributed between $M(z^k)$ and $I(z)$. Formally we require

$$(1+\delta_{p,M})(1+\delta_{p,I}) = 1+\delta_{p,H}$$

If the ripple $\delta_{p,H}$ is small, then $\delta_{p,M} + \delta_{p,I} \cup \delta_{p,H}$. One possibility for solving this equation that works in practice is to choose $\delta_{p,M} = \delta_{p,I} = 0.5\delta_{p,H}$.
Provided δ_p is small it is possible to make the further approximation $\delta_{s,M} = \delta_{s,I} \cup \delta_s$.

Lowpass IFIR Example

Consider implementing the lowpass filter characterized in Figure 9.

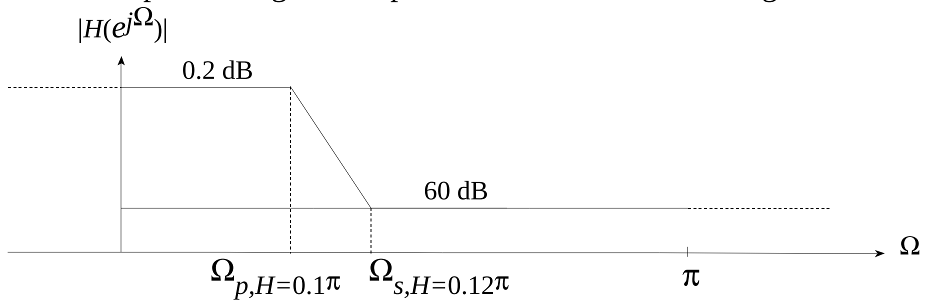
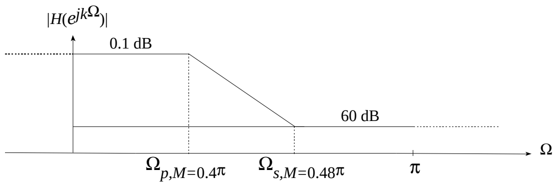


Figure 9: Narrow-Band lowpass filter example - magnitude response requirements.

Using a normalized sample rate of 2π radians, the passband edge frequency is $\Omega_{p,H} = 0.05 \leftrightarrow 2\pi$, the stopband edge frequency is $\Omega_{s,H} = 0.06 \leftrightarrow 2\pi$, the passband ripple is to be 0.2 dB and the required minimum stopband attenuation is 60 dB. Using filter design software the number of filter taps, *N*, required to meet the filter requirements is 260. An efficient technique for implementing inner-product calculations using distributed memory FPGA architectures like the Xilinx XC4000 series, is *distributed arithmetic* [11]. The mechanization of this type of filter, as well as many other DSP related functions is facilitated by the *Core Generator* [14] design automation tool. The largest FIR module that can be produced automatically is an 80-tap filter. However, these modules are cascadable. The required 260-tap filter can be easily built from available modules by cascading three 80-tap filters and a 20-tap section. Exploiting optimizations based on the symmetry in the filter coefficient data, a single 80-tap filter employing 16-bit arithmetic requires 320 CLBs. A 20-tap filter using 16-bit precision occupies 95 CLBs. Cascading these modules to produce the required 260-tap filter gives a CLB count of 1055.

Now consider an IFIR realization of the same filter. The model filter passband and stopband specifications are $\Omega_{p,M} = k\Omega_{p,H}$ and $\Omega_{s,M} = k\Omega_{s,H}$. To support the required passband ripple requirement $\delta_{p,H}$ of the original filter $H(z)$, the model and image rejection filters do not simply inherit the same value. They must be designed using a smaller value for passband ripple, so that the series cascade combination provides the required performance. For this example the passband ripple requirement will be equally distributed between $M(z)$ and $I(z)$. So



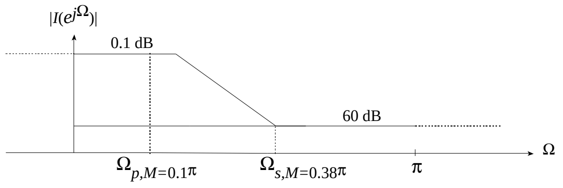


Figure 10: Model filter specifications for the Lowpass IFIR example, *k*=4.

