Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2010, Article ID 357978, 14 pages
doi:10.1155/2010/357978

Research Article

Partial Reconfigurable FIR Filtering System Using Distributed Arithmetic

Daniel Llamocca,[1] Marios Pattichis,[1] and G. Alonzo Vera[2]

[1] *Electrical and Computer Engineering Department, The University of New Mexico, Albuquerque, NM 87131, USA*
[2] *Microelectronics Research and Development Corporation, Albuquerque, NM 87110, USA*

Correspondence should be addressed to Daniel Llamocca, dllamocca@ieee.org

Received 2 March 2010; Revised 8 July 2010; Accepted 20 November 2010

Academic Editor: Viktor K. Prasanna

Dynamic partial reconfiguration (DPR) allows us to adapt hardware resources to meet time-varying requirements in power, resources, or performance. In this paper, we present two new DPR systems that allow for efficient implementations of 1D FIR filters on modern FPGA devices. To minimize the required partial reconfiguration region (PRR), both implementations are based on distributed arithmetic. For a smaller required PRR, the first system only allows changes to the filter coefficient values while keeping the rest of the architecture fixed. The second DPR system allows full FIR-filter reconfiguration while requiring a larger PR region. We investigate the proposed system performance in terms of the dynamic reconfiguration rates. At low reconfiguration rates, the DPR systems can maintain much higher throughputs. We also present an example that demonstrates that the system can maintain a throughput of 10 Mega-samples per second while fully reconfiguring about seventy times per second.

1. Introduction

Dynamically reconfigurable systems offer unique advantages over nondynamic systems. Dynamic adaptation provides us with the ability to adapt hardware resources to match real-time varying requirements. The majority of the 1D FIR filtering literature is dominated by static implementations. Here, we use the term static to refer to both CMOS implementations (e.g., [1–5]) and reconfigurable hardware (nondynamic) (e.g., [6, 7]). Some implementations use the label *reconfigurable* in the sense of having the capability to load different filter coefficients on demand (e.g., [2–5]). In the context of this paper, such implementations are considered static since the underlying hardware is not changed or reconfigured.

For reconfigurable hardware, the most efficient implementations are based on Distributed Arithmetic (DA) [8]. These filters have coefficients fixed or hardwired within the filter's logic. This approach allows fast and efficient implementations while sacrificing some flexibility, since coefficients cannot be changed at run time. Dynamic partial reconfiguration (DPR) can be used in this scenario to provide the flexibility of coefficients' values changes without having to turn off the device and only rewriting a section of the configuration memory. The efficiency of DPR over the full reconfiguration alternative and the savings in terms of power and resources is a function of the relative size of the portion being reconfigured [9].

We consider a DPR approach that allows us to change the filter's structural configuration and/or the number of taps. The proposed approach provides a level of flexibility that cannot be efficiently accomplished with traditional static implementations. In particular, we develop a dynamically reconfigurable DA-based FIR system that uses DPR to adapt the number and value of the coefficients, the filter's symmetry, and output truncation scheme. Two systems are presented that allow the flexibility to change all these filter's characteristics: (i) a system that only allows changes to the coefficients values and (ii) a system that allows changes to the number and value of the coefficients, the symmetry, and the output truncation scheme.

Previous research on dynamically reconfigurable FIR filters has focused on multiply-accumulate-based implementations and coarse reconfiguration. The first system

Table 1: Fir filter implementation savings due to the use of filter blocks.

Implementation	Resource requirements
1 Filter block of size M (LUTs have M inputs)	size $I \times 2^M$ words
M/L filter blocks of size L (LUTs have L inputs)	size $I \times 2^L \times M/L$ words

Table 2: Hardware utilization of Virtex-4 XC4VFX20-11FF672 for coefficient-only reconfiguration.

Module	FF	(%)	Slice	(%)	LUT	%
PRR	0	0%	180	2%	360	2%
Static Region	5303	31%	6130	72%	8698	51%
PRR interface	1313	8%	786	9%	885	5%
Overall	5203	31%	6310	74%	9058	53%

described in this paper is based on dynamically reconfiguring at a coarse level, that is, the entire FIR filter. The second system is based on dynamically reconfiguring at the finest possible level, the LUTs that store the coefficients, with a small dynamic reconfiguration area. We have demonstrated a related, LUT-based approach in a dynamically reconfigurable pixel processor [10]. The paper also explores different ways to execute dynamic partial reconfiguration and elaborates on the impact over reconfiguration time overhead of the different approaches.

This paper provides an extended version of the conference paper presented in [11]. The paper has been extended to provide: (i) extended background information, (ii) more implementation details, (iii) extended methodology, (iv) architectural extensions to allow changes on the filter's internal structure, and (v) new results.

The rest of the paper is organized as follows: Section 2 presents background and related work. Section 3 describes the FIR filter core implementation. Section 4 introduces the dynamically reconfigurable system. Results and conclusions are presented in Sections 5 and 6, respectively.

2. Background and Related Work

Reconfigurable logic has established itself as a popular alternative to implement digital signal processing algorithms [12]. Furthermore, a number of articles have been published on using DPR to implement different signal processing algorithms [9, 11, 13, 14]. In particular, [15–17] report different approaches for taking advantage of DPR in FIR filter implementations. The capability of reconfiguring a filter at run time is of special interest for applications such as wireless communications and software radio.

Hardware realizations of FIR filters can be divided into constant coefficients and multiplier-based implementations [15]. In the latter case, DPR is mainly used to change a filter's overall structure [16, 17], or other filter-wide characteristic. At a higher level, DPR is also used to simply change the level of parallelism of an implementation by changing the number of filter cores in an application's critical path. In all these cases, changes are usually initiated from a desire to implement a new filter, based on power or resources considerations, or simply to obtain new functionality. A change in coefficients does not require reconfiguration for this type of filter implementation. Thus, for these cases, DPR has milder constraints in terms of reconfiguration speed and reconfigurable logic partition.

The case of constant coefficients implementation is considerably more complex, since DPR is used to change inner characteristics of the filters (coefficients are not easily isolated within the filter structure). This requires more complex schemas to segment logic into reconfigurable tiles and more efficient reconfiguration mechanism in order to reduce the amount of time it takes to reconfigurable a filter.

DA filters in Xilinx FPGAs are introduced in [18, 19], where the authors exploit common characteristics between the Xilinx's FPGA architecture and the filter architecture. In [7], the authors present other approaches for flexible FPGA implementations of FIR filters by combining pipelined multipliers and parallel, distributed arithmetic.

In [15], the authors consider different DPR architectures for extending constant-coefficients approaches to implement adaptive filters. This relatively early study already provides insights on the advantages of using run-time partial reconfiguration to modify a filter's behavior at run time. The study used an earlier device (currently unavailable) and explored architectures different than DA, which were a natural fit for such device. Their results in terms of performance cannot be compared to our results due the inherent difference between the reconfigurable devices used.

In [17], the authors describe a self-reconfigurable adaptive FIR filter system composed of up to three multiplier-based filter modules. These modules can be reconfigured at run time by a control manager that uses System ACE to store and fetch the corresponding partial bitstream. This system only allows a full-filter reconfiguration instead of finer reconfiguration schemas such as coefficient-only reconfiguration. In this paper, speed results are not clearly presented. The authors report different reconfiguration overhead times for different filters that apparently occupy the same reconfigurable region in the device. These results are surprising, since reconfiguration time overhead depends mainly on the bitstream size, which depends on the size of the partial reconfigurable area, not on the number of resources used within that area. It is also worth mentioning that reconfiguration speeds reported are slower than speeds reported on other DPR papers [20, 21].

In [16], a similar system is described although in this case, it is not self-reconfigurable and uses an external PC to perform reconfiguration. Reconfiguration times reported are also considerably slower than other reported methods.

In [22], the authors describe a tool-flow to map applications to a self-reconfiguring application. The authors use a 32-tap MAC-based FIR filter as an example. The paper compares the performance of simply reloading coefficients by writing over specific registers and using DPR to reconfigure the whole filter. In this paper, the reconfiguration time

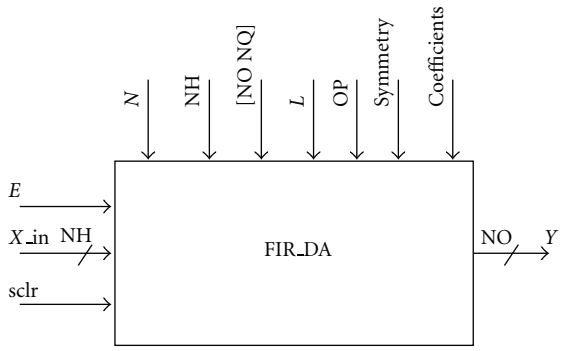

Figure 1: Generalized FIR DA module.

Table 3: Hardware utilization of Virtex-4 XC4VFX20-11FF672 for full-filter reconfiguration.

Module	FF	(%)	Slice	(%)	LUT	%
PRR	1324	8%	818	10%	1306	8%
Static Region	4017	24%	5515	65%	8072	47%
PRR interface	6	0%	5	0%	107	1%
Overall	5341	31%	6333	74%	9378	55%

overhead is large but dismissed as an acceptable handicap for the paper's goals.

