Evolvable FIR Filter Design in Hardware and Software

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by: Edward James Luckett

MEng Field Advisor: Dr. Bruce Land

Degree Date: May 2016

Table of Contents

Abstract

Master of Engineering Program
School of Electrical and Computer Engineering
Cornell University
Design Project Report

Project Title: Evolvable FIR Filter design in Hardware and Software

Author: Edward Luckett

Abstract: Algorithms based on the evolutionary mechanisms of genetic mutation and crossover have the potential to yield substantial advantages compared to conventional design. In the specific case of filter design, evolutionary convergence potentially allows a solution that requires less energy and hardware to implement. However, the potential benefits of exploiting evolution in hardware are still being researched. To explore the characteristics of such algorithms in hardware, an evolutionary digital filtering algorithm was created and optimized on MATLAB. The filter design algorithm eventually achieved performance comparable to conventional Finite Impulse Response (FIR) filters, but suffered from limited design speed due to a large amount of software overhead. The advantage was that the evolved filter lent itself to a higher degree of customizable parameters, as the performance evaluation module was relatively simplistic compared to those of a Least-Squares or Parks-McClellan FIR filter. After evaluating the performance of the algorithm against conventional filtering algorithms, the subset of the algorithm was written in a Hardware Description Language (HDL) and simulated to determine the constraints on a hardware platform such as a Field Programmable Gate Array (FPGA). The results from these studies show that the evolutionary algorithm will reap advantages in unique filter yield and code simplicity for similar performance to conventional filter designs. In essence, the evolved filter platform lends itself to a greater degree of customizability, as the parameters for performance evaluation can easily be changed without modifying the rest of the algorithm.

Executive Summary:

Algorithms based on the evolutionary mechanisms of genetic mutation and crossover have the potential to yield substantial advantages compared to conventional design. In the specific case of filter design, evolutionary convergence potentially allows a solution that requires less energy and hardware to implement. However, the potential benefits of exploiting evolution in hardware are still being researched.

To explore the characteristics of such algorithms in hardware, an evolutionary digital filtering algorithm was created and optimized on MATLAB. The filter design algorithm eventually achieved performance comparable to conventional Finite Impulse Response (FIR) filters, but suffered from limited design speed due to a large amount of software overhead. For instance, running one version of the MATLAB code for 10 thousand generations would take several hours to reach completion. This level of delay is not only unacceptable for most applications, but also inhibits the ability to explore the potential gains made by optimizing the crossover, mutation, and performance evaluation modules. These considerations motivated the move to custom code on a hardware platform such as an FPGA. However, the advantage was that the evolved filter lent itself to a higher degree of customizable parameters, as the performance evaluation module was relatively simplistic compared to those of a Least-Squares or Parks-McClellan FIR filter. Unlike conventional methods, the evolved filter allows the user greater direct control of the magnitude and phase response for the desired filter, and allows for multiple unique solutions, as multiple iterations can find multiple local solutions.

After evaluating the performance of the algorithm against conventional filtering algorithms, the subset of the algorithm was written in a Hardware Description Language (HDL) and simulated to determine the constraints on a hardware platform such as an FPGA. The results from these studies show that the evolutionary algorithm will reap advantages in hardware and code simplicity for similar performance to conventional filter designs. In essence, the evolved filter platform lends itself to a greater degree of customizability, as the parameters for performance evaluation can easily be changed without modifying the rest of the algorithm. However, if the user so chooses, parameters in the mutation and crossover modules can be tuned to control the speed of convergence in the algorithm.

Introduction:

The main method of generating low-error FIR filters for digital signal processing (DSP) is applying a pre-determined filter that seeks to minimize certain characteristics of input signals, such as mean-squared error or ripple in the filter's passband or stopband. Examples of commonly used filters that optimize these characteristics are the Parks-McClellan filter and Least Mean Squares filter. For the Parks-McClellan filter, an iterative algorithm applies several steps to generate a set of filter coefficients that minimizes band ripple. The adaptive Least Mean Squares algorithm, however, seeks to generate coefficients that reduce the mean squared error in the bands. Overall, this class of filters allows for a deterministic design based in either linear algebra or differential equations, and are characterized by standardized implementations in hardware for successful operation.

In contrast, evolutionary algorithms present an alternative method of finding the optimum solution for a DSP problem. Primarily relying on stochastic processes, the algorithm emulates genetic mutation and crossover to converge to an optimal set of filter coefficients. After several thousand generations of evaluating the "fitness" of the algorithm, the evolved filter should achieve similar performance in magnitude response to filters generated by the Parks-McClellan or Least Squares algorithms. However, due to the non-deterministic nature of the algorithm, the filter has the potential to result in a varied set of hardware topologies and energy constraints. These factors can lead to an optimum filter with reduced hardware requirements for a particular application. However, evolutionary filter design relies on the user setting ideal requirements for solution convergence in order to produce an effective filter. In addition, as evolutionary processes do not lend themselves to repeatability, the evolved filters generated with this algorithm would likely represent several local results in the solution space. Overall, this project serves as an exploration and assessment of the capabilities of such algorithms.

Elaboration of Evolutionary Mutation and Crossover:

As mentioned in the **Introduction** section, the evolutionary filter algorithm relies on the processes of gene mutation and crossover to achieve an ideal solution. While the "genes" used in the project were actually the coefficients of each additional filter, a brief explanation of the actual genetic processes will aid in the comprehension of the design algorithm.

Genetic mutation, in essence, is a structural change of a block of DNA of a living cell. These changes generally result in the permanent change in the gene's structure and "can affect anywhere from a single DNA building block … to a large segment of a chromosome" (NIH). In actuality, these mutations are either inherited from previous generations or influenced by environmental factors, but for the purposes of the project, are simulated with a random process. Genetic mutations in populations are of particular interest, as the propagation of their effects across generations can result in the genetic drift away from the population's original genome structure. It is this trait that the algorithm is designed to exploit, as the genetic drift will result in several filters that can be analyzed for accuracy. In the algorithm, those filters that demonstrate the highest performance will be considered "evolutionarily fit" and be used to create the next generation of filters.

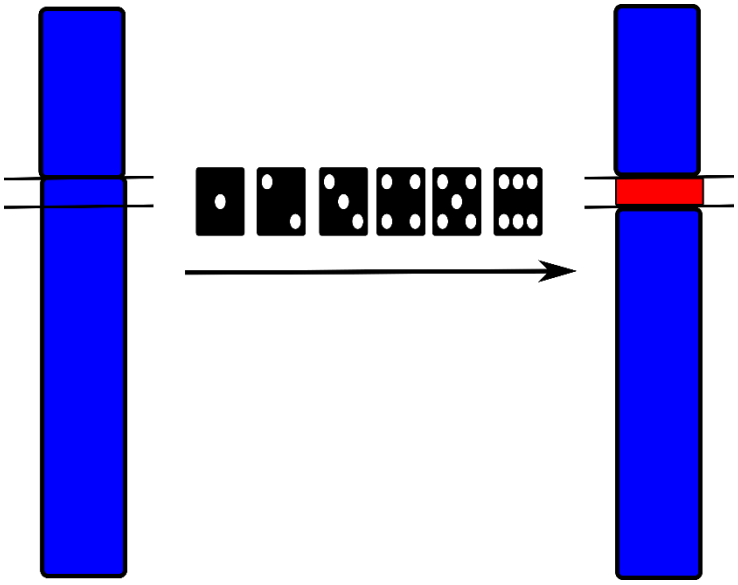

Figure 1: Random Chance of Genetic Mutation in Filters

Genetic crossover, however, is the mixing of genes within one generation in order to propagate genetic diversity onto the next generation. For the members of a population that have the highest evolutionary fitness, their genes are combined in a manner similar to Figure 2 in

order to create the next generation. While the process of evolutionary mutation promotes genetic drift, crossover increases the genetic diversity within a population. Therefore, for the purposes of the algorithm, the combination of genetic drift and crossover results in an iterative process that can result in multiple, unique filters that can be analyzed and evaluated for performance.

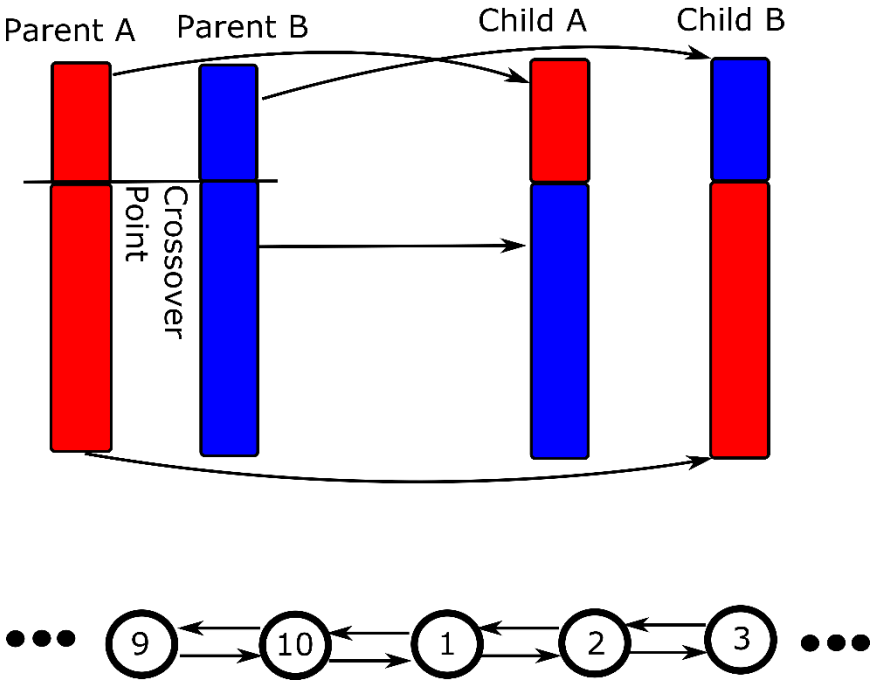

Figure 2: Crossover of Parents' Genes to Create Children

Background on Genetic Algorithm (GA) Applications:

The first major instance of a genetic algorithm being used for engineering applications is attributed to Adrian Thompson from the University of Sussex. In 1996, Thompson created an algorithm that would "evolve a circuit to discriminate between square waves of 1kHz and 10kHz at the input" of an FPGA without the use of a system clock (Thompson 3). An interesting characteristic of this evolved circuit was that the algorithm created a unique loading sub-circuit that had no output to another part of the system. However, the system would fail to function without this sub-circuit, which speaks to the unpredictable outcomes possible in evolved circuitry.

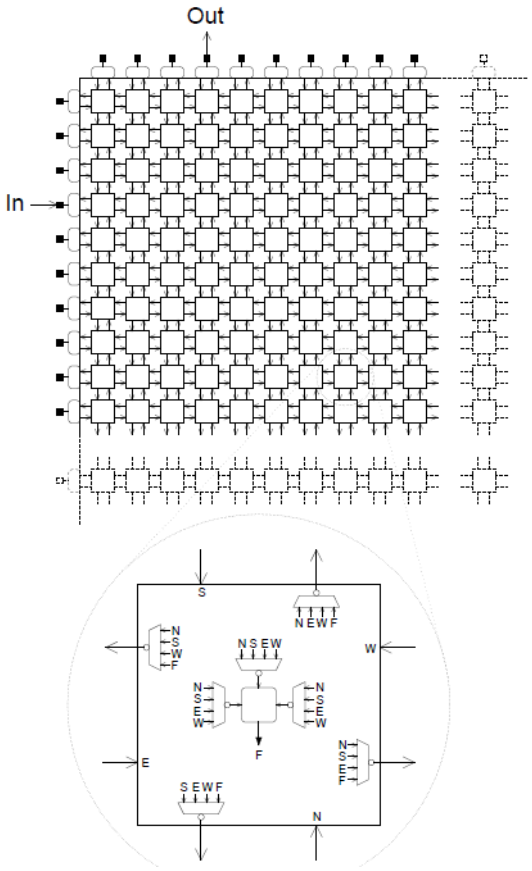

Figure 3: FPGA Setup in Thompson Experiment

The success of Thompson's test inspired a wide range of projects using genetic algorithms for applications primarily focused on operations optimization or computer-aided design. In particular, a multitude of projects surrounding the application of genetic algorithms to filter design. A paper by Ken Sharman and Anna Esparcia-Alcazar, for instance, details several

methods for generating filters from evolutionary processes, as well as highlights how the inspiration of nature-based design can lead to unique filters not possible with deterministic mathematical solutions.

In addition to these works in digital filter design, general optimization techniques in genetic algorithms must be considered in order to make an effective filter generation tool. While genetic mutation and crossover allow for a diversity of filter solutions, the performance evaluation used is the main control on the speed at which the algorithm converges to the optimum solution. Therefore, the overall speed and design energy consumption, especially in a hardware implementation, hinges on the effectiveness of determining the evolutionary fitness in filters. While most implementations set one performance evaluation condition that is used throughout all generations, researchers such as Vassilios Petridis have conducted experiments to determine the effectiveness of varying the fitness functions used across generations. This method in particular can be useful in filter design for parametrizing, for example, the desired magnitude response and group delay displayed by a filter. Overall, there exists a wide range of literature on the subject that suggests several methods for achieving optimal filter performance.

