Journal of VLSI Signal Processing 28, 63–83, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Application of Reconfigurable Computing to a High Performance Front-End Radar Signal Processor*

DAVID R. MARTINEZ, TYLER J. MOELLER AND KEN TEITELBAUM
MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420-9108

Received July 1999; Revised December 1999

Abstract. Many radar sensor systems demand high performance front-end signal processing. The high processing throughput is driven by the fast analog-to-digital conversion sampling rate, the large number of sensor channels, and stringent requirements on the filter design leading to a large number of filter taps. The computational demands range from tens to hundreds of billion operations per second (GOPS). Fortunately, this processing is very regular, highly parallel, and well suited to VLSI hardware. We recently fielded a system consisting of 100 GOPS designed using custom VLSI chips. The system can adapt to different filter coefficients as a function of changes in the transmitted radar pulse. Although the computation is performed on custom VLSI chips, there are important reasons to attempt to solve this problem using adaptive computing devices. As feature size shrinks and field programmable gate arrays become more capable, the same filtering operation will be feasible using reconfigurable electronics. In this paper we describe the hardware architecture of this high performance radar signal processor, technology trends in reconfigurable computing, and present an alternate implementation using emerging reconfigurable technologies. We investigate the suitability of a Xilinx Virtex chip (XCV1000) to this application. Results of simulating and implementing the application on the Xilinx chip is also discussed.

Keywords: VLSI rader signal processor, front end high performance filtering, reconfigurable hardware, digital filtering mapped to reconfigurable computing, commercial FPGA hardware

1. Introduction

The radar signal processing trend is to bring the analog-to-digital converter (ADC) closer to the radar antenna elements, and process the incoming signals digitally instead of using more conventional analog approaches. The digital hardware offers more robust system stability, avoiding unnecessary calibrations characteristic of analog hardware due to component drifts with time and temperature. The availability of digital hardware also provides very important benefits, such as, more flexibility in waveform design, an ability to reconfigure the hardware by downloading different system coefficients, and an easier upgrade path as digital electronics continues to advance at an exponential rate commensurate with Moore's law.

The benefits of digital hardware, in advanced radar signal processing systems, come at the expense of very potent front-end signal processors needs. The computational and data throughputs result from the large amount of incoming data processed over a very short time interval. The computational throughputs are in the tens to hundreds of billion operations per second (GOPS). The data throughput bandwidths are in the hundreds of megabytes per second (MBytes/sec). In addition to stringent computational and data throughputs, the digital hardware must be deployed in a small size, low weight, and low power. Furthermore, the hardware must also comply with demanding environmental,

*This work was sponsored by the Air Force under contract F19628-95-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

shock, vibration, humidity, and temperature specifications. Examples of platforms requiring these capabilities are today's airborne early warning radars, unmanned air vehicles (UAVs), fighters, and spaceborne surveillance and targeting radars.

For the past several years, commercially available digital signal processors (DSPs) or RISC microprocessors have not met the system requirements for cases where we need in excess of 10 GOPS/ft^3 and $\geq$0.5 GOPS/Watt, operating on at least 16-bit data. Fortunately, the front-end signal processing functions are very regular, well structured, and often times independent of the incoming data. Therefore, most implementations to date have used either commercially available, yet dedicated, compute engines (e.g., INMOS A100 or SHARP LH9124 FFT chip), or custom VLSI designs. These approaches have allowed us to meet the computational throughputs and data throughputs, within the form factor and environmental requirements.

In this paper, we describe a recently fielded front-end signal processor capable of 100 GOPS. Since this system is channel parallel, it can be scaled to many hundreds of GOPS as we increase the number of radar input channels. The system can also be reconfigured in a few milliseconds for a different set of coefficients. Although the system requirements led us to choose a custom VLSI implementation, we believe that this application is very well matched to reconfigurable computing hardware. At the time when the custom VLSI design started (circa 1996–1997), the commercially available field programmable gate arrays were not able to meet the system requirement by a large margin. However, the rapid growth in computational capability of more recent reconfigurable devices provides a new opportunity for meeting the requirements without depending on point solutions.

Reconfigurable computing will offer many of the same benefits provided by commercial general purpose microprocessors without sacrificing overall system performance. These devices will be cost effective, since the same hardware can be reused for many different applications. The economies of scale will result in a very cost-effective solution. Furthermore, for a given design, the implementation cycle is shorter than custom VLSI solutions. If the first design contains flaws, one can redo the hardware configuration in software. These changes would be very costly with custom VLSI. Finally, because of the large number of devices, reconfigurable hardware manufactures can rip the benefits of the latest reductions in lithography feature size.

We are not implying that all front-end radar signal processing can be solved using reconfigurable computing. There are still system specifications, for example, in spaceborne applications, where custom VLSI solutions are further ahead than reconfigurable hardware, particularly when we need solutions with throughputs per unit power $\geq$20 GOPS/W, data throughputs exceeding GBytes/sec, and compliant with radiation hardening requirements.

The example presented in this paper represents an advanced prototype that only a few years back would have been unacceptable to attempt to solve using field programmable gate arrays. This example serves as a measure of how far reconfigurable technologies have come in just three years. As discussed in Section 4, under technology trends, this rapid advance can be credited to several factors, such as, smaller lithography size, advancements in development tools, lower voltages, and algorithms that exploit the distributed memory architecture of field-programmable gate arrays (FPGAs).

The paper is divided as follows. In the next section, we describe the front-end radar digital filtering and the associated computational requirements. In Section 3, the custom VLSI approach is presented. Section 4 addresses the technology trends and drivers fueling the advances experienced with reconfigurable electronics compared to more general purpose computing. In Section 5, we review and contrast several architecture approaches to solve the front-end radar signal processing posed in Section 2. In Section 6, we discuss other future candidate applications for reconfigurable computing technologies. Finally, in Section 7 we present our conclusions and summary.

2. Front-End Digital Filtering

A typical radar signal processing flow starts at the output of the ADC with real-time data processed through a set of digital filters. The digital filters are needed to first convert the data from real ADC samples to complex digital in-phase and quadrature (DIQ) samples. This operation is commonly known in the radar community as DIQ sampling [1]. The DIQ sampling is necessary to preserve the target Doppler information. From the Doppler information one can discern the target velocity.

As illustrated in Fig. 1, the digital filtering is typically the first step in the signal processing chain. The remaining processing, Doppler filtering, adaptive nulling, and

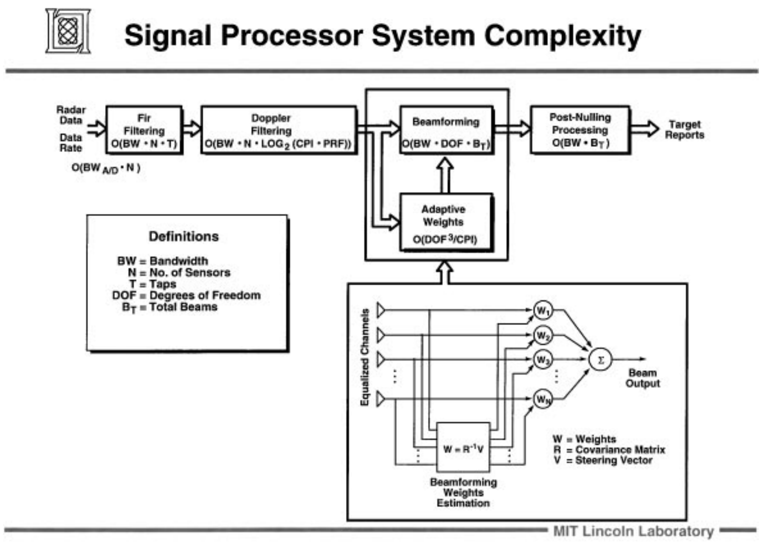


Figure 1. Typical radar signal processing flow.

post-nulling processing, can also be very demanding. However, in this paper we only focus in on DIQ sampling. J. Ward [2] presents a thorough exposition of the typical radar signal processing flow, with particular emphasis on space-time adaptive processing. DIQ sampling is proportional in computational operations to the data bandwidth of the system, the number of filter taps, and the number of channels, as shown in Fig. 1. In the next subsection we describe in more detail the filter specifications.

2.1. DIQ Sampling Architecture

The radar data after the ADC will exhibit a set of spectrum replicates with periodicity equal to the Nyquist sampling (1/2 the sampling rate). Because the data are real samples, at this stage the spectrum maintains equal images with respect to 0 Hz. The DIQ filtering will serve the purpose of extracting one of the sidebands, mapping the sideband spectrum to baseband, and filtering all remaining images including any DC offsets introduced by the ADC. The filter coefficients will vary depending on the characteristics of the bandwidth present in the transmitted radar pulse. The resulting output contains a single sideband spectrum replicated every increment of the sampling frequency. The single sideband represents the complex signal (in-phase and quadrature components) characteristic of a Hilbert transform performed in a demodulation operation [3].

The analog receiver filtering, prior to the ADC converter, is selected such that the information bandwidth is preserved without aliasing. The signal instantaneous bandwidth is typically several factors below the ADC sampling frequency. This oversampling leads to very simple DIQ architectures.

For example, if the ADC sampling is four times the instantaneous frequency of the radar [4], the mapping of the spectrum to baseband reduces to multiplication by $+1$, 0, and -1 for the in-phase component, and 0, $+1$, and -1 for the quadrature component. Once the data are mapped to baseband, the complex signal can be filtered and decimated to match the signal bandwidth. This simplification leads to the typical DIQ sampling architecture shown in Fig. 2. We use two filter banks of equal length, each operating on the in-phase and quadrature samples, respectively. The odd samples are used to form the in-phase component, and the even samples are used to form the quadrature components.

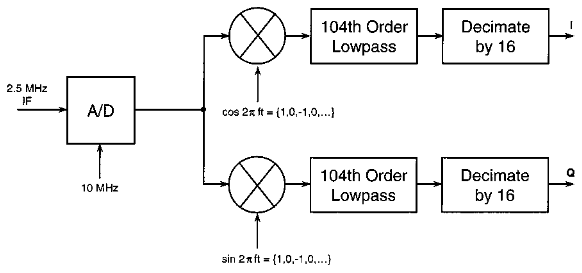


Figure 2. Digital in-phase and quadrature sampling architecture.