In general, the reconfiguration time overhead is an important factor in the evaluation of systems using DPR. Several approaches exist to deal with the overhead. One approach is to hide it by using efficient hardware scheduling strategies (e.g., [23]). A more simplified approach is to select carefully the elements of an architecture that requires reconfiguration for a desired change in functionality (e.g., [11, 21]). By doing so, one can reduce drastically the size of the partial bitstream used to execute the DPR, thus reducing the reconfiguration time overhead. Finally, there is also the approach of maximizing the access speed to the configuration memory (e.g., [20]). Unfortunately, this approach has a limit determined by the device. In the case of Virtex-4 FPGAs, the maximum speed is 3.2 Gbps (32 bit wide bus at 100 MHz). A combination of the last two approaches is used in this paper to deal with reconfiguration time overhead.

Our paper seeks to extend prior research in this area by primarily focusing on developing, analyzing, and improving DPR systems in terms of the dynamic reconfiguration rate on modern devices. This leads us to consider a DA implementation that allows efficient implementations with small hardware footprints on modern FPGA devices. Then, we consider a scalable approach where we have two systems: (i) a DPR system that allows for faster dynamic reconfigurations of coefficient values while fixing the number of taps and (ii) a second DPR system that allows flexibility in the number of taps, the filtering structure, and truncation characteristics while allowing for a slower dynamic reconfiguration rate.

Table 4: PRR measures for both dynamic partial reconfiguration system realizations.

Dynamic realization	PRR size (Slices)	Bitstream size (bytes)
(1) Coefficient-only reconfiguration	$90 \times 6 = 540$	43000
(2) Full-filter reconfiguration	$44 \times 20 = 880$	83000

3. Stand-Alone FIR Filter Core Implementation

A high-performance FIR implementation based on Distributed Arithmetic is described in this section (see also [11]). The approach was coded in VHDL, so as to achieve a level of portability. Specific LUT primitives are employed when the system is compiled in Xilinx devices. We will consider two dynamic realizations based on this core in Section 4.

3.1. Description. The FIR filter module is shown in Figure 1. It shows the FIR filter module with its inputs, outputs, and parameters. Signal "E" controls the input validity. Clearing the register chain ("sclr" signal) at will is an important requirement when performing filtering on finite size signals.

We present two filter implementations in Figure 2. A simplified approach is possible for symmetric filters (see Figure 2) [24]. The more general, nonsymmetric case is also presented in Figure 2.

Here, N denotes the number of taps, NH represents the input/coefficients bitwidth, L is the LUT input size (explained in next subsection). We also use OP for controlling the output truncation scheme: (i) LSB truncation then saturation, (ii) LSB and MSB truncation, and (iii) no truncation. We use the parameter format [NO NQ] to denote the fixed-point output format for NO bits with NQ fractional bits. The filter coefficients are specified in an input text file.

We define $M = \lceil N/2 \rceil$, size I = NH + 1 for symmetric filters, and $M = N$, size I = NH for nonsymmetric filters. The inputs/coefficients format is set at [NH NH-1], which restricts values to $[-1, 1)$. As a result, the maximum number of output integer and fractional bits results

$$\left[2(\mathrm{NH}-1) + \left\lceil \log_2(N+1) \right\rceil + 1 \quad 2(\mathrm{NH}-1)\right]. \qquad (1)$$

3.2. FIR DA Implementation. The Distributed Arithmetic technique rearranges the input sequence samples (be it $x[n]$ or $s[n]$) into vectors of length M, which require an array of size IM-input LUTs. This becomes prohibitively expensive when M is large. For efficient implementation, we divide the filter into M/L filter blocks [24], as illustrated in Figure 2. Each filter block works on L coefficients requiring size IL-input LUTs (each vector of size L goes to one L-input LUT, see Figure 3). Table 1 summarizes the resources savings associated with the filter blocks approach. An advantage of using FIR filter blocks is that it allows for efficient routing while mapping the implementation to the specific LUT primitives found in an FPGA. As shown in [10], the approach is scalable in that can be easily ported to different LUT sizes.

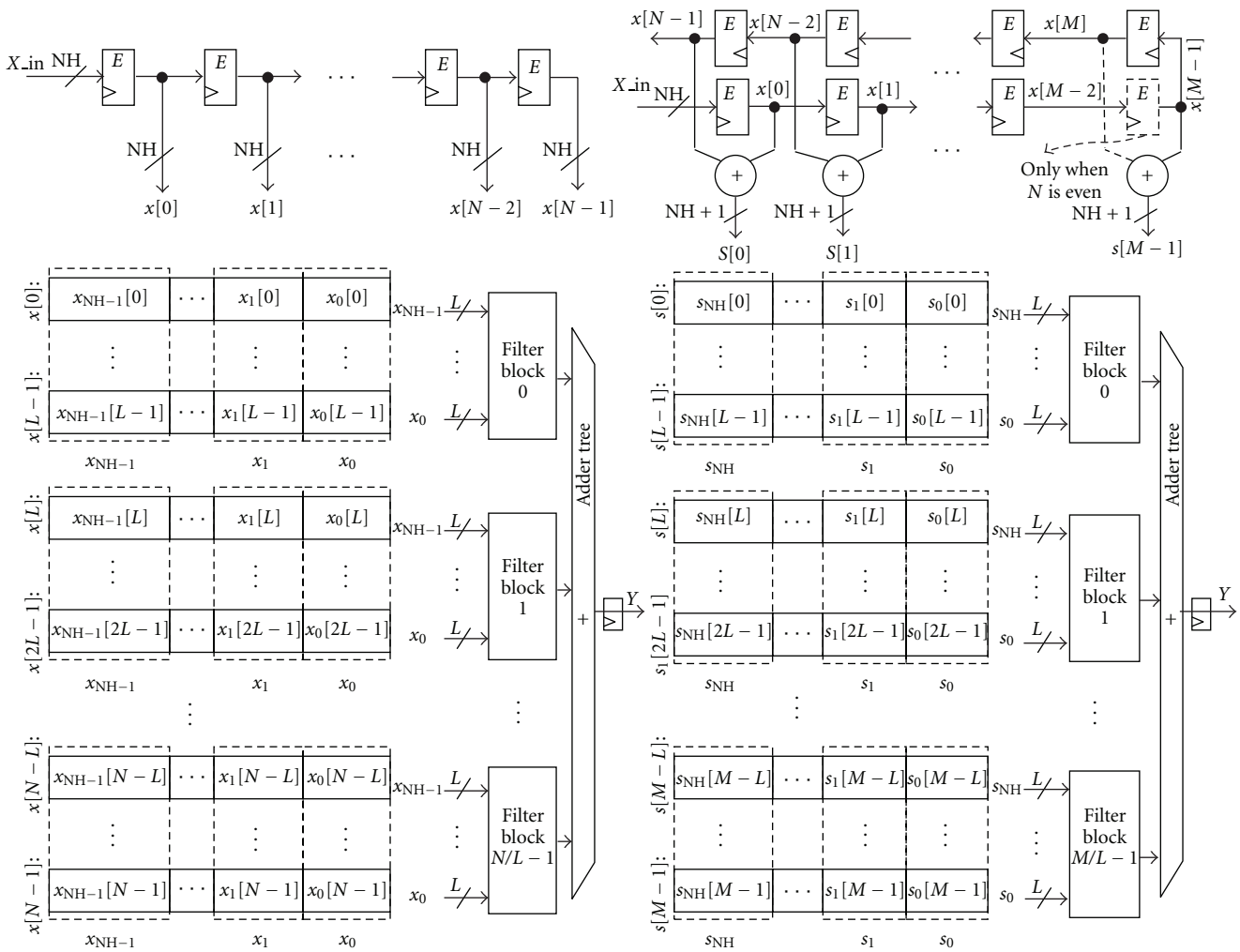

Figure 2: High-performance DA implementation based on the underlying LUT input size (L). Nonsymmetric filter (left) symmetric filter (right).

To demonstrate the savings, we consider a particular example. Using the formulae of Table 1, for $M = 16$, $L = 4$, we have significant savings since $2^{16} \gg 2^4 \times 16/4$. It does require an additional adder tree structure (see Figure 2). However, compared to the savings, the overhead is not significant.

A pipelined implementation of a symmetric filter block example is shown in Figure 3. Here, we have the parameters SYMMETRY = YES and NH = 8. It consists of an array of L-input LUTs, an adder tree, shifters, and registers. The number of register levels is given by the following formula:

$$\#\text{ of register levels in Filter Block} = \left\lceil \log_2(\text{size } I) \right\rceil. \quad (2)$$

The L-input LUT subblocks are shown in Figure 4. Here, the output word size of each L-input LUT is given by $\text{LO} = \text{NH} + \lceil \log_2(L) \rceil$. It also shows its decomposition into LO L-to-1 LUTs, useful for efficient FPGA implementation. Xilinx FPGA devices contain L-to-1 LUT primitives with $L = 4$ (Spartan-3, Virtex-II Pro, Virtex-4) and $L = 6$ (Virtex-5). Thus, $L = 4$ or $L = 6$ are optimum values of choice. Moreover, as explained in [10] for Virtex-4, optimal LUT implementations can also be obtained for $L = 5, 6, 7, 8$.

Figure 5 depicts the internal pipelined architecture of the adder tree that is used for adding the filter blocks outputs. The result is stored in an output register. The number of register levels of the adder structure is given by

$$\#\text{ of register levels in Filter Adder Structure} = \left\lceil \log_2\left(\frac{M}{L}\right) \right\rceil. \quad (3)$$

Since we can quantize the LUT table values (i.e., the summations), rather than the coefficients, this FIR DA Implementation is slightly less sensitive to quantization noise than a normal implementation, with quantized coefficients.