Algorithm Design Considerations:

In general, the main concern of the evolutionary algorithm is to optimize the factors affected by each module of the overall filter generation code. Therefore, separate parameters must be considered for the mutation, crossover, and performance evaluation modules. Overall, however, these modules contribute to meeting the top-level system specifications, which focus on the performance of the final filter, execution speed, and algorithm complexity compared to conventional FIR filter designs.

Mutation Module

As mentioned in the **Elaboration of Evolutionary Mutation and Crossover** section, the mutation module serves as the main control of genetic drift. Therefore, the frequency of random changes of filter coefficients must maintain a balance to achieve proper system convergence. If the mutation rate were too low, then the population of filters used would not reach a local solution within an acceptable number of generations. For example, in the figure below, the evolved filter failed to converge due to a low mutation rate, and thus actually amplifies potential stopband signals. However, a mutation rate that is too high can result in potential divergence away from a potential optimized filter. In nature, most mutation rates for living beings are low, with viruses having a rate of approximately 1% per generation. Therefore, I started my exploration of mutation rates with this value. In addition, the size of the mutation can have similar effects. The number of coefficients that the module changes during one mutation event can also prevent the algorithm from returning an ideal filter.

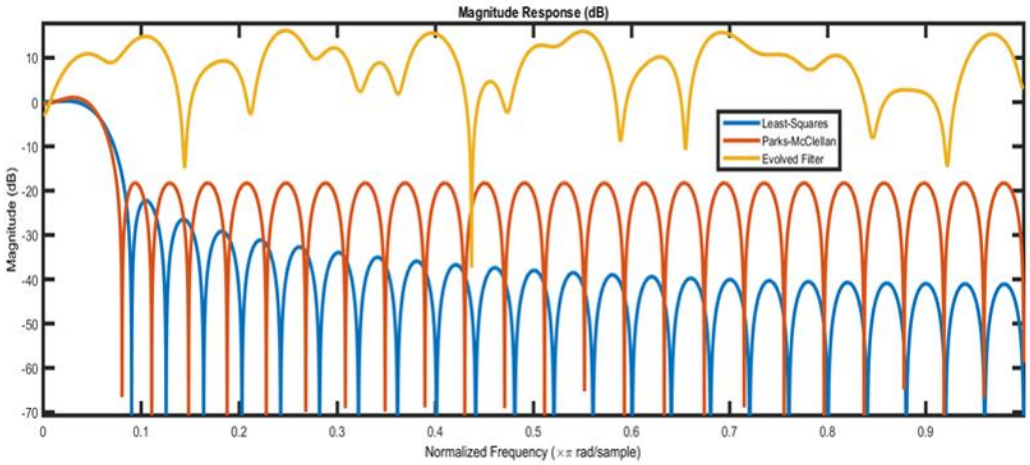

Figure 4: Example of Failed Convergence due to Low Mutation Rate

Crossover Module

The genetic diversity provided by the crossover module is based primarily on the ranking system given to filters after performance evaluation. As highlighted in Sabbir Ahmad's *Application of Genetic Algorithms for the Design of Digital Filters*, the ranking system used for the filters that pass the fitness function can be used to determine their crossover. For example, one potential crossover scheme is crossing the genes of the filters ranked from best to worst. On the other hand, one could also implement a system where all filters that meet the fitness spec pick a random partner with which to perform crossover. In first case, that particular crossover scheme could potentially achieve a higher convergence rate at the expense of filter diversity. The second crossover scheme achieves the opposite result.

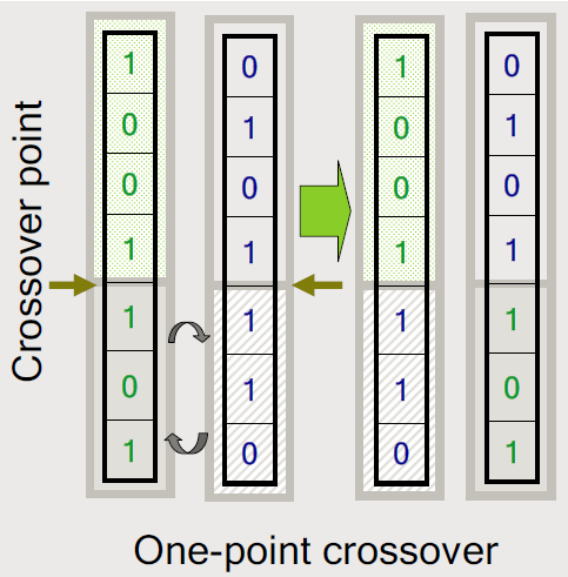

Figure 5: Effect of Crossover Point on Filters (Ahmad 19)

Another key consideration in the crossover module design is choosing the crossover point within the filter coefficients. The initial choice would likely be to choose the midpoint for filter coefficients. However, another choice that could potentially foster more genetic diversity in the filter population would be selecting a random point for each generation.

Performance Evaluation Module

The part of the evolvable filter that has the most significant impact on performance is the evaluation function, so the implementation of this module will primarily determine the rate of successful filter generation. While the fitness function used depends on what aspects of the filter

that the user values most, I will limit the discussion in this section to the magnitude response of the output filter. While several methods of evaluating the magnitude response exist, many of the functions that lend themselves best to simple software and hardware implementations involve quantifying the error in reconstructing a signal corrupted by noise. Therefore, the common methods used for the fitness function are the total squared error, mean squared error, and log error. An advantage of using a form of squared error over total error or log error is that large deviations in performance are given more importance in evaluation. In addition, using mean squared error versus total squared error also allows for a lower variance in the filter performance. The advantage of using log error, especially in conjunction with root mean square, provides greater differentiation between filters that have high evolutionary fitness, as opposed to the ones that have high error.

Conventional FIR Design Algorithm Analysis:

In order to better define the differences between the genetic algorithm, and more conventional FIR designs, it helps to analyze the specifications that these designs attempt to optimize. While numerous dedicated design solutions exist for FIR filter design, this section will focus on the filters used in the MATLAB implementation as a performance baseline, the Parks-McClellan and the Recursive Least Squares filters.

Parks-McClellan Filter Design

The Parks-McClellan filter design algorithm refers to a class of filters based on the Alternation Theorem. The Alternation Theorem, in essence, provides a method of finding the optimum filter coefficients with respect to error in a Chebyshev class filter. The Parks-McClellan filter therefore uses the Alternation Theorem in an iterative algorithm that attempts to converge to these coefficients as shown in the figure below.

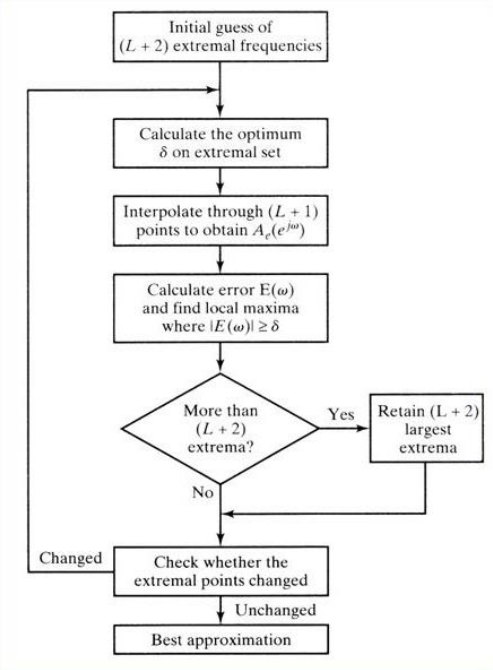

Figure 6: Parks-McClellan Flowchart (Oppenheim and Schafer 569)

After an initial guess of the optimal alternation frequencies using polynomial interpolation, the algorithm calculates the ripple in the bands of the filter. Using interpolation,

the alternation frequencies, and the prior ripple calculations, the algorithm then finds the transfer function, and determines whether the filter meets the ripple specifications for the desired band frequencies. This process continues, with adjustments to the alternation frequencies, until the desired ripple and band frequencies are achieved.

Recursive Least Squares Design

Where the Parks-McClellan attempts to minimize the ripple error in the passband and stopband of filters, the Recursive Least Squares Filter attempts to minimize the total squared error of the entire filter. By comparing an initial FIR filter design to the desired frequency response, the algorithm creates an error function from the difference and uses linear algebra to determine a Kalman gain vector for the system. This gain vector is then used to update coefficients for the next iteration until the total squared error is within the desired tolerance.

2.2 Summary of the RLS Filter Coefficient Algorithm

With a new input sample $f(n)$, and desired output value $d(n)$,

1. Update the input history vector $\mathbf{f}(n)$.
2. Compute the filter output using the previous set of filter coefficients $\mathbf{b}(n-1)$

$$y(n) = \mathbf{f}^T(n)\mathbf{b}(n-1)$$

3. Compute the error

$$e(n) = d(n) - y(n)$$

4. Compute the Kalman gain vector

$$\mathbf{k}(n) = \frac{\mathbf{R}^{-1}(n-1)\mathbf{f}(n)}{\lambda + \mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\mathbf{f}(n)}$$

5. Update the matrix $\mathbf{R}^{-1}(n)$ for the next iteration

$$\mathbf{R}^{-1}(n) = \lambda^{-1}\left[\mathbf{R}^{-1}(n-1) - \mathbf{k}(n)\mathbf{f}^T(n)\mathbf{R}^{-1}(n-1)\right]$$

6. Update the filter coefficients for the next iteration.

$$\mathbf{b}(n) = \mathbf{b}(n-1) + \mathbf{k}(n)e(n)$$

Figure 7: Description of Recursive Least Squares Algorithm (Dowell 3)

MATLAB Implementation and Results:

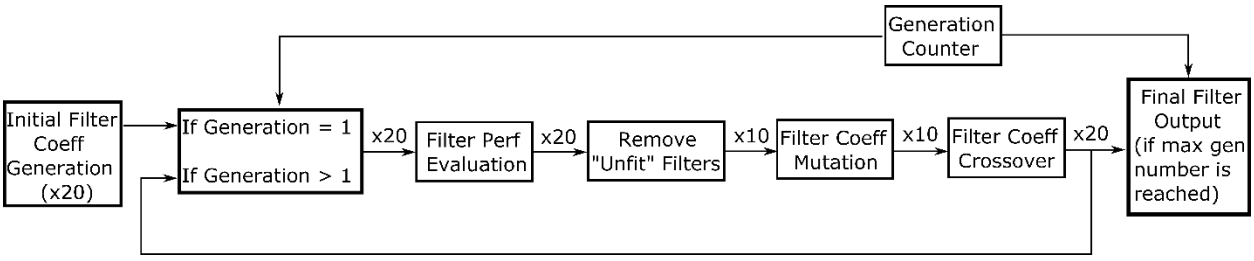

Figure 8: Flowpath of Genetic Algorithm

The figure above shows a high-level diagram of the proposed evolutionary algorithm. The filter will generate a set of random filter coefficients and evaluate the performance based on an input metric. For instance, the current method used to evaluate performance is to measure the logarithm of the mean squared error in the filter output across a set of test frequencies. This metric is used to remove half of the total filters that perform the worst on the given evaluation method. After removal, the best performing filters are "mutated," meaning that a random coefficient within the filters is assigned a random value. The crossover module splits the coefficients of each filter along a random point and recombines these coefficients to create a new set of filters. These filters are fed to the beginning of the loop to run through the modules again. This process is repeated until the generation counter module reaches a maximum number set by the designer.

As an initial proof of concept, a version of the algorithm was created in MATLAB to provide insight into potential implementation issues in hardware. The results from a 10 thousand generation run are shown below. In addition, the code is included in the attached appendix. The current version of the MATLAB code implements a version of the algorithm shown in Figure 8. For the initial filter generation, the code creates 60 sets of 50-coefficient FIR filters comprised of random numbers between -1 and 1. After initial generation, the code implements a mutation function that has a 1% chance of mutating any coefficient in the sixty evolvable filters. The code then uses a performance evaluation function to determine the ranking of the generated filters. The current method used to evaluate the performance of the filters involves first generating a set of test sine waves, which vary depending on the desired cutoff frequency. For testing, the cutoff frequency was set to 500Hz. The test frequencies used are shown in the table below. The sine waves are then corrupted with Gaussian white noise such that the result has an SNR of 5dB.

After corrupting the sine wave, the function saves the resulting wave generated from filtering the noisy sine wave with each evolved filter, which will henceforth be referred to as the filter output. To evaluate the performance of each filter, the function first compares the filter output to the ideal sine wave of the corresponding frequency, and then calculates the logarithm of the mean-squared error for each filter. The values for each test frequency point are then added together to create a performance index for each filter.

Test Freq. Formula	*Test Freq. Used (Hz)*
$f_c/100$	5
$f_c/10$	50
f_c	500
f_c*10	5,000
f_c*100	50,000
$f_c*1{,}000$	500,000
$f_c*10{,}000$	5,000,000

Table 1: Performance Evaluation Frequency Test Points

After generating a unique performance index for each filter, the MATLAB code sends the filters into a function that combines the processes of unfit filter removal and crossover. Using the previously generated performance indices, the function sorts the filters and removes the 30 filters that perform the worst. After removal, in order to repopulate the filter "population", the function implements crossover. The figure below shows the process of crossover. Essentially, at a random point within the coefficients of a filter, two filters split at that point and recombine according to the process shown in the figure. To repopulate to 60 filters, a filter performs crossover with the filters before and after it in the ordered list of filters. For example, the 2nd filter performs crossover with the 1st and 3rd filters similar to the process shown in Figure 2. After performing crossover, the code repeats until the desired number of generations is reached. In general, the number of generations is set to the minimum needed to achieve convergence, but can be set lower to save execution time. For the particular filter performance evaluation function used, the maximum number of generations was set to 10,000.