Figure11: Image rejection filter specications a lowpass IFIR, *k*=4.

$$\delta_{p,M} = \delta_{p,I} = \frac{1}{2}\left(10^{0.2/20} - 1\right) = 0.0116464$$

Selecting the up-sampling factor as *k*=4 results in the model filter specification defined in Figure 10. The image rejection filter $I(z)$ design frequencies are $\Omega_{p,I} = \Omega_{p,H} = 0.1\pi$ and

$\Omega_{s,I} = 2\pi / k - \Omega_{s,H} = 0.38\pi$. This filter is illustrated in Figure 11. To meet the model filter requirements a 69-tap filter is needed. Figure 12a shows the magnitude frequency response for $M(z)$. Upsampling $M(z)$ by a factor *k*=4 produces the response in Figure 12b. The image rejection filter specifications can be met with a 22-tap filter. The magnitude response for $I(z)$ is shown in Figure 12c. The net frequency response of this combination is obtained by multiplying $M(z^4)$ with $I(z)$. This result is shown in Figure 12d.

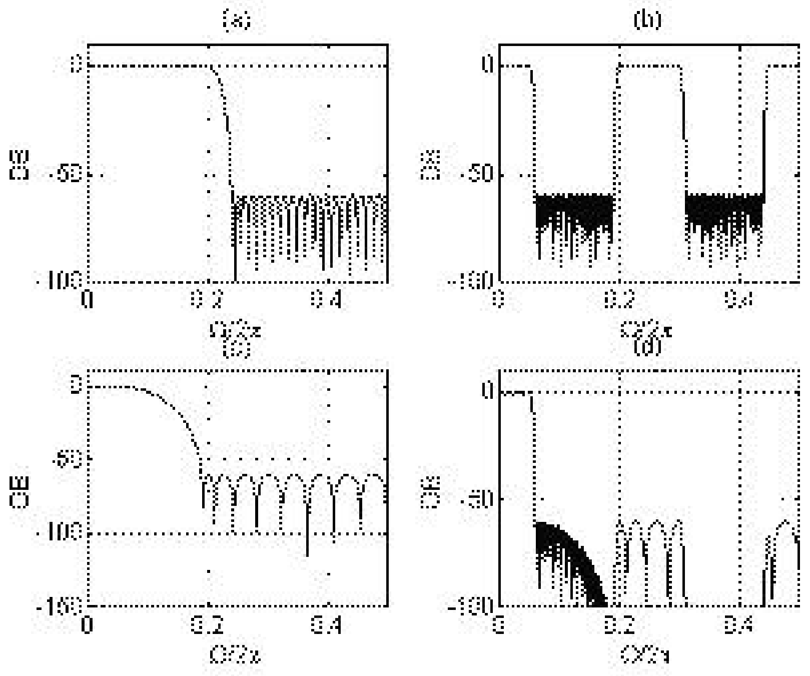
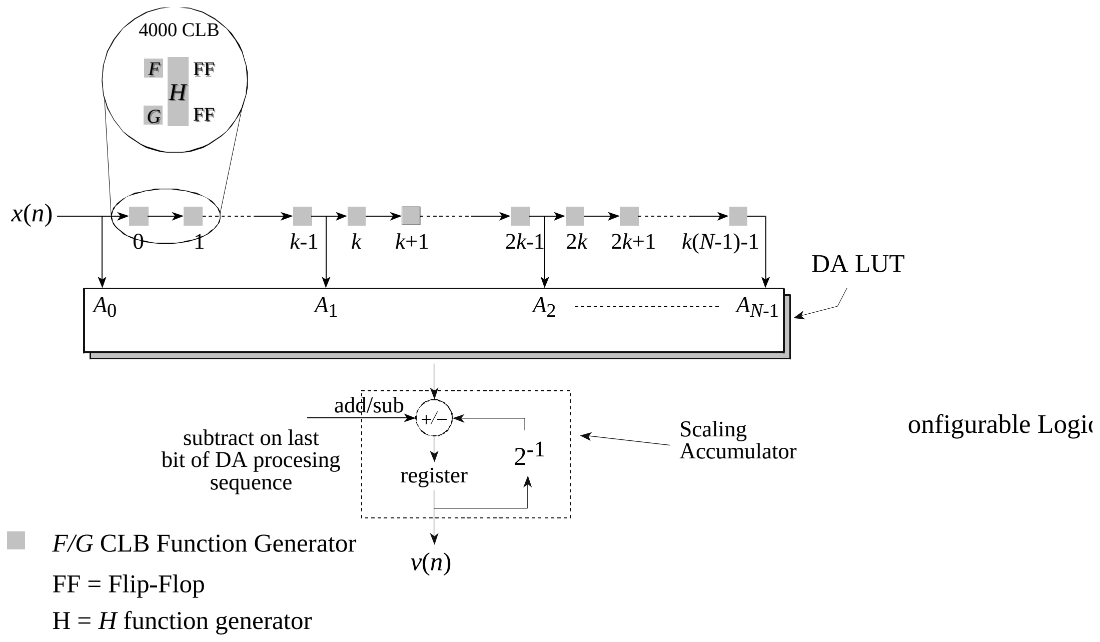