The latency of the pipelined system is shown in Figure 6. The latency (input-output delay) is given by REG_LEVELS = $\lceil \log_2(\text{size } I) \rceil + \lceil \log_2(M/L) \rceil + 2$ cycles, where REG_LEVELS is the number of register levels between the input and the output.

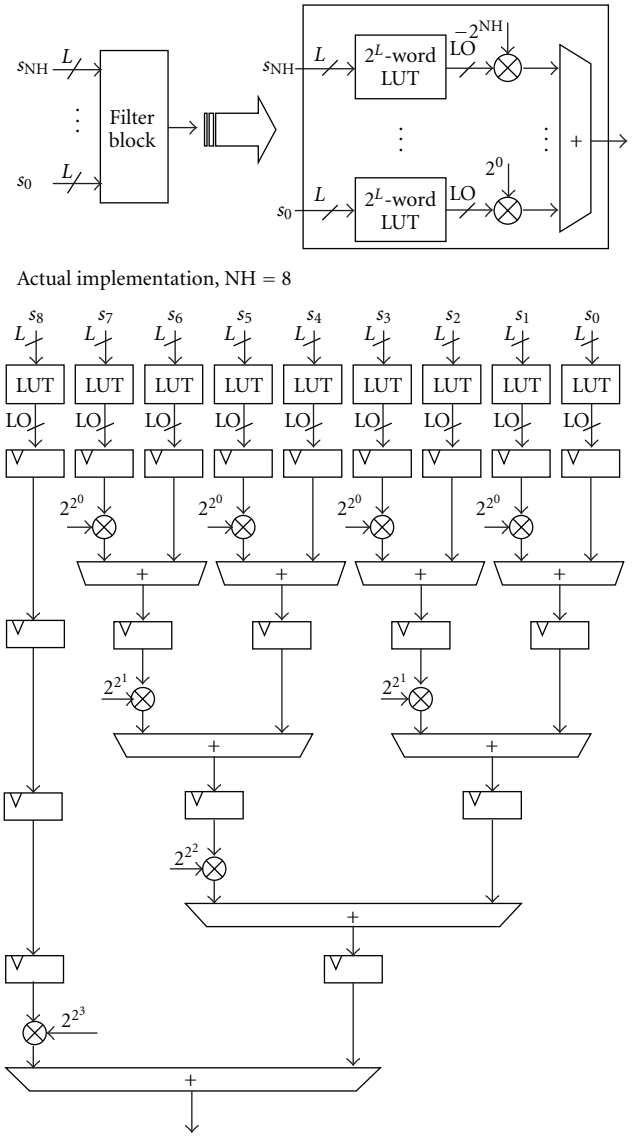

Figure 3: Filter block architecture. SYMMETRY = YES.

Table 5: Reconfiguration time for both DPR system realizations. the 43 kb bitstreamcorresponds to coefficient-only reconfiguration case. The 83 kb bitstream corresponds to full-filter reconfiguration.

Scenario	Reconfiguration speed	Reconfiguration time	
		43 KB bitstream	83 KB bitstream
(1) Current	3.28 MB/s	13.10 ms	25.30 ms
(2) Custom [20]	295.4 MB/s	0.145 ms	0.280 ms
(3) Ideal	400 MB/s	0.107 ms	0.207 ms

4. Dynamically Reconfigurable FIR Filtering System

We now extend the basic FIR filter core to be dynamically reconfigurable. We allow for the dynamic reconfiguration of both the number and the filter coefficients themselves in an embedded system. The basic system is shown in Figure 7. By means of dynamic partial reconfiguration, we turn a constant coefficient FIR filter into an adaptive FIR filter.

The basic approach requires that we prespecify the Partial Reconfiguration Region (PRR). We consider two dynamically reconfigurable realizations.

(1) *Coefficient-only reconfiguration*: The PRR allows modifications to the filter coefficient values, while keeping the rest of the architecture intact.

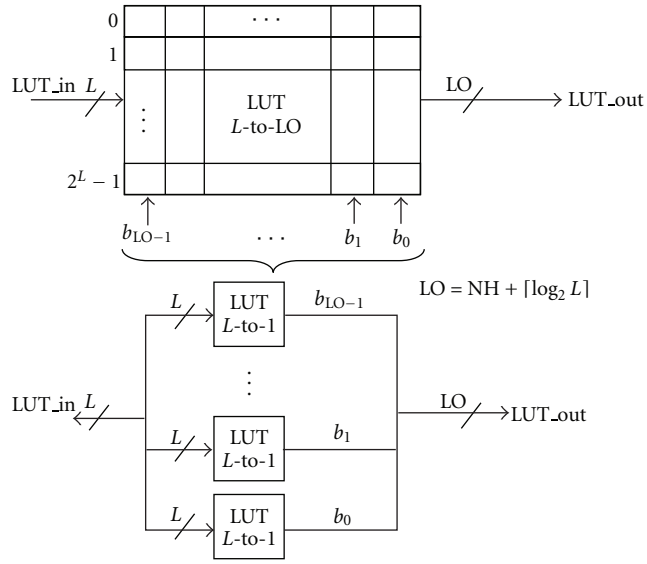

Figure 4: Realization of an L-to-LO LUT using LO L-to-1 LUTs.

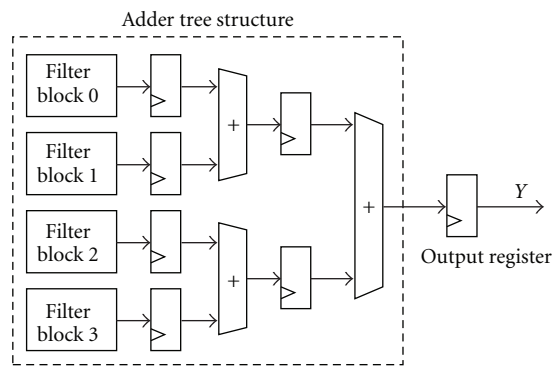

Figure 5: Adder tree structure for filter blocks' outputs. $M/L = 4$.

Table 6: DPR system throughput (MSPS) as function of delay between reconfigurations for 1D fir filtering with full filter reconfiguration.

Scenario	Number of samples between reconfigurations				
	2048 K	1024 K	409.6 K	204.8 K	102.4 K
	Amount of time between reconfigurations				
	68 ms	34 ms	13.6 ms	6.8 ms	3.4 ms
(1) Current	21.9	17.2	10.5	6.3	3.5
(2) Custom [20]	29.9	29.8	29.5	28.9	27.8
(3) Ideal	30.0	29.9	29.6	29.2	28.3

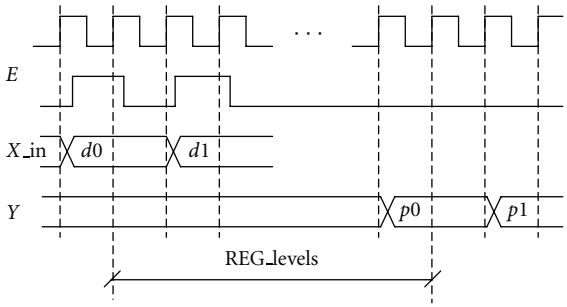

Figure 6: Latency measured from the moment "$d0$" is input until its correspondent output "$p0$" is available.

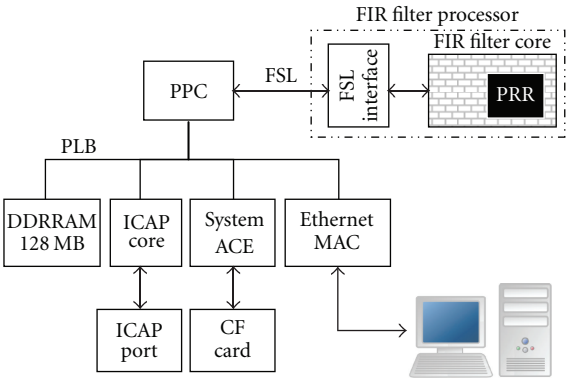

Figure 7: System block diagram.

(2) *Full-filter reconfiguration:* The PRR allows modification to the number of coefficients, the coefficient values, and the filter symmetry.

We start by describing the system architecture and FIR filter dataflow, which are not affected by the PRR definition. Then, we explain each of the dynamic realizations by providing a detailing representation of the PRR in the context of the FIR filter architecture.

4.1. System Architecture. From Figure 7, we can see that the dynamic FIR core and the PowerPC (PPC) communicate using the high speed FSL bus. The Partial Reconfiguration Region (PRR) is dynamically reconfigured via the internal configuration access port (ICAP), driven by the ICAP controller core.

The DDRRAM stores volatile data needed at run time, for example, input streams, processed streams, and partial bitstreams. At power-up, SystemACE reads a Compact Flash (CF) Card that stores the partial bitstreams and input streams. The processed streams are written back to the DDRRAM. The Ethernet core provides reliable communication with a PC and allows us to get new partial bitstreams or new input streams and to send processed streams to the PC for its verification or storage. Also, it serves as an interface for throughput measurements and system status.