2.2. Digital Filter Specifications

The DIQ filter is designed to suppress all out of band images and spurious signals, including any DC offset from the ADC. The filter must maintain a minimum ripple across the passband with a sidelobe level sufficiently down to attenuate any unwanted data outside the signal bandwidth. The filter skirts determine the number of taps in the filter. For our application, we used the Parks-McClellan Remez algorithm [3] available in Matlab to design a finite impulse response filter with linear phase. The characteristics of the filter were as follows [5]:

- Equiripple passband $= 0.18$ dB
- Stopband level ≥ 80 dB
- Unattenuated passband $= 0.25$ MHz
- Bandwidth at 3 dB points $= 0.381$ MHz

The above specifications led to a filter with 208 complex taps (104 taps for the real and 104 taps for the imaginary components, respectively). In order to meet the filter stopband level the coefficients must have a finite precision of at least 15 bits.

The ADC used was a DATEL ADS945 sampling at 10 MHz, with a precision of 14 bits. After the digital filter, shown in Fig. 2, the radar data were decimated by a factor of 16 or, equivalently, a factor of 8 complex samples. Thus, the output sampling rate out of the DIQ filter was down to 0.625 MHz complex data. Although this output data rate was twice what was needed to maintain the signal information bandwidth, it simplified the following filtering stage past the DIQ sampling resulting in a simpler filter design with a smaller number of taps.

On a per channel basis, as shown in Fig. 1, the computational throughput is proportional to the data bandwidth and the number of filter taps. For the 208 complex filter taps and an input data rate of 10 MHz, the DIQ filter must perform at least 2.08 GOPS, based on the architecture shown in Fig. 2. The reduction in sampling rate by a factor of 8 could have allowed us to only process the samples needed at the output. However, the hardware designer [6] concluded that the custom VLSI implementation was more regular and easier to layout by processing all the incoming samples and decimating at the end. For a 48-channel system the total computation throughput was 100 GOPS.

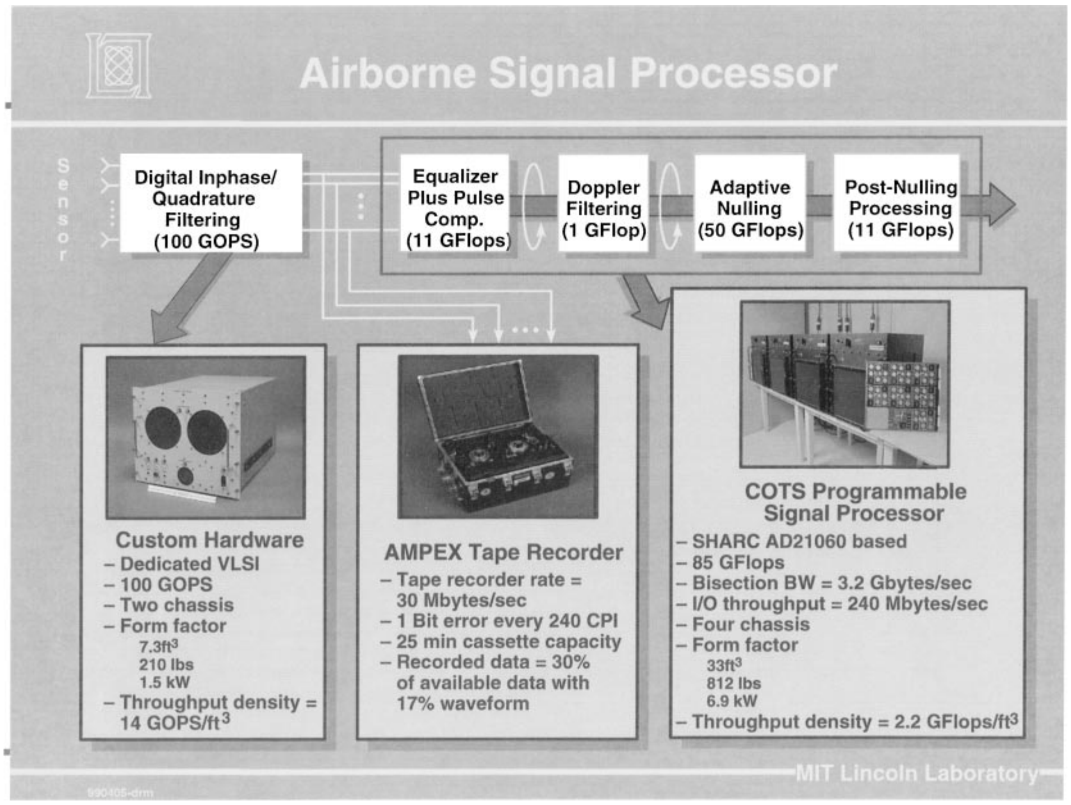
3. Custom VLSI Implementation

The front-end signal processor design begun in 1996. The system was completed in 1998. At the start of the project, a detailed search was done to determine if either general purpose hardware or commercially available dedicated chips could be used to solve the DIQ sampling requirements. We concluded that no DSP or RISC chips would be suitable to meet the processing requirements, particularly when you account for the loss in processing efficiencies caused by programmable hardware. There were several commercially specialized chips designed specifically to perform FIR filtering (e.g., Gray chip GC2011, Harris HSP43168, GEC Plessey PDSP16256). We concluded that none of these chips met the requirements in all the critical dimensions, such as, input data precision bits, coefficient precision bits, internal arithmetic accuracy, and computation throughput at low power.

The decision was made to proceed with the design of a custom VLSI chip based on a mix of standard cells and datapath multiply-accumulate (MAC) cells. The details on this design are further discussed in the latter sections. The final front-end signal processor was integrated into two 9U-VME chassis together with additional control electronics, a single-board computer (SBC), and interface boards to the follow on processing and an instrumentation recorder. The airborne signal processor subsystems are shown in Fig. 3. In this paper, we only discuss the Digital In-Phase and Quadrature Filtering Subsystem. The following sections elaborate in more detail on the hardware developed for this subsystem.

3.1. DIQ Hardware Subsystem

The DIQ hardware subsystem consisted of three unique board designs. There were two 9U-VME chassis; each chassis performed 50 GOPS. A 9U-VME chassis housed 10 boards. The internal architecture of this subsystem is shown in Fig. 4. One board slot was dedicated to a SBC control computer used to download radar coefficients to the processing boards and to initialize all the board registers. Another slot was dedicated to a distribution board used for subsystem control. This board interfaced to the radar control processor and provided control signals to each of the subsequent boards in the chassis. There were 6 DIQ processing boards. Each DIQ board handled 4 input channels. The outputs from 2 DIQ boards were then sent to a data


Airborne Signal Processor
Sensor
Digital Inphase/ Quadrature Filtering (100 GOPS)
Equalizer Plus Pulse Comp. (11 GFlops)
Doppler Filtering (1 GFlop)
Adaptive Nulling (50 GFlops)
Post-Nulling Processing (11 GFlops)
MIT Lincoln Laboratory
990405-drm

Custom Hardware
- Dedicated VLSI
- 100 GOPS
- Two chassis
- Form factor
  - 7.3ft^3
  - 210 lbs
  - 1.5 kW
- Throughput density = 14 GOPS/ft^3

AMPEX Tape Recorder
- Tape recorder rate = 30 Mbytes/sec
- 1 Bit error every 240 CPI
- 25 min cassette capacity
- Recorded data = 30% of available data with 17% waveform

COTS Programmable Signal Processor
- SHARC AD21060 based
- 85 GFlops
- Bisection BW = 3.2 Gbytes/sec
- I/O throughput = 240 Mbytes/sec
- Four chassis
- Form factor
  - 33ft^3
  - 812 lbs
  - 6.9 kW
- Throughput density = 2.2 GFlops/ft^3

Figure 3. Airborne signal processor subsystems.

output board (DOB board). Each DOB board packed the data and provided the same interface outputs to a back-end programmable processor and to an instrumentation recorder. These custom boards were based on a 14-layer PCB design of size 9U $\times$ 220 mm. The overall characteristics of this subsystem for two 9U-VME chassis were:

- 100 GOPS
- Size = 7.3 ft^3
- Weight = 210 lbs
- Power = 1.5 KW
- Chassis throughput/Power = 67 MOPS/W
- Throughput density = 14 GOPS/ft^3
- Power density = 205 W/ft^3

There were 24 channels input to each DIQ subsystem chassis. The total data rate to the chassis was 420 MBytes/sec. These data were then distributed to 6 DIQ processing boards over a custom backplane. Each DIQ board received 70 MBytes/sec of data. There were 4 custom VLSI chips per DIQ board; each chip processed one channel of data. The input data was mapped to 18-bit data on the DIQ board prior to going into the VLSI chip. The output of the chip was a 24-bit data word. A 16-bit word was then selected and sent over the custom backplane to the DOB board. Thus, the output of the DIQ board was reduced down to 10 MBytes/sec (4 channels, 16-bit complex words, at 0.625 MHz). The next section describes in more detail the custom VLSI hardware.

