A Dynamically Reconfigurable Platform for Fixed-Point FIR Filters

Daniel Llamocca, Marios Pattichis
Electrical and Computer Engineering Department
The University of New Mexico
Albuquerque, NM, USA
dllamocca@ieee.org, pattichis@ece.unm.edu

G. Alonzo Vera
Microelectronics Research and Development Corp.
Albuquerque, NM, USA
alonzo@ieee.org

***Abstract*— Many DSP, image and video processing applications use Finite Impulse Response (FIR) filters as basic computing blocks. Our paper introduces an efficient dynamically reconfigurable FIR system that can adapt the number of filter coefficients, and their values, in real time. Here, dynamic reconfiguration is used to switch between different, pre-computed, fixed-point realizations of different digital filters. Our platform relies on the use of Distributed Arithmetic blocks, mapped to the specific LUTs of the underlying FPGA. Dynamic reconfiguration of the coefficients is limited to changing a small number of relevant LUT contents, while leaving the rest of the architecture intact. We investigate the dynamic system throughput as a function of the dynamic reconfiguration rate.**

Keywords: FPGA, dynamic partial reconfiguration, distributed arithmetic, FIR filters, hardware.

I. INTRODUCTION

Over the last three decades, there have been a large number of static implementations of one dimensional digital filters, both in CMOS technology (e.g. [1]) and reconfigurable hardware (e.g. [2]).

Highly-efficient, multiplier-less, FIR filter implementations can be achieved using the Distributed Arithmetic (DA) technique [3]. This approach however, requires the coefficients to be fixed or hardwired in the filter implementation.

This paper introduces an efficient dynamically reconfigurable platform for implementing fixed-point FIR filters using DA, whose coefficients can be dynamically changed at run-time. Dynamic Partial Reconfiguration (DPR) [4] is used to modify otherwise fixed filter coefficients at the expense of partial reconfiguration time overhead. The efficiency of DPR relies on the fact that it does not require the device to be turned off and that only a portion of the device is reconfigured, which saves time and power with respect to full static reconfiguration.

The introduction of a dynamically reconfigurable platform allows us to re-use resources in real-time. A dynamic reconfiguration can be initiated from a desire to implement a new filter, based on power or resources considerations, or simply to implement new functionality. While we will not investigate such specific applications, we will investigate performance bounds based on the reconfiguration rate. We believe that these performance bounds can be used to understand a wide variety of possible future applications.

Our proposed system is based on dynamically reconfiguring at the finest possible level, the LUTs that store the coefficients, with a small dynamic reconfiguration area. This approach of dynamically modifying LUT-based structures was also presented for a dynamically reconfigurable pixel processor in [5].

The rest of the paper is organized as follows: Section II describes the stand-alone FIR Filter architecture, its parameters, its operation, and its benefits. Section III describes the dynamic embedded system created with the FIR Filter as the core, details its operation, and explains how DPR is performed on this system. Section V provides results in terms of: 1) resource utilization, 2) accuracy of the fixed-point FIR Filter implementation, and 3) throughput as a function of the reconfiguration rate. Section V summarizes the paper and discusses further work.

II. A GENERAL DA FIR FILTER IMPLEMENTATION

A high performance FIR implementation based on Distributed Arithmetic is described in this section. We encoded our approach directly in VHDL, so as to achieve a level of portability. When Xilinx ® FPGAs are employed, the code makes use of its specific LUT primitives. We will consider a dynamic adaptation of this system in Section III.