Figure 8 depicts the interfacing of the FIR filter processor and the PPC for both dynamic realizations. The FIR filter processor, as shown in Figure 7, consists of the FIR filter core and a control unit that provides interfacing with the 32-bitwide FSL bus. Figure 8 shows a special case when the filter input size is NH = 8 bits. Here, the input is processed sample by sample (one byte at a time). After 32 output samples are computed, they are transmitted through the FSL bus. Other input/output bit-width configurations require different logic and control.

We next provide a description of the different possible modes of operation. First, we note that an FIR filter with N coefficients and NX input values can output a maximum of $\mathrm{NX} + N - 1$ values. The three modes of operation are implemented through a finite state machine as follows.

(i) *Basic output mode*: The system computes the first NX output values. This mode is useful for finite 1D signals.

(ii) *Symmetric output mode*: The system computes the central NX output samples (i.e., in the range $\lfloor N/2 \rfloor + 1 : \mathrm{NX} + \lfloor N/2 \rfloor$). This mode is useful when performing 2D separable convolution on images.

(iii) *Streaming mode*: with infinite number of input samples, that is, $\mathrm{NX} = \infty$.

4.2. FIR Filter Processor Data Flow. The FIR Filter processor receives and sends 32 bits at a time via the FSL bus. Due to the FIFO-like nature of the FSL bus [25], the PPC processor sends a data stream to FIFOw to be grabbed by the FIR filter processor that in turn writes an output data stream on FIFOr to be retrieved by the PPC processor (see Figure 8).

We optimize FSL bus usage by letting the PPC write a large block of data on FIFOw. The FIR filter processor then processes the data and writes the results on FIFOr in a pipelined fashion. After reading all data in FIFOr, the PPC writes another large block of data on FIFOw, that is, the PowerPC is busy only when reading/writing each large block of data. In addition, the FIR filter processor starts reading the next available block of data on FIFOw right after writing a processed chunk of data on FIFOr. Each FIFO depth has been set to 64 words (32-bit words).

4.3. Dynamic Partial Reconfiguration Setup. Figure 8 presents two dynamically reconfigurable systems and the associated PRRs. In the full-filter reconfiguration case, we do not allow any changes to the I/O bit-width. Here, we note that a change to the I/O bit-width would also require a generalized FSL interface to be included in the PRR, further complicating the design. Despite the complexity of doing so, this will be of interest for allowing us to build a dynamic precision system.

The static region is defined by everything else outside the PRR, including FSL interface, FSL circuitry, peripheral controllers, and the FIR filter core static portion (coefficient-only reconfiguration).

All signals between the dynamic region (PRR) and the static part are connected by prerouted bus macros in order to lock the wiring. Also, the PRR I/Os are registered as the reconfiguration guidelines advise [26]. To perform DPR, the partial bitstreams are read from a CF card and stored in

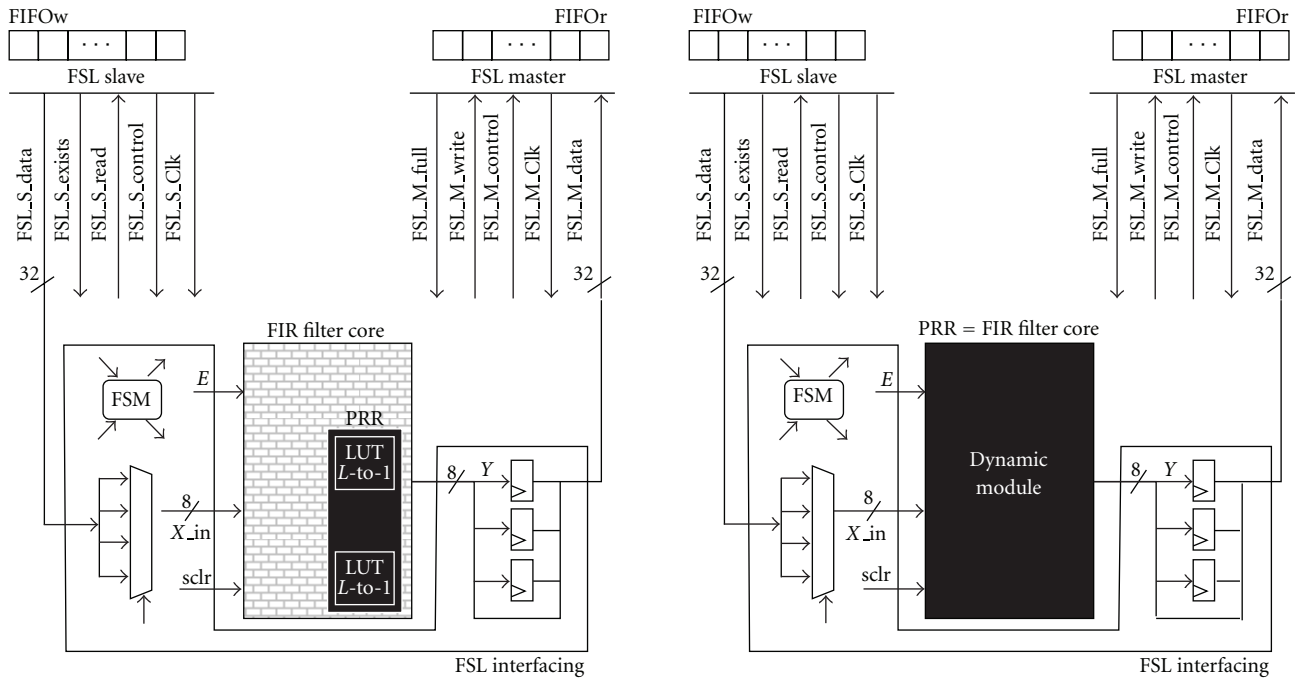

Figure 8: Dynamic FIR filter processor interfacing with FSL. PRR for dynamic reconfiguration of the coefficients (left) and PRR for dynamic reconfiguration of the number of coefficients, their values, and symmetry (right).

DDRRAM. When needed, they are written to the ICAP port. This fairly simple technique is explained in [21].

For throughput measurement purposes, the partial bitstreams and the input set of streams reside on DDRRAM. The streams are sent to the FIR Filter processor, and the output streams are written back to the DDRRAM. This process is repeated with different partial reconfiguration bitstreams loaded at specific rates, so as to get different filter responses and measure performance as the reconfiguration rate varies.

4.3.1. Coefficient-Only Reconfiguration. In this dynamic realization, the dynamic region is made of $(M/L) \times \text{size}\, IL$-to-1 LUTs, resulting in a PRR with $(M/L) \times \text{size}\, I \times L$ inputs and $(M/L) \times \text{size}\, I \times \text{LO}$ outputs. Figure 9 depicts the PRR along with the bus macros when SYMMETRY = NO, NH = 8, $N = 8$, $L = 4$. The PRR is depicted in the context of the FIR filter core.

This realization is very useful for applications that only require filter coefficients modification, and it exhibits a smaller reconfiguration time overhead than the full reconfiguration case. Also, since only the LUT values are modified, the routing inside the PRR does not change. This has potential advantages in the area of run-time bitstream generation, as there is no need for run-time place-and-route operation. Fast routing is a very demanding task, and in most cases, it cannot be performed at run time [27].

4.3.2. Full-Filter Reconfiguration. In this case, the PRR involves the entire FIR filter core. It enables us dynamically modify the coefficients, number of coefficients, symmetry, and LUT input size. Figure 10 depicts the PRR along with the bus macros in the context of the FIR filter processor (with the FSL interface). We can see that the PRR has NH+2 inputs and NH outputs.

5. Results

5.1. Stand-Alone Fir Filter Core. Figure 11 shows hardware resource utilization as a function of the number of coefficients (N), input bitwidth (NH), and symmetry (dotted lines: nonsymmetric filters, solid lines: symmetric ones). Also, we set OP = 0, $L = 4$. Here, we use the XC4VFX20-11FF672 Virtex-4 device, with 8544 slices.

In addition, for each input bitwidth, we are considering the largest output format attainable (in the range $[-1, 1)$). The output format ([NO NQ]) plays a negligible role in resource consumption (a difference of at most 12 slices).

Regarding frequency of operation, the goal of 200 MHz minimum frequency of operation was attained in all cases.

In addition, an error analysis is performed for the same parameters. Figure 12 shows the relative error curves for three cases (input stream = 1024 sinusoid samples). The error metric is

$$\text{Relative error} = \left| \frac{\text{ideal value} - \text{FPGA output}}{\text{ideal value}} \right|. \quad (4)$$

Figure 12 shows that in most cases the relative error is below 5%. The peaks correspond to FPGA values of zero and ideal values close to zero, resulting in a deceptive 100% error.

5.2. Embedded System. Results are shown using the following FIR Filter core parameters: $N = 32$, NH = 8, [NO NQ] = [8 7], $L = 4$, OP = 0, SYMMETRY = YES.