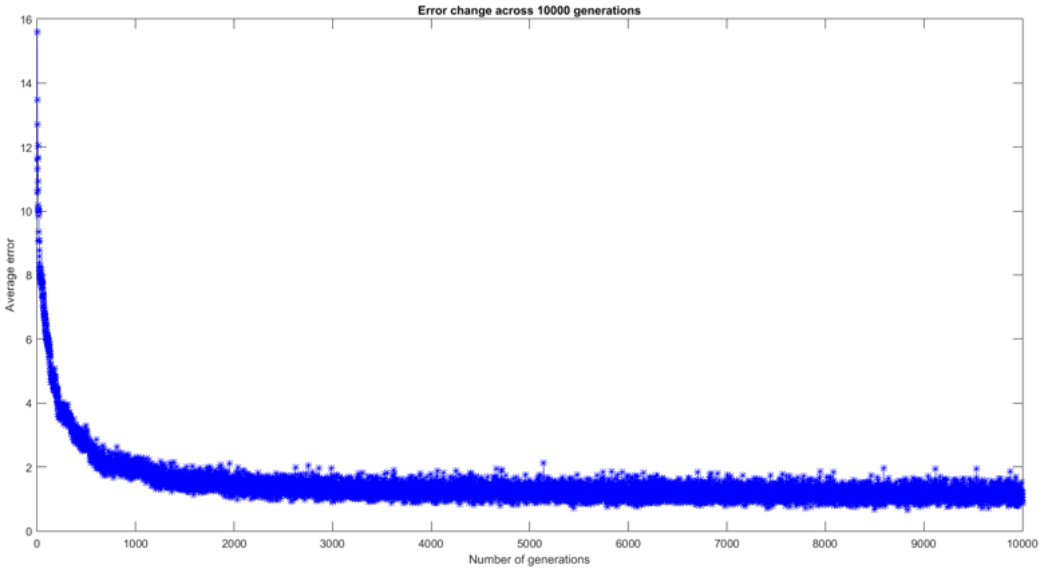

Figure 9: Convergence in Average Error across Generations

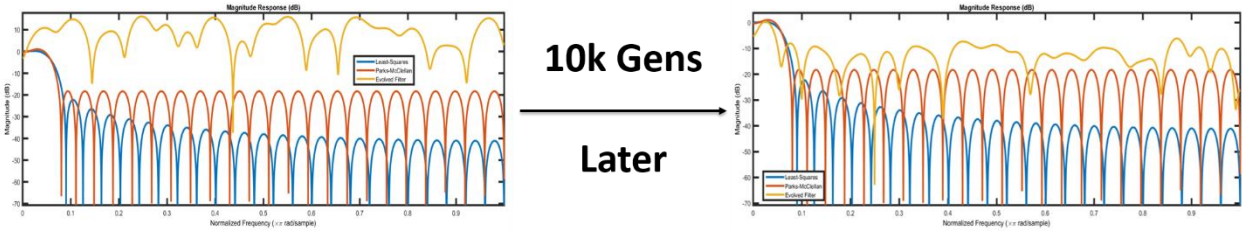

Figure 10: Comparison of Magnitude Response of Filters

Figures 9 and 10 show the results from running the evolutionary algorithm with the above MATLAB code. As shown, the algorithm had achieved the majority of convergence after 2500 generations, but still showed a minor decrease in error for the rest of the 10,000-generation run. After 10,000 generations, the final filter showed the desired performance in the passband and a trend towards acceptable performance in the stopband. As expected, increasing the number of coefficients per filter lengthened execution time and increased attenuation in the stopband. In addition, the performance evaluation methods listed in **Algorithm Design Considerations** section were tested, but all achieved a similar level of average error by 10,000 generations. As shown, for higher frequencies, the evolved filter shows worse performance compared to the Least-Squares and Parks-McClellan FIR filters, but that might be due to a lack of test points at

higher frequencies. An increase in test points across the stopband might yield better convergence to an ideal solution.

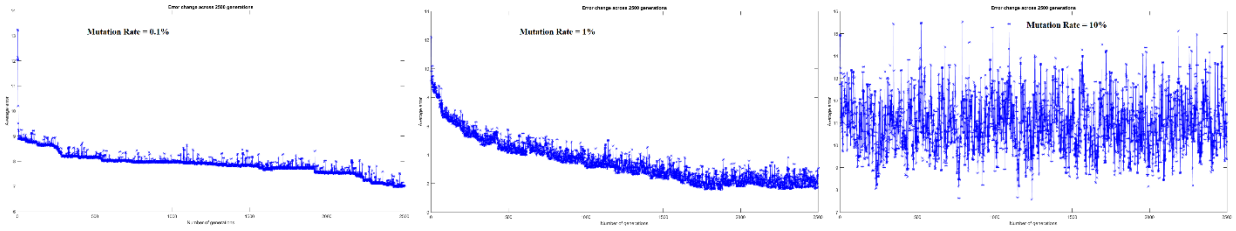

Figure 11: Effect of Mutation Rate on Average Filter Error (Left: 0.1%; Middle: 1%; Right: 10%)

Figure 11 above shows a series of tests on the effect of mutation rates on filter convergence. As mentioned in the **Algorithm Design Considerations** section, the initial guess for an appropriate mutation rate, which seemed to be valid. For mutation rates on the order of 0.1%, the lack of genetic drift drastically lowers the convergence rate. However, while a high mutation rate of 10% could potentially find an optimum filter, the filters in the next generation would likely drift too far away from the solution space. Therefore, mutation rates around 1% seem to provide acceptable results.

In addition, several tests of the algorithm showed that increasing the number of filters in the initial population would improve the performance of the final filter. In essence, the increase in filters would allow the algorithm to find more local minimums in the error space. The main tradeoff is a significant decrease in computation speed. To elaborate, MATLAB does not present an optimized platform to test the evolved filter algorithm due to a large amount of software overhead. In the instance of the 10,000-generation run, the execution time can total to several hours. The majority of the execution time is spent on the performance evaluation module, as each of the 60 filters are tested across several filtered sinusoids. This is the primary motivation for developing Verilog code to execute the evolutionary algorithm on a hardware platform.

Verilog Implementation and Expected Results:

As mentioned in the previous section, the execution time of the evolvable filter algorithm makes it an unattractive option compared to conventional FIR filter designs. However, implementing the algorithm in an HDL would allow for greater optimization in a hardware platform, such as an FPGA. This section details the design process for the submodules created for this purpose. The attached appendix contains the Verilog code created for the algorithm.

On a high level, the Verilog code follows the same format as shown in Figure 8, with several adjustments. The general format used for encoding the filters was to assign each filter 16 coefficients, where each coefficient is 8 bits. Each coefficient follows a <8, 6> fixed-point format, allowing shift operations to be used instead of divide operations. Since a byte limits the precision of coefficients for certain applications, a <16,14> fixed point format can also be used to still allow for shift operations and maintain the addressability of coefficients for debugging. A multiplier would still be required to complete the convolution operation for performance evaluation. Other minor changes were made to general modules needed throughout the algorithm. For instance, the sinusoids created for testing would have to be created using a direct digital synthesis (DDS) submodule and the random numbers used throughout the algorithm are provided by a Linear Feedback Shift Register (LFSR). In addition, the Verilog code allows the user to exploit several properties unique to hardware implementations. The following paragraphs describe the exact changes made to modules that are more specialized in order to accommodate a hardware implementation.

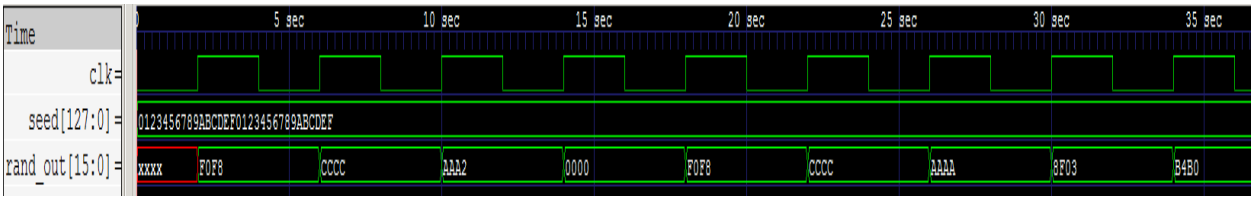

Figure 14: Example of LFSR Operation

Crossover Module

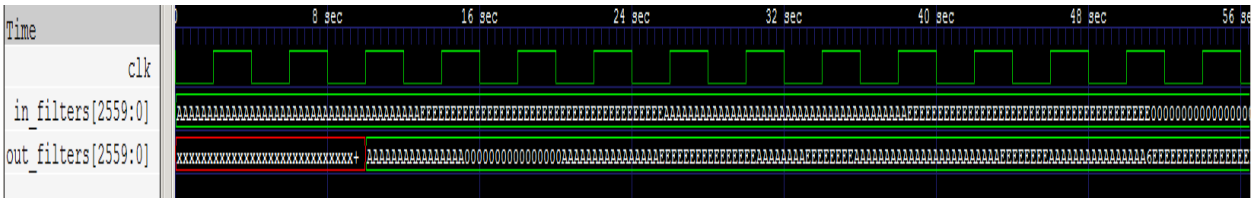

Figure 15: Example of Crossover Module Operation

The figure above displays a simulation of the Verilog crossover module. As shown, filters 1-5 and 15-20 are hardcoded to meet the performance threshold in order to induce crossover. This figure shows that the Verilog implementation still follows the format set in Figure 2. A Finite State Machine (FSM) controls the main operation of the crossover module with four main states: IDLE, INIT, CALC, and DONE. The IDLE and DONE states mainly control the handshaking protocol between the crossover and top-level modules. The INIT state performs initialization operations before the main crossover function. In particular, the initialization stage removes filters that do not meet the fitness threshold, while storing the fit filters in a different register for use in the CALC stage. In the CALC stage, the module performs crossover based on a random crossover point chosen from an LFSR input. This random input from the LFSR acts as the selection bits for a mux that feeds into a crossover structure similar to the one in Figure 2. Both the INIT and CALC stage require one clock cycle each, so the crossover module will not dominate the execution time.

Mutation Module

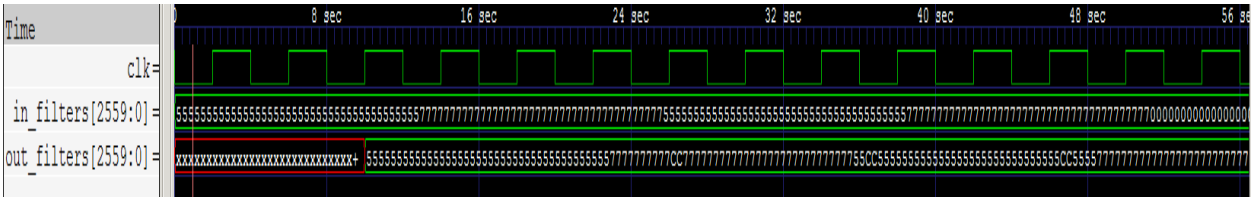

Figure 16: Example of Mutation Module Operation

Figure 16 above shows the simulated operation of the mutation module in Verilog. In this particular example, the original coefficients in each filter contain a value of either 0x55 or 0x77. A coefficient in several filters is hardcoded to mutate to a value of 0xCC. Similar to the crossover module, the mutation module utilizes an FSM with IDLE, INIT, CALC, and DONE stages. In the INIT stage for this module, however, one input from the LFSR chooses which 8-bit

coefficient to change in a filter. In the CALC stage, each filter receives an independent chance of mutation, as each filter is assigned a separate random number from the LFSR. In the current configuration, each filter has a 0.8% chance of undergoing mutation. If this check passes, then the coefficient chosen during the INIT stage is changed for a particular filter. The mutation module also only requires four clock cycles for computation, so its operation will not act as a bottleneck for system performance.

Performance Evaluation Module

Although the performance module was not implemented, several assumptions about its operation can be made. This module will make significant use of both the DDS and LFSR modules in order to create sinusoids to filter and test. The initialization of the test waveforms would likely require at least one state in an FSM. After creating the sinusoids, another state would be needed to perform a convolution between the noise-corrupted sinusoids and each filters' response. To reduce computation time of these operations in terms of cycles, the convolutions for each filter can be parallelized at the expense of hardware, especially for the multiply operations. The results of these convolutions would then need to be compared to pure sine waves of the same frequencies, and the log of the mean squared error would be computed for each filter. Considering the number of points needed for an accurate measurement, this step would likely not be parallelized and would take a cycle per filter. In the current implementation, this state would require 20 clock cycles for completion. After finding the fitness of each, the midpoint would have to be found using a sorting algorithm, so that the crossover module could reliably remove unfit filters. This additional SORT state would be dependent on the number of filters used. If the worst-case of a bubble sort is assumed, then the state would require an additional 190 cycles.

As described, it is blatantly obvious that the performance evaluation module dominates the execution time. However, even in a lower-end modern FPGA, such as the Lattice iCE40 with a clock of 12 MHz, the total execution time is on the order of hundreds of milliseconds, making the evolvable filter more competitive as a solution for FIR filter design.