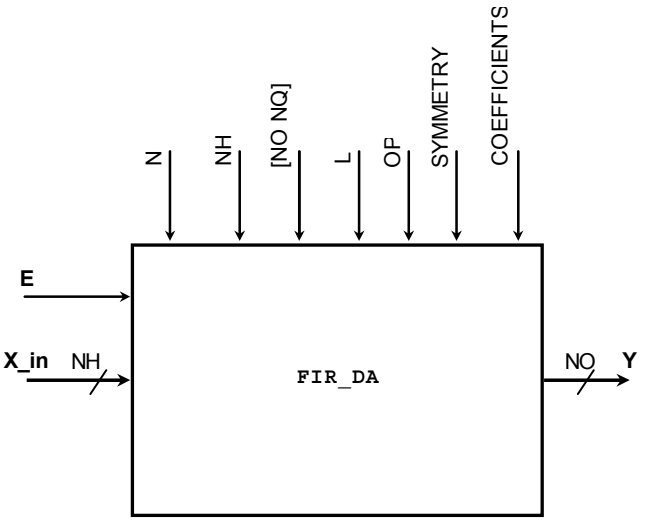

Figure 1. Generalized FIR DA Module

978-0-7695-3917-1/09 $26.00 © 2009 IEEE
DOI 10.1109/ReConFig.2009.43

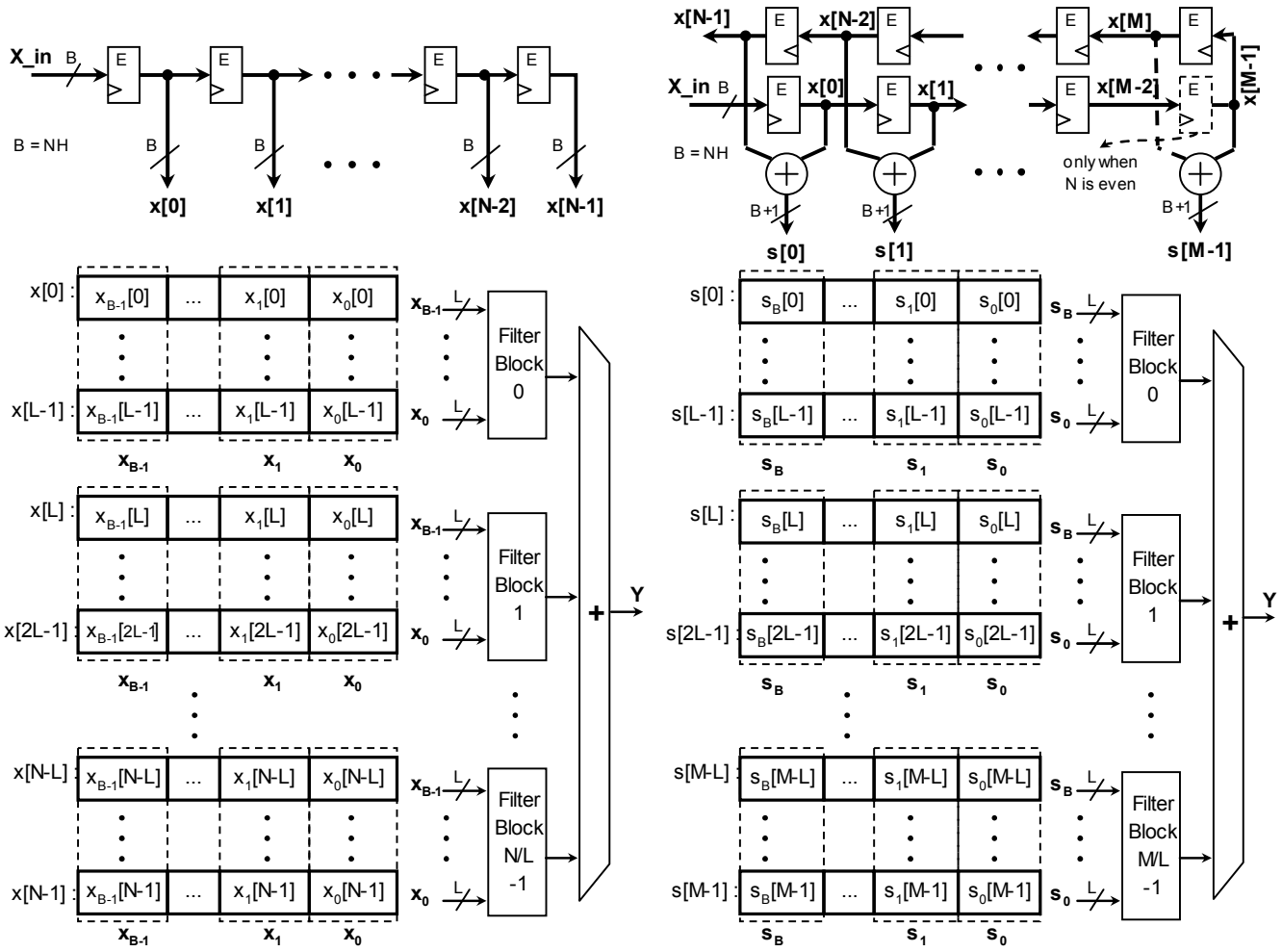

Figure 2. High-performance DA implementation based on the underlying LUT input size (L) Non-symmetric Filter (left), Symmetric Filter (right)

