A VHDL Core for Intrinsic Evolution of Discrete Time Filters with Signal Feedback

David A. Gwaltney
NASA Marshall Space Flight Center
Huntsville, AL 35812
david.a.gwaltney@nasa.gov

Kenneth Dutton
Jacobs Sverdrup Company
Huntsville, AL 35812
kenneth.dutton@nasa.gov

Abstract

The design of an Evolvable Machine VHDL Core is presented, representing a discrete-time processing structure capable of supporting control system applications. This VHDL Core is implemented in an FPGA and is interfaced with an evolutionary algorithm implemented in firmware on a Digital Signal Processor (DSP) to create an evolvable system platform. The salient features of this architecture are presented. The capability to implement IIR filter structures is presented along with the results of the intrinsic evolution of a filter. The robustness of the evolved filter design is tested and its unique characteristics are described.

1 Introduction

The capability for automated design and adaptation are often cited as the motivation for investigation of intrinsic hardware evolution. Indeed, an evolutionary algorithm's capability to create designs from high level specifications without real knowledge of the underlying configurable medium makes evolvable hardware attractive when seeking autonomy in the generation of innovative designs. In systems where unknown or unanticipated forces may conspire to impair the performance of an electronic design, or render it inoperable, the ability to find a solution using whatever functional components exist draws much attention to evolvable hardware. The work presented here is concerned with the implementation of an evolution accessible discrete time computation medium in reconfigurable hardware, and the application of an evolutionary algorithm to the design of structures for filtering and control. The motivation is to provide an autonomous system that can configure itself to enable on-line design, adaptation and self-repair.

Discrete time filtering is widely used for signal processing and control applications. The same basic structures in finite impulse response (FIR) filters and infinite impulse response (IIR) filters can be applied to low-pass, band-pass and high-pass filtering, dynamic system representation and controllers. For control applications, structures with output feedback, such as the IIR, are preferred because they implement the pole-zero representation used in control system design and analysis. FIR structures (feed-forward) are frequently used in filtering applications due to their ease of implementation, flexibility of attainable filtering and linear phase characteristics. However, the design process for determining FIR coefficients is more involved than that of the design of IIR filters [1]. While this paper reports on the evolution of filters, rather than dynamic system representation or controllers, the goal is to use hardware evolution in control applications, so structures with signal feedback are considered here.

Using hardware evolution for the design of discrete time filters has been a topic of study by other researchers. So has the implementation of adaptive filters for filtering and dynamic system parameter identification. The majority of these investigations use feed-forward structures. The ones involving intrinsic evolution all use feed-forward structures.

In [2], the authors concern themselves with evolving the transfer functions of high order filter designs using multi-objective criterion. The fitness of the designs is determined analytically. Other notable work includes the implementation of FIR filters using primitive operator filter (POF) design [3]. This work utilized a simulated reconfigurable hardware platform that was designed for fault tolerance via redundancy. The platform consisted of an array of programmable arithmetic logic units (PALUs), including a shifter and adder/subtracter. The PALUs operated on 16-bit data words. Evolution was shown to configure the hardware to produce desired results in the presence of induced component faults. In [4], more work was done using POFs to implement FIR filters using a multi-objective approach. This work used automatically generated Verilog netlists to simulate a filter design employing blocks consisting of evolved multipliers and adders. Recently, Particle Swarm Optimization (PSO) was used to identify the parameters of fixed IIR structures used to represent a dynamic system. The simulated results showed that PSO was able to rapidly converge to a reasonable solution, and should be considered for on-line adaptive applications [5]. Other researchers have evolved

digital filters for imaging applications. Using a simulated array of Configurable Function Blocks (CFBs) with a variety of selectable arithmetic and Boolean functions, the authors in [6] evolved image filters designed to remove noise from corrupted images. The CFBs operated on 8-bit data words representing pixels from a digital image. The CFB array is configured such that the data moves forward through the array.