Conclusions and Future Work:

While this project serves as a brief exploration on the potential of evolvable filters, additional work can be conducted to provide a more robust framework for the genetic algorithm. The obvious next step would be the completion and implementation of the Verilog code in order to confirm the results shown in this report. In addition, the exploration of fitness functions that consider phase response would provide interesting insights into the evolutionary process. While I attempted to create a fitness function in MATLAB that attempted to achieve a zero-mean phase while meeting the magnitude constraints I set, the resulting run of the algorithm failed to achieve any significant convergence. A fitness function that instead analyzed the linearity of the phase, particularly in the passband, would likely achieve greater success.

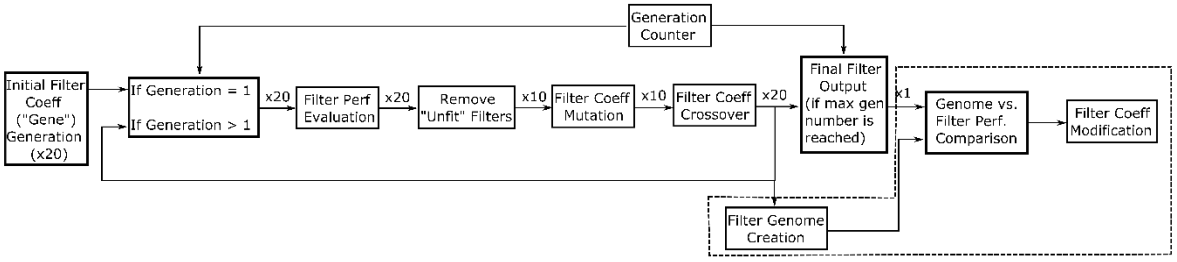

Figure 17: Genetic Algorithm Flowpath with Epigenetic Modules

Another potential avenue of exploration is the idea of creating an “epigenetic” algorithm. The idea of epigenetics is based on environmental stressors controlling whether certain genes in a living being turn on or off during one generation. The implementation in case of evolved filters would be a set of modules that could monitor the performance of the final filter created by the main genetic algorithm and essentially act as a predictor of future performance. These modules would then be able to calibrate the coefficients if accuracy fell out of an acceptable range.

References:

[1] Sekanina, L. and Vasicek, Z. “On the Practical Limits of the Evolutionary Digital Filter Design at the Gate Level”. Faculty of Information Technology. Brno University of Technology, Czech Republic.

[2] National Institute of Health. “What is a gene mutation and how do mutations occur?” May 10, 2016. Web.

[3] Thompson, Adrian. “An evolved circuit, intrinsic in silicon, entwined with physics”. 1st International Conference on Evolvable Systems.

[4] Sharman, K., and Esparcia-Alcazar, A. “Evolutionary Methods for Designing Digital Filters”. Contemporary Music Review, 22:3, 5-19, DOI:10.1080/0749446032000150843

[5] Mostafa et al. “Hardware Implementation of Genetic Algorithm on FPGA”. 21st National Radio Science Conference.

[6] Petridis et al. “Varying Fitness Functions in Genetic Algorithm Constrained Optimization: The Cutting Stock and Unit Commitment Problems”. IEEE Transactions on Systems, Man, and Cybernetics, vol. 28, no. 5, October 1998.

[7] Drake, J. and Holland, J. “Mutation rates among RNA viruses”. Proceedings of the National Academy of Sciences of the United States of America, vol. 96, no. 24, November 1999.

[8] Ahmad, S., and Antoniou, A. “Application of Genetic Algorithms for the Design of Digital Filters”. Dept. of Electrical and Computer Engineering. University of Victoria, Canada.

[9] Oppenheim, A. and Schafer, R. “Discrete Time Signal Processing”. Upper Saddle River: Pearson, 2010. Print.

[10] Rowell, Derek. “Introduction to Recursive-Least-Squares (RLS) Adaptive Filter”. Dept. of Mechanical Engineering. Massachusetts Institute of Technology.

[11] Chauhan, R. and Arya, S. “An Optimal Design of FIR Digital Filter Using Genetic Algorithm”. Communications in Computer and Information Science, vol. 168.

[12] Tang, P. and Lee, G. “An Adaptive Fitness Function for Evolutionary Algorithms using Heuristics and Prediction”. World Automation Congress, 2006.

Appendix A: MATLAB Code:

```
function evo_filt = Evo_Alg_Test_Driver_final
close all;

% Generate initial parameters (i.e. number of coefficients, the passband/
% stopband, the amplitude at each band, the initial population for genetic
% algorithm implementation, number of generations, freq. for tests)
num_coeffs = 50;
freq_band = [0 0.05 0.075 1];
freq_amp  = [1 1 0 0];
evo_population = 50;
evo_generations = 10*10^3;
cutoff = 500;
Fs = 10^5;
freq_weights = 1.*ones(1,7);

% Generate Least-Squared and Parks_McClellan FIR filter for comparison
[ls_filt, pm_filt] = Filter_comparison_init(num_coeffs,freq_band,freq_amp);

% Run evolvable filter function
evo_filt = Evolutionary_filter_module(num_coeffs,evo_population...
    ,evo_generations,cutoff,Fs,freq_weights);

% Plot Evolved, Least Squares, and Parks-McClellan Filters
hfvt = fvtool(ls_filt,1,pm_filt,1,evo_filt,1);
legend(hfvt,'Least-Squares','Parks-McClellan','Evolved Filter')

save('test_evo_filt.mat','evo_filt');
return

function [ls, pm]=Filter_comparison_init(num_coeffs,freq_band,freq_amp)
% Generate Least-Squares Filter
ls = firls(num_coeffs-1,freq_band,freq_amp);
% Generate Parks-McClellan Filter
pm = firpm(num_coeffs-1,freq_band,freq_amp);
return

function evo_filt = Evolutionary_filter_module(num_coeffs,evo_population...
    ,evo_generations,cutoff,Fs,freq_weights)

% Generate initial random set of numbers for evolution
evo_filters = 2.*rand(evo_population,num_coeffs)-1;

% Evaluate filter performance
perf_idx = Eval_Performance(evo_filters,cutoff,Fs,freq_weights);

prob_mutation = 0.01;
prob_crossover = 0.5;

% Track evolved filter performance across generations (Track at cutoff)
perf_idx_tracking = zeros(size(perf_idx,1),evo_generations);
```

```
perf_idx_tracking(:,1) = perf_idx;

% Run for multiple generations (exact # specified in input)
filt_count = 2;
for n = 2:(evo_generations)
    % This is for debugging
    test1 = size(perf_idx,1);

    % Run mutation function
    evo_filters = Mutate_generation(evo_filters,prob_mutation);
    % Run crossover function
    evo_filters = Cross_over_generation(evo_filters,prob_crossover,...
        perf_idx);
    % Run performance evaluation module
    perf_idx = Eval_Performance(evo_filters,cutoff,Fs,freq_weights);

    % This is for debugging as well
    if(size(perf_idx,1) ~= test1)
        pause;
    end

    % This is for tracking the error reduction in the evolved filters
    % across generations
    perf_idx_tracking(:,filt_count) = perf_idx;

    % These lines are for tracking the filter count and the progress
    % from the command window
    filt_count = filt_count+1;
    display(filt_count);
end

% Pick best filter and send as output
[~,idx] = min(perf_idx);
evo_filt = evo_filters(idx,:);

% FOR PERFORMANCE ANALYSIS
% Analyze how error drops over subsequent generations
perf_mean = zeros(1,evo_generations);
for n = 1:evo_generations
    perf_mean(n) = mean(perf_idx_tracking(:,n));
end
figure;
plot(1:evo_generations,perf_mean,'b-*')
title(['Error change across ',num2str(evo_generations),' generations'])
xlabel('Number of generations')
ylabel('Average error')
figure;
plot(1:evo_generations,perf_mean./max(perf_mean),'b-*')
title(['Error change across ',num2str(evo_generations),' generations'])
xlabel('Number of generations')
ylabel('Average normalized error')
return

function mutated_filters = Mutate_generation(evo_filters,prob_mutation)
```

```
% Determine which point to mutate based on probability of mutation
mutated_filters = evo_filters;
rand_mat = rand(size(evo_filters,1),size(evo_filters,2));
thresh_mat = rand_mat >= (1-prob_mutation);
[thresh_x,thresh_y] = find(thresh_mat > 0);

% If the mutation matrix is not empty, change appropriate points to random
if(~isempty(thresh_x))
    for n = 1:length(thresh_x)
        mutated_filters(thresh_x(n),thresh_y(n)) = ((2)*rand-1);
    end
end
return

function new_gen = Cross_over_generation(old_gen,prob_crossover,perf_idx)
% Find "most fit" filters to pass their "genes to the next generation"
pop_size = size(old_gen,1);
perf_cutoff = floor(pop_size*prob_crossover);
perf_idx_sorted = sort(perf_idx);

% % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % % %

[rows,~] = find(perf_idx <= perf_idx_sorted(perf_cutoff));
rows = rows(1:round(pop_size/2));
temp_mat = old_gen(rows,:);

% Create children from "fit" filters
num_coeffs = size(old_gen,2);
crossover_point = floor(rand*num_coeffs);
new_gen = zeros(pop_size,num_coeffs);
for n=1:floor(pop_size/2)
    a = mod(2*n-1,length(rows));
    b = mod(2*n,length(rows));
    if(a == 0)
        a = length(rows);
    end
    if(b == 0)
        b = length(rows);
    end
    new_gen(2*n-1,:) = [temp_mat(a,1:crossover_point), ...
        temp_mat(b,crossover_point+1:end)];
    new_gen(2*n,:) = [temp_mat(b,1:crossover_point), ...
        temp_mat(a,crossover_point+1:end)];
end
return

function perf_idx = Eval_Performance(evo_filters,cutoff,Fs,freq_weights)
% Generate initial sine waves
t = linspace(0,1,Fs);
sin_test = zeros(7,length(t));
sin_test(1,:)  = sin(2*pi*(cutoff/100)*t);
sin_test(2,:)  = sin(2*pi*(cutoff/10)*t);
sin_test(3,:)  = sin(2*pi*(cutoff)*t);
sin_test(4,:)  = sin(2*pi*(cutoff*10)*t);
```

```
sin_test(5,:)  = sin(2*pi*(cutoff*100)*t);
sin_test(6,:)  = sin(2*pi*(cutoff*1000)*t);
sin_test(7,:)  = sin(2*pi*(cutoff*10000)*t);

% Corrupt copy of sine waves with noise for testing
sin_test_corrupted = zeros(7,length(t));
sin_test_corrupted(1,:) = awgn(sin_test(1,:),5,'measured');
sin_test_corrupted(2,:) = awgn(sin_test(2,:),5,'measured');
sin_test_corrupted(3,:) = awgn(sin_test(3,:),5,'measured');
sin_test_corrupted(4,:) = awgn(sin_test(4,:),5,'measured');
sin_test_corrupted(5,:) = awgn(sin_test(5,:),5,'measured');
sin_test_corrupted(6,:) = awgn(sin_test(6,:),5,'measured');
sin_test_corrupted(7,:) = awgn(sin_test(7,:),5,'measured');

% Create loop to evaluate performance of filters and apply weights to freq.
filt_num = size(evo_filters,1);
perf_idx_mul = zeros(filt_num,11);
filt_out = zeros(filt_num,length(t));
% For passband signals
for m = 1:3
    for n = 1:filt_num
        filt_out(n,:) = filter(evo_filters(n,:),1,sin_test_corrupted(m,:));
        perf_idx_mul(n,m) = log1p(sum((filt_out(n,:)-sin_test(m,:)).^2)/length(t));
    end
    perf_idx_mul(:,m) = perf_idx_mul(:,m).*freq_weights(m);
end
% For stopband signals
for m = 4:7
    for n = 1:filt_num
        filt_out(n,:) = filter(evo_filters(n,:),1,sin_test_corrupted(m,:));
        perf_idx_mul(n,m) = log1p(sum((filt_out(n,:)-0).^2)/length(t));
    end
    perf_idx_mul(:,m) = perf_idx_mul(:,m).*freq_weights(m);
end


% Sum performance at each frequency to determine final performance index
perf_idx = sum(perf_idx_mul,2);
return
```

Published with MATLAB® R2015a

Appendix B: Verilog Code:

LFSR

```
//=============================================================================
// Verilog_LFSR2
//=============================================================================
// Updated LFSR based on one used in link below for stochastic chemical
// reaction simulation.
// Link: http://people.ece.cornell.edu/land/courses/ece5760/Chemical_
// Simulation/Two_reaction_per_step/VGA_320x240_chem_sim_2_MM_rand127bit.v

`ifndef MENG_PROJECT_LFSR2
`define MENG_PROJECT_LFSR2

module Verilog_LFSR2
(
      input  wire [3:0]   state_in,
      input               clk,
      input               reset,
      input  wire [127:0] seed,
      output wire [15:0]  rand_out
);
      // Individual shift registers
      reg [8:1] sr1, sr2, sr3, sr4, sr5, sr6, sr7, sr8, sr9, sr10, sr11,
                  sr12, sr13, sr14, sr15, sr16;

      // State names
            // State for start of module
      parameter mod_start = 4'h0;

      // Generate random number
      assign rand_out = {sr1[7],sr2[7],sr3[7],sr4[7],sr5[7],sr6[7],sr7[7],

sr8[7],sr9[7],sr10[7],sr11[7],sr12[7],sr13[7],
                                 sr14[7],sr15[7],sr16[7]};

      always @(posedge clk)
      begin
            if(reset)
            begin
                  // Initialize RNG
                  sr1  <= seed[8:1]    ;
                  sr2  <= seed[16:9]   ;
                  sr3  <= seed[24:17]  ;
                  sr4  <= seed[32:25]  ;
                  sr5  <= seed[40:33]  ;
                  sr6  <= seed[48:41]  ;
                  sr7  <= seed[56:49]  ;
                  sr8  <= seed[64:57]  ;
                  sr9  <= seed[72:65]  ;
                  sr10 <= seed[80:73]  ;
                  sr11 <= seed[88:81]  ;
                  sr12 <= seed[96:89]  ;
                  sr13 <= seed[104:96] ;
```

```
                sr14 <= seed[112:105];
                sr15 <= seed[120:113];
                sr16 <= {1'b0,seed[127:121]} ;
        end
        else
        // Update all shift registers in parallel
        begin
                // if(state_in == mod_start)
                // begin
                sr1  <= {sr1[7:1],  sr1[7]^sr16[7]} ;
                sr2  <= {sr2[7:1],  sr2[7]^sr1[8]}  ;
                sr3  <= {sr3[7:1],  sr3[7]^sr2[8]}  ;
                sr4  <= {sr4[7:1],  sr4[7]^sr3[8]}  ;
                sr5  <= {sr5[7:1],  sr5[7]^sr4[8]}  ;
                sr6  <= {sr6[7:1],  sr6[7]^sr5[8]}  ;
                sr7  <= {sr7[7:1],  sr7[7]^sr6[8]}  ;
                sr8  <= {sr8[7:1],  sr8[7]^sr7[8]}  ;
                sr9  <= {sr9[7:1],  sr9[7]^sr8[8]}  ;
                sr10 <= {sr10[7:1], sr10[7]^sr9[8]} ;
                sr11 <= {sr11[7:1], sr11[7]^sr10[8]};
                sr12 <= {sr12[7:1], sr12[7]^sr11[8]};
                sr13 <= {sr13[7:1], sr13[7]^sr12[8]};
                sr14 <= {sr14[7:1], sr14[7]^sr13[8]};
                sr15 <= {sr15[7:1], sr15[7]^sr14[8]};
                sr16 <= {sr16[6:1], sr16[7]^sr15[8]};
                // end
        end
    end
endmodule

    `endif /* MENG_PROJECT_LFSR */
```

Mutation Module

```
//=============================================================================
// Mutation_Mod
//=============================================================================
// Module that handles the mutation of genes in the genetic algorithm.
// For each input filter into the module, the mutation module triggers
// based on the value of an input random number. If that number is below
// ~1-2% of its maximum value, then a random coefficient in the filter is
// changed to a random value.
//
// Current setup will used fixed point of the form (fixed<8,6>)
// (MSB determines sign) and assumes a set of 20 filters with
// 16-coefficients each (=> 128 bits/filter)

`ifndef MENG_PROJECT_MUTATE
`define MENG_PROJECT_MUTATE

`include "Verilog_RegRst.v"
`include "Verilog_Mux_2.v"
`include "Verilog_Reg.v"
`include "Verilog_Incr.v"

module Mutation_Mod(
```

```
        input                           clk,
        input               [2559:0] in_filters,
        input  wire [255:0]      LFSR_out_idx,
        input  wire [159:0]      LFSR_out_rplc_bits,
        output reg  [2559:0] out_filters,
        input                    reset,
        output reg               req_rdy,
        output reg               resp_val,
        input                         req_val,
        input                         resp_rdy
);
        // Initialize signals for FSM control
        reg                INIT_IS_DONE;
        reg                CALC_IS_DONE;

        // Initialize temp signals used for mutation operation in CALC
        reg  [6:0]   coeff_idx_LSB;
        reg    [127:0] temp_filt;

        //------------------------------------------------------------------------
        // Parameters that set the indexing for the individual filters
        parameter filt0_MSB  = 127;
        parameter filt0_LSB  = 0;
        parameter filt1_MSB  = 255;
        parameter filt1_LSB  = 128;
        parameter filt2_MSB  = 383;
        parameter filt2_LSB  = 256;
        parameter filt3_MSB  = 511;
        parameter filt3_LSB  = 384;
        parameter filt4_MSB  = 639;
        parameter filt4_LSB  = 512;
        parameter filt5_MSB  = 767;
        parameter filt5_LSB  = 640;
        parameter filt6_MSB  = 895;
        parameter filt6_LSB  = 768;
        parameter filt7_MSB  = 1023;
        parameter filt7_LSB  = 896;
        parameter filt8_MSB  = 1151;
        parameter filt8_LSB  = 1023;
        parameter filt9_MSB  = 1279;
        parameter filt9_LSB  = 1152;
        parameter filt10_MSB = 1407;
        parameter filt10_LSB = 1280;
        parameter filt11_MSB = 1535;
        parameter filt11_LSB = 1408;
        parameter filt12_MSB = 1663;
        parameter filt12_LSB = 1536;
        parameter filt13_MSB = 1791;
        parameter filt13_LSB = 1664;
        parameter filt14_MSB = 1919;
        parameter filt14_LSB = 1792;
        parameter filt15_MSB = 2047;
        parameter filt15_LSB = 1920;
        parameter filt16_MSB = 2175;
        parameter filt16_LSB = 2048;
        parameter filt17_MSB = 2303;
        parameter filt17_LSB = 2176;
```

```
parameter filt18_MSB = 2431;
parameter filt18_LSB = 2304;
parameter filt19_MSB = 2559;
parameter filt19_LSB = 2432;
//------------------------------------------------------------------------

//------------------------------------------------------------------------
// Parameters that set the indexing for the filter coefficients
parameter coeff0_MSB    = 7;
parameter coeff0_LSB    = 0;
parameter coeff1_MSB    = 15;
parameter coeff1_LSB    = 8;
parameter coeff2_MSB    = 23;
parameter coeff2_LSB    = 16;
parameter coeff3_MSB    = 31;
parameter coeff3_LSB    = 24;
parameter coeff4_MSB    = 39;
parameter coeff4_LSB    = 32;
parameter coeff5_MSB    = 47;
parameter coeff5_LSB    = 40;
parameter coeff6_MSB    = 55;
parameter coeff6_LSB    = 48;
parameter coeff7_MSB    = 63;
parameter coeff7_LSB    = 56;
parameter coeff8_MSB    = 71;
parameter coeff8_LSB    = 64;
parameter coeff9_MSB    = 79;
parameter coeff9_LSB    = 72;
parameter coeff10_MSB   = 87;
parameter coeff10_LSB   = 80;
parameter coeff11_MSB   = 95;
parameter coeff11_LSB   = 88;
parameter coeff12_MSB   = 103;
parameter coeff12_LSB   = 96;
parameter coeff13_MSB   = 111;
parameter coeff13_LSB   = 104;
parameter coeff14_MSB   = 119;
parameter coeff14_LSB   = 112;
parameter coeff15_MSB   = 127;
parameter coeff15_LSB   = 120;
parameter coeff16_MSB   = 135;
parameter coeff16_LSB   = 128;
parameter coeff17_MSB   = 143;
parameter coeff17_LSB   = 136;
parameter coeff18_MSB   = 151;
parameter coeff18_LSB   = 144;
parameter coeff19_MSB   = 159;
parameter coeff19_LSB   = 152;
//------------------------------------------------------------------------

//------------------------------------------------------------------------
// Parameters for the mutation chance indexing
parameter LFSR_idx0_MSB  = 6;
parameter LFSR_idx0_LSB  = 0;
parameter LFSR_idx1_MSB  = 13;
parameter LFSR_idx1_LSB  = 7;
parameter LFSR_idx2_MSB  = 20;
```

```
parameter LFSR_idx2_LSB  = 14;
parameter LFSR_idx3_MSB  = 27;
parameter LFSR_idx3_LSB  = 21;
parameter LFSR_idx4_MSB  = 34;
parameter LFSR_idx4_LSB  = 28;
parameter LFSR_idx5_MSB  = 41;
parameter LFSR_idx5_LSB  = 35;
parameter LFSR_idx6_MSB  = 48;
parameter LFSR_idx6_LSB  = 42;
parameter LFSR_idx7_MSB  = 55;
parameter LFSR_idx7_LSB  = 49;
parameter LFSR_idx8_MSB  = 62;
parameter LFSR_idx8_LSB  = 56;
parameter LFSR_idx9_MSB  = 69;
parameter LFSR_idx9_LSB  = 63;
parameter LFSR_idx10_MSB = 76;
parameter LFSR_idx10_LSB = 70;
parameter LFSR_idx11_MSB = 83;
parameter LFSR_idx11_LSB = 77;
parameter LFSR_idx12_MSB = 90;
parameter LFSR_idx12_LSB = 84;
parameter LFSR_idx13_MSB = 97;
parameter LFSR_idx13_LSB = 91;
parameter LFSR_idx14_MSB = 104;
parameter LFSR_idx14_LSB = 98;
parameter LFSR_idx15_MSB = 111;
parameter LFSR_idx15_LSB = 105;
parameter LFSR_idx16_MSB = 118;
parameter LFSR_idx16_LSB = 112;
parameter LFSR_idx17_MSB = 125;
parameter LFSR_idx17_LSB = 119;
parameter LFSR_idx18_MSB = 132;
parameter LFSR_idx18_LSB = 126;
parameter LFSR_idx19_MSB = 139;
parameter LFSR_idx19_LSB = 133;
wire [3:0]   LFSR_init = LFSR_out_idx[139:136];
//----------------------------------------------------------------------


//======================================================================
// State Transition Logic
//======================================================================

// State elements
wire [2:0] STATE_IDLE;
assign STATE_IDLE = 3'b000;
wire [2:0] STATE_INIT;
assign STATE_INIT = 3'b001;
wire [2:0] STATE_CALC;
assign STATE_CALC = 3'b010;
wire [2:0] STATE_DONE;
assign STATE_DONE = 3'b011;

reg   [2:0] state_reg_in;
wire  [2:0] state_reg_out;

// Register for state storage
```

```
	Verilog_RegRst #(3, 0) state_reg
	(
		.clk	(clk),
		.reset	(reset),
		.in	(state_reg_in),
		.out	(state_reg_out)
	);

	logic [2:0] curr_state;
	logic [2:0] next_state;

	always @(*)
	begin
		curr_state = state_reg_out;
		next_state = state_reg_out;
		// Transitions out of IDLE state
		if( curr_state == STATE_IDLE ) begin
			if( req_val ) begin
				next_state = STATE_INIT;
			end
		end
		// Transitions out of INIT state
		if( curr_state == STATE_INIT ) begin
			if( INIT_IS_DONE ) begin
				next_state = STATE_CALC;
			end
		end
		// Transitions out of CALC state
		if( curr_state == STATE_CALC ) begin
			if( CALC_IS_DONE ) begin
				next_state = STATE_DONE;
			end
		end
		// Transitions out of DONE state
		if( curr_state == STATE_DONE ) begin
			if( resp_rdy ) begin
				next_state = STATE_IDLE;
			end
		end
		state_reg_in = next_state;
	end

	logic [2:0] current_state;
	always @(*)
	begin
		current_state = state_reg_out;
		// In IDLE state, wait for inputs to arrive and latch them
		if( current_state == STATE_IDLE ) begin
			req_rdy			= 1;
			resp_val		= 0;
			CALC_IS_DONE		= 0;
			INIT_IS_DONE		= 0;
		end
		// In INIT state, determine coefficient to mutate for filters
used
		// in the CALC state
		else if( current_state == STATE_INIT ) begin
```

```
                  case(LFSR_init)
                        4'd0:  coeff_idx_LSB = 0;
                        4'd1:  coeff_idx_LSB = 8;
                        4'd2:  coeff_idx_LSB = 16;
                        4'd3:  coeff_idx_LSB = 24;
                        4'd4:  coeff_idx_LSB = 32;
                        4'd5:  coeff_idx_LSB = 40;
                        4'd6:  coeff_idx_LSB = 48;
                        4'd7:  coeff_idx_LSB = 56;
                        4'd8:  coeff_idx_LSB = 64;
                        4'd9:  coeff_idx_LSB = 72;
                        4'd10: coeff_idx_LSB = 80;
                        4'd11: coeff_idx_LSB = 88;
                        4'd12: coeff_idx_LSB = 96;
                        4'd13: coeff_idx_LSB = 104;
                        4'd14: coeff_idx_LSB = 112;
                        4'd15: coeff_idx_LSB = 120;
                  endcase
                  INIT_IS_DONE       = 1;
            end
            // In CALC state, check LFSR output and determine if any
            // filters undergo mutation
            else if( current_state == STATE_CALC ) begin