Figure 12: (a) Model filter magnitude response. (b) Upsampled model filter, *k*=4. (c) Image filter response. (d) IFIR $M(z^4)I(z)$.

IFIR FPGA Implementation

The architecture of the upsampled filter $M(z^k)$ implemented in an FPGA is shown in Figure 13. Zero-packing the filter impulse response is of course achieved in practice by replacing the unit delay z^{-1} in the model filter with *k* units of delay. The filter supports a history of the previous $k(N_M - 1)$ input samples, but only N_M multiply-accumulates are required to compute an output value. Taking advantage of the highly



efficient implementation of FIR filters in Xilinx FPGAs using distributed arithmetic results in the architecture shown in Figure 13.

Logic Requirements

The IFIR implementation requires an input sample delay-line or shift-register, and two FIR structures for $M(z)$ and $I(z)$. The delay-line must have taps at intervals of the up-sampling factor *k*. This structure supports the input sample precision *B*, and its length is the product of *k* and the model filter length $N_{M(z)}$. The shift-register is implemented with Xilinx technology using RAM-cell based shift-registers - function generators. Two shift-register stages of up to 16-bit precision can be supported in a single XC4000 series configurable logic block (CLB). Excluding the minimal amount of control required for the input shift-register, the CLB count Γ for an IFIR filter is

$$\Gamma = \frac{kN_{M(z)}}{2} + \Xi(M(z)) + \Xi(I(z))$$

where $\Xi(f)$ denotes the number of CLBs required to implement the functional unit *f*. Using the LPF example above, with *B*=16, the input delay-line is $4 \leftrightarrow 69 = 276$ samples deep, and requires 136 CLBs to implement. The filter $M(z)$ is implemented with 270 CLBs and $I(z)$ requires 99 CLBs. The total CLB count is $\Gamma_1 = 509$. Comparing this figure to the 1010 CLB requirement for a direct implementation of $F(z)$ for the above example shows that the IFIR approach requires only $509/1055 \leftrightarrow 100 = 48\%$ of the logic resources. This is a considerable saving.

Performance - Computation Rate The entire IFIR architecture will support a sample rate of 6 MHz. This equates to a computation throughput of 1.5 Giga-MACs (multiply-accumulates) per second. Re-phrasing this number, it is equivalent to 3 Mega-MACs per CLB - this is a large amount of filtering using only a modest amount of silicon real estate!

DIGITAL COMMUNICATIONS

FPGAs are being employed in high-performance communication systems at an exponential growth rate. Pulse shaping filters, modulators, convolutional encoders, Viterbi decoders, interleavers and de-interleavers, transforms and adaptive equalizers are just a few of the types of functionality this technolgy is being used for.

It is not always apparent how best to best implement a desired type of DSP functionality in an FPGA. Many digital signal processing engineers may be knowledgeable on implementing signal processing algorithms using software programmable digital signal processors. However, a direct translation of this approach to the FPGA domain may not produce area efficient high-performance FPGA designs: alternative algorithms and architectures may be called for.