3.2. *Custom VLSI Hardware*

The VLSI custom front-end chip, designed using 1996 process technology, is shown in Fig. 5. The chip was fabricated at the National Semiconductor facility in Arlington, Texas, under the C050 process. The key features of this chip were:

- 2.08 billion operations per second
- 18-bit input data and coefficients; 24-bit output data
- 585 mil $\times$ 585 mil die size
- 1.5 million transistors

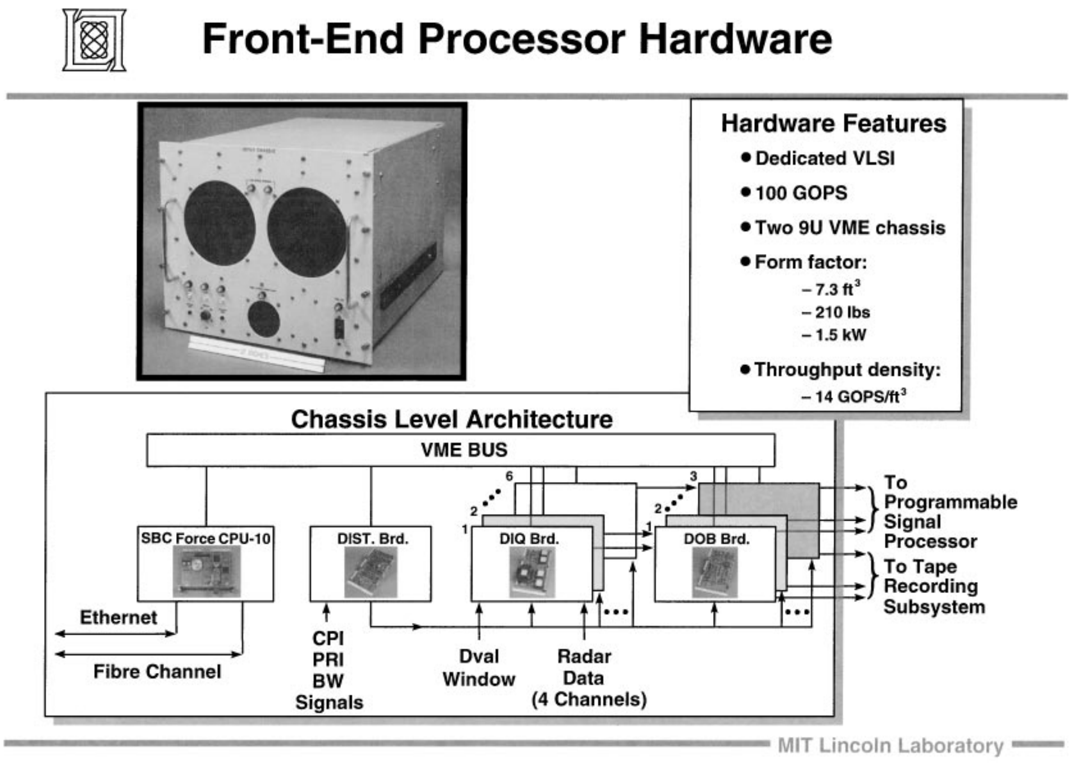


Figure 4. DIQ subsystem architecture.

– 0.65 μm feature size
– CMOS using three-layer metal
– Designed for 4 watts power dissipation; measured 3.2 watts in operation
– Throughput/Power = 0.65 GOPS/W

The designed was based on a combination of standard cells and datapath blocks [6]. The standard cells were used in the chip control interface, barrel bit selector, and downsampler. The datapath blocks formed the multiply and add (MAC) cells. There were a total of 64 MACs available on the chip. These MACs could be used either as 64 complex taps or as maximum 512 real taps. Since we were interested in computing the in-phase (I) and quadrature (Q) data from real ADC data, each MAC processed up to 128 real taps for the I samples and, in parallel, another up to 128 taps for the Q samples, at 10 MHz. We could also have used the chip to process at 5 MHz using 512 taps.

ADC samples arrived at 10 MHz. Every other sample went to the I computation and the Q computation, respectively (see Fig. 2). Thus, two sets of outputs were computed at 10 MHz (5 MHz for I and Q), each applying 104 real taps. Since there were two real operations per tap (a multiply and an add), the total computation throughput for these parameters was 2.08 GOPS. These operations were performed on 18-bit sign extended data (from 14-bit ADC samples) using 18-bit coefficients. The resulting significant 16-bit outputs (I and Q samples) were selected on the DIQ board from a 24-bit data output from the VLSI chip.

Figure 5 illustrates a die photograph, the packaged chip using a pin grid array (PGA) with 238 pins, and the results of processing simulated radar data. The performance results contain three plots. Two of the curves illustrate the output from the hardware superimposed over the output of Matlab, respectively. Matlab simulates the filtering operation using full precision arithmetic. The third plot shows the difference between the hardware output and the Matlab results. The difference between the hardware output and the Matlab full precision simulation is about 100 dB relative to the square waveform peak. These results are very good and quite commensurate with what one would expect using 18-bit finite precision arithmetic.

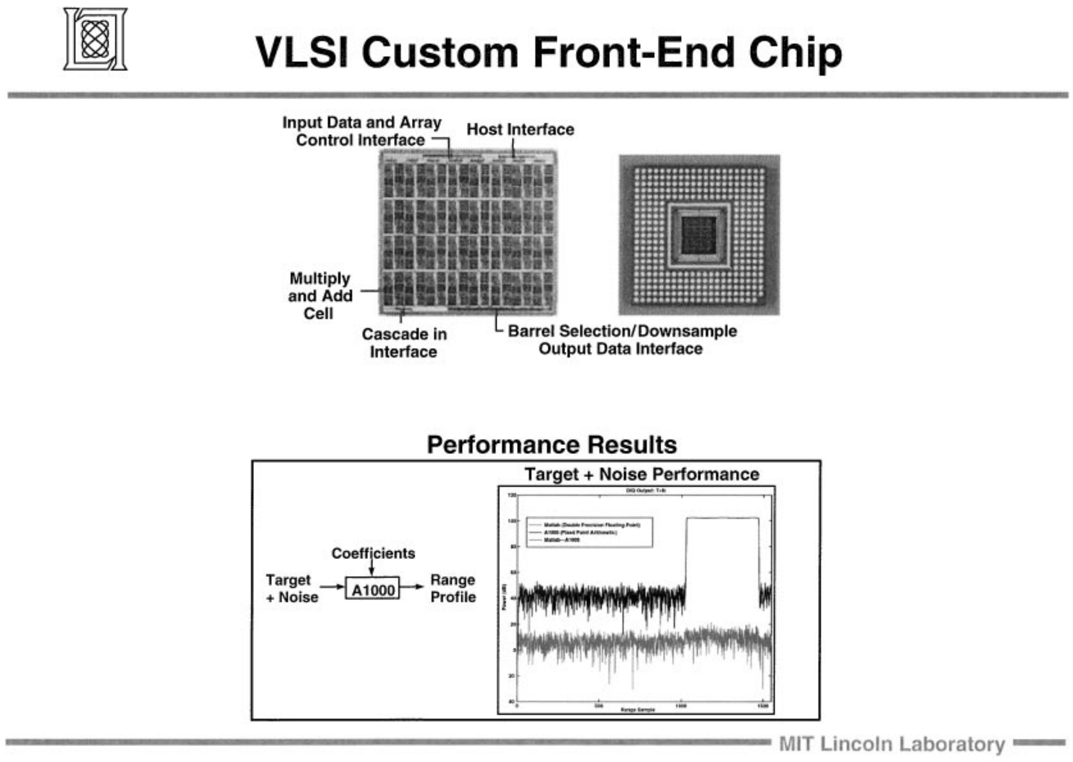


Figure 5. Custom VLSI front-end chip.

If we compare the characteristics of the DIQ subsystem, described in the previous section, against the characteristics of the custom VLSI chip, several observations can be made. The DIQ subsystem has a computation throughput per unit power of only 67 MOPS/W. The custom VLSI chip has a throughput/power of 0.65 GOPS/W. This decrease of one order of magnitude in throughput/power performance is typical of a hardware integrated as part of an overall subsystem, contrasted to a single chip performance. Additional power is needed for the control and interface boards, backplane drivers, and cooling. Another important observation is the lag between the time the VLSI chip was designed (circa 1996 fabrication process) to the time the system was deployed (circa 1998). This two-year lag was used to integrate the overall system. Therefore, the system was over one generation old in feature size process technology at deployment time. Furthermore, for fabrication orders consisting of few wafer lots (in circa 1996) the only available process technology was 0.65 μm. This feature size was sufficient for this one of a kind prototype technology demonstration. Larger wafer lots fabrication, often found in commercial chip designs, have access to more advanced fabrication process lines. These are areas that we expect to improve with the availability of more advanced reconfigurable computing technologies.

As FPGA technologies become more capable, we can now expect to perform the same filtering operations as previously delegated to custom VLSI devices. The FPGA implementations would have several major benefits:

1. Reduction in design and manufacturing costs relative to custom VLSI designs.
2. Flexibility to implement different filtering functions using the same FPGA devices.
3. Ability to upgrade the design to higher ADC sampling rates by substituting more capable FPGA devices commensurate with the Moore's law progression in silicon technology.
4. Wider vendor sources of FPGA technologies than available for custom VLSI designs.

The major limiting factor in the use of FPGAs to date, for these classes of applications, has been their inability to reach computation throughputs on the order of several GOPS, with 18-bit data and coefficient inputs, and a throughput per unit power greater than 0.5 GOPS/W. Most FPGA demonstrations to date have been limited to a few bits of precision. The custom VLSI implementation, described in this section, serves as an existing proof and benchmark of capabilities we would like to see in FPGAs. In the following sections we discuss the recent observed FPGA technology trends. We also present implementation techniques and results showing what is feasible to date with the most advanced FPGA devices. We show how close this hardware comes to meeting our FIR filtering requirements.

4. Technology Trends in Field Programmable Gate Arrays

The increasing sophistication of radar signal processing techniques has paralleled improvements in digital computing techniques. Computational requirements of future radar systems will approach 10 TOPS (trillion operations per second) during the next decade, placing it in the domain occupied by the leading edge of commercial massively parallel processors consisting of hundreds or thousands of individual CPUs. Unfortunately, the sheer bulk of these systems will preclude their use in most military radar applications where platform constraints place stringent requirements on size, weight and power. Reconfigurable computing has the potential to mitigate this problem by offloading computationally intensive tasks for execution onto FPGAs.

Compared to the digital computer industry, FPGAs are in their infancy. Xilinx, the leading manufacturer of FPGAs produced its first field programmable gate array 15 years ago [7]. The initial devices were primitive by today's standards, and FPGAs were mainly used for "glue logic." The programmable nature of the device made it possible for engineers to make circuit design changes without costly and time-consuming modifications to printed circuit boards, and the fledgling industry prospered. As technology progressed and the process geometry shrank, both the capacity and speed of FPGAs have increased. These improvements have enabled FPGAs to perform the arithmetic functions necessary for digital signal processing (DSP). The goal of reconfigurable computing is to design application-independent hardware based on FPGA technology, and download application-specific algorithms to handle a wide variety of potential applications. Several manufacturers currently offer such products with many FPGAs, external memory, and industry-standard interfaces.

4.1. Building Blocks

In simplest terms, FPGAs are nothing more than large arrays of look-up tables (LUTs) with a flexible interconnect which allows building complex circuits with multiple LUTs. Once the circuit is described (either with VHDL or with schematic capture), the chore of mapping of these circuits onto the sea of LUTs is accomplished with automated tools for synthesis, placement, and routing.