                  // For each filter, perform a check on whether the random
number
                  // assigned to it is below 2 out of 128 (=> ~0.78%). If it
meets
                  // this condition, then replace coefficient chosen in the
INIT
                  // stage with an input random number

                  // Filt 1
                  if(LFSR_out_idx[LFSR_idx0_MSB:LFSR_idx0_LSB] < 2) begin
                        temp_filt = in_filters[filt0_MSB:filt0_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
```

```
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff0_MSB:coeff0_LSB];
                        endcase
                        out_filters[filt0_MSB:filt0_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt0_MSB:filt0_LSB] =
in_filters[filt0_MSB:filt0_LSB];
                  end

                  // Filt 2
                  if(LFSR_out_idx[LFSR_idx1_MSB:LFSR_idx1_LSB] < 2) begin
                        temp_filt = in_filters[filt1_MSB:filt1_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff1_MSB:coeff1_LSB];
                        endcase
                        out_filters[filt1_MSB:filt1_LSB] = temp_filt;
                  end
```

```
                  else begin
                        out_filters[filt1_MSB:filt1_LSB] =
in_filters[filt1_MSB:filt1_LSB];
                  end

                  // Filt 3
                  if(LFSR_out_idx[LFSR_idx2_MSB:LFSR_idx2_LSB] < 2) begin
                        temp_filt = in_filters[filt2_MSB:filt2_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff2_MSB:coeff2_LSB];
                        endcase
                        out_filters[filt2_MSB:filt2_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt2_MSB:filt2_LSB] =
in_filters[filt2_MSB:filt2_LSB];
                  end

                  // Filt 4
                  if(LFSR_out_idx[LFSR_idx3_MSB:LFSR_idx3_LSB] < 2) begin
                        temp_filt = in_filters[filt3_MSB:filt3_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
```

```
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff3_MSB:coeff3_LSB];
                        endcase
                        out_filters[filt3_MSB:filt3_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt3_MSB:filt3_LSB] =
in_filters[filt3_MSB:filt3_LSB];
                  end

                  // Filt 5
                  if(LFSR_out_idx[LFSR_idx4_MSB:LFSR_idx4_LSB] < 2) begin
                        temp_filt = in_filters[filt4_MSB:filt4_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
```

```
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff4_MSB:coeff4_LSB];
                        endcase
                        out_filters[filt4_MSB:filt4_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt4_MSB:filt4_LSB] =
in_filters[filt4_MSB:filt4_LSB];
                  end

                  // Filt 6
                  if(LFSR_out_idx[LFSR_idx5_MSB:LFSR_idx5_LSB] < 2) begin
                        temp_filt = in_filters[filt5_MSB:filt5_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
```

```
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff5_MSB:coeff5_LSB];
                        endcase
                        out_filters[filt5_MSB:filt5_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt5_MSB:filt5_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                  end

                  // Filt 7
                  if(LFSR_out_idx[LFSR_idx6_MSB:LFSR_idx6_LSB] < 2) begin
                        temp_filt = in_filters[filt6_MSB:filt6_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff6_MSB:coeff6_LSB];
                        endcase
                        out_filters[filt6_MSB:filt6_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt6_MSB:filt6_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                  end

                  // Filt 8
```

```
                  if(LFSR_out_idx[LFSR_idx7_MSB:LFSR_idx7_LSB] < 2) begin
                        temp_filt = in_filters[filt7_MSB:filt7_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff7_MSB:coeff7_LSB];
                        endcase
                        out_filters[filt7_MSB:filt7_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt7_MSB:filt7_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                  end

                  // Filt 9
                  if(LFSR_out_idx[LFSR_idx8_MSB:LFSR_idx8_LSB] < 2) begin
                        temp_filt = in_filters[filt8_MSB:filt8_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
```

```
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff8_MSB:coeff8_LSB];
                        endcase
                        out_filters[filt8_MSB:filt8_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt8_MSB:filt8_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                  end

                  // Filt 10
                  if(LFSR_out_idx[LFSR_idx9_MSB:LFSR_idx9_LSB] < 2) begin
                        temp_filt = in_filters[filt9_MSB:filt9_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
```

```
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff9_MSB:coeff9_LSB];
                        endcase
                        out_filters[filt9_MSB:filt9_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt9_MSB:filt9_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                  end

                  // Filt 11
                  if(LFSR_out_idx[LFSR_idx10_MSB:LFSR_idx10_LSB] < 2) begin
                        temp_filt = in_filters[filt10_MSB:filt10_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff10_MSB:coeff10_LSB];
                        endcase
                        out_filters[filt10_MSB:filt10_LSB] = temp_filt;
                  end
```

```
                  else begin
                        out_filters[filt10_MSB:filt10_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                  end

                  // Filt 12
                  if(LFSR_out_idx[LFSR_idx11_MSB:LFSR_idx11_LSB] < 2) begin
                        temp_filt = in_filters[filt11_MSB:filt11_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff11_MSB:coeff11_LSB];
                        endcase
                        out_filters[filt11_MSB:filt11_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt11_MSB:filt11_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                  end

                  // Filt 13
                  if(LFSR_out_idx[LFSR_idx12_MSB:LFSR_idx12_LSB] < 2) begin
                        temp_filt = in_filters[filt12_MSB:filt12_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
```

```
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff12_MSB:coeff12_LSB];
                        endcase
                        out_filters[filt12_MSB:filt12_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt12_MSB:filt12_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                  end

                  // Filt 14
                  if(LFSR_out_idx[LFSR_idx13_MSB:LFSR_idx13_LSB] < 2) begin
                        temp_filt = in_filters[filt13_MSB:filt13_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
```

```
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff13_MSB:coeff13_LSB];
                        endcase
                        out_filters[filt13_MSB:filt13_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt13_MSB:filt13_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                  end

                  // Filt 15
                  if(LFSR_out_idx[LFSR_idx14_MSB:LFSR_idx14_LSB] < 2) begin
                        temp_filt = in_filters[filt14_MSB:filt14_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
```

```
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff14_MSB:coeff14_LSB];
                        endcase
                        out_filters[filt14_MSB:filt14_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt14_MSB:filt14_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                  end

                  // Filt 16
                  if(LFSR_out_idx[LFSR_idx15_MSB:LFSR_idx15_LSB] < 2) begin
                        temp_filt = in_filters[filt15_MSB:filt15_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff15_MSB:coeff15_LSB];
                        endcase
                        out_filters[filt15_MSB:filt15_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt15_MSB:filt15_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                  end

                  // Filt 17
```

```
                  if(LFSR_out_idx[LFSR_idx16_MSB:LFSR_idx16_LSB] < 2) begin
                        temp_filt = in_filters[filt16_MSB:filt16_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff16_MSB:coeff16_LSB];
                        endcase
                        out_filters[filt16_MSB:filt16_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt16_MSB:filt16_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                  end

                  // Filt 18
                  if(LFSR_out_idx[LFSR_idx17_MSB:LFSR_idx17_LSB] < 2) begin
                        temp_filt = in_filters[filt17_MSB:filt17_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
```

```
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff17_MSB:coeff17_LSB];
                        endcase
                        out_filters[filt17_MSB:filt17_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt17_MSB:filt17_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                  end

                  // Filt 19
                  if(LFSR_out_idx[LFSR_idx18_MSB:LFSR_idx18_LSB] < 2) begin
                        temp_filt = in_filters[filt18_MSB:filt18_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
```

```
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff18_MSB:coeff18_LSB];
                        endcase
                        out_filters[filt18_MSB:filt18_LSB] = temp_filt;
                  end
                  else begin
                        out_filters[filt18_MSB:filt18_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                  end

                  // Filt 20
                  if(LFSR_out_idx[LFSR_idx19_MSB:LFSR_idx19_LSB] < 2) begin
                        temp_filt = in_filters[filt19_MSB:filt19_LSB];
                        case(coeff_idx_LSB)
                              7'd0:temp_filt[7:0]       =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd8:temp_filt[15:8]      =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd16:temp_filt[23:16]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd24:temp_filt[31:24]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd32:temp_filt[39:32]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd40:temp_filt[47:40]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd48:temp_filt[55:48]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd56:temp_filt[63:56]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd64:temp_filt[71:64]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd72:temp_filt[79:72]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd80:temp_filt[87:80]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd88:temp_filt[95:88]    =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd96:temp_filt[103:96]   =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd104:temp_filt[111:104] =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd112:temp_filt[119:112] =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                              7'd120:temp_filt[127:120] =
LFSR_out_rplc_bits[coeff19_MSB:coeff19_LSB];
                        endcase
                        out_filters[filt19_MSB:filt19_LSB] = temp_filt;
                  end
```

```
                  else begin
                        out_filters[filt19_MSB:filt19_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                  end

                  CALC_IS_DONE       = 1;
            end
            // In DONE stage, wait for output transaction to occur
            else if ( current_state == STATE_DONE ) begin
                  req_rdy                  = 0;
                  resp_val           = 1;
            end
      end

endmodule

      `endif /* MENG_PROJECT_MUTATE */
```