Structures that save multiplications and implement transcendental functions without multiplications can be easily mapped to FPGAs. One simple example is polar-to-rectangular and rectangular-to-polar conversions. These conversions are used in communication receivers and modulators as complex heterodynes. A signal presented to the front end of a receiver is often subjected to a spectral translation, i.e. a heterodyne, implemented as a complex multiplication with a complex sinusoid. In conventional receivers, forming the complex sinusoid and applying it to the input data are two different tasks as shown in Figure 14.

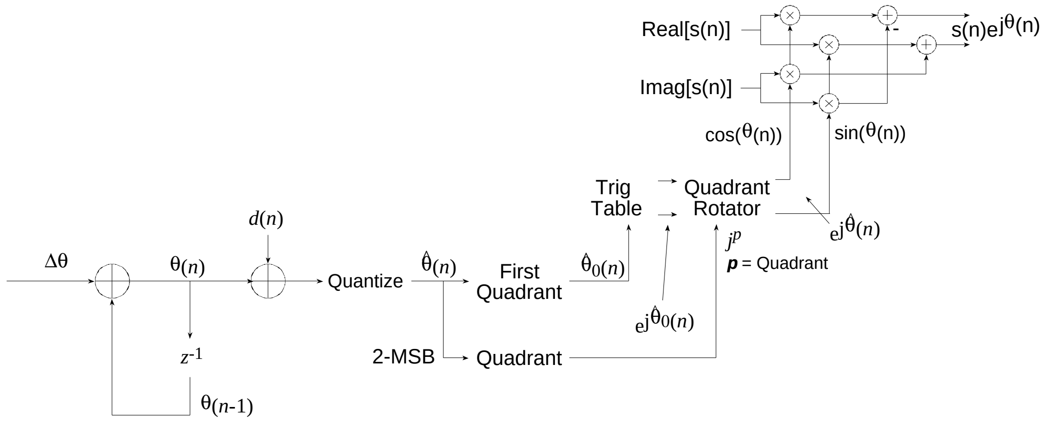


Figure 14; Conventional heterodyne employing a direct digital synthesizer and complex multiplier.

For a complex input signal, the heterodyne requires 4 multiplications and 2 additions, as well as the cost of generating the complex heterodyning waveform by a direct digital synthesizer (DDS). The major consumer of logic resources in the DDS is the sin/cosine lookup-table (LUT). Using Xilinx FPGA technology, this module is placed in distributed RAM in XC4000 devices. Using Virtex [13] technology the LUT may be located in on-chip block memory or distributed RAM.

The complex product performed in the heterodyne circuit performs a rotation of the complex input samples. This functionality can be performed in an alternative manner using coordinate rotation digital computer (CORDIC) arithmetic [15]. A CORDIC based heterodyne is shown in Figure 15.

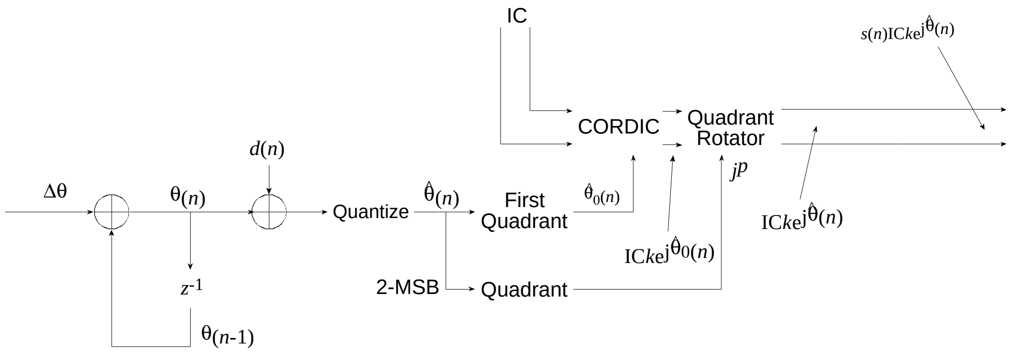


Figure 15: CORDIC based heterodyne.