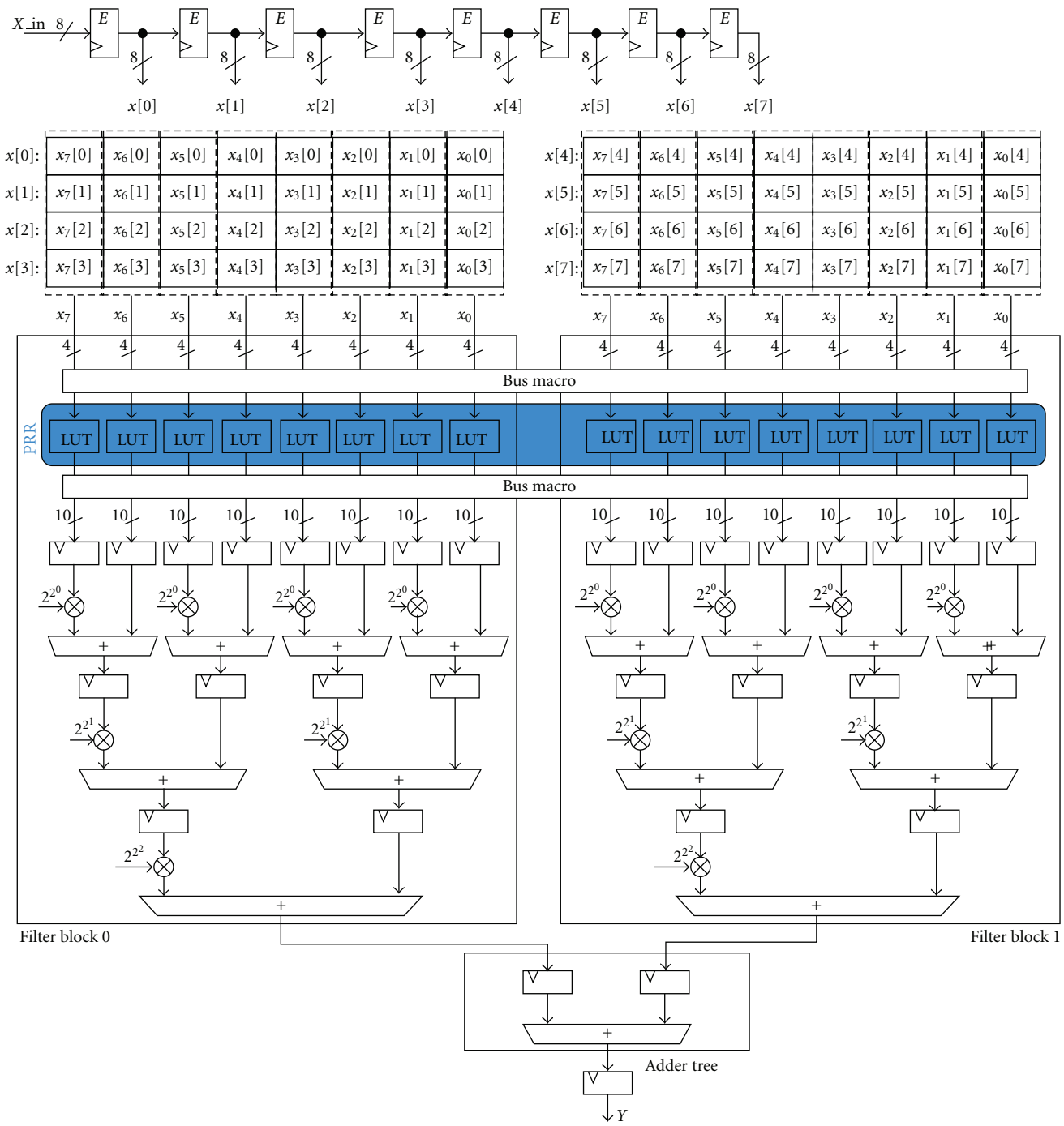

Figure 9: FIR filter core where the PRR and bus macros can be appreciated. Here, we can modify only the coefficients via the LUTs.

The system is implemented on the ML405 Xilinx Development Board that houses a XC4VFX20-11FF672 Virtex-4 FPGA. The PPC is clocked at 300 MHZ and the peripherals run at 100 MHz.

In order to improve performance, the DDRRAM memory space is cached. Also, the dynamic systems are tested in the basic output mode; that is, only the first NX outputs are considered.

5.2.1. Hardware Resource Utilization. Results for this section depend on the specific dynamic realization. Tables 2 and 3 show hardware resource utilization for two DPR systems: (i) coefficient-only reconfiguration and (ii) full-filter reconfiguration. It shows the static region, dynamic region and the entire system resource usage. The module "PRR interface" is the gluing static logic needed to join the static and dynamic regions.

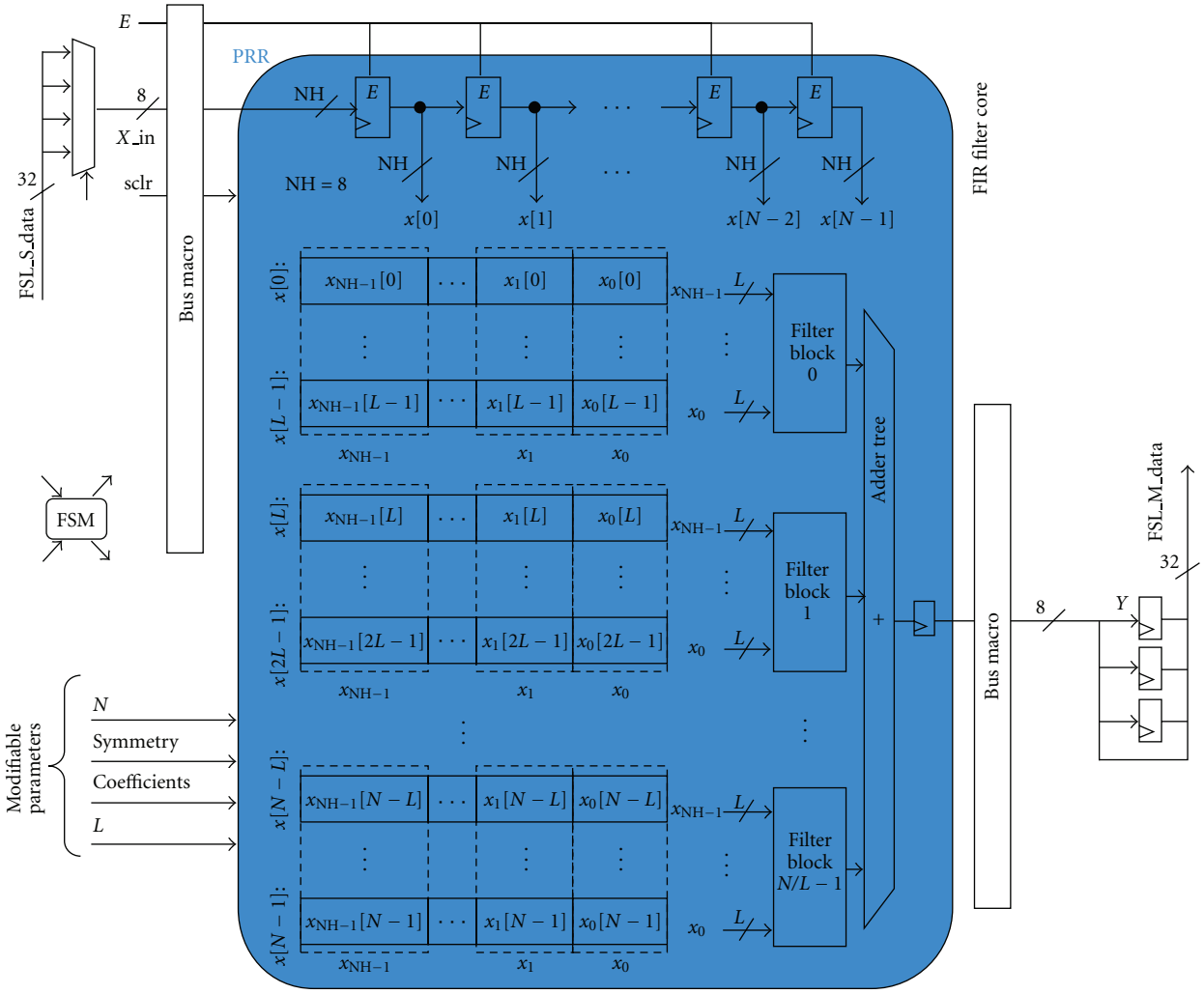

Figure 10: FIR filter processor where the PRR is the FIR Filter core. Note the parameters we can modify.

As expected, the overall resource utilization is about the same. What varies is the static region size, which is larger in the coefficient-only reconfiguration case.

Table 4 shows the reconfiguration size and its partial bitstream size. Note that the PRR in the first case is somewhat larger than expected (about 62% of the second case). This can also be appreciated in Figure 13 that shows the dynamic region (PRR) for both realizations, which are functionally the same.

The reason for the large PRR in the first case is the large number of required bus macros I/Os. In the coefficient-only reconfiguration case, the system needs access to the LUTs (see Section 4.3.1). As a result, for the special case shown, we require $(M/L) \times \text{size}\,I \times L = 4 \times 9 \times 4 = 144$ inputs and $(M/L) \times \text{size}\,I \times \text{LO} = 4 \times 9 \times 10 = 360$ outputs.

As explained in Section 4.3.2, in the second case (full-filter reconfiguration), we only need $\text{NH} + 2 = 10$ inputs and $\text{NH} = 8$ outputs. So, the PRR in the first case is larger than what it is actually needed for the L-to-1 LUT array, thereby wasting hardware resources in order to accommodate the large number of bus macros I/Os.