Several researchers have evolved filters intrinsically. FIR filters have been evolved in hardware using an implementation the authors called Complete Hardware Evolution (CHE). In this implementation, the FIR filter structure and a pipelined Genetic Algorithm (GA) were configured on a Field Programmable Gate Array (FPGA). The authors demonstrated the capability to adapt the coefficients of the FIR filter on-line in response to changing input signals [7]. A hardware system for evolution of image filters has been implemented in an FPGA. The system includes a genetic processing unit (GPU) that implements an evolutionary strategy, and an array of processing elements that have selectable Boolean and arithmetic functions. Similar to [6], the array is feed-forward and the processing elements act on 8-bit data words that represent pixels from a digital image. Filters were evolved that removed Gaussian noise from images to recover the original image [8]. Using an array of processing elements with Boolean operators and an embedded evolutionary algorithm, researchers were able to intrinsically evolve a 3x3 bit multiplier and a 4x3 bit multiplier. Here the goal is to develop a design environment that configures an FPGA with a complete evolvable system based on characteristics selected by the user. In this way, the user can tailor the configuration of the reconfigurable elements to the application. At the time of writing, the available reconfigurable element selection was limited to an array of pipelined processing elements [9].

In this paper, the design of Evolvable Machine VHDL Core, which permits feed back of output and interior signals, is presented. This VHDL Core is implemented in an FPGA and is interfaced with an evolutionary algorithm executed in firmware on a Digital Signal Processor (DSP) to create an evolvable system platform. The salient features of this architecture are presented. The goal is to implement a discrete-time processing structure that supports control system applications. The capability to implement IIR filter structures will be presented along with the initial results of intrinsic evolution of a filter on the VHDL Core.

2 Evolvable Machine VHDL Core

The evolvable machine is implemented as a portable VHDL core and will be referred to as the EMVCore in the remainder of the paper. It consists of an array of configurable function tiles. These tiles can be configured to perform one of several different Boolean and arithmetic operations, or functions, on one or two sixteen bit inputs. Both Boolean and arithmetic operations provide the flexibility to implement many different types of overall functions. The tiles are interconnected with neighboring tiles via multiplexed sixteen bit busses. A diagram of the EMVCore configuration used is shown in Figure 1. Each tile is restricted to making interconnections with its four neighbors, except in the case of some edge tiles. In general the input to a tile may be the output of up to two of the neighboring tiles to the north, south, east or west. Outputs from all of the neighboring tiles are connected to any given tile, with the selection of inputs being made by internal multiplexers. A tile may have the output from one of the neighboring cells be selected as input one (in1) and input two (in2) via the multiplexer arrangement. A tile may also use an internally stored value as an input to a function, rather than a signal from a neighboring tile. This allows the use of a constant for arithmetic functions or a mask for bitwise Boolean functions. In the case of some edge tiles, inputs may come from other sources. Each tile requires 3 configuration data words; a 4 bit word for external signal select, a 7-bit word for selection of the tile function input and operator and a 16-bit word for the internal data constant.

A 3x3 array of nine functional tiles is used, with delay registers on either side of the array. In Figure 1, each tile is labeled in the lower left hand corner with its location in the array. It is labeled in the upper right hand corner with the number of multiplexed inputs to that tile. The functional tiles can be configured with the functions listed in Table 1.

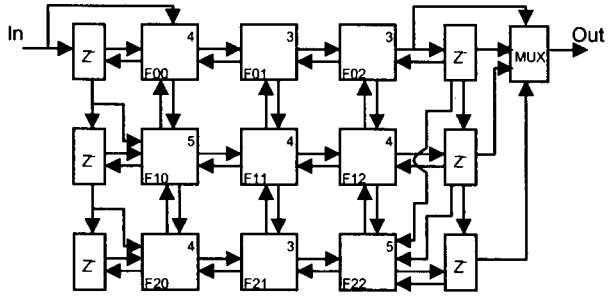

Figure 1 Diagram of the EMVCore configuration

A goal of this work is to implement discrete time filters and controllers, so delay registers are included at the west and east ends of the array. These delay registers can be cascaded to provide the delayed input and output values that are needed for a discrete time filter or controller. These delay registers can also be used to delay the output of neighboring function tiles. Note that the tiles labeled

F00, F20 and F22 have multiplexed inputs from several of the input/output delay registers. F00 has a direct connection to the input. To allow the EMVCore to be used in sampled data applications, there is a programmable clock divider that is used to set the data sampling rate in the delay registers in relation to the rate of data propagation through the tile array. Since each tile output is registered, a clock is required to move the data through the array. The programmable clock divider is a means of setting the number of clocks provided to the tile array between data samples. The use of the EMVcore in a digital filtering application will be covered in Section 4. The size of the tile array can be easily increased by replication of existing components used in the VHDL design.