A. Description

Fig.1 shows the FIR Filter module with its inputs, outputs and parameters. Signal 'E' controls the input validity. Fig. 2 depicts two implementation schemes depending on the filter symmetry. For symmetric filters, we halve the number of effective taps by adding the corresponding samples [6]. Non-symmetric filters are also considered, since they are useful in many instances, e.g. polyphase filter implementations.

Here, N denotes the number of taps, NH represents the input values and coefficients bit width, L is the LUT input size, OP is the output truncation scheme (LSB Truncation then Saturation, LSB and MSB Truncation, and no Truncation), [NO NQ] denotes the output format: NO bits with NQ fractional bits, and the coefficients are given in a text file. We define $M = \lceil N/2 \rceil$, $sizeI = NH + 1$ for symmetric filters, and $M = N$, $sizeI = NH$ for non-symmetric filters.

The inputs/coefficients format is set at [NH NH-1], i.e. the values are restricted to $[-1,1)$. As a result, the maximum number of output integer and fractional bits results:

$$[\,2(NH-1)+\lceil \log_2(N+1)\rceil+1 \qquad 2(NH-1)] \tag{1}$$

B. FIR DA Implementation

The Distributed Arithmetic technique rearranges the input sequence (be it x[n] or s[n]) into vectors of length M, which require an array of *sizeI* M-input LUTs. This becomes prohibitively expensive when M is large. For efficient implementation, we divide the filter into M/L filter blocks [6], as illustrated in Fig. 2. Each filter block has L coefficients requiring *sizeI* L-input LUTs (in Fig.2 each vector of size L goes to one L-input LUT). Table 1 illustrates the resource savings attained when performing division of the filter in filter blocks. The advantage of the FIR Filter block is that it allows for efficient routing while mapping the implementation to the specific LUT primitives found in an FPGA. As shown in [5], the approach is scalable in that it can be easily ported to different LUT sizes.

TABLE 1. LUT SPACE COMPARISONS

	Total space required
1 Filter block of size M LUTs have M inputs	$sizeI \times 2^M$ words
M/L filter blocks of size L LUTs have L inputs	$sizeI \times 2^L \times M/L$ words