For the Xilinx devices, the basic building block is called a Configurable Logic Block (CLB) that consists of two 4-input LUTs, one 3-input LUT, and two registers. DSP functions consist principally of multiply and add operations that must be constructed of multiple CLBs. A b-bit adder needs b LUTs and b registers, and thus consumes $b/2$ CLBs. A $b \times b$ bit multiplier requires $2b$ b-bit adders consuming b^2 CLBs. A FIR filter tap (multiply-add, 2 real operations) consumes $(b^2 + b/2)$ CLBs. For 16-bit arithmetic, an adder would consume only 8 CLBs, but a multiplier would take 264 CLBs.

In order to promote a more industry-standard approach to counting FPGA capacities, Xilinx has proposed the notion of a Logic Cell [8] which consists of a 4-input LUT and a register. For comparison purposes, the Xilinx XC4000 family CLB is equivalent to 2.375 logic cells.

Since the parallel multiplier has been so expensive in terms of resources used (CLBs) compared to resources available on a single FPGA, several alternative strategies for implementing DSP functions have evolved. These strategies tend to exploit the LUT-based architecture of the FPGA.

When multiplying a data stream by a constant that is known a priori, a constant coefficient multiplier [9] may be used to look up the product of the data sample and coefficient. The data sample effectively becomes the address to the look-up table that stores the product of the coefficient and data sample for each possible value of the data sample. Also assuming a priori knowledge of multiplier constants, Serial Distributed Arithmetic (SDA) [10] rearranges the order of computations to facilitate table lookup of a 1-bit sum of products. Distributed arithmetic has also been used to implement Fast-Fourier Transforms (FFTs) [11].

These techniques work well for many digital-filtering applications where the filter coefficients (impulse response) are fixed. For adaptive filters where the coefficients are changed in response to the data, other approaches are necessary. It is possible to build SDA filters with two banks of filter taps that are bank swapped when filter coefficients are updated [12]. In the latter sections under implementation, this is the approach used for handling multiple filter coefficients in real-time. A table generator computes the LUT contents in real time while the new filter coefficients are downloaded. While this is obviously less resource efficient than the straightforward SDA approach, it is still an improvement over having to use parallel multipliers.

4.2. *Benchmarks*

In assessing the computational requirements of DSP algorithms, and the capabilities of DSP technologies, the metric commonly used is the number of arithmetic operations to be executed per unit time. This is typically expressed in MOPS (million operations per second) or GOPS (billion operations per second). For considering the performance of FPGAs, this metric is normalized by the "area" consumed by the circuit (the number of CLBs) to yield MOPS/CLB. If the multiply-adder can operate at a clock rate of f_C, then the number of operations per second executed per CLB is:

$$2^* f_C/(b^2 + b/2) \tag{1}$$

FPGAs are much more effective for small word lengths since computation rate per unit area (MOPS/CLB) falls off as the square of the word length in bits. This is clearly evident in the graph in Fig. 6, which is based on the Xilinx CORE benchmarks [13].

Figure 6 also illustrates the performance advantage achievable with SDA compared to parallel multiply-add architectures. The product of the MOPS/CLB metric and the device capacity (number of CLBs) give a notion of the device capability in aggregate. For example, if Xilinx's new Virtex device (XCV1000) which has 12,288 CLBs (equivalent CLBs with respect to Xilinx 4000 series) could achieve 1 MOPS/CLB, it would be able to perform about 12 GOPS per chip.

In many DSP applications, power consumption is heavily constrained. For these applications, MOPS/Watt is a convenient performance metric. Xilinx [14] has suggested a simple formula for estimating the

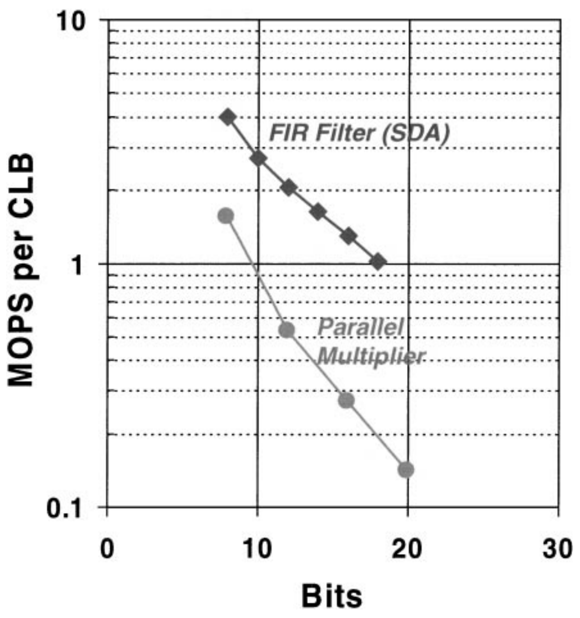


Figure 6. Xilinx core benchmarks.

power consumption of a device:

$$P = V_{cc}{}^* K_p{}^* f_C{}^* N_{LC}{}^* \mathrm{Tog}_{LC} \tag{2}$$

Where V_{cc} is the supply voltage, K_p is a device-specific power factor, f_C is the maximum clock frequency, N_{LC} is the number of logic cells and Tog_{LC} is the fraction of logic cells toggling on each clock (typically about 20%). Rearranging (1) and (2), and assuming 2.375 logic cells per CLB yields the following estimate for MOPS/Watt:

$$\mathrm{MOPS/Watt} = 4.2e^{-6}/[V_{cc}{}^* K_p{}^* (b^2 + b/2)] \tag{3}$$

Typical values for several generations of Xilinx parts are shown in Table 1 and assume 16-bit arithmetic. Power efficiencies for several 32-bit floating point programmable processors are shown in Table 2 for comparison.

Table 1. Power efficiency (MOPS/W) for several FPGA families.

Part number	V_{cc}	$K_p \times 10^{12}$	MOPS/W
XC4000E	5	72	43.0
XC4000EX	5	47	65.9
XC4000XL	3.3	28	167.5
XC4000XV	2.5	13	476.3
XCV (Virtex)	2.5	6.8	961.2

Table 2. Power efficiency (MFLOPS/W) for several floating-point programmable microprocessors.

Processor	Peak MFLOPS	Power (W) (Typical)	MFLOPS/W
DEC Alpha 21164	1200	46	26
INTEL Pentium III	1000	28	35.7
Motorola Power PC 750	800	8	138
Analog Devices 21062 (SHARC)	120	2	60
Analog Devices 21160	600	2	300
Texas Instruments TMS320C6701	1000	2.8	357

4.3. *Technology Trends*

Shrinking process geometry has fueled the continued growth in capability for FPGAs in DSP applications by increasing both density and clock rate. Supply voltage has dropped to reduce power consumption. The net result has been enormous growth in terms of computational capability (MOPS) and power efficiency (MOPS/W). The growth in computational capability versus process geometry is shown in Fig. 7. Curves are shown for FPGAs, the Motorola Power PC [15,16], and custom VLSI for comparison purposes. The data for custom VLSI is based on several development efforts at MIT Lincoln Laboratory [17] over the past several years.

It is clear from Fig. 7 that FPGAs offer an intermediate capability between that offered by programmable processors and custom VLSI. At present FPGAs are about an order of magnitude more capable than the programmable processors, and an order of magnitude less capable than full custom VLSI. Furthermore, the gap between FPGAs and microprocessors is widening. This is easily explained. FPGAs benefit directly from the increase in density and clock rate. Using an FIR filter as an example, successively finer geometries result in increases both in the number of filter taps which can be implemented in parallel, and in the sample rate at which they process data. The Power PC, on the other hand, has maintained two floating-point pipelines despite the shrinkage of process geometry, relying on increases in clock frequency to improve peak MFLOPS. The Power PC has taken advantage of the increases in density to increase memory caches and shrink die sizes.

The growth in the computational capability of FPGAs for DSP applications as a function of time is shown in Fig. 8 in billions of 16-bit arithmetic operations per second (GOPS). Within the last 5 years the computational capability of FPGAs have increased by

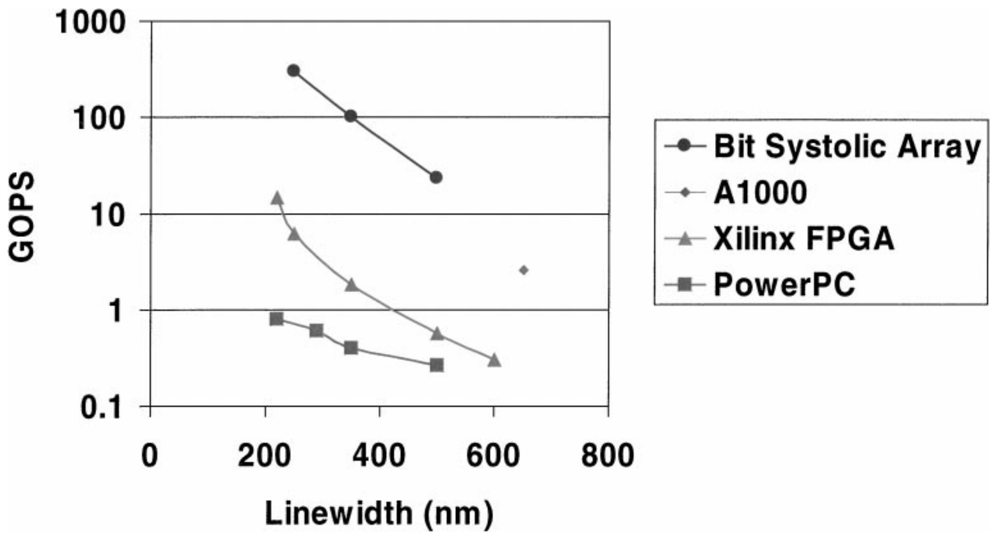


Figure 7. Growth in FPGA computation rate versus process geometry.

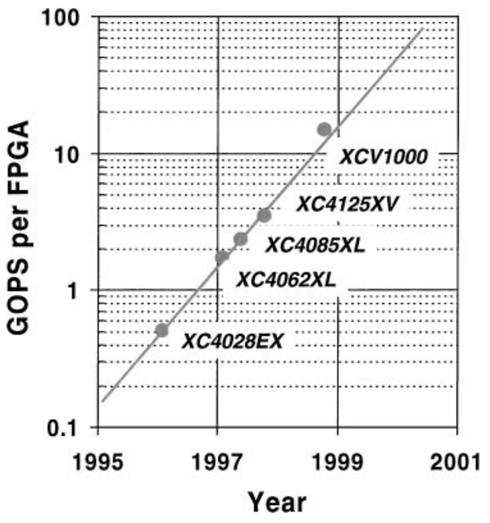


Figure 8. Growth of computational capability for Xilinx FPGAs (16-bit arithmetic operations).

an order of magnitude every two years, and we have reached the point where it is now feasible to explore the implementation of complex DSP systems using FPGAs as the computational building blocks. In the next sections we present implementation approaches for the FIR digital filtering function shown in Fig. 1 using the Xilinx Virtex XCV1000 as an example of state-of-the-art FPGAs.