Code	Function
0	out = 0
1	out = in1
2	out = in2
3	out = in1 and in2
4	out = not(in1 and in2)
5	out = in1 or in2
6	out = not(in1 or in2)
7	out = in1 + in2
8	out = in1 - in2
9	out = in1 x in2

Table 1 Configurable operations for function tiles

Because feedback paths are allowed to be established, the outputs of the tiles are registered. Otherwise, when the VHDL design code is synthesized, the worst case timing is excessive due to the long signal paths that are possible. Further, the Xilinx XST synthesis tool complains about the possibility for what amounts to direct connection from cell output to input. This is typically not done in digital design because the results are usually considered undesirable. In the case of the EMVCore, the possibility for long signal paths would make the timing variable and unpredictable, since the worst case path may be selected via evolution for some individuals, but not for others. While only outputs that can be fed back to tiles in a previous column need to be registered, all outputs are registered to maintain predictable timing within the tile array.

The EMVCore is implemented on a Xilinx Virtex XCV600E Field Programmable Gate Array (FPGA). The design consumes 47% of the CLB slices available on the FPGA.

3 Evolvable System Architecture

The FPGA configured with the EMVCore is part of an evolvable system architecture consisting of a carrier board for the FPGA, a Digital Signal Processor based single board computer (DSP SBC), A Personal Computer (PC) and the software to execute an evolutionary algorithm and to provide a graphical user interface (GUI). A diagram of the configuration is shown in Figure 2.

The DSP SBC is a custom designed board using a Texas Instruments TMS320VC33PGE150 floating point device, and the carrier board for the FPGA is a Xilinx AFX-PQ-240 prototyping board. The DSP SBC communicates with the PC through a RS-232 serial link, so the user can control and observe the execution of the evolution. The DSP SBC interfaces with the EMVcore via the DSP external memory interface, so the EMVcore is addressed like memory space. The clocks for the EMVcore are generated by the DSP to provide complete control over the data flow through it. Input data can be supplied to the EMVcore externally or from the DSP. In either case the DSP can access the input and output data for use in the evolutionary algorithm.

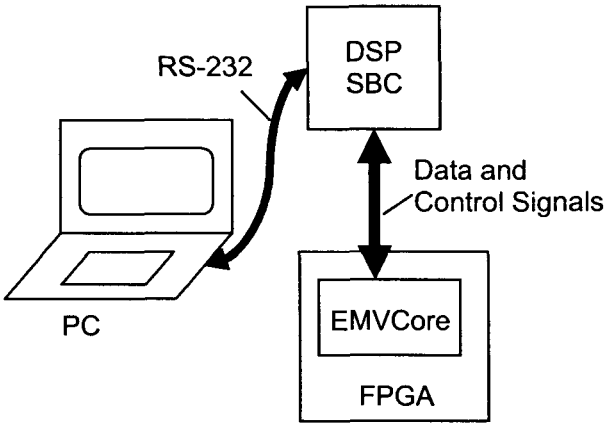

Figure 2 Diagram of the Evolvable System Architecture

The firmware that executes the evolutionary algorithm is resident on the DSP. At this time, it is using a genetic algorithm (GA) with selection, crossover, mutation and fitness operators that are selected by the user. The firmware design is modular so that the user can add, or exchange, operator modules as desired. For instance, users can select from proportional, tournament or stochastic universal sampling selection. There are a variety of crossover and mutation operators as well. Users can define fitness operator modules and then add them to a menu for later selection as desired in experiments. The configuration and execution of the GA is managed through the GUI.

The GUI allows the user to control the evolution and to observe progress. The GUI provides the capability for user selection of evolutionary operators from lists of available choices as shown in Figure 3. The user can start, stop, modify and restart the evolution at any time.

Templates for the GA configuration can be created and loaded. The user can stop the evolution and save the GA configuration and population, then later reload it to start where the evolution was stopped. The GUI displays plots of fitness and average fitness, as well as the output from the EMVcore.