5.2.2. FIR Filter Processor Performance Bounds. The maximum throughput of this particular FIR filter processor ($\text{NH} = 8$) is given by

$$\text{Max.Throughput} = \frac{1\,\text{byte}}{1\,\text{cycle}} = \frac{8\,\text{bits}}{10\,\text{ns}} = 0.8\,\text{Gbps}. \quad (5)$$

Note that since the system is pipelined, there is an initial setup delay that becomes negligible over time. Actual throughput depends on many factors, such as cache size, PPC instruction execution, and FSL usage. Note that the maximum throughput of (5) cannot be attained since the PPC cannot read and write into the FIFOs at the same time.

5.2.3. Reconfiguration Time. Table 5 shows the reconfiguration time for 3 scenarios. Both dynamic realizations are included. In our setup, called Scenario 1, we used the Xilinx ICAP core and obtained a reconfiguration average speed of 3.28 MB/s. The reconfiguration time of Scenario 2 is computed based on the speed results reported in [22]. The dramatic improvement in reconfiguration lies on the use of a custom ICAP controller, DMA access, and burst transfers.

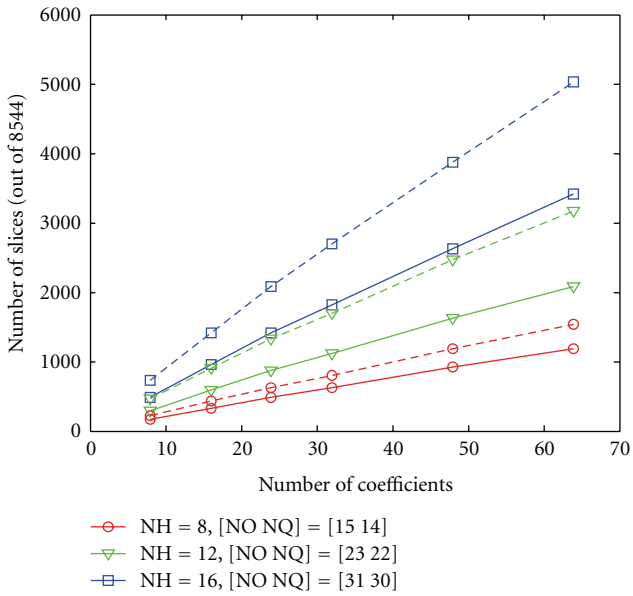

Figure 11: Resources versus number of coefficients and input bitwidth. Solid lines represent the symmetric case. Dotted lines represent the nonsymmetric case.

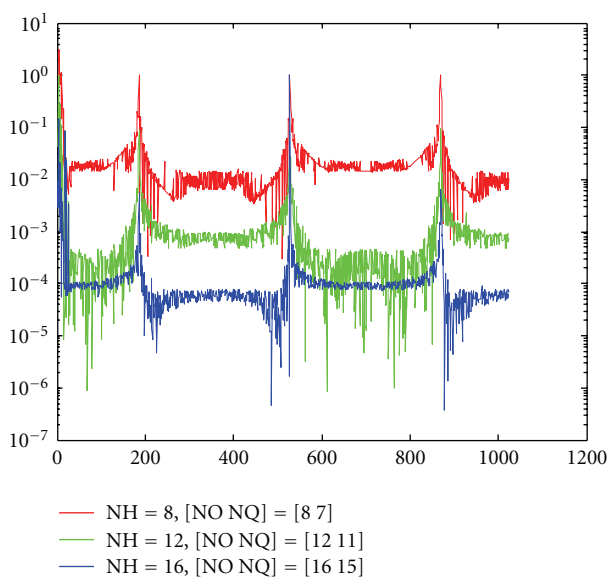

Figure 12: Relative error, $N = 32$. Three bitwidth cases.

Scenario 3 is the maximum theoretical throughput, which for the Virtex-4 is 400 MB/s [21].

5.2.4. Dynamic Performance. We use software timers to measure the elapsed time from the moment we start reading the input stream from DDRRAM until the processed stream is written back on DDRRAM. We are considering sinusoids as our inputs. Here, we refer to Section 4.3 for some of the details that will be discussed in this section.

In order to evaluate the dynamic performance of the system, we use a stream of 102400 samples (1 sample = 8 bits). The stream is processed a number of times (100 runs). Within the 100 runs, partial bitstreams are loaded at a specific rate. Each partial bitstream amounts to a different filter response.

Note that for the coefficient-only reconfiguration case, we only load a different set of coefficient values.

For the full-reconfiguration case, we switch between a filter with $N = 32$ coefficients and one with $N = 16$ coefficients. The PRR size is defined to be sufficiently large so as to allow implementation of the larger filter; that is, the $N = 32$ filter case. The filter with $N = 16$ requires only one fewer latency cycle (3). As a result, the static performance improvement of the smaller filter is not significant.

We report the average throughput over the 100 runs. Here, we define the dynamic reconfiguration rate in terms of the inverse of the number of samples that are being processed prior to a hardware reconfiguration. For better visualization, we report throughput in terms of the number of processed Mega samples per second (MSPS). This corresponds to the inverse of the reconfiguration rate.

Figures 14 and 15 show the dynamic performance over 100 runs for both dynamic realizations. There are 3 curves that correspond to the 3 scenarios shown in Table 5. In the limit, at zero reconfiguration rate, we have static performance. The performance results converge for the static case.

From Figure 14 (coefficient-only reconfiguration), we see that for Scenario 1 (our actual measurements), the static performance resulted in 29.25 MSPS. At the maximum reconfiguration rate (1 reconfiguration every stream), the dynamic performance resulted in 6.1651 MSPS. The other curves (Scenarios 2 and 3) provide performance bounds based on the static performance and reconfiguration speeds of Table 5.

We can appreciate that the dynamic performance of the full-filter reconfiguration case is slightly lower than the coefficient-only reconfiguration. This is due to differences in the PRR size. But as we increase the number of samples before a reconfiguration, or use a scenario other than the first one, this effect is less noticeable.

As expected, the dynamic performance heavily depends on reconfiguration speed and input stream size. Better reconfiguration speeds offset the reconfiguration time overhead (Scenarios 2 and 3). We have the same effect for smaller dynamic regions. The slower reconfiguration rates due to longer data streams help to offset the reconfiguration overhead as well.

In Table 6, we present the full-filter reconfiguration system throughput as a function of the time between reconfigurations. It is quite clear from the results that even for the slowest scenario, we can maintain throughputs over ten MSPS while dynamically reconfiguring seventy times per second.

5.3. Experimental Results with ECG Processing. We present an example application for electrocardiogram (ECG) characterization (R-wave detection). Here, we consider coefficient-only reconfiguration for implementing a 3-channel, 1D

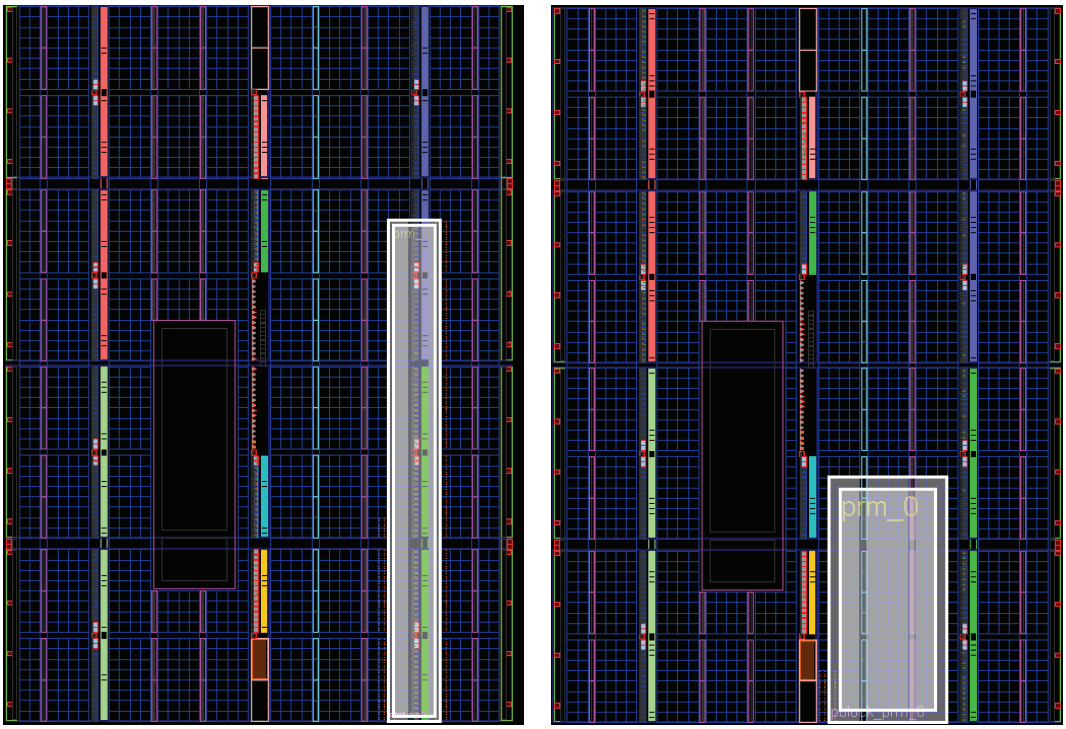

Figure 13: Dynamic reconfiguration region for (i) coefficient-only reconfiguration system (left), and (ii) full-filter reconfiguration system (right).

filterbank. We make use of the embedded system detailed in Section 5.2. Each channel filter is symmetric, with 32 8-bit coefficients for 8-bit I/O, using truncation (saturation) arithmetic for the outputs. Our approach here is to implement a variation of the ECG processing algorithms presented in [28].