Crossover Module

```
//=============================================================================
// Crossover_Mod
//=============================================================================
// Module that handles the crossover of genes in the genetic algorithm.
// the module will pick a random point in the bitstream representing the
// "genes" in a filter and combine separate filters to create a new
// generation.
//
// For example, for filters A and B, each with 5 coefficients,
// suppose the crossover split point is 3. Therefore, the new filters
// C and D will have the following coefficients:
//                C: [A1, A2, A3, B4, B5]
//                D: [B1, B2, B3, A4, A5]
//
// Current setup will used fixed point of the form (fixed<8,6>)
// (MSB determines sign) and assumes a set of 20 filters with
// 16-coefficients each (=> 128 bits/filter)
//
// Will also need performance threshold and a bitstream with the
// performance levels of each filter

`ifndef MENG_PROJECT_CROSS
`define MENG_PROJECT_CROSS

`include "Verilog_Mux_2.v"
`include "Verilog_Reg.v"
`include "Verilog_Incr.v"
`include "Verilog_LFSR2.v"
`include "Verilog_RegRst.v"

module Crossover_Mod(
      input                         clk,
      input             [2559:0] in_filters,
      input             [99:0]       in_filt_perf,
      input             [4:0]    perf_thresh,
      input  wire [3:0]  LFSR_crossover,
      output reg  [2559:0] out_filters,
```

```
        input                       reset,
        output reg                  req_rdy,
        output reg                  resp_val,
        input                       req_val,
        input                       resp_rdy
);
        //------------------------------------------------------------------------
        // Parameters that set the indexing for the individual filters
        parameter filt0_MSB  = 127;
        parameter filt0_LSB  = 0;
        parameter filt1_MSB  = 255;
        parameter filt1_LSB  = 128;
        parameter filt2_MSB  = 383;
        parameter filt2_LSB  = 256;
        parameter filt3_MSB  = 511;
        parameter filt3_LSB  = 384;
        parameter filt4_MSB  = 639;
        parameter filt4_LSB  = 512;
        parameter filt5_MSB  = 767;
        parameter filt5_LSB  = 640;
        parameter filt6_MSB  = 895;
        parameter filt6_LSB  = 768;
        parameter filt7_MSB  = 1023;
        parameter filt7_LSB  = 896;
        parameter filt8_MSB  = 1151;
        parameter filt8_LSB  = 1023;
        parameter filt9_MSB  = 1279;
        parameter filt9_LSB  = 1152;
        parameter filt10_MSB = 1407;
        parameter filt10_LSB = 1280;
        parameter filt11_MSB = 1535;
        parameter filt11_LSB = 1408;
        parameter filt12_MSB = 1663;
        parameter filt12_LSB = 1536;
        parameter filt13_MSB = 1791;
        parameter filt13_LSB = 1664;
        parameter filt14_MSB = 1919;
        parameter filt14_LSB = 1792;
        parameter filt15_MSB = 2047;
        parameter filt15_LSB = 1920;
        parameter filt16_MSB = 2175;
        parameter filt16_LSB = 2048;
        parameter filt17_MSB = 2303;
        parameter filt17_LSB = 2176;
        parameter filt18_MSB = 2431;
        parameter filt18_LSB = 2304;
        parameter filt19_MSB = 2559;
        parameter filt19_LSB = 2432;
        //------------------------------------------------------------------------

        //------------------------------------------------------------------------
        // Parameters that set the indexing for crossover point
        parameter cross_1  = 7;
        parameter cross_2  = 15;
        parameter cross_3  = 23;
        parameter cross_4  = 31;
        parameter cross_5  = 39;
```

```
parameter cross_6  = 47;
parameter cross_7  = 55;
parameter cross_8  = 63;
parameter cross_9  = 71;
parameter cross_10       = 79;
parameter cross_11       = 87;
parameter cross_12       = 95;
parameter cross_13       = 103;
parameter cross_14       = 111;
parameter cross_15       = 119;
parameter cross_16       = 127;
//----------------------------------------------------------------------

// Initialize wire that stores high-performing filters
reg  [1279:0] good_filters;
reg  [11:0]   idx_MSB;
reg  [11:0]   idx_LSB;
reg  [4:0]    good_filt_count;
// Initialize temp regs and muxes used for crossover
reg  [95:0]   temp_filt1;
reg  [95:0]   temp_filt2;

//======================================================================
// State Transition Logic
//======================================================================

// State elements
wire [2:0] STATE_IDLE;
assign STATE_IDLE = 3'b000;
wire [2:0] STATE_INIT;
assign STATE_INIT = 3'b001;
wire [2:0] STATE_CALC;
assign STATE_CALC = 3'b010;
wire [2:0] STATE_DONE;
assign STATE_DONE = 3'b011;

reg               INIT_IS_DONE;
reg               CALC_IS_DONE;
reg   [2:0] state_reg_in;
wire  [2:0] state_reg_out;


// Register for state storage
Verilog_RegRst #(3, 0) state_reg
(
      .clk    (clk),
      .reset  (reset),
      .in     (state_reg_in),
      .out    (state_reg_out)
);

logic [2:0] curr_state;
logic [2:0] next_state;

always @(*)
begin
      curr_state = state_reg_out;
```

```
            next_state = state_reg_out;
            // Transitions out of IDLE state
            if( curr_state == STATE_IDLE ) begin
                  if( req_val ) begin
                        next_state = STATE_INIT;
                  end
            end
            // Transitions out of INIT state
            if( curr_state == STATE_INIT ) begin
                  if( INIT_IS_DONE ) begin
                        next_state = STATE_CALC;
                  end
            end
            // Transitions out of CALC state
            if( curr_state == STATE_CALC ) begin
                  if( CALC_IS_DONE ) begin
                        next_state = STATE_DONE;
                  end
            end
            // Transitions out of DONE state
            if( curr_state == STATE_DONE ) begin
                  if( resp_rdy ) begin
                        next_state = STATE_IDLE;
                  end
            end
            state_reg_in = next_state;
      end


      //========================================================================
      // State Output Logic
      //========================================================================
      logic [2:0] current_state;

      always @(*)
      begin
            current_state = state_reg_out;
            // In IDLE state, wait for inputs to arrive and latch them
            if( current_state == STATE_IDLE ) begin
                  req_rdy                 = 1;
                  resp_val          = 0;
                  INIT_IS_DONE      = 0;
                  CALC_IS_DONE      = 0;
                  good_filt_count   = 0;
            end
            // In INIT state, put high_performing filters into one wire
            else if( current_state == STATE_INIT ) begin
                  req_rdy                 = 0;
                  resp_val          = 0;

                  // Filt 1
                  if(in_filt_perf[4:0] > perf_thresh)
                  begin
                        good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt0_MSB:filt0_LSB];
                        good_filt_count = good_filt_count+1;
                  end
```

```
                        // Filt 2
                        if(in_filt_perf[9:5] > perf_thresh)
                        begin
                              if(good_filt_count) begin
                                    good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt1_MSB:filt1_LSB];
                              end
                              else begin
                                    good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt1_MSB:filt1_LSB];
                              end
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 3
                        if(in_filt_perf[14:10] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt2_MSB:filt2_LSB];
                                    5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt2_MSB:filt2_LSB];
                                    5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt2_MSB:filt2_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 4
                        if(in_filt_perf[19:15] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt3_MSB:filt3_LSB];
                                    5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt3_MSB:filt3_LSB];
                                    5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt3_MSB:filt3_LSB];
                                    5'd3: good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt3_MSB:filt3_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 5
                        if(in_filt_perf[24:20] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt4_MSB:filt4_LSB];
                                    5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt4_MSB:filt4_LSB];
                                    5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt4_MSB:filt4_LSB];
                                    5'd3: good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt4_MSB:filt4_LSB];
```

```
                                    5'd4: good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt4_MSB:filt4_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 6
                        if(in_filt_perf[29:25] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                                    5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                                    5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                                    5'd3: good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                                    5'd4: good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                                    5'd5: good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt5_MSB:filt5_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 7
                        if(in_filt_perf[34:30] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                                    5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                                    5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                                    5'd3: good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                                    5'd4: good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                                    5'd5: good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                                    5'd6: good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt6_MSB:filt6_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 8
                        if(in_filt_perf[39:35] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                                    5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt7_MSB:filt7_LSB];
```

```
                                5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                                5'd3: good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                                5'd4: good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                                5'd5: good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                                5'd6: good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                                5'd7: good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt7_MSB:filt7_LSB];
                        endcase
                        good_filt_count = good_filt_count+1;
                end

                // Filt 9
                if(in_filt_perf[44:40] > perf_thresh)
                begin
                        case(good_filt_count)
                                5'd0: good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd1: good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd2: good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd3: good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd4: good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd5: good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd6: good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd7: good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                                5'd8: good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt8_MSB:filt8_LSB];
                        endcase
                        good_filt_count = good_filt_count+1;
                end

                // Filt 10
                if(in_filt_perf[49:45] > perf_thresh)
                begin
                        case(good_filt_count)
                                5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt9_MSB:filt9_LSB];
```

```
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt9_MSB:filt9_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 11
                        if(in_filt_perf[54:50] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt10_MSB:filt10_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 12
                        if(in_filt_perf[59:55] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt11_MSB:filt11_LSB];
```

```
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt11_MSB:filt11_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 13
                        if(in_filt_perf[64:60] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt12_MSB:filt12_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 14
                        if(in_filt_perf[69:65] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt13_MSB:filt13_LSB];
```

```
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt13_MSB:filt13_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 15
                        if(in_filt_perf[74:70] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt14_MSB:filt14_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 16
                        if(in_filt_perf[79:75] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt15_MSB:filt15_LSB];
```

```
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt15_MSB:filt15_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 17
                        if(in_filt_perf[84:80] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt16_MSB:filt16_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 18
                        if(in_filt_perf[89:85] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt17_MSB:filt17_LSB];
```

```
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt17_MSB:filt17_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 19
                        if(in_filt_perf[94:90] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                                    5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt18_MSB:filt18_LSB];
                              endcase
                              good_filt_count = good_filt_count+1;
                        end

                        // Filt 20
                        if(in_filt_perf[99:95] > perf_thresh)
                        begin
                              case(good_filt_count)
                                    5'd0:  good_filters[filt0_MSB:filt0_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                                    5'd1:  good_filters[filt1_MSB:filt1_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                                    5'd2:  good_filters[filt2_MSB:filt2_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                                    5'd3:  good_filters[filt3_MSB:filt3_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                                    5'd4:  good_filters[filt4_MSB:filt4_LSB] =
in_filters[filt19_MSB:filt19_LSB];
```

```
                              5'd5:  good_filters[filt5_MSB:filt5_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                              5'd6:  good_filters[filt6_MSB:filt6_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                              5'd7:  good_filters[filt7_MSB:filt7_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                              5'd8:  good_filters[filt8_MSB:filt8_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                              5'd9:  good_filters[filt9_MSB:filt9_LSB] =
in_filters[filt19_MSB:filt19_LSB];
                        endcase
                        good_filt_count = good_filt_count+1;
                  end

                  INIT_IS_DONE       = 1;
            end
            // In CALC state, perform the crossover on the good filters to
            // increase population to original size
            else if( current_state == STATE_CALC ) begin

                  case(LFSR_crossover)
                        4'd0:
                        begin // If crossover point is 0th coefficient [bit
0]
                              out_filters = in_filters;
                        end
                        4'd1:
                        begin // If crossover point is 1st coefficient [bit
8]
                              out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_1+1],good_filters[filt9_LSB+cross_1:f
ilt9_LSB]}; // {gf0:gf9}
                              out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_1+1],good_filters[filt1_LSB+cross_1:f
ilt1_LSB]}; // {gf0:gf1}
                              out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_1+1],good_filters[filt0_LSB+cross_1:f
ilt0_LSB]}; // {gf1:gf0}
                              out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_1+1],good_filters[filt2_LSB+cross_1:f
ilt2_LSB]}; // {gf1:gf2}
                              out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_1+1],good_filters[filt1_LSB+cross_1:f
ilt1_LSB]}; // {gf2:gf1}
                              out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_1+1],good_filters[filt3_LSB+cross_1:f
ilt3_LSB]}; // {gf2:gf3}
                              out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_1+1],good_filters[filt2_LSB+cross_1:f
ilt2_LSB]}; // {gf3:gf2}
                              out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_1+1],good_filters[filt4_LSB+cross_1:f
ilt4_LSB]}; // {gf3:gf4}
                              out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_1+1],good_filters[filt3_LSB+cross_1:f
ilt3_LSB]}; // {gf4:gf3}
```

```
                                out_filters[filt9_MSB:filt9_LSB]    =
{good_filters[filt4_MSB:filt4_LSB+cross_1+1],good_filters[filt5_LSB+cross_1:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_1+1],good_filters[filt4_LSB+cross_1:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_1+1],good_filters[filt6_LSB+cross_1:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_1+1],good_filters[filt5_LSB+cross_1:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_1+1],good_filters[filt7_LSB+cross_1:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_1+1],good_filters[filt6_LSB+cross_1:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_1+1],good_filters[filt8_LSB+cross_1:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_1+1],good_filters[filt7_LSB+cross_1:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_1+1],good_filters[filt9_LSB+cross_1:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_1+1],good_filters[filt8_LSB+cross_1:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_1+1],good_filters[filt0_LSB+cross_1:f
ilt0_LSB]}; // {gf9:gf0}
                        end
                        4'd2:
                        begin // If crossover point is 2nd coefficient [bit
16]
                                out_filters[filt0_MSB:filt0_LSB]    =
{good_filters[filt0_MSB:filt0_LSB+cross_2+1],good_filters[filt9_LSB+cross_2:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]    =
{good_filters[filt0_MSB:filt0_LSB+cross_2+1],good_filters[filt1_LSB+cross_2:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]    =
{good_filters[filt1_MSB:filt1_LSB+cross_2+1],good_filters[filt0_LSB+cross_2:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]    =
{good_filters[filt1_MSB:filt1_LSB+cross_2+1],good_filters[filt2_LSB+cross_2:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]    =
{good_filters[filt2_MSB:filt2_LSB+cross_2+1],good_filters[filt1_LSB+cross_2:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]    =
{good_filters[filt2_MSB:filt2_LSB+cross_2+1],good_filters[filt3_LSB+cross_2:f
ilt3_LSB]}; // {gf2:gf3}
```

```
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_2+1],good_filters[filt2_LSB+cross_2:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_2+1],good_filters[filt4_LSB+cross_2:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_2+1],good_filters[filt3_LSB+cross_2:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_2+1],good_filters[filt5_LSB+cross_2:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_2+1],good_filters[filt4_LSB+cross_2:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_2+1],good_filters[filt6_LSB+cross_2:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_2+1],good_filters[filt5_LSB+cross_2:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_2+1],good_filters[filt7_LSB+cross_2:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_2+1],good_filters[filt6_LSB+cross_2:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_2+1],good_filters[filt8_LSB+cross_2:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_2+1],good_filters[filt7_LSB+cross_2:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_2+1],good_filters[filt9_LSB+cross_2:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_2+1],good_filters[filt8_LSB+cross_2:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_2+1],good_filters[filt0_LSB+cross_2:f
ilt0_LSB]}; // {gf9:gf0}
                        end
                        4'd3:
                        begin // If crossover point is 3rd coefficient [bit
24]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_3+1],good_filters[filt9_LSB+cross_3:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_3+1],good_filters[filt1_LSB+cross_3:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_3+1],good_filters[filt0_LSB+cross_3:f
ilt0_LSB]}; // {gf1:gf0}
```

```