5. Reconfigurable Hardware for FIR Digital Filtering

To demonstrate the current state of FPGA technology and its application to front-end signal processing, an FPGA design meeting the design requirements of MIT Lincoln Laboratory's custom VLSI FIR chip was created [19]. Although the actual VLSI chip was capable of processing, in the DIQ mode, 256 taps (I and Q components) for input data at 10 MHz, the design could accommodate up to 512-tap real filter with 5 MHz input data. Therefore, the FPGA demonstration was presented with the requirement to also accommodate up to 512 taps at a maximum input data rate of 5 MHz. This requirement is equivalent to 10 MHz input data with a maximum of 256 taps. The FPGA requirements were:

- Perform 512-tap real FIR filtering.
- Accept a minimum of 16-bit data at a maximum input rate of 5 MHz.
- Operate with a maximum chip clock frequency of 40 MHz (eight times the input data rate).
- Output data with the same precision as the custom VLSI chip (the VLSI chip outputs 24-bits of data, but simulations showed 18-bits of accurate precision was sufficient).
- Use two swappable banks of 18-bit coefficients, one active and one loadable.
- Fit into the largest Xilinx FPGA available, the Virtex XCV1000.
- Coefficient banks must be able to be switched every 1 ms (i.e. reloading all 512 coefficients must take place in less than 1 ms), and the new bank of coefficients must become active instantaneously.

The 16-bit inputs were chosen for the FPGA implementation as the custom VLSI chip is currently being used with an ADC with a 14-bit output, and it is unlikely the VLSI chip would operate with an ADC of more than 16-bits in the future. The only reason for having the custom VLSI chip designed to accommodate 18-bit input data was to facilitate word growth if several custom A1000 chips were used in a cascaded mode. This is not a requirement imposed on the FPGA implementation, because for the DIQ filtering function there was no need to cascade multiple chips.

Two banks of coefficients were used in the custom VLSI chip. This capability is also required for the FPGA; so that one bank stores the active filter coefficients while the other bank is being loaded with new coefficients. The banks may then be swapped by an external control so that the new coefficients may become instantly active.

Although the need to change coefficients could be viewed as an ideal application for reconfiguration, using swappable coefficient banks is more efficient. It has been proposed that the ability of an FPGA to be reconfigured in-system be used to implement a single filter with fixed coefficients that is reconfigured when a coefficient switch is desired. The design requirements specify an instantaneous switch between the active coefficients and the new set of loaded coefficients. However, this prohibits using a single bank of coefficients that could be reconfigured via the FPGA's reconfiguration ability. The filter would be inactive during the reconfiguration time, which is unacceptable. This slow reconfiguration is one of the important limitations with today's reconfigurable technologies.

One solution might be to have one filter operational while a second filter was being configured in the same chip. This would require the FPGA to have partial reconfiguration ability. In addition, the filter coefficients would have to be known in advance so that configurations using constant coefficients could be mapped, placed, and routed by the Xilinx software to be ready for loading into the FPGA. These configurations would then have to be stored off-chip to be recalled and loaded into the FPGA. In most applications, the filter responses are not known beforehand, so creating and storing every configuration for every set of possible coefficients is not feasible.

5.1. *Implementation Techniques on Reconfigurable Hardware*

Several different implementation techniques were investigated to find a design with the best performance (or at least satisfying the design requirements) with the minimum amount of FPGA utilization. These techniques included: a parallel MAC structure (much like that used in the custom A1000 VLSI design), a bit-level systolic structure (similar to that used in another MIT Lincoln Laboratory custom VLSI chip), and a bit-serial approach using Distributed Arithmetic (DA). The technique that was discovered to have the best performance for the smallest area was the bit-serial DA approach [19].

In addition to the DA approach, we also investigated the fast FIR algorithm (FFA) and filtering in the frequency domain. Although these are techniques that could be used in a VLSI solution, they would not lend themselves as easily to a multi-mode chip. The custom VLSI A1000 chip was designed to be able to perform real, complex, or DIQ digital filtering with the same hardware. The FPGA configuration does not need to support multiple modes. Instead, a specific implementation can be developed for each mode, and the correct FPGA configuration can be loaded into the reconfigurable hardware for the mode desired.

The next section describes the implementation of DIQ filtering using the DA approach [18]. Additionally, a custom layout tool is discussed. This tool, developed at MIT Lincoln Laboratory, aids in the automatic placement of regular and pipelined FPGA VHDL designs [19]. Currently, there is no way, through the Xilinx development flow, to describe how synthesized components from a design written in VHDL should be laid out within a FPGA. A designer may develop a highly pipelined design that requires only short net lengths between logic and pipeline stages. However, by breaking the design into small, regular, systolic cells, the development tools have no knowledge of the fact that a single cell needs to be laid out only once and that cells that communicate with each other should be placed next to each other. Presently, the tools place every single synthesized component individually, resulting in inefficient designs with long net lengths that lead to low clock performance results. The MIT Lincoln Laboratory custom tool will read, through the VHDL hierarchy, the user-defined placement constraints embedded within the VHDL for low-level, regular structures. The tool will also develop an overall placement scheme that will keep the same layout for identical repeated systolic cells, and will place connecting cells adjacent to each other. The DA implementation and the FPGA complexity are described in the following section.

5.2. *DIQ Filtering Implementation Based on Distributed Arithmetic*

The abundance of small distributed RAM blocks throughout the Xilinx FPGA chip enables the user to pre-calculate partial products, and to load these into the distributed RAM, thereby eliminating the large amounts of logic needed to compute multiplication results in a non-distributed approach. A Distributed Arithmetic (DA) architecture is a versatile approach to using this distributed RAM [10, 20–24].

The 16×1 RAM units within the Xilinx CLBs are good candidates for this DA scheme. One bit of a single 4-input LUT can fit into one of these units with no unused logic. For FIR filters larger than 4-taps, the filter can be broken into four tap groups. For example, a 16-tap FIR is shown in Fig. 9. To eliminate overflow, each adder stage must grow by one bit, and the scaling accumulator must also grow accordingly in size (of course, the scaling accumulator could drop the lower bits in its accumulation if less precision is required).

The shift registers required by each four-tap group can be implemented in the Xilinx Virtex distributed 16×1 RAM for 16-bit (or less) inputs. Each 16×1 RAM block may be configured to act as (up to) a 16-bit shift register. The length of the register is specified by the value written into the RAM block's 4-bit address. Each clock cycle a bit is shifted into the front of the register from the RAM's data input, and a bit is shifted out of the end of the register to the RAM's data output. With this technique, a single CLB can hold the entire 16-bit input for two taps. This is much more

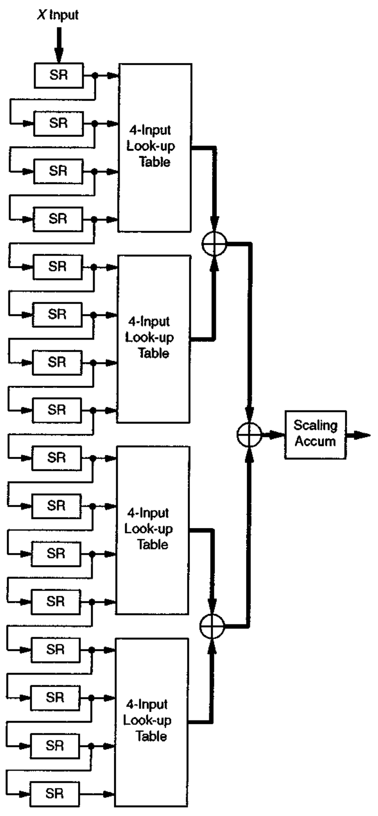


Figure 9. 16-Tap serial distributed FIR.

efficient than using registers to form each tap's shift register.

Figure 9 could be expanded until a 512-tap filter has been constructed, but two problems would exist. First of all, the coefficients are not constant as required by the design. Secondly, the filter above requires B clock cycles to process a single input sample. For a 5 MHz input, this means a filter with a 16-bit input must run at 80 MHz, which is twice as fast as the design specifications allow.

To solve the first problem, two banks of LUTs are used for each four-tap group. One bank is used as the active LUT. This LUT is addressed by the shift-register outputs. Then, another bank is loadable by the user. Therefore, the user can load all of the LUT values into one bank of the FIR filter in the background while the old LUT values stored in the other bank are active. By toggling a bank select line, the two banks are switched so that the previously loadable bank is now active, and the previously active bank is now loadable.

Figure 10 shows the complete block diagram for a single four-tap group including the shift registers and the double-banking LUTs as described above. The four-tap group accepts a serial data input (from the last tap's shift register of the previous group), and produces a serial data output for the next group's first tap's shift register. A bank selection line selects (through multiplexors) which bank of LUTs are active, and which are used for loading new coefficients. Each bank is 20-bits long as described above. The active bank is addressed by the four shift register outputs. The bank's output is the group's output. The load bank is addressed by an external set of coefficient address lines that select which of the 16 bank addresses is being written. A group coefficient load enable line selects whether this group is to have its coefficients updated versus another group, and a bank write clock line writes the data into the correct bank (the bank being loaded). A separate clock was used for each bank's LUT write clock to minimize the amount of logic local to a group.

Pipelining has been inserted so that the cumulative delay between pipeline registers has been kept to a minimum to increase performance. In addition, the bank selection line has been pipelined between 4-tap groups. This prevents a single bank selection line from having to drive all the mutiplexers in every 4-tap group, which would lead to a very high fan-out, and a very slow signal, decreasing the overall system performance. One drawback to this is that changing coefficient banks will take 128 clock cycles during which the new bank selection switch is propagated through its pipelining registers. Any outputs produced during that time will consist of outputs from both coefficient banks, and will be incorrect outputs from either bank's filter. This drawback was resolved using a linear-network summer tree [19]. Another implementation constraint is that coefficients in a given 4-tap's stand-by registers cannot be altered after a bank selection switch until the bank selection signal has propagated to that 4-tap. Otherwise, the wrong set of coefficients will be changed, since that tap has the wrong bank selected. This constraint was easily met since the entire bank change takes only 128 clock cycles, after which any 4-tap group's coefficients can be changed. A single four-tap group requires 36 CLBs.