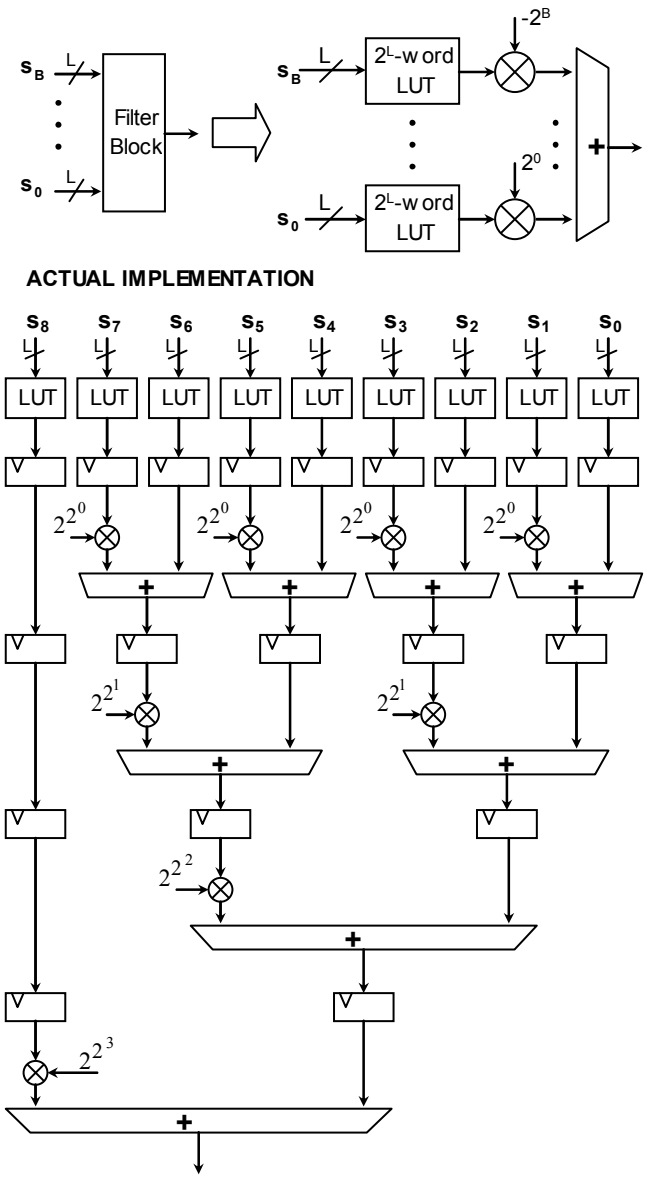

Figure 3. Filter Block architecture. NH = 8

From Table 1, we can see that if, for example, M = 16, L = 4, then $2^{16} >> 2^4$x16/4. Thus, dividing the filter into M/L filter blocks saves a great deal of hardware resources. It only needs an extra adder structure, (see Fig. 2), to add up all filter block outputs.

Fig. 3 depicts the internal pipelined architecture of a Filter Block when SYMMETRY = YES. It consists of an L-input LUT array, an adder tree, shifters, and registers. The number of register levels is given by the following formula:

$$\#\text{ of register levels in Filter Block} = \lceil \log_2(\text{sizeI}) \rceil \quad (2)$$

Fig. 4 shows the structure of a L-input LUT (like those shown in Fig. 3). The word size (output bits) of each L-input LUT was computed as $LO = NH + \lceil \log_2(L) \rceil$. It also shows its decomposition into LO L-to-1 LUTs, useful for efficient FPGA implementation.

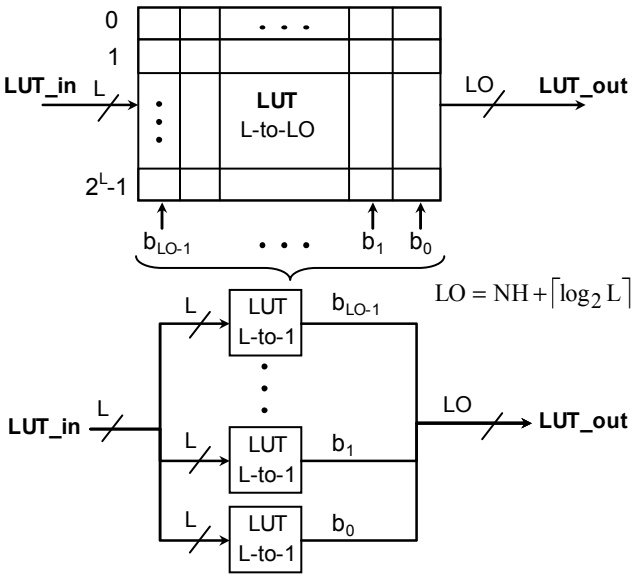

Figure 4. Decomposition of a L-to-LO LUTs into LO L-to-1 LUTs

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 [5] for Virtex-4, optimal LUT implementations can also be obtained for L = 5, 6, 7, 8.

Fig. 5 depicts the internal pipelined architecture of the extra adder structure that adds up all the Filter Block outputs. A final output register is also shown. The number of Filter blocks is set to M/L = 4. The number of register levels of the adder structure is given by:

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

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

The FIR Filter architecture has an input-output delay of $REG\_LEVELS = \lceil \log_2(\text{sizeI}) \rceil + \lceil \log_2(M/L) \rceil + 2$ cycles, i.e. REG_LEVELS is the number of register levels between input and output.