The complex input sample is loaded as the initial condition (IC) to the CORDIC engine. The angle of rotation is generated by the integrator. The interesting points to observe here are that the circuit does not employ any multipliers. The CORDIC algorithm only requires a sequence of shift and add/subtract operations. This type of arithmetic is easily and efficiently accommodated using FPGAs. The design illustrates an approach to solving a DSP problem that would not normally be employed in a software programmable DSP solution, but that can afford logic resource savings in an FPGA design.

Run-Time Reconfiguration

Xilinx SRAM based FPGAs are configurable by downloading a configuration bit-stream. The bit-stream defines the device functionality and in most systems the configuration memory is static during operation. In more recent times a new facet has been introduced to this aspect of system design with FPGAs: *dynamic* or {*run-time reconfiguration* (RTR). With this hardware model, selected parts of the configuration memory can be updated at execution time. During the reconfiguration period, the parts of the datapath that are not influenced by the changes in configuration memory remain active, and are still usefully processing data or maintaining state information. The approach offers the potential for true silicon resource sharing. A useful analogy can be drawn between RTR and the type of resource sharing afforded by a multi-user operating system.

The recent Virtex family from Xilinx provides RTR hardware support. An early example application that explored the potential for RTR was the DISC (dynamic instruction set computer) [16] project. DISC essentially consisted of a processor augmented with configurable hardware for performing application specific tasks. The application specific functions were supported by a set of complex instruction that executed on the FPGA hardware. So, in an image processing environment, for example, a 2-D convolution instruction would be designed that executed on the FPGA hardware. This feature would be made available to an application as an extension to the host processor's instruction set. An example of RTR in a DSP context is presented as a solution to an automatic target recognition problem is described in [17]. The utility of RTR for DSP is further explored in the next example.

Consider the digital receiver architecture defined in Figure 16.

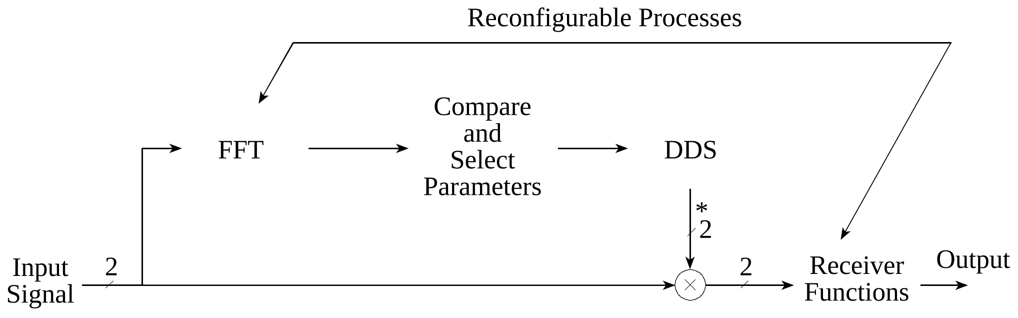


Figure 16: Run-time reconfigurable digital receiver.

The receiver operates in two distinct modes-fast acquisition and post acquisition detection. Here the transform generates a pre-processing estimate of system parameters such as center frequency in a multi-carrier system, bandwidth estimate in a bandwidth on demand system, spreading code parameters in a spread spectrum system such as GPS and CDMA, and Doppler frequency estimate for systems with high relative velocities (such as low earth orbit satellites (LEOS)). The receiver cannot be processing data until system parameters are passed to it from the transform based pre-processor and select subsystem. After the parameters have been estimated and passed to the receiver, the transform serves no purpose and can be disabled. Here we see two distinct and non-overlapping processes required for this system to operate. In a reconfigurable machine, the same resources can be applied to the two distinct tasks. After the transform has finished with its task, it can be disassembled and reassembled as the back end processor of the receiver. The disassembly and reassembly can occur during the compare and select process.

For the interested reader, several very useful overviews of reconfigurable computing and RTR can be found in references [18] and [19].

CONCLUSION

This paper has provided an overview of FPGAs and their utility in bringing both flexibility and performance to a DSP design.

FPGAs put the silicon back in the hands of the signal processing engineer and expose a diverse range of design opportunities. This was illustrated in the filter design example. The IFIR approach is not optimal for a software programmable DSP platform, but was shown to be a useful technique for FPGA based hardware.