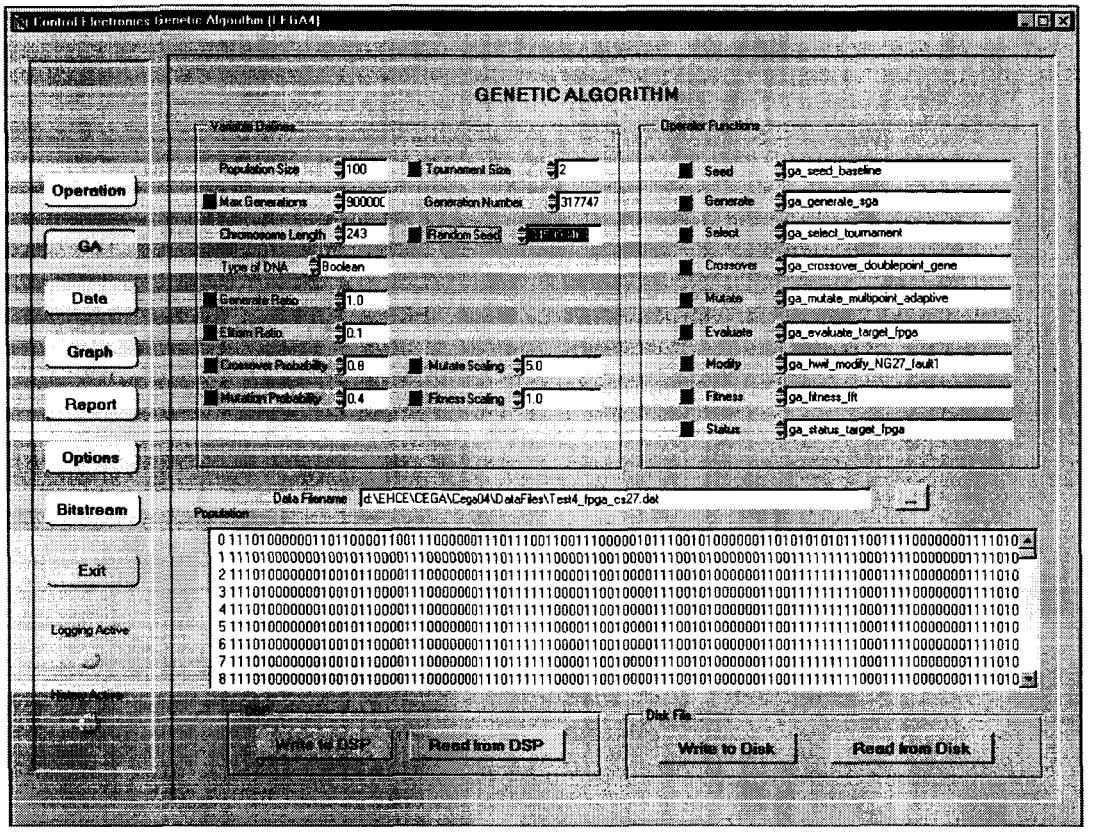

Figure 3 Screen shot of the GUI page for selection of evolutionary operators

4 Second Order Filter Test Case

In order to verify the EMVcore design, and provide a test case for evolutionary design, a second order infinite impulse response (IIR) filter was implemented. The IIR filter can be implemented using the Direct Form II Transpose structure shown in Figure 4 [1]. The general equation that is represented by this diagram is given in Equation 1.

$$y(k) = \sum_{i=0}^{2} b_i x(k-i) - \sum_{i=1}^{2} a_i y(k-i) \qquad (1)$$

In equation 1, the index k represents sampling instants in time as multiples of the signal sampling period. The filtering characteristics of this equation are invariant at different sampling rates, with the pass bands being set by the sampling rate (period) and the coefficients, a_i and b_i.

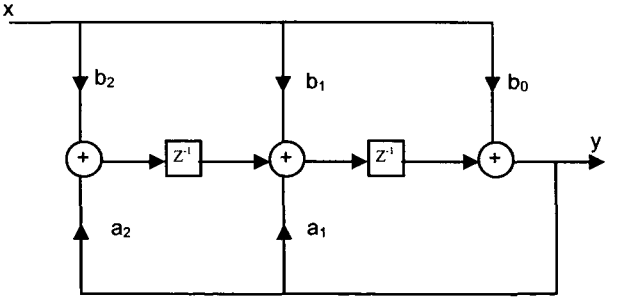

Figure 4 Direct form II transpose structure for 2nd order IIR filter

A low-pass filter designed to provide a cut-off frequency of 2 KHz at a sampling rate of 10 KHz (sampling period 0.1

milliseconds), will provide a cutoff frequency of 20 KHz at a sampling rate of 100 KHz. This feature is utilized in this test case in that the input data is sampled at a specific rate and the filter is designed for that sampling rate, but the filter is tested with a sampling rate that is much higher (230 KHz) using stored data input samples supplied as input at each sample update time.