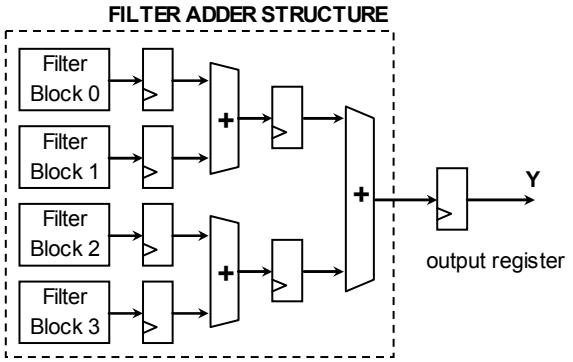

Figure 5. Final tree adder structure. M/L = 4

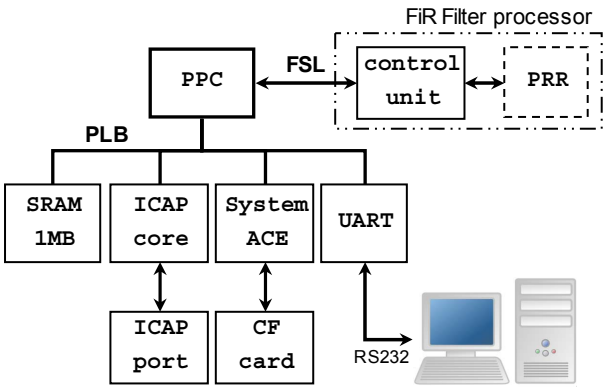

Figure 6. System Block Diagram

III. DYNAMIC EMBEDDED FIR FILTER SYSTEM

The FIR filter processor consists of the FIR Filter core and a state machine that provides interfacing with the Fast Simplex Link (FSL) bus. We process 4 bytes in parallel since the FSL bus width is 32 bits. The system can be dynamically reconfigured to perform an arbitrary filter response.

A. System Architecture

Fig. 6 shows the system block diagram. The dynamic FIR Core and the PowerPC (PPC) are linked by the high speed FSL Bus. The Partial Reconfiguration Region (PRR) holds the LUTs for every filter block and it is dynamically reconfigured via the internal configuration access port (ICAP), driven by the ICAP controller core. The SRAM stores volatile data needed at run-time, e.g.: input streams, processed streams and partial bitstreams. SystemACE reads a Compact Flash (CF) Card that stores the partial bitstreams and input streams at power-up. The processed streams are written back into the SRAM for throughput measurements. To verify correctness, the processed stream is stored in the CF Card for further evaluation. The UART interface provides a display interface for throughput measurements and current system status.

Fig. 7 depicts the internal architecture of the FIR Filter processor. The PRR is made of $(M/L)\times sizeI$ L-to-1 LUTs; the PRR I/Os are registered as the reconfiguration guidelines advise [7].

A FIR Filter with N coefficients and NX input values can output a maximum of NX+N-1 values. The state machine provides three ways of selecting the output values:

- We output the first NX values. This mode is useful for common 1D signals.
- We output NX values in the range $\lfloor N/2 \rfloor + 1 : NX + \lfloor N/2 \rfloor$. This is the case when performing 2D separable convolution on images.
- Streaming mode with infinite number of input samples, i.e. NX = ∞.

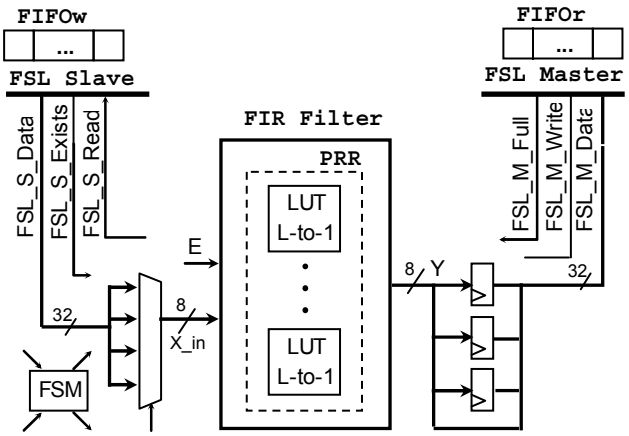

Figure 7. FIR Filter processor in detail

B. FIR Filter system operation

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 [8], 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. 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, i.e. 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).