A 512-tap filter using an expanded version of Fig. 9 with the four-tap group in Fig. 10 has been built. This filter was designed with full-precision, meaning that every adder stage grew by one bit (so no rounding was required), and the scaling accumulator was large enough to hold the entire 43-bit result. This 512-tap filter required 6,203 CLBs.

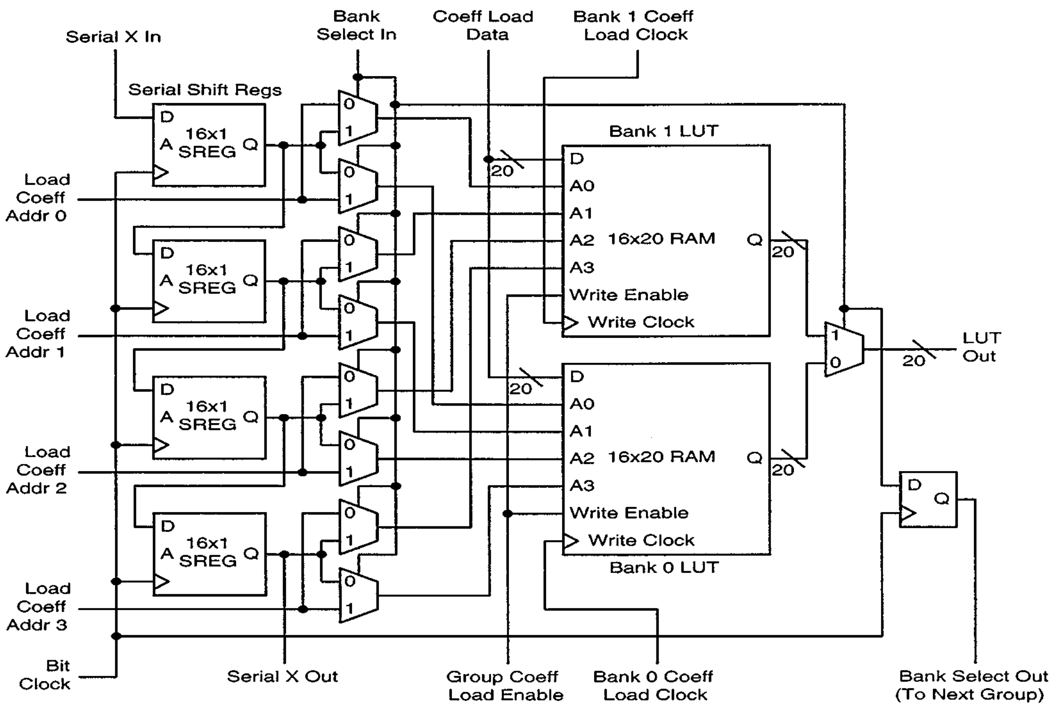


Figure 10. Block diagram of four-tap serial DA group.

To verify the functionality of the design, a custom 512-tap FIR filter was constructed in C++ that could output full-precision fixed-point two's-complement results.[1] The output of this simulation was compared, bit-for-bit, with the output of the simulated FPGA design. No errors were found for random sets of input data and coefficients.

5.3. *Xilinx Virtex XCV 1000 Performance and Area Results*

The serial distributed arithmetic design, as described above, requires a clock rate 16 times faster than the input data rate. For a 5 MHz data rate, this means the serial filters must run at 80 MHz. The design specifications require a clock 8 times faster than the input rate, or 40 MHz to be compatible with the DIQ subsystem shown in Fig. 4. Two solutions were investigated to address this requirement. The first approach operated the internal SDA filter at 80 MHz, using the Virtex Digital Locked Loop (DLL) to multiply the external 40 MHz clock up to a 80 MHz internal clock rate. The second technique placed two 512-tap SDA filters on the chip and had each filter, in parallel, operate on a single bit of the input data at a time. Two bits of the input data are processed per clock (this technique is called 2-bit parallel distributed arithmetic, or 2-bit PDA). An eight times faster clock rate would be required for this design, and the internal filters would only have to operate at 40 MHz, yet the design would require twice as much area as a SDA design.

The tradeoff between the two techniques was speed versus area. The DLL design required the internal filter to work twice as fast as the 2-bit PDA design, whereas the 2-bit PDA design required twice as much area. Unfortunately, the 2-bit PDA design required 12,756 CLBs for the 512-tap linear-network filter alone (without the scaling accumulator and additional glue logic), which was larger than what was available in the Virtex XCV1000. For this reason, the DLL SDA design was chosen.

A 512-tap linear-network SDA design using a Virtex DLL to double the external clock rate required 6,446 CLBs slices with the scaling accumulator and glue logic added to the design. The Xilinx timing analyzer software predicted a maximum worst-case clock rate of 85.9 MHz with the −6 speed grade. In the Xilinx 4000XL series of FPGAs (the Virtex was based on the 4000 design), real performance results were often noted to exceed those of the timing analyzer software.

It is very likely that the filter design here would meet the 80 MHz clock rate for 5 MHz input rate design requirements.

The Xilinx place and route routines did not optimally place the FIR filter. The optimized design that was chosen as the final design consisted of identical 4-tap groups connected in a linear fashion. The only connections to a given group were to the two adjacent groups. Only the clock signals were routed to all the groups, which could be handled by the Virtex's global clock interconnect. The design had also been highly pipelined so that placing adjacent groups next to each other on the FPGA would result in very short route lengths.

The place and route tools would only need to place a 4-tap group's internal components once, then replicate that group 128 times. However, the tools did not have the knowledge that the design was made this way, so all the various components of the design were placed in a haphazard fashion, even with timing constraints applied to the design. As a result, many of the wire lengths were longer than necessary, decreasing the filter's performance. A short description of the MIT Lincoln Laboratory placement tool is described in the next section (for more details refer to [19]).

5.4. *MIT Lincoln Laboratory Custom VHDL Placement Tool*

One of the benefits of using VHDL to describe a design is the ability to create parameterized modules that can be instantiated in higher levels of the design hierarchy. For example, a delay line could be created in VHDL with a variable number of bits and a variable number of delay stages. When this delay line module is instantiated, the exact sizing of a particular instance is specified at compile time in the VHDL as part of the instantiation code.

The problem with using VHDL for FPGA designs is that there is no way presently (at least with the Xilinx Foundation software) to communicate the desired layout of a VHDL design to the placement software. In the designs discussed above, a single small cell was often replicated many times in a systolic fashion. For example, in the DA approach, the 4-tap group was replicated 128 times. Each group only connected to the two adjacent groups, creating a linear systolic network. Because of this, care and time could be taken to place the components that make up one group, and then this placement could be replicated to all the instantiations of the 4-tap group. Ensuring that consecutive groups were placed adjacent to each other would result in the most efficient design.

Unfortunately, VHDL has no method of describing this process, and the placement tools do not have the knowledge that the design was created in a systolic fashion. As a result, the components from different 4-tap groups and the rest of the logic were interspersed among each other in a seemingly haphazard fashion. Many signals that should have been short if the systolic approach was taken ended up long, as components that should have been near each other were far apart on the chip. Constraining the design placement would drastically improve its performance.

Although some tools (for example, Synopsys) allow attributes (such as placement constraints) to be passed from VHDL to the synthesized netlist for instantiated components, this feature is very limited. Components created as part of a generate statement (for example, a bank of registers x-bits long could be created by using a generate statement to duplicate a single register x times) are created during the synthesis process, and cannot have attributes attached to them. In addition, writing a long string of attribute statements can be tedious and error-prone.

A custom tool referred to as Cell_Maker [19] was developed to read in VHDL code, extract basic placement constraints added as comments by the user, and create a user constraints file (UCF) describing the placement constraints for every component in the VHDL code. This tool allows a cell that is replicated many times to be placed once, and the replication strategy (e.g. linear systolic) to be specified in order to create the best placement.

The only limitation is that instantiated library components from the Xilinx FPGA library (e.g. registers, LUTs, RAMs, adder primitives etc) must be used within the VHDL code, instead of high-level synthesizable code (for example, using the $+$ operator for addition). The reason for this limitation is that the tool cannot infer the components that would be generated by high-level code. All the components must be explicitly instantiated so that placement constraints can be attached to them. With better placement strategies, it is expected that the linear systolic network DA design performance could improve significantly. In the following section we illustrate the layout improvements achieved using Cell_Maker.

5.4.1. Placement and Routing Using Cell_Maker Custom Tool. Using Xilinx placement and routing

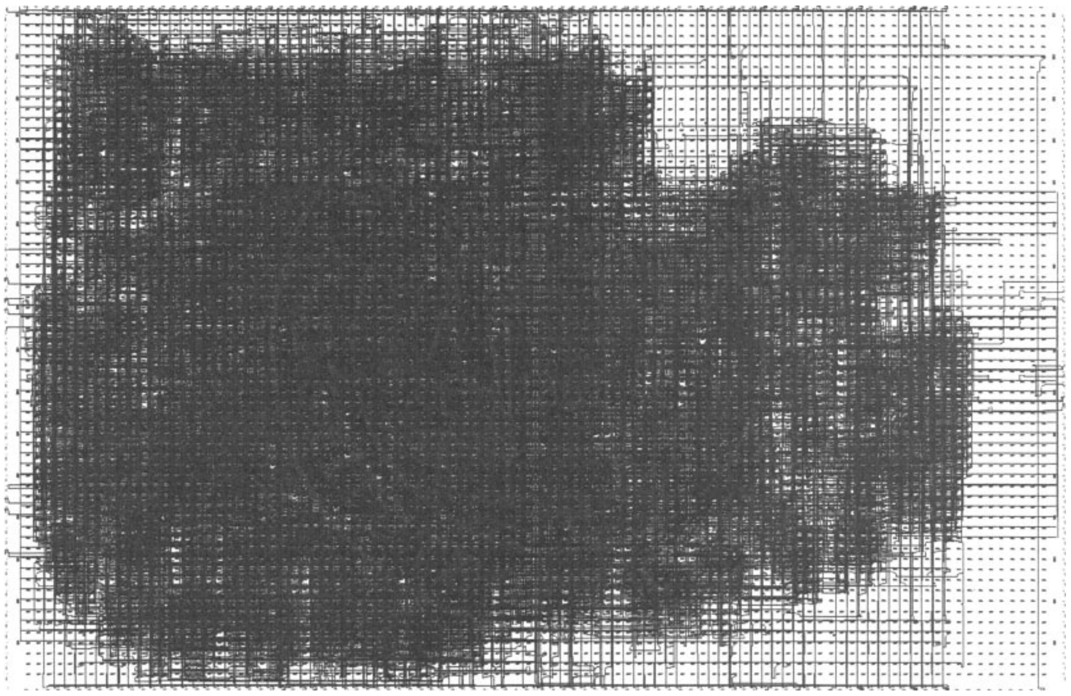


Figure 11. Placed and routed linear systolic DA design without Cell_Maker.

tools, the linear systolic design was not placed well within the Xilinx Virtex XCV1000, resulting in long route lengths that decreased performance. The placed and routed version of this systolic design layout, using this approach, is shown in Fig. 11. This design had the best performance out of twenty different placements run by the Xilinx tools via their multi-pass place and route feature. As the figure shows, the placement strategy did not take advantage of any of the systolic, regular, design features built into the linear DA design.