For the filter test case, a second order low pass filter designed for a cut-off frequency of 200Hz at a sample rate of 5 KHz is implemented. The filter coefficients are selected using the Mathworks MATLAB Filter Design and Analysis tool to give a filter that sets b_2 to zero, thus requiring only four coefficients for implementation [10]. The coefficients are listed in Table 2.

Coefficient	Value
b_0	0.02651057178
b_1	0.02651057178
b_2	0.0
a_1	-1.647848081
a_2	0.7008692244

Table 2 Coefficients for the low pass filter

The EMVcore is configured as shown in Figure 5. This figure illustrates the use of internally stored constants as inputs to tiles F00, F10, F12 and F22 to create the multiplication of the sampled input and output data by a coefficient. The implementation requires eight tiles with F20 being left unused. In this test configuration, the register controlling the configuration of the delay registers is set to cascade the sampled input and output data through the delay registers. Also, the value of the clock divider register is set to six. This means input and output data is sampled on every 7th clock pulse. The tile array is given 6 clock pulses to settle to a calculated value.

A two's complement data representation is used for the sampled data and the coefficients. The sixteen bit data word represents a floating point number in the range [-2,(2-$1/2^{15}$)] or an integer in the range [-32768, 32767]. The floating point coefficients are converted to the two's complement format by dividing them by two and then converting to two's complement. Figure 6 shows input and output data time and frequency response. The data is presented in the signed integer format, as this is utilized in fitness calculations during evolution. The input to the filter is an added sine wave with two frequencies, one at 100 Hz and the other at 500 Hz. The input sine wave was generated in MATLAB and sampled at a rate of 5 KHz to give a 128 point input vector that could be presented to the EVMCore by the DSP SBC firmware. The data plot includes an FFT for both input and output and clearly shows the filtering characteristics of the test case filter implemented on the EVMCore.

Second order filters are frequently used as building blocks in digital filtering applications. A cascade of second order filters can be used to minimize deviation in desired filtering characteristics due to coefficient quantization effects in high order filters [1]. This test case will serve as the target response for evolution in the next section.

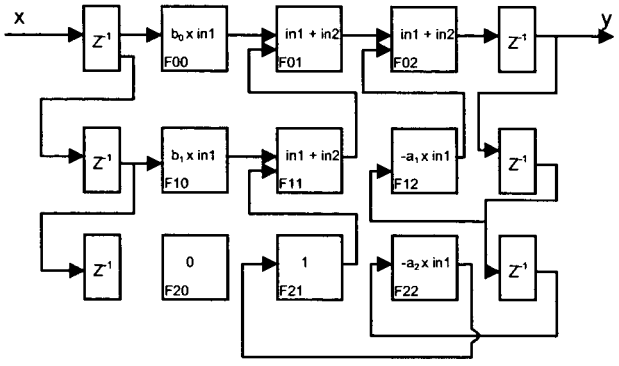

Figure 5 Diagram of the low pass filter implementation on the EMVCore

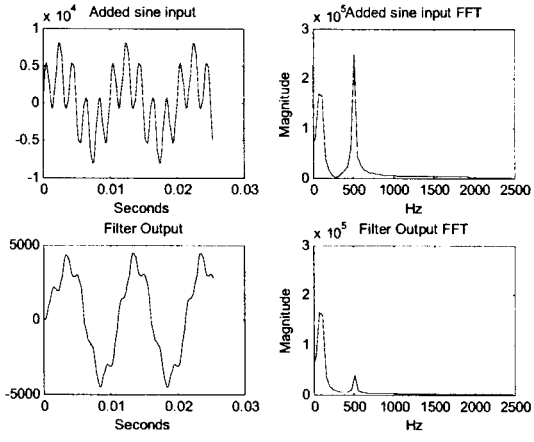

Figure 6 Time and frequency response for low pass filter implemented on the EMVCore

5 Evolved Filter

Evolution is conducted using the evolvable system architecture presented in Section 3. The added sine input from the test case is used again as input to the EVMCore. The goal is to provide the same filtering characteristics as the test case filter. So, the FFT magnitude of the output of the evolved configurations for EVMCore is compared directly with the FFT magnitude of the filter output from the test case in Section 4. Fitness is based on the square of the error between the frequency responses.

The chromosome consists of the 27 configuration data words required for the 3x3 array of tiles. The chromosome is structured such that the three data words required to configure each of the tiles are grouped together. Taken as a group, there are nine groups of data ordered row wise, so the configuration for F00 appears first, then F01, then F02, then F10 and so on. The registers controlling the configuration of the delay registers and the clock divider are fixed at the same configurations used in the test case.