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

C. Dynamic Partial Reconfiguration Setup

All signals between the dynamic region and the static part are connected by pre-routed Bus Macros in order to lock the wiring [7]. To perform DPR, the partial bitstreams are read from a CF card and stored in SRAM. When needed, they are written to the ICAP port. This fairly simple technique is explained in [9].

IV. RESULTS

A. Stand-Alone FIR Filter core

Fig. 8 shows hardware utilization and Fig. 9 shows the maximum frequency of operation. Both curves are a function of the number of taps and the output format, with OP = 0, L = 4, SYMMETRY = YES. We use the XC4VFX20-11FF672 Xilinx® Virtex-4 device, with 8544 Slices. The architecture exhibits small resource utilization (worst case: 73% of the device) and frequencies of operation above 200 MHz.

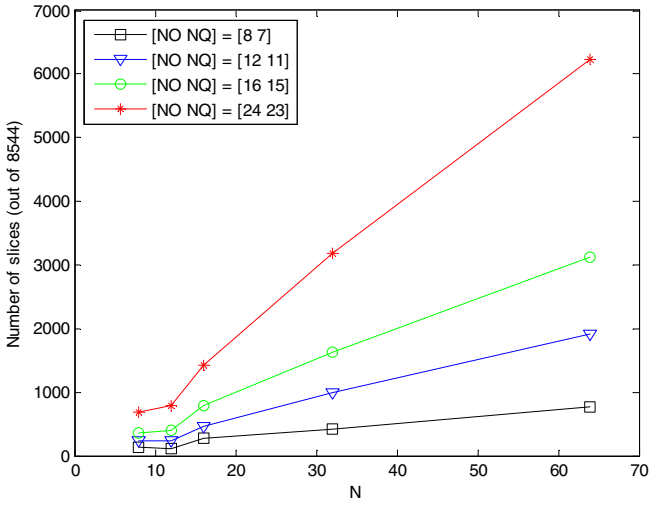

Figure 8. Resources vs. number of coefficients and output fixed-point representation

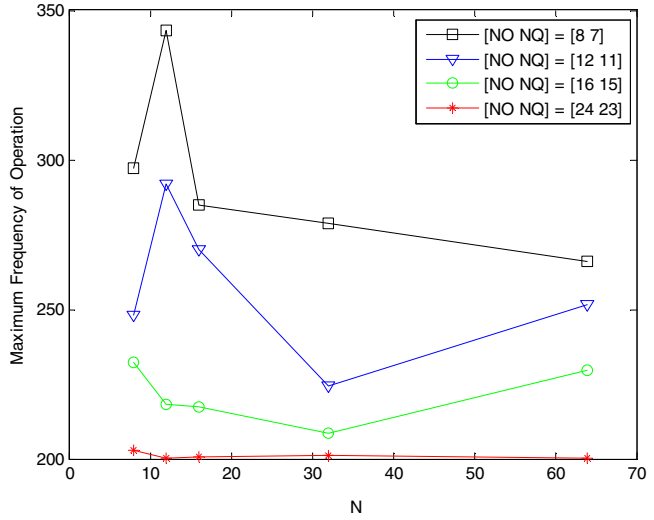

Figure 9. Max. Frequency of operation (MHz) vs. number of coefficients and output fixed-point representation

An error analysis is performed for the same parameters. Fig. 10 shows the relative error curves for two cases (input stream = 1024 sinusoid samples). The error metric is:

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

Fig. 10 shows the relative error to be below 5%. The peaks occur when FPGA values are zero and the ideal values are close to zero, resulting in a deceptive 100% error.

B. *Dynamic Embedded System*

The results are shown for: N = 32, NH = 8, [NO NQ] = [8 7], L = 4, OP = 0, SYMMETRY = YES (parameters defined in Subsection II.A). The Partial Reconfiguration Area is made of $(M/L)\times sizeI = 36$ 4-to-1 LUTs. It occupies a tightly packed area of 8x45=360 CLBs with a 48000 bytes bitstream. Also, the dynamic system is tested in the 1D signals mode, i.e. only the first NX outputs are considered.