The linear systolic DA design was then constrained with the Cell_Maker constraints so that the four tap groups were placed in a linear chain as shown in Fig. 12. A parallel-to-serial shift register at the input of the first four-tap group changes the 16-bit X input into serial data for the DA algorithm, and a scaling accumulator at the output of the last four-tap group produced full 43-bit outputs.

The final placed and routed design with Cell_Maker constraints is shown in Fig. 13. The performance increased from 86 MHz to 118 MHz with the improved systolic layout; a 37% improvement due to layout alone. It is also evident that the overall layout is much more regular and consistent with a systolic short length interconnects.

The linear systolic SDA design constrained with Cell_Maker, as shown in Fig. 13, had a maximum clock rate of 117.5 MHz, which allowed a maximum sample rate of 7.3 MHz. With a sample rate of 7.3 MHz, and 512 multiply and add operations (two operations) performed each clock cycle, the linear systolic DA design could perform 7.475 GOPS. This throughput represents about a 49% increase over the older custom VLSI design described in Section 3 (with a maximum capability of 5.02 GOPS).

For the linear systolic DA design, about 6,700 flip-flops and shift registers were clocked; so the power consumption for the design was estimated at 12.14 watts with a 117 MHz clock [19]. With 7.475 billion operations performed per second at 12.14 watts, the throughput/power factor for the linear systolic DA design was 0.62 GOPS/Watt. In comparison, the custom VLSI chip throughput/power, for its 512-tap real mode, was estimated at 1.57 GOPS/Watt, or 2.5 times better than the FPGA linear systolic DA design.

One important lesson learned from this implementation experience is that generic tools can compromise the mapping of the desired architecture structure to the available FPGA hardware. For best utilization of the FPGA resources, with the goal of achieving highest

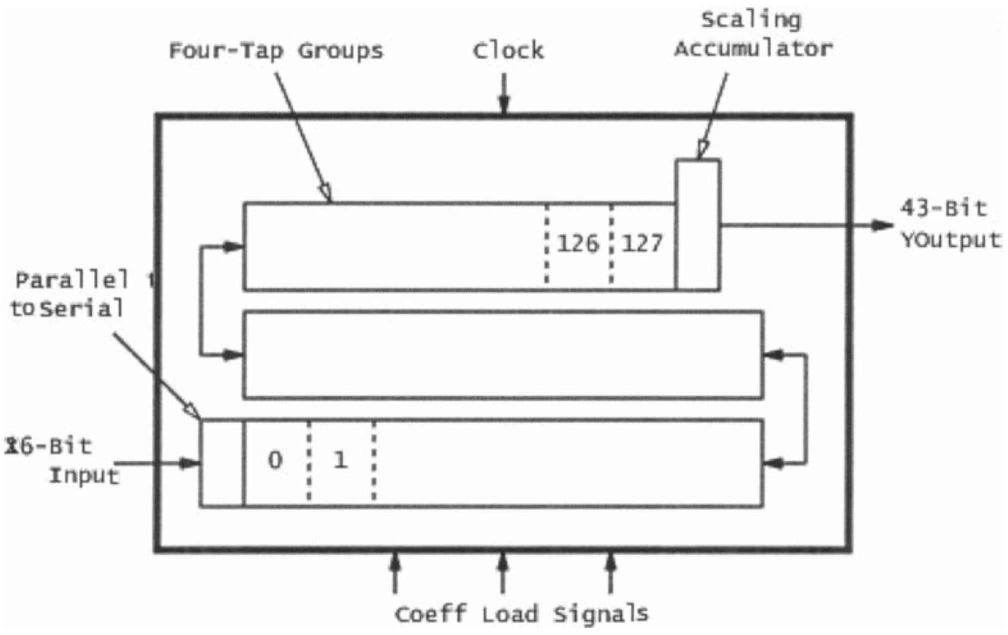


Figure 12. Linear systolic DA Cell_Maker layout.

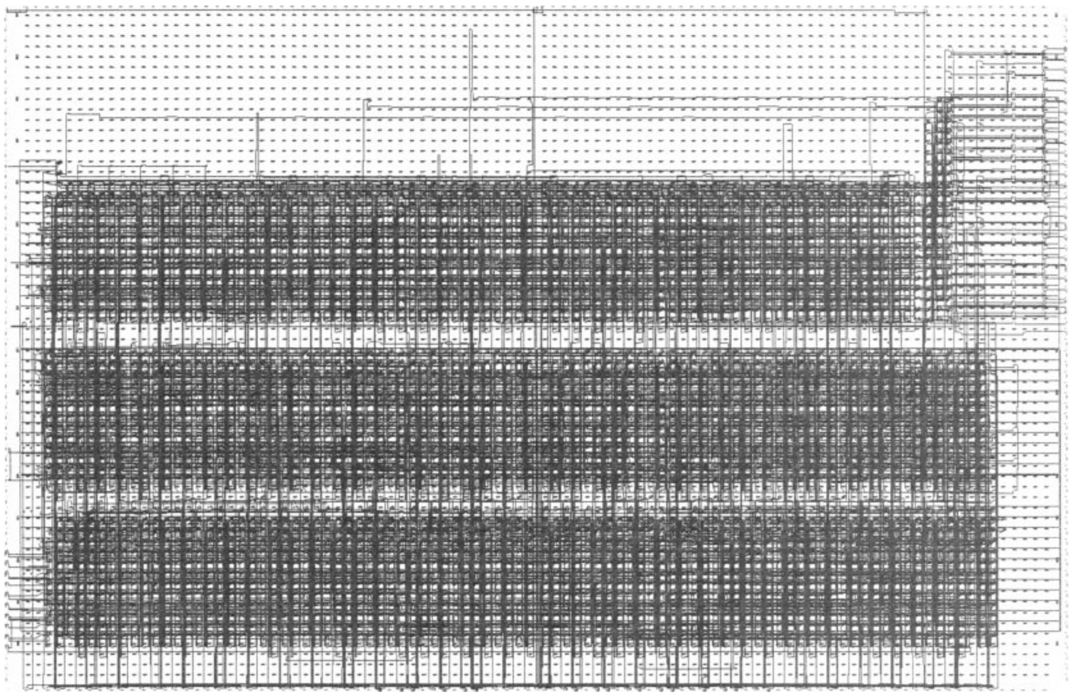


Figure 13. Linear systolic DA final placement and routing after Cell_Maker.

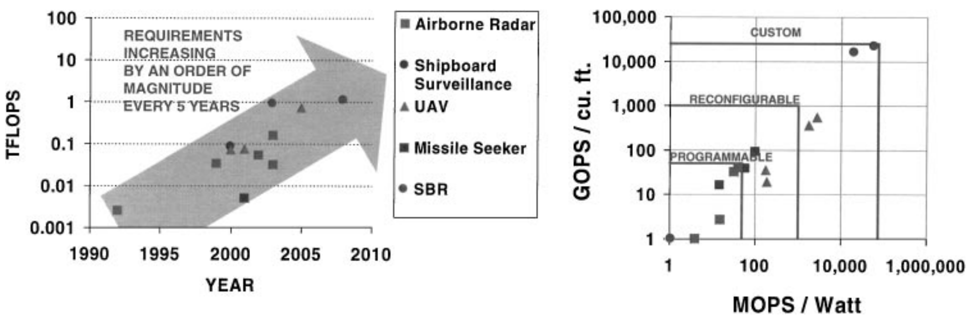


Figure 14. Growth of embedded radar signal processing throughput.

performance over minimum area, the placement and routing tools should exploit architecture specific constraints.

6. Future Candidate Applications for Reconfigurable Computing Technologies

In an effort to explore possible application areas for FPGA computing, computational requirements for various military signal-processing applications were surveyed. These applications included shipboard radar, airborne radar, radars for unmanned air vehicles (UAVs), missile seekers, and space-based radar (SBR). For each application the processing throughput in teraflops (TFLOPS $= 10^{12}$ operations/sec.) was determined, along with an estimate of when (approximate year) the technology was required, and what constraints (volume, power) were imposed by the platform. The results are shown graphically in Fig. 14.

The bulk of near-term requirements fall in the 50-500 GFLOPS range (sustained). There is a general trend which shows computational requirements increasing approximately an order of magnitude every five years. At this pace computational requirements will exceed 10 TFLOPS in the 2005–2010 time frame.

The ratios GOPS/ft^3 and MOPS/Watt express the platform constraints of volume and power, respectively. For the various application areas, these requirements span almost six orders of magnitude. These requirements will dictate the technology required for implementation. Several technologies were surveyed, based on current and previous experience at MIT Lincoln Laboratory. Programmable processors (SHARC, PowerPC, DEC Alpha, etc.) are capable of up to approximately 50 GOPS/ft^3 (PEAK) and 50 MOPS/Watt, assuming they are packaged for embedded use. Typical efficiencies run about 25%, so these numbers should be scaled accordingly. Reconfigurable computing based on FPGAs should be able to reach 1000 GOPS/ft^3 and 1000 MOPS/Watt. This projection assumes embedded packaging with MCMs (Multi-Chip Modules). Efficiencies are likely to be substantially higher than with programmable processors, but less than unity. Full custom VLSI with embedded packaging is the most capable technology, reaching out to approximately 20,000 GOPS/ft^3 and 75,000 MOPS/Watt.