For the evolution, tournament selection with a size of 2 is used, along with a two point crossover operator. The two crossover points are chosen at random. An adaptive mutation operator is used that is based on the mutation operator found in [8]. In this work, a mutation probability is used to determine if mutation will occur. If mutation occurs, the number of bits that are mutated is governed by Equation 2.

$$N = c \times l \times \left(\frac{F_P}{F_A} \right) \qquad (2)$$

Where N is the number of mutations, c is scaling factor, l is the limit based on the length of the chromosome, F_P is the max fitness of the parents and F_A is the max worst case fitness.

After 543,640 generations the evolution resulted in a configuration that has nearly the frequency response desired. A set of plots comparing the test case filter output data and the evolved filter output data is shown in Figure 7. The test case output data is plotted in black and the evolved filter output data is plotted in gray. The time response shows some large differences between the data values, but the general shapes of the responses are similar. However, the errors in the frequency response shown in the plot of the FFT data visually appear to be lower and are on the order of 10% as shown in the FFT magnitude error plot. The results do not perfectly match the test case, but a low pass filter has indeed been evolved.

The evolved filter configuration is shown in Figure 7. It is clear that a conventional digital filter has not been evolved. The evolved filter uses one of the bitwise Boolean operations, "not (in1or in2)," in tile F12 and the input of tile F11 is squared. Additionally, there are two feedback loops present. The evolved filter does not actually require delayed sampled input or output data as the IIR filter implementation does. It could use sampled data input directly rather than two-sample delayed input, as it is shown using in Figure 8. It is a non-linear mathematical function that is dependent on the data circulating in tiles F20, F21 and F22. These tiles act somewhat like cascaded integrators, due to the registered output of the tiles. It is also notable that the evolved filter uses six tiles rather than eight and one less multiplier.

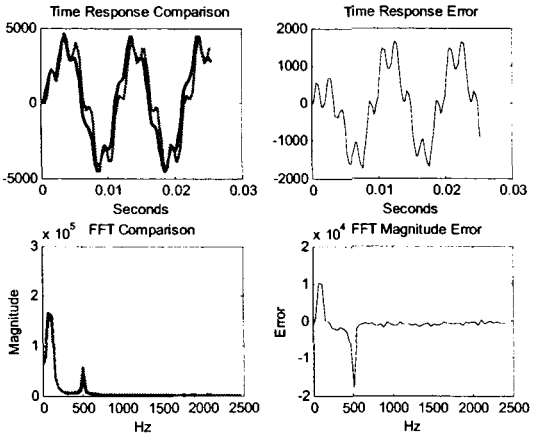

Figure 7 Comparison of time and frequency responses. Test case output is black and evolved filter output is gray

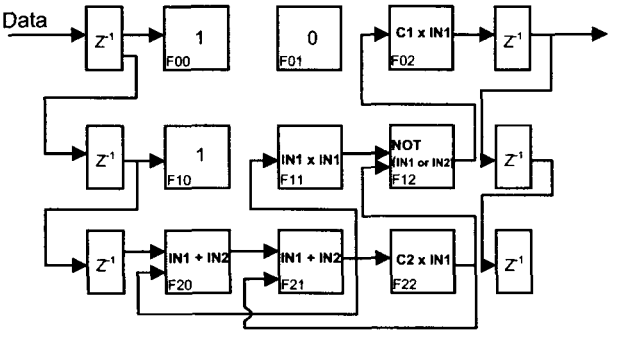

Figure 8 Diagram of the evolved filter on the EMVCore

In order to test the robustness of the evolved design, two other input signals are used. One of the two inputs is an added sine, with sinusoids at 100 Hz, 200Hz, 300Hz, 400Hz and 500 Hz. The other is a sinusoidal sweep frequency content from DC to 1200 Hz. The time sequence of both input signals are shown in Figure 9, along with the frequency content of each as determined by an FFT.