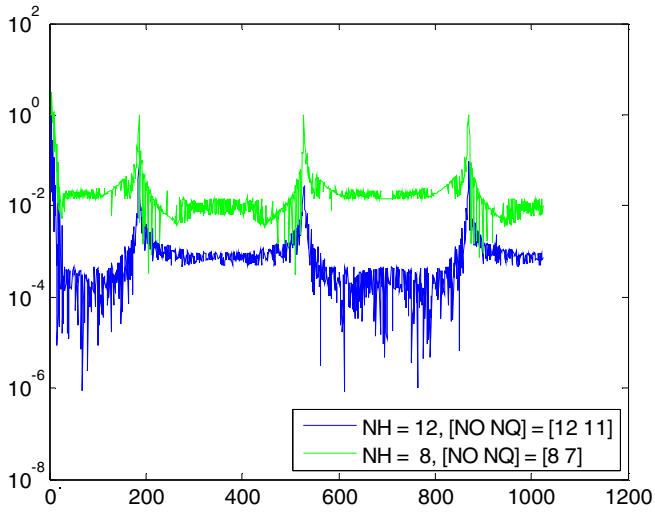

Figure 10. Relative error, N = 32. Two bit width cases

The system is implemented in 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.

1) Hardware resource utilization: Table 2 shows the hardware resource utilization of the static part, dynamic region and the whole system. The Static Region includes the FIR Filter processor and every other peripheral controller shown in Fig. 6. ST_FIR is the portion of the Static Region that implements only the FIR Filter Processor.

2) FIR Filter Processor performance bounds: The maximum throughput of the FIR Filter processor is given by:

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

Note that since the system is pipelined, there is an initial latency that will become negligible over time. Actual throughput depends on many factors, e.g.: cache size, PPC instruction execution, and FSL usage. The maximum throughput can not be attained since the PPC can not read and write into the FIFOs at the same time.

3) Reconfiguration Time: Table 3 shows the reconfiguration time for 3 scenarios. In our setup, called Scenario 1, we used the Xilinx® ICAP core and obtained a reconfiguration time of 25.96 ms yielding a reconfiguration speed of 3.23 MB/s. The reconfiguration time of Scenario 2 is computed based on the speed results reported in [10]. The dramatic improvement in reconfiguration speed lies on the use of a custom ICAP controller, DMA access, and burst transfers. Scenario 3 is the maximum theoretical throughput, which for Virtex-4 is 400 MB/s [9].

TABLE 2. HARDWARE UTILIZATION ON VIRTEX-4 XC4VFX20-11FF672

Module	FF	(%)	Slice	(%)	LUT	%
PRR	0	0%	948	11%	985	5%
Static Region	3326	19%	4399	51%	6307	36%
ST_FIR	1286	7%	815	9%	626	3%
Overall	3326	19%	5347	62%	7292	41%

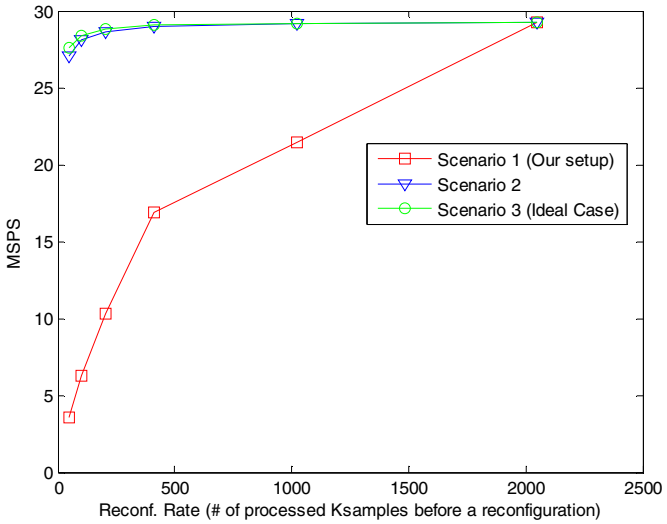

Figure 11. Throughput vs. Reconfiguration Rate, N = 32

4) Throughput measurements: The system operation when measuring throughput was explained in Subsection III.B. By using timer functions in a software routine, time was measured from the moment we start reading the input stream from the SRAM until the processed stream is written in the SRAM. Out input streams consist of sinusoids.