The wavelet transform offers a new method for signal processing in time/frequency (space/frequency) domain with applications toward digital communications and imaging. VLSI architectures were proposed which make effective use of silicon area for real time processing.

Run-time reconfiguration techniques are at an early stage of evolution. Even so this technology offers exciting new possibilities for future generation DSP hardware.

ACKNOWLEDGEMENTS

Some of the work focused on the application of FPGA technology to communication system was performed in conjunction with Professor fred harris, Cubic Signal Processing Chair, Communications and Signal Processing Institute, San Diego State university, San Diego, CA, USA.

REFERENCES

[1] Burrus, C. S.; Gopinath, R. A.; and Guo, H., Introduction to Wavelets and Wavelet Transforms: A Primer. Prentice Hall Inc., Upper Saddle River, New Jersey, 1998.

[2] Cohen, A.; Daubechies, I.; and Feauveau, J. C., *Biorthogonal bases of compactly supported wavelets*, Communications on Pure and Applied Mathematics. Vol. 45, pp. 485-560, 1992.

[3] Daubechies, I., *Ten Lectures on Wavelets*. SIAM, Philadelphia, Pennsylvania, 1992.

[4] Strang, G. and Nguyen, T., Wavelets and Filter Banks. Wellesley-Cambridge Press, Wellesley, Massachusetts, 1996.

[5] Sweldens, W., *Wavelets: what next?* Proceedings of the IEEE, Vol. 84, No. 4, pp. 680-685, April 1996.

[6] Vaidyanathan, P. P. and Djokovic, I., *Wavelet transforms, in The Circuits and Filters Handbook*. Edited by W. K. Chen, Chapter 6, pp. 134-219, CRC Press and IEEE Press, Boca Raton, Florida, 1995.

[7] Vaidyanathan, P. P., Multirate Systems and Filter banks. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1993.

[8] Yu, C. and Chen, S., *VLSI Implementation of 2-D Discrete Wavelet Transform For Real-Time Video Signal Processing.* IEEE Transactions on Consumer Electronics, Vol. 43, N0. 4, November 1997.

[9] Grzeszcsak, A., Mandal, M., Panchanathan, S., Yeap, T., *VLSI Implementation of Discrete Wavelet Transform.* IEEE Transactions on VLSI Systems, Vol. 4, No. 4, December 1996.

[10] Parhi, K., Nishitani, T., *VLSI Architectures for Discrete Wavelet Transforms.* IEEE Transactions on VLSI Systems, Vol. 1, No. 2, June 1993.

[11] A. Peled and B. Liu, “A New Hardware Realization of Digital Filters”, *IEEE Trans. on Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.

[12] L. Mintzer, “FIR Filters with Field Programmable Gate Arrays”, *Journal of VLSI Signal Processing*, No. 6, pp. 120-127, 1993.

[13] Xilinx Inc., Virtex Product Specification, 1998.

[14] Xilinx Inc., Core Solutions Data Book, 1998.

[15] J. E. Volder, “The CORDIC Trigonometric Computing Technique”, *IRE Trans. on Electronic Computers*, Vol. 8 no. 3, pp. 330-334, 1959.

[16] M. J. Wirthlin and B. L. Hutchings, “DISC: The Dynamic Instruction Set Computer”, In J. Schewel, editor, Proc. of the International Society of Optical Engineering (SPIE). Field Programmable Gate Arrays (FPGAs) for Fast Board

Development and Reconfigurable Computing, Vol. 2607, pp. 92-103, Philadelphia, PA, Oct. 1995.

[17] J. Villasenor, B. Schoner, K. N. Chia, C. Zapata, H. J. Kim, C. Jones, S. Lansing and B. Mangione-Smith, "Configurable Computing Solutions for Automatic Target Recognitio"', In J. Arnold and K. J. Pocek, editors, *Proc. of IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 70-79, Napa, CA, April 1996.

[18] J. Villasenor and B. Hutchings, "The Flexibility of Configurable Computing", *IEEE Signal Processing Magazine*, Vol. 15, No. 5, pp. 67-84, Sept. 1998.

[19] J. Villasenor and W. H. Mangione-Smith, "Configurable Computing", *Scientific American*, Vol. 276, No. 6, pp. 66-71, June 1997.