ECG signal processing is of great interest for emergency applications, including the detection of cardiac arrhythmias [29] and stenosis assessment for atherosclerotic plaque video analysis [30]. A popular approach based on [28] is to use the outputs of a Wavelet filterbank for ECG analysis.

As in Wavelet analysis, we design a dyadic filterbank to cover the entire, discrete frequency space. We have a high-pass filter with a positive frequency pass-band from $\pi/2$ to π, a band-pass filter from $\pi/4$ to $\pi/2$, and a low-pass filter for frequencies up to $\pi/4$. For each channel filter, we consider efficient implementations using 32 8-bit coefficients. The magnitude response of the designed filterbank is shown in Figure 16.

For testing the implementation, we use the first recording (record 100) from the MIT arrhythmia database [31]. In this record, we have 2 channels with 650 K samples sampled at 360 Hz and quantized at 11-bits over a 10 mV range. We further quantized the input down to 8 bits, downloaded them to the DDRRAM using the Ethernet core and tested using the procedure outlined in Figure 17.

Based on [28], we implemented a simple R-wave detection algorithm. For detection, we look for thresholds in the outputs. In the example of Figure 18, we threshold as follows: low-pass filter ($[-1/32, 1/32]$), band-pass ($[-1/32, 1/32]$), and high-pass ($>1/64$). This results in perfect R-wave detection for the first 5 cycles of the second channel (1500 samples). We refer to [28] for more details on how to adjust thresholds in such algorithms for near-perfect results verified over the entire database. Our goal is to simply demonstrate the DPR FIR system on real signals.

The detection algorithm is included in the embedded PowerPC software routines, and the resulting signal is stored into the DDRRAM. We note that performance improves with larger input signals (see Figure 14). The detection algorithm is performed at the end of the operations and takes about 80 ms. Dynamic reconfiguration of a channel filter requires 13.1 ms. At a sampling rate of 360 Hz, the system allows significant time for implementing real-time detection algorithms and DPR. As a result, the number of samples that are processed prior to reconfiguration can be significantly reduced. By processing every 2000 samples, the processing rate stands at 4.62 MSPS (2000 samples takes 140 ms to process). Thus, after 5.5 seconds spent in acquiring 2000 samples, we get a detection response in 140 ms.

5.4. Comparison with Other PR Systems For FIR Filtering. The majority of previously reported work on FIR filtering is based on multiply-and-add approaches [14, 16, 17, 22]. In [14], the authors reported a reconfiguration time of 1.5 ms for changing the coefficients and their number on a Virtex-II (74.7 KB bitstream). In [16, 17], the authors presented a DPR FIR system that only allowed for changes in the number of coefficients. Reconfiguration time for 8794 slices for a 20-tap filter required 700 ms. The filter presented in [22] most

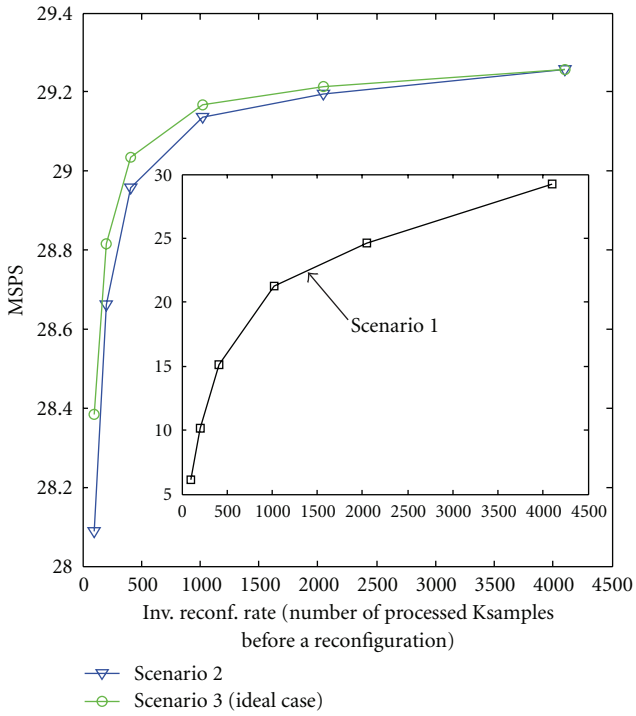

Figure 14: DPR system performance for coefficient-only reconfiguration.

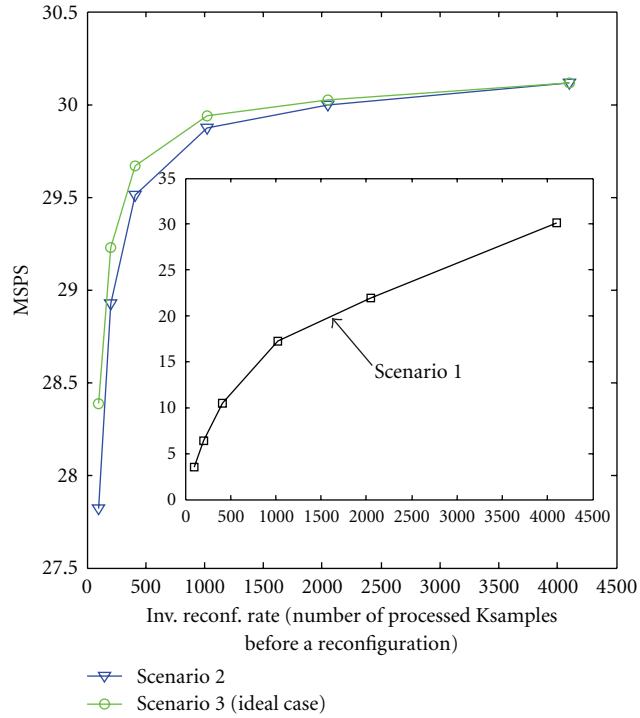

Figure 15: DPR system performance for full-filter reconfiguration.

closely resembles our FIR filter: 32-taps, 8-bit coefficients, 8-bit input, but with multiply-and-add approach. It required 1985 LUTs for a 13.1 ms reconfiguration time. We can change the entire filter using a 83 KB bitstream for a reconfiguration time of 25.3 ms.

As mentioned earlier, for FPGA implementations, the distributed arithmetic presented here is far better suited than these multiply-and-add approaches. DA approaches allow for efficient use of hardware resources. Beyond this, multiply-and-add approaches tend to have fixed input/output characteristics as opposed to the flexible, dynamically reconfigurable arithmetic representations presented here.

The constant-coefficient filter with DPR is mentioned in [15], but the work is more theoretical and the results are noncomparable with ours, as stated in Section 2.

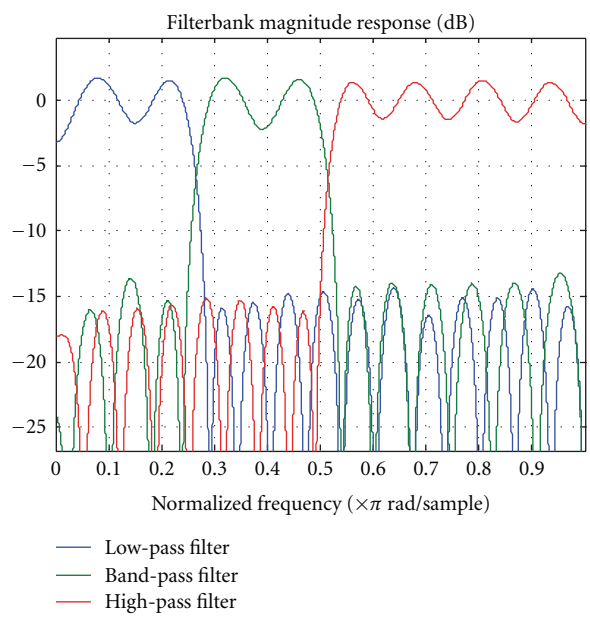

Figure 16: Filterbank used for R-wave detection.

6. Conclusions

We presented two efficient dynamic partial reconfiguration systems that allow us to implement a wide range of 1D FIR filters. Requiring a significant smaller partial reconfiguration region, the first system allows changes to the FIR filter coefficients while keeping the rest of the architecture intact. Using a larger partial reconfiguration region, the second system allows full-filter reconfiguration. This system can be used to switch between FIR filters based on power, performance, and resources considerations.

For both systems, the required partial reconfiguration region is kept small by using Distributed Arithmetic implementations. System performance is evaluated in terms of the dynamic reconfiguration rate. For a representative example, it is shown that we can process over ten Mega samples per second while dynamically reconfiguring about seventy times per second. The introduction of faster dynamic reconfiguration controllers can lead to much higher throughputs for the same number of reconfigurations per second. Alternatively, we can maintain much higher throughputs at much lower reconfiguration rates.