The computational requirements of many of the applications surveyed fall outside of the capabilities of programmable processors. While the capabilities of such processors are increasing with each passing year, so are the computational requirements. There will always be applications for which the platform constraints push signal processing requirements into the realm of reconfigurable computing and custom VLSI hardware. Furthermore, the bulk (60%–90%) of the processing for these applications are the "easy ops" of FIR filtering, FFTs, and beam forming (exclusive of weight computation) and are well suited to hardware implementation either in FPGAs or custom silicon.

7. Conclusion

Historically, front-end signal processors for radar systems have demanded computational throughputs well beyond any contemporary microprocessor. Therefore, system designers have required to develop custom VLSI devices to meet these throughput demands. Recently there has been a major evolution in reconfigurable computing making viable solving these front-end signal processing problems with commercial-off-the-shelf FPGAs.

As FPGA technologies become more capable, we can now expect to perform the same digital filtering operations previously delegated to custom VLSI designs.

The FPGA technology offers several important benefits to these military applications, such as:

– Reduction in design and manufacturing costs relative to custom VLSI designs.
– Flexibility to implement different filtering functions using the same FPGA devices.
– Ability to upgrade the design to higher ADC sampling rates by substituting more capable FPGA devices commensurate with the Moore's law progression in silicon technology.
– Wider vendor sources of FPGA technologies than available for custom VLSI designs.

In the past several years, FPGAs have been successfully used for either microprocessor emulation or digital control functions. For digital control functions, the FPGAs have replaced many discrete integrated circuits, simplifying the complexity and providing more flexibility. People have also demonstrated the application of FPGAs signal processing for cases where the arithmetic operations only required a few bits of precision.

In this paper, we presented a recently fielded high-performance signal processor built to provide an aggregate throughput of 100 GOPS based on a custom VLSI design, with an input precision of 18 bits and an output precision of 24 bits. We used this example as an existence proof. It also provided the design goals needed to be demonstrated with FPGA hardware. Only a few years back, it would have been unacceptable to attempt to solve this problem with FPGAs. This limitation was due to very anemic number of transistors on a die, the slow clock frequency, excessive downloading times, inability to operate on a large data word, and poor placement and routing tools that could efficiently use the limited hardware available. These areas have advanced exponentially in the past years. In this paper, we have demonstrated that the same front-end digital filtering requirements could be met with the latest Xilinx Virtex XCV 1000 device, without sacrificing system performance.

The implementation of the digital in-phase and quadrature filtering using the custom VLSI design was based on a parallel group of multiply-add cells. This approach was not feasible in a reconfigurable hardware. Instead we opted to implement the digital filtering function using a serial distributed arithmetic approach. This technique, previously published in the literature, offered maximum utilization of the RAM based FPGA maintaining the throughput performance requirement.

However, we had to assist the Xilinx placement tools by creating a user constraint file (UCF) describing the placement constraints for every component in the VHDL code. This tool allowed a cell that is replicated many times to be placed once, and the replication strategy (e.g. linear systolic) to be specified in order to create the best placement across the array of CLBs.

We concluded the paper by presenting several important applications with the potential to be impacted by FPGA technology. Reconfigurable computing based on FPGAs was predicted to reach 1000 GOPS/ft^3 and 1000 MOPS/Watt within a couple of years, with the assumption of using embedded packaging with MCMs (Multi-Chip Modules). Throughput efficiencies were also predicted to be substantially higher than with programmable processors, but less than unity.

We do not imply that all front-end radar signal processing can be solved using reconfigurable computing. There are still system specifications, for example, in spaceborne applications, where custom VLSI solutions are further ahead than reconfigurable hardware, particularly when we need solutions with throughputs per unit power $\geq$20 GOPS/W, data throughputs exceeding GBytes/sec, and compliant with radiation hardening requirements.

Acknowledgments

We would like to thank several people at MIT Lincoln Laboratory for their help with various aspects of the technologies discussed in this paper. Bob Ford, Bill Song, Michael Killoran, and Huy Nguyen provided many insights into the application of FPGAs to front-end digital filtering. The architecture for the custom front-end radar signal processor was developed by Paul McHugh. The custom chip specifications were provided by Bob Pugh, and the design was implemented by Joe Greco. The chip was fabricated by National Semiconductor at the Arlington, Texas facility. Finally, the back-end chip design and design rule verifications were performed by Mentor Graphics, Inc. The authors are also thankful to the anonymous reviewers for their inputs and constructive comments.

Note

1. Courtesy of Michael Killoran, MIT Lincoln Laboratory.

References

1. G.W. Stimson, *Introduction to Airborne Radar, 2nd edn.*, Scitech Publishing, Inc., 1998.
2. J. Ward, "Space-Time Adaptive Processing for Airborne Radar," Technical Report #1015, MIT Lincoln Laboratory, Dec. 1994.
3. A.V. Oppenheim and R.W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Inc., 1989.
4. K. Teitelbaum, "A Flexible Processor for a Digital Adaptive Array," in *Proceedings of the 1991 IEEE National Radar Conference*, March 1991.
5. E.D. Baranoski, "Pre-Processor Specifications," Internal Memorandum, 21 April 1995.
6. J. Greco, "A1000 Critical Design Review," MIT Lincoln Laboratory, June 1996.
7. "Xilinx Celebrates 15th Year of Continuous Innovation in Programmable Logic," Xilinx Press Release, Feb. 1999, http://www.xilinx.com/company/anniversary.htm.
8. "The Future of FPGAs," Xilinx White Paper, http://www.xilinx.com/prs_rls/5yrwhite.htm.
9. K. Chapman, "Building High Performance FIR Filters Using KCMs," Xilinx App Note, July 1996, http://www.xilinx.com/appnotes/kcm_fir.pdf.
10. G.R. Goslin, "A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application—Specific Digital Signal Processing Performance," Xilinx, Dec. 1995, http://www.xilinx.com/appnotes/dspguide.pdf.
11. L. Mintzer, "Large FFTs in a Single FPGA," in *Proceedings of the 7th International Conference on Signal Processing Applications & Technology*, Boston, MA, 7–10 Oct. 1996.
12. B. Allaire and B. Fischer, "Adaptive Filters in FPGAs," in *Proceedings of the 7th International Conference on Signal Processing Applications & Technology*, Boston, MA, 7–10 Oct. 1996.
13. "Xilinx Core Solutions Data Book," Xilinx, 2/98, http://www.xilinx.com/products/logicore/core_sol.htm.
14. "A Simple Method of Estimating Power in XC4000XL/EX/E FPGAs," Xilinx Application Brief XBRF014 v 1.0, June 1997, http://www.xilinx.com/xbrf/xbrf014.pdf.
15. "PowerPC 603e Microprocessors, Motorola," http://www.motorola.com/SPS/PowerPC/products/semiconductor/cpu/603.html.
16. "PowerPC 750 and PowerPC 740 Microprocessors," Motorola, http://www.motorola.com/SPS/PowerPC/products/semiconductor/cpu/750.html.
17. W. Song, "A Two Trillion Operations per Second Minature Mixed Signal Radar Receiver/Processor," in *Asilomar Conference on Signals, Systems, and Computers*, Nov. 1998.
18. T.J. Moeller and D.R. Martinez, "Field Programmable Gate Array Based Front-End Digital Signal Processing," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM'99, April 1999.
19. T.J. Moeller, "Field Programmable Gate Array for Front-End Digital Signal Processing," Master of Engineering Thesis, Massachusetts Institute of Technology, May 1999.
20. G.R. Goslin, "Using Xilinx FPGAs to Design Custom Digital Signal Processing Devices," in *DSPX 1995 Technical Proceedings*, 12 Jan. 1995, p. 595.
21. G.R. Goslin, "Implement DSP Functions in FPGAs to Reduce Cost and Boost Performance," EDN, 1996.
22. B. New, "A Distributed Arithmetic Approach to Designing Scalable DSP Chips," EDN, 17 Aug. 1995.
23. Xilinx Publications, "The Role of Distributed Arithmetic in FPGA-based Signal Processing," Technical Report.
24. S.A. White, "Application of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSP Magazine*, July 1989.

David R. Martinez received a B.S. degree in Electrical Engineering from New Mexico State University in 1976. He received an M.S. and E.E. degree in Electrical Engineering from MIT, jointly with the Woods Hole Oceanographic Institution in 1979. Mr. Martinez also completed an MBA degree from the Southern Methodist University in 1986. He worked at the Atlantic Richfield Co. in seismic signal processing from 1979 to 1988. During this time, Mr. Martinez worked on algorithm development and technology field demonstrations. While at Atlantic Richfield Co., he received a Special Achievement Award for the conception, management, and implementation of a multidisciplinary project. He holds three U.S. patents relating to seismic signal processing hardware. He has worked at MIT Lincoln Laboratory since 1988. His areas of interest are in VLSI signal processing and high performance parallel processing systems. He has been responsible for managing the development of several complex real-time signal processor systems. Mr. Martinez is Associate Division Head in the Air Defense Technology Division. For the last three years, he has been the chairman for a national workshop on high performance embedded computing, held at MIT Lincoln Laboratory. He also served as an Associate Editor for the IEEE Signal Processing Magazine.

Tyler J. Moeller grew up in Alexandria, VA, where his interest in electrical engineering was sparked over several summers of internships at the Army's Night Vision and Electro-Optics Laboratory during High School. He then attended MIT, where he received his Bachelors degree in Electrical Engineering and Computer Science. During the summers, he was an intern at MIT Lincoln Laboratory, where he worked on the Laboratory's Miniaturized Digital Receiver project, designing the digital filtering multi-chip module for the project. Tyler then received his Master's degree in Electrical Engineering and Computer Science from MIT while working on his thesis, Field Programmable Gate Arrays for Radar Front-End Digital Signal Processing, at Lincoln Laboratory with Dave Martinez. He is now a lead developer at carOrder.com, a spin-off company of Trilogy Software.

Kenneth Teitelbaum received the B.S.E.E. degree in 1977 from the State University of New York at Stony Brook and the M.S. degree in 1979 from the University of Illinois at Urbana-Champaign. Since then he has been employed by M.I.T. Lincoln Laboratory in Lexington, MA, and currently holds the position of Senior Staff Member in the Embedded Digital Systems Group. His interests include radar systems design, adaptive signal processing, and parallel computing.