The evolved filter response is compared with that of the test case filter for each signal. The responses to the added sine input are shown in Figure 10. The response of the test case low-pass filter is plotted in black, while the response of the evolved filter is in gray. Visually, the plots of the time response and frequency magnitudes are very close. Plots of the error between the outputs of the two filters are included. Figure 10 shows that the evolved filter provides a low pass filtering action for frequencies between the two used in the evolution. The comparison of output responses for the sinusoidal sweep input is shown in figure 11. There is a noticeable degradation in the output response of the

evolved filter. There are significantly large errors between the time and frequency responses for the test case filter and the evolved filter. While a low-pass filtering action is being provided by the evolved filter, the response would not be acceptable in a signal processing application.

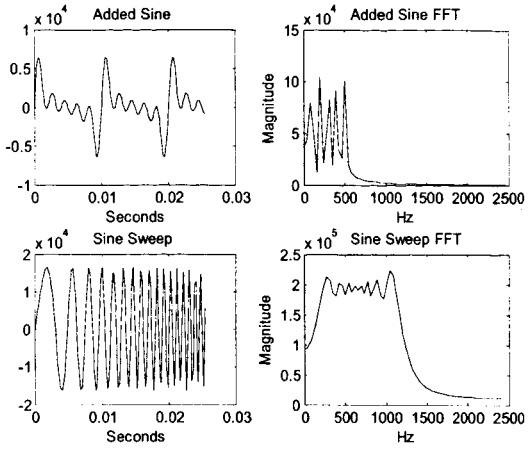

Figure 9 Plots of the test input signals used after evolution

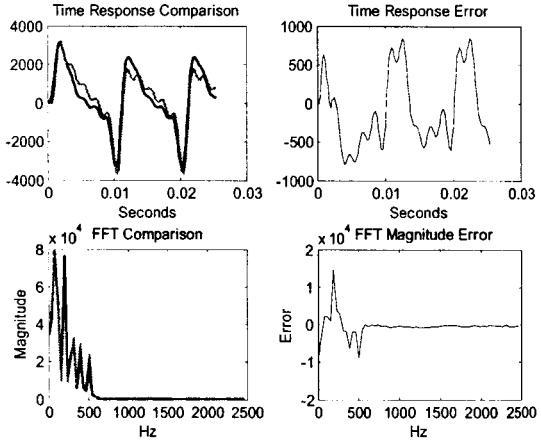

Figure 10 Comparison of time and frequency responses for input consisting of five added sines. Test case output is black and evolved filter output is gray

It appears that using an input consisting of two added sines during the evolution is not sufficient to provide a robust filtering characteristic. The sine sweep contains significant frequencies beyond the 100 Hz and 500 Hz input used during the evolution. Additionally, it has a different dynamic characteristic than several sine waves added together. A conclusion is drawn that the stimulus used during evolution should be selected to provide good coverage of the inputs that will be experienced by the evolved design during deployment. The evolutionary algorithm will tend to optimize the design for the conditions under which evolution is conducted.

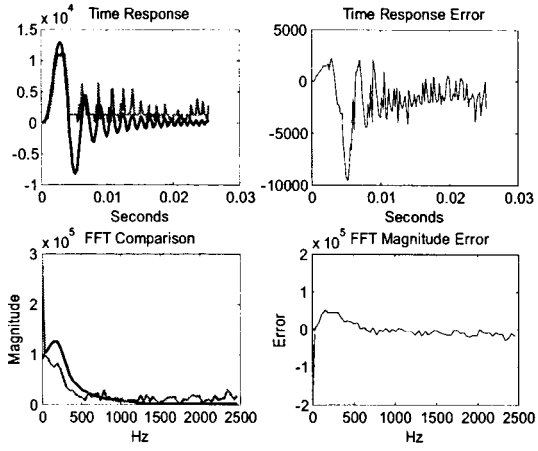

Figure 11 Comparison of time and frequency responses for the sine sweep input. Test case output is black and evolved filter output is gray

6 Summary

The design of an evolvable machine VHDL core (EMVCore) has been presented. The EMVCore is designed to have the components needed for a standard approach to discrete time signal processing and to have additional components that may be used in an innovative design. The evolvable system architecture in which the EMVCore is included has also been presented. The evolvable system provides a high degree of flexibility through user selectable and definable evolutionary operators.

The ability to implement standard IIR filter designs is shown in the low-pass filter test case. Other filter functions can be implemented via a change in coefficient selection. The ability to implement discrete time filters with data feedback, rather than just data feed forward processing, enables the implementation of discrete time controllers, as well.