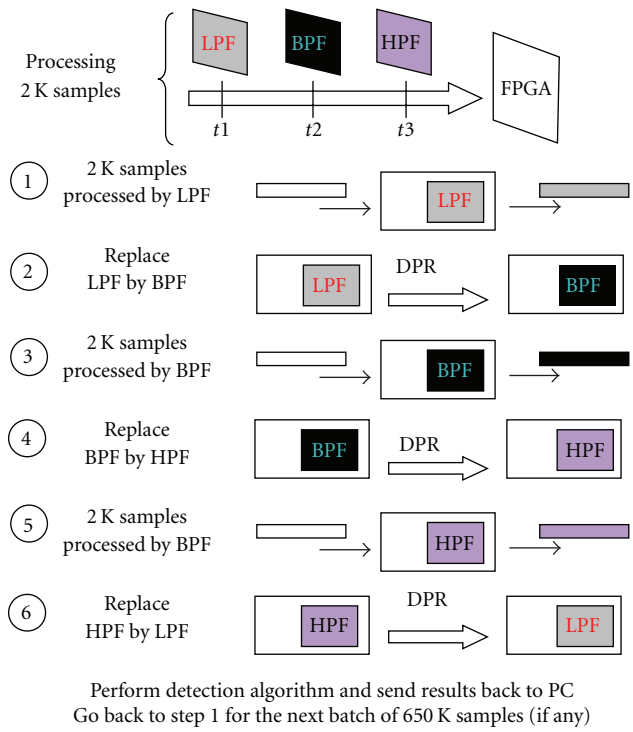

Figure 17: Filterbank data processing. LPF: low-pass filter, BPF: band-pass filter, HPF: high-pass filter.

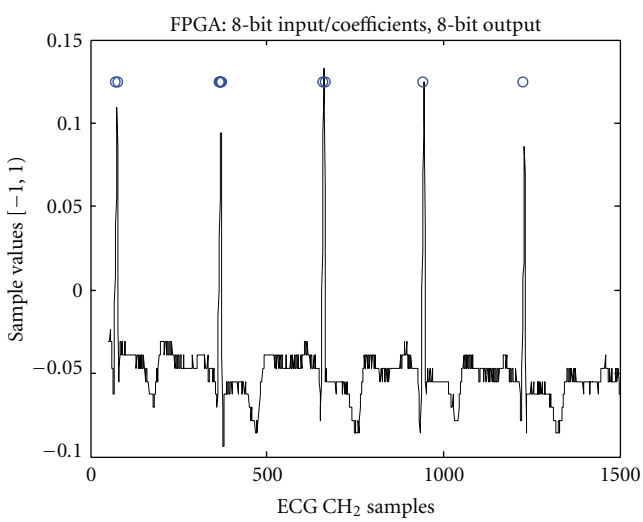

Figure 18: Perfect detection of R-waves for the first 5 ECG cycles.

The results have encouraged us to explore the use of dynamically reconfigurable filtering for digital image and video processing applications. As seen from the results of this paper, it is possible to dynamically reconfigure at real-time frame rates. For such applications, the DPR systems can be extended to separate implementations of 2D dynamically reconfigurable filterbanks.

Acknowledgment

The research presented in this paper has been funded by the Air Force Research Laboratory under Grant no. QA9453-060C-0211.

References

[1] M. Hatamian and G. L. Cash, "A 70 MHz 8 bit x 8 bit parallel pipelined multiplier in 2.5 μm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 21, no. 4, pp. 505–513, 1986.

[2] K. G. Smitha and A. P. Vinod, "A reconfigurable high-speed RNS-FIR channel filter for multi-standard software radio receivers," in *Proceedings of the 11th IEEE Singapore International Conference on Communication Systems (ICCS '08)*, pp. 1354–1358, Guangzhou, China, 2008.

[3] F. Gallazzi, G. Torelli, P. Malcovati, and V. Ferragina, "A digital multistandard reconfigurable FIR filter for wireless applications," in *Proceedings of the 14th IEEE International Conference on Electronics, Circuits and Systems (ICECS '07)*, pp. 808–811, December 2007.

[4] H. Bruce, R. Veljanovski, V. Öwall, and J. Singh, "Power optimization of a reconfigurable FIR-filter," in *Proceedings of the IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 321–324, October 2004.

[5] R. Mahesh and A. P. Vinod, "New reconfigurable architectures for implementing FIR filters with low complexity," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 275–288, 2010.

[6] C. Chou, S. Mohanakrishman, and J. B. Evans, "FPGA implementation of digital filters," in *Proceedings of Signal Processing Applications Technology*, Santa Clara, Calif, USA, 1993.

[7] T. Do, H. Kropp, C. Reuter, and P. Pirsch, "A flexible implementation of high-performance FIR filter on xilinx FPGAs," in *Proceedings of the 8th International Workshop on Field-Programmable Logic and Applications. From FPGAs to Computing Paradigm (FPL '98)*, pp. 441–445, 1998.

[8] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," *IEEE Transactions on Acoustics, Speech and Signal Processing Magazine*, vol. 6, no. 3, pp. 4–19, 1989.

[9] G. A. Vera, D. Llamocca, M. S. Pattichis, and J. Lyke, "A dynamically reconfigurable computing model for video processing applications," in *Proceedings of the 43rd Asilomar Conference on Signals, Systems and Computers*, pp. 327–331, Pacific Grove, Calif, USA, November 2009.

[10] D. Llamocca, M. Pattichis, and A. Vera, "A dynamically reconfigurable parallel pixel processing system," in *Proceedings of the 19th International Conference on Field Programmable Logic and Applications (FPL '09)*, pp. 462–466, Prague, Czech Republic, August-September 2009.

[11] D. Llamocca, M. Pattichis, and A. Vera, "A dynamically reconfigurable platform for fixed-point FIR filters," in *Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig '09)*, pp. 332–337, Cancun, Mexico, December 2009.

[12] E. C. Tan, A. Wahab, and K. K. Wong, "Programmable DSP systems using FPGA," in *Proceedings of the IEEE Conference on Digital Signal Processing Applications*, vol. 2, pp. 654–658, Perth map, Australia, November 1996.

[13] R. Sidhu and V. K. Prasanna, "Efficient metacomputation using self-reconfiguration," in *Proceedings of the 12th International Conference on Field-Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream (FPL '02)*, pp. 85–92, Montpellier, France, 2002.

[14] J.-P. Delahaye, J. Palicot, C. Moy, and P. Leray, "Partial reconfiguration of FPGAs for dynamical reconfiguration of a software radio platform," in *Proceedings of the 16th IST Mobile and Wireless Communications Summit*, pp. 1–5, Budapest, Hungary, July 2007.

[15] T. Rissa, R. Uusikartano, and J. Niittylahti, "Adaptive FIR filter architectures for run-time reconfigurable FPGAs," in *Proceedings of IEEE International Conference on Field-Programmable Technology*, pp. 52–59, 2002.

[16] C. S. Choi and H. Lee, "An reconfigurable FIR filter design on a partial reconfiguration platform," in *Proceedings of the 1st International Conference on Communications and Electronics (ICCE '06)*, pp. 352–355, Hanoi, Vietnam, October 2006.

[17] C. S. Choi and H. Lee, "A self-reconfigurable adaptive FIR filter system on partial reconfiguration platform," *IEICE Transactions on Information and Systems*, vol. E90-D, no. 12, pp. 1932–1938, 2007.

[18] K. Chapman, *Constant Coefficient Multipliers for the XC4000E*, Xilinx, San Jose, Calif, USA, 1996, Xilinx's Application Note 054.

[19] *Distributed Arithmetic FIR Filter (DS240)*, Xilinx, San Jose, Calif, USA, v9.0 edition, 2005.

[20] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hübner, and J. Becker, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL '08)*, pp. 535–538, Heidelberg, Germany, September 2008.

[21] G. A. Vera, *A dynamic arithmetic architecture: precision, power, and performance considerations*, Ph.D. dissertation, University of New Mexico, Albuquerque, NM, USA, May 2008.

[22] K. Bruneel, F. Abouelella, and D. Stroobandt, "Automatically mapping applications to a self-reconfiguring platform," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE '09)*, pp. 964–969, April 2009.

[23] S. Chevobbe and S. Guyetant, "Reducing reconfiguration overheads in heterogeneous multicore RSoCs with predictive configuration management," *International Journal of Reconfigurable Computing*, vol. 2009, Article ID 390167, 7 pages, 2009.

[24] *Implementing FIR Filters in FLEX Devices (AN73)*, Altera Corp., San Jose, Calif, USA, v1.01 edition, 1998.

[25] *Fast Simplex Link (FSL) Bus Product Specification (DS449)*, Xilinx, San Jose, Calif, USA, v2.11a edition, 2007.

[26] *Early Access Partial Reconfiguration User Guide for ISE 9.204i (UG208)*, Xilinx, San Jose, Calif, USA, v1.2 edition, 2008.

[27] A. DeHon, R. Huang, and J. Wawrzynek, "Hardware-assisted fast routing," in *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '02)*, April 2002.

[28] J. S. Sahambi, S. N. Tandon, and R. K. P. Bhatt, "Using wavelet transforms for ECG characterization," *IEEE Engineering in Medicine and Biology Magazine*, vol. 16, no. 1, pp. 77–83, 1997.

[29] E. Kyriacou, C. Pattichis, M. Pattichis et al., "An m-Health monitoring system for children with suspected arrhythmias," in *Proceedings of the 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS '07)*, pp. 1794–1797, 2007.

[30] A. Panayides, M. S. Pattichis, C. S. Pattichis, C. P. Loizou, M. Pantziaris, and A. Pitsillides, "Robust and efficient ultrasound video coding in noisy channels using H.264," in *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS '09)*, pp. 5143–5146, Minneapolis, MN, USA, September 2009.

[31] G. B. Moody and R. G. Mark, "The MIT-BIH Arrhythmia Database on CD-ROM and software for use with it," in *Proceedings of Computers in Cardiology*, pp. 185–188, September 1990.