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. We report the average throughput over the 100 runs. Reconfiguration rate is defined as the number of processed samples before a reconfiguration is performed. Throughput is defined in MSPS (Mega samples per second).

Fig. 11 shows the dynamic performance over 100 runs. There are 3 curves that correspond to the 3 scenarios shown in Table 3. Static performance, i.e. no reconfiguration, corresponds to the value at which the 3 curves converge. 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 3.51 MSPS. The other curves (Scenarios 2 and 3) provide performance bounds based on the static performance and reconfiguration speeds of Table 3.

We see that dynamic performance heavily depends on reconfiguration speed and input stream size. Better reconfiguration speeds offset the reconfiguration time overhead. (Scenarios 2 and 3). Longer data streams help to offset the reconfiguration time overhead.

TABLE 3. RECONFIGURATION TIME FOR A 84KB BITSTREAM

Scenario	Reconfiguration Speed	Reconfiguration Time
1. Current	3.23 MB/s	25.96 ms
2. Custom [10]	295.4 MB/s	0.284 ms
3. Ideal	400 MB/s	0.210 ms

V. CONCLUSIONS AND FURTHER WORK

A high-performance and fully parameterizable FIR Filter has been presented. The small resource utilization is due to the fixed-nature of coefficients and the efficient use of the Distributed Arithmetic technique. The dynamic implementation was evaluated in terms of performance and resource consumption. It was shown that besides the obvious factor of the reconfiguration speed, the dynamic performance depends on the input stream size. In general, the dynamic performance is less sensitive to the effects of reconfiguration time overhead as the stream size increases.

Further work consists on making the number of taps dynamic so as to control power dynamically. On dynamic performance, we can dramatically improve the reconfiguration time by using a faster ICAP controller, demonstrated by Scenario 2 [10] in Fig. 11. On static performance, there is room for improvement: simulation runs show that the processor takes a long time writing and reading on the FSL bus. By using DMA and PLB burst transfers so as to stream data to the system at a faster rate, better performances can be achieved. In future applications, the dynamic system can be used to switch between FIR filters based on power, performance, resources considerations.

ACKNOWLEDGMENT

The research presented in this paper has been funded by the Air Force Research Laboratory under grant number 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 J. Solid-State Circuits, vol. SC-21, pp. 505-513, Aug. 1986.

[2] C. Chou, S. Mohanakrishman, and J. B. Evans, "FPGA implementation of digital filters", Proceedings of Signal Processing Applications Technol., Santa Clara, CA, 1993.

[3] S. White, "Applications of Distributed Arithmethic to Digital Signal Processing: A Tutorial Review", IEEE Transactions on Acoustics, Speech and Signal Processing Magazine, 4-19, 1989

[4] J. Becker, M. Hubner, G. Hettich, R. Constapel, J. Eisenmann, and J. Luka, "Dynamic and Partial FPGA Exploitation", Proc IEEE, vol. 95, no. 2, pp. 438-452, 2007.

[5] D. Llamocca, M. Pattichis, and A. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", Proceedings of FPL'2009, Prague, Czech Republic, Sept. 2009.

[6] "Implementing FIR Filters in FLEX Devices (AN73)", v1.01 ed., Altera Corp., 101 Innovation Drive, San Jose CA 95134, Feb. 1998.

[7] "Early Access Partial Reconfiguration User Guide for ISE 9.204i (UG208)", v1.2 ed., Xilinx Inc., 2100 Logic Drive, San Jose CA 95124, Sept. 2008.

[8] "Fast Simplex Link (FSL) Bus Product Specification (DS449)", v2.11a ed., Xilinx Inc., 2100 Logic Drive, San Jose CA 95124, Jun. 2007.

[9] Guillermo A. Vera, "A Dynamic Arithmetic Architecture: Precision, Power, and Performance Considerations", Ph.D. Dissertation, University of New Mexico, Albuquerque, NM, USA, May 2008.

[10] C. Claus, B. Zhang, W. Stechele, L. Braun, M. Hubner, and J. Becker, "A Multi-Platform Controller allowing for maximum dynamic partial reconfiguration throughput", Proceedings of FPL'2008, Heidelberg, Germany, Sept. 2008, pp. 535-538.