Evolution of a novel low-pass filter design has been presented, along with an assessment of its capabilities. This filter performed well with the two added sines used as input during the evolution. It also performs well, when the input includes more sines at frequencies between the two used during evolution. The filter fails to function properly when the input is changed to a signal that is significantly different from that used during evolution, i.e. a sine sweep with wider bandwidth. This illustrates the importance of

designing the evolution to be representative of the environment that will be seen by the evolved design during deployment.

The evolved design is interesting in that it uses a bitwise Boolean operation and a square operation in its design. This is non-standard in the field of digital signal processing. Also, it uses 6 tiles rather than the 8 tiles used by the IIR filter implementation. The use of non-standard operators and fewer resources leads one to the conclusion that an EMVcore as described here can be used to implement more compact representations of digital filters and to provide fault tolerance by implementing a new solution in the remaining tiles after some are damaged. In addition to investigating the design of innovative digital filters, future work will include implementation of controllers and experiments in repair of damaged configurations due to faults in the electronic device.

The EMVcore can also be used to implement functions other than digital filters. The delay registers can be by-passed to implement numerical calculations for mathematical functions other than difference equations. If operators other than those defined in Table 1 are required they can be added, subject to the limitations of implementation in VHDL. It is possible to implement fuzzy logic processing and neural networks in an FPGA using an array of tiles which include the functions required for such structures.

Large FPGAs are currently available at a moderate price that enable and encourage the implementation of configurations with parallel processing resources. The Xilinx Virtex II devices feature hardware multipliers to facilitate parallel implementation of digital signal processing. The largest of these devices provides 168 18-bit by 18-bit multiplier blocks [11]. The latest Virtex 4 devices include the SX family with Xtreme DSP slices. These slices contain an 18-bit by 18-bit multiplier blocks, an adder and an accumulator. The largest of these devices contain 192 such slices [12]. The Altera Stratix II devices include columns of DSP blocks with multipliers, adders and accumulators. Each block can implement one 36 x 36 bit multiplier, or several multipliers with smaller word sizes. The largest of these devices contains 96 DSP blocks [13]. Industry is now providing high level functional components in commercial devices, which can support the implementation of complex evolution oriented intellectual property cores.

Acknowledgments

This work was supported in part by funding received from the Evolvable Systems Group at NASA Ames Research Center. Mr. Gwaltney was supported under a Marshall Space Flight Center Director's Discretionary Fund Task.

References

[1] Oppenheim, Alan V, and Shafer, Ronald W., Digital Signal Processing, Prentice-Hall Inc, Englewood Cliffs, NJ, 1975.

[2] Schnier, T., Yao, X. and Liu, P., *Digital Filter Design using Multiple Pareto Fronts*, Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware, Long Beach, CA, July 2001, pp 136-145

[3] Hounsell, B. I. and Arslan, T,. *Evolutionary Design and Adaptation of Digital Filters within an Embedded Fault Tolerant Hardware Platform*, Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware, Long Beach, CA, July 2001, pp.127-135.

[4] Thomson, R and Arslan, T, *Evolvable Hardware for the Generation of Sequential Filter Circuits*, Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware, Alexandria, VA, July 2002, pp 17-25

[5] Krusienski, D. J. and Jenkins, W. K., *Particle Swarm Optimization for Adaptive IIR Filter Structures*, Proceedings of the 2004 Congress on Evolutionary Computation, Portland, OR, June 2004, pp.965-970.

[6] Sekanina, L. and Ruzicka, R., *Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers*, Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware, Chicago, IL, July 2003, pp 135-144.

[7] Tufte, G. and Haddow, P. C., *Evolving and Adaptive Digital Filter*, Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware, Palo Alto, CA, July 2000, pp 143-150.

[8] Zhang, Y., Smith, S. L., and Tyrell, A. M., *Digital Circuit Design using Intrinsic Evolvable Hardware*, Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, June 2004, pp. 55-62.

[9] Sekanina, L. and Friedl, S., *On Routine Implementation of Virtual Evolvable Devices Using Combo6*, Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, WA, June 2004, pp 63 – 70.

[10] Mathworks, Signal Processing Toolbox Users Guide Version 6, October 2004. (http://www.mathworks.com/access/helpdesk/help/pdf_doc/signal/signal_tb.pdf)

[11] Xilinx Corporation, "Virtex II Platform FPGAs: Complete Data Sheet," v3.3, June 24, 2004.

[12] Xilinx Corporation, "Virtex -4 Family Overview," v1.2, December 8, 2004.

[13] Altera Corporation, "Stratix II Device Handbook, Volume I," January 2005.