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_3+1],good_filters[filt2_LSB+cross_3:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_3+1],good_filters[filt1_LSB+cross_3:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_3+1],good_filters[filt3_LSB+cross_3:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_3+1],good_filters[filt2_LSB+cross_3:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_3+1],good_filters[filt4_LSB+cross_3:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_3+1],good_filters[filt3_LSB+cross_3:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_3+1],good_filters[filt5_LSB+cross_3:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_3+1],good_filters[filt4_LSB+cross_3:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_3+1],good_filters[filt6_LSB+cross_3:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_3+1],good_filters[filt5_LSB+cross_3:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_3+1],good_filters[filt7_LSB+cross_3:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_3+1],good_filters[filt6_LSB+cross_3:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_3+1],good_filters[filt8_LSB+cross_3:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_3+1],good_filters[filt7_LSB+cross_3:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_3+1],good_filters[filt9_LSB+cross_3:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_3+1],good_filters[filt8_LSB+cross_3:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_3+1],good_filters[filt0_LSB+cross_3:f
ilt0_LSB]}; // {gf9:gf0}
                            end
                            4'd4:
                            begin // If crossover point is 4th coefficient [bit
32]
```

```
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_4+1],good_filters[filt9_LSB+cross_4:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_4+1],good_filters[filt1_LSB+cross_4:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_4+1],good_filters[filt0_LSB+cross_4:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_4+1],good_filters[filt2_LSB+cross_4:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_4+1],good_filters[filt1_LSB+cross_4:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_4+1],good_filters[filt3_LSB+cross_4:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_4+1],good_filters[filt2_LSB+cross_4:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_4+1],good_filters[filt4_LSB+cross_4:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_4+1],good_filters[filt3_LSB+cross_4:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_4+1],good_filters[filt5_LSB+cross_4:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_4+1],good_filters[filt4_LSB+cross_4:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_4+1],good_filters[filt6_LSB+cross_4:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_4+1],good_filters[filt5_LSB+cross_4:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_4+1],good_filters[filt7_LSB+cross_4:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_4+1],good_filters[filt6_LSB+cross_4:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_4+1],good_filters[filt8_LSB+cross_4:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_4+1],good_filters[filt7_LSB+cross_4:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_4+1],good_filters[filt9_LSB+cross_4:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_4+1],good_filters[filt8_LSB+cross_4:f
ilt8_LSB]}; // {gf9:gf8}
```

```
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_4+1],good_filters[filt0_LSB+cross_4:f
ilt0_LSB]}; // {gf9:gf0}
                            end
                            4'd5:
                            begin // If crossover point is 5th coefficient [bit
40]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_5+1],good_filters[filt9_LSB+cross_5:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_5+1],good_filters[filt1_LSB+cross_5:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_5+1],good_filters[filt0_LSB+cross_5:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_5+1],good_filters[filt2_LSB+cross_5:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_5+1],good_filters[filt1_LSB+cross_5:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_5+1],good_filters[filt3_LSB+cross_5:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_5+1],good_filters[filt2_LSB+cross_5:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_5+1],good_filters[filt4_LSB+cross_5:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_5+1],good_filters[filt3_LSB+cross_5:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_5+1],good_filters[filt5_LSB+cross_5:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_5+1],good_filters[filt4_LSB+cross_5:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_5+1],good_filters[filt6_LSB+cross_5:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_5+1],good_filters[filt5_LSB+cross_5:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_5+1],good_filters[filt7_LSB+cross_5:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_5+1],good_filters[filt6_LSB+cross_5:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_5+1],good_filters[filt8_LSB+cross_5:f
ilt8_LSB]}; // {gf7:gf8}
```

```
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_5+1],good_filters[filt7_LSB+cross_5:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_5+1],good_filters[filt9_LSB+cross_5:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_5+1],good_filters[filt8_LSB+cross_5:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_5+1],good_filters[filt0_LSB+cross_5:f
ilt0_LSB]}; // {gf9:gf0}
                            end
                            4'd6:
                            begin // If crossover point is 6th coefficient [bit
48]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_6+1],good_filters[filt9_LSB+cross_6:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_6+1],good_filters[filt1_LSB+cross_6:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_6+1],good_filters[filt0_LSB+cross_6:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_6+1],good_filters[filt2_LSB+cross_6:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_6+1],good_filters[filt1_LSB+cross_6:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_6+1],good_filters[filt3_LSB+cross_6:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_6+1],good_filters[filt2_LSB+cross_6:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_6+1],good_filters[filt4_LSB+cross_6:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_6+1],good_filters[filt3_LSB+cross_6:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_6+1],good_filters[filt5_LSB+cross_6:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_6+1],good_filters[filt4_LSB+cross_6:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_6+1],good_filters[filt6_LSB+cross_6:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_6+1],good_filters[filt5_LSB+cross_6:f
ilt5_LSB]}; // {gf6:gf5}
```

```
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_6+1],good_filters[filt7_LSB+cross_6:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_6+1],good_filters[filt6_LSB+cross_6:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_6+1],good_filters[filt8_LSB+cross_6:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_6+1],good_filters[filt7_LSB+cross_6:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_6+1],good_filters[filt9_LSB+cross_6:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_6+1],good_filters[filt8_LSB+cross_6:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_6+1],good_filters[filt0_LSB+cross_6:f
ilt0_LSB]}; // {gf9:gf0}
                        end
                        4'd7:
                        begin // If crossover point is 7th coefficient [bit
56]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_7+1],good_filters[filt9_LSB+cross_7:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_7+1],good_filters[filt1_LSB+cross_7:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_7+1],good_filters[filt0_LSB+cross_7:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_7+1],good_filters[filt2_LSB+cross_7:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_7+1],good_filters[filt1_LSB+cross_7:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_7+1],good_filters[filt3_LSB+cross_7:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_7+1],good_filters[filt2_LSB+cross_7:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_7+1],good_filters[filt4_LSB+cross_7:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_7+1],good_filters[filt3_LSB+cross_7:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_7+1],good_filters[filt5_LSB+cross_7:f
ilt5_LSB]}; // {gf4:gf5}
```

```
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_7+1],good_filters[filt4_LSB+cross_7:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_7+1],good_filters[filt6_LSB+cross_7:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_7+1],good_filters[filt5_LSB+cross_7:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_7+1],good_filters[filt7_LSB+cross_7:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_7+1],good_filters[filt6_LSB+cross_7:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_7+1],good_filters[filt8_LSB+cross_7:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_7+1],good_filters[filt7_LSB+cross_7:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_7+1],good_filters[filt9_LSB+cross_7:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_7+1],good_filters[filt8_LSB+cross_7:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_7+1],good_filters[filt0_LSB+cross_7:f
ilt0_LSB]}; // {gf9:gf0}
                        end
                        4'd8:
                        begin // If crossover point is 8th coefficient [bit
64]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_8+1],good_filters[filt9_LSB+cross_8:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_8+1],good_filters[filt1_LSB+cross_8:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_8+1],good_filters[filt0_LSB+cross_8:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_8+1],good_filters[filt2_LSB+cross_8:f
ilt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_8+1],good_filters[filt1_LSB+cross_8:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_8+1],good_filters[filt3_LSB+cross_8:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_8+1],good_filters[filt2_LSB+cross_8:f
ilt2_LSB]}; // {gf3:gf2}
```

```
                                out_filters[filt7_MSB:filt7_LSB]    =
{good_filters[filt3_MSB:filt3_LSB+cross_8+1],good_filters[filt4_LSB+cross_8:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]    =
{good_filters[filt4_MSB:filt4_LSB+cross_8+1],good_filters[filt3_LSB+cross_8:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]    =
{good_filters[filt4_MSB:filt4_LSB+cross_8+1],good_filters[filt5_LSB+cross_8:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_8+1],good_filters[filt4_LSB+cross_8:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_8+1],good_filters[filt6_LSB+cross_8:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_8+1],good_filters[filt5_LSB+cross_8:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_8+1],good_filters[filt7_LSB+cross_8:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_8+1],good_filters[filt6_LSB+cross_8:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_8+1],good_filters[filt8_LSB+cross_8:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_8+1],good_filters[filt7_LSB+cross_8:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_8+1],good_filters[filt9_LSB+cross_8:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_8+1],good_filters[filt8_LSB+cross_8:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_8+1],good_filters[filt0_LSB+cross_8:f
ilt0_LSB]}; // {gf9:gf0}
                        end
                        4'd9:
                        begin // If crossover point is 9th coefficient [bit
72]
                                out_filters[filt0_MSB:filt0_LSB]    =
{good_filters[filt0_MSB:filt0_LSB+cross_9+1],good_filters[filt9_LSB+cross_9:f
ilt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]    =
{good_filters[filt0_MSB:filt0_LSB+cross_9+1],good_filters[filt1_LSB+cross_9:f
ilt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]    =
{good_filters[filt1_MSB:filt1_LSB+cross_9+1],good_filters[filt0_LSB+cross_9:f
ilt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]    =
{good_filters[filt1_MSB:filt1_LSB+cross_9+1],good_filters[filt2_LSB+cross_9:f
ilt2_LSB]}; // {gf1:gf2}
```

```
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_9+1],good_filters[filt1_LSB+cross_9:f
ilt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_9+1],good_filters[filt3_LSB+cross_9:f
ilt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_9+1],good_filters[filt2_LSB+cross_9:f
ilt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_9+1],good_filters[filt4_LSB+cross_9:f
ilt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_9+1],good_filters[filt3_LSB+cross_9:f
ilt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_9+1],good_filters[filt5_LSB+cross_9:f
ilt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_9+1],good_filters[filt4_LSB+cross_9:f
ilt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_9+1],good_filters[filt6_LSB+cross_9:f
ilt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_9+1],good_filters[filt5_LSB+cross_9:f
ilt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_9+1],good_filters[filt7_LSB+cross_9:f
ilt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_9+1],good_filters[filt6_LSB+cross_9:f
ilt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_9+1],good_filters[filt8_LSB+cross_9:f
ilt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_9+1],good_filters[filt7_LSB+cross_9:f
ilt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_9+1],good_filters[filt9_LSB+cross_9:f
ilt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_9+1],good_filters[filt8_LSB+cross_9:f
ilt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_9+1],good_filters[filt0_LSB+cross_9:f
ilt0_LSB]}; // {gf9:gf0}
                        end
                        4'd10:
                        begin // If crossover point is 10th coefficient [bit
80]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_10+1],good_filters[filt9_LSB+cross_10
:filt9_LSB]}; // {gf0:gf9}
```

```
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_10+1],good_filters[filt1_LSB+cross_10
:filt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_10+1],good_filters[filt0_LSB+cross_10
:filt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_10+1],good_filters[filt2_LSB+cross_10
:filt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_10+1],good_filters[filt1_LSB+cross_10
:filt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_10+1],good_filters[filt3_LSB+cross_10
:filt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_10+1],good_filters[filt2_LSB+cross_10
:filt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_10+1],good_filters[filt4_LSB+cross_10
:filt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_10+1],good_filters[filt3_LSB+cross_10
:filt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_10+1],good_filters[filt5_LSB+cross_10
:filt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_10+1],good_filters[filt4_LSB+cross_10
:filt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_10+1],good_filters[filt6_LSB+cross_10
:filt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_10+1],good_filters[filt5_LSB+cross_10
:filt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_10+1],good_filters[filt7_LSB+cross_10
:filt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_10+1],good_filters[filt6_LSB+cross_10
:filt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_10+1],good_filters[filt8_LSB+cross_10
:filt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_10+1],good_filters[filt7_LSB+cross_10
:filt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_10+1],good_filters[filt9_LSB+cross_10
:filt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_10+1],good_filters[filt8_LSB+cross_10
:filt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_10+1],good_filters[filt0_LSB+cross_10
:filt0_LSB]}; // {gf9:gf0}
```

```
                        end
                        4'd11:
                        begin // If crossover point is 11th coefficient [bit
88]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_11+1],good_filters[filt9_LSB+cross_11
:filt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_11+1],good_filters[filt1_LSB+cross_11
:filt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_11+1],good_filters[filt0_LSB+cross_11
:filt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_11+1],good_filters[filt2_LSB+cross_11
:filt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_11+1],good_filters[filt1_LSB+cross_11
:filt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_11+1],good_filters[filt3_LSB+cross_11
:filt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_11+1],good_filters[filt2_LSB+cross_11
:filt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_11+1],good_filters[filt4_LSB+cross_11
:filt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_11+1],good_filters[filt3_LSB+cross_11
:filt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_11+1],good_filters[filt5_LSB+cross_11
:filt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_11+1],good_filters[filt4_LSB+cross_11
:filt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_11+1],good_filters[filt6_LSB+cross_11
:filt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_11+1],good_filters[filt5_LSB+cross_11
:filt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_11+1],good_filters[filt7_LSB+cross_11
:filt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_11+1],good_filters[filt6_LSB+cross_11
:filt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_11+1],good_filters[filt8_LSB+cross_11
:filt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_11+1],good_filters[filt7_LSB+cross_11
:filt7_LSB]}; // {gf8:gf7}
```

```
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_11+1],good_filters[filt9_LSB+cross_11
:filt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_11+1],good_filters[filt8_LSB+cross_11
:filt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_11+1],good_filters[filt0_LSB+cross_11
:filt0_LSB]}; // {gf9:gf0}
                            end
                            4'd12:
                            begin // If crossover point is 12th coefficient [bit
96]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_12+1],good_filters[filt9_LSB+cross_12
:filt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_12+1],good_filters[filt1_LSB+cross_12
:filt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_12+1],good_filters[filt0_LSB+cross_12
:filt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_12+1],good_filters[filt2_LSB+cross_12
:filt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_12+1],good_filters[filt1_LSB+cross_12
:filt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_12+1],good_filters[filt3_LSB+cross_12
:filt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_12+1],good_filters[filt2_LSB+cross_12
:filt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_12+1],good_filters[filt4_LSB+cross_12
:filt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_12+1],good_filters[filt3_LSB+cross_12
:filt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_12+1],good_filters[filt5_LSB+cross_12
:filt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_12+1],good_filters[filt4_LSB+cross_12
:filt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_12+1],good_filters[filt6_LSB+cross_12
:filt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_12+1],good_filters[filt5_LSB+cross_12
:filt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_12+1],good_filters[filt7_LSB+cross_12
:filt7_LSB]}; // {gf6:gf7}
```

```
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_12+1],good_filters[filt6_LSB+cross_12
:filt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_12+1],good_filters[filt8_LSB+cross_12
:filt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_12+1],good_filters[filt7_LSB+cross_12
:filt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_12+1],good_filters[filt9_LSB+cross_12
:filt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_12+1],good_filters[filt8_LSB+cross_12
:filt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_12+1],good_filters[filt0_LSB+cross_12
:filt0_LSB]}; // {gf9:gf0}
                            end
                            4'd13:
                            begin // If crossover point is 13th coefficient [bit
104]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_13+1],good_filters[filt9_LSB+cross_13
:filt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_13+1],good_filters[filt1_LSB+cross_13
:filt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_13+1],good_filters[filt0_LSB+cross_13
:filt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_13+1],good_filters[filt2_LSB+cross_13
:filt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_13+1],good_filters[filt1_LSB+cross_13
:filt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_13+1],good_filters[filt3_LSB+cross_13
:filt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_13+1],good_filters[filt2_LSB+cross_13
:filt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_13+1],good_filters[filt4_LSB+cross_13
:filt4_LSB]}; // {gf3:gf4}
                                out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_13+1],good_filters[filt3_LSB+cross_13
:filt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_13+1],good_filters[filt5_LSB+cross_13
:filt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_13+1],good_filters[filt4_LSB+cross_13
:filt4_LSB]}; // {gf5:gf4}
```

```
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_13+1],good_filters[filt6_LSB+cross_13
:filt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_13+1],good_filters[filt5_LSB+cross_13
:filt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_13+1],good_filters[filt7_LSB+cross_13
:filt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_13+1],good_filters[filt6_LSB+cross_13
:filt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_13+1],good_filters[filt8_LSB+cross_13
:filt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_13+1],good_filters[filt7_LSB+cross_13
:filt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_13+1],good_filters[filt9_LSB+cross_13
:filt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_13+1],good_filters[filt8_LSB+cross_13
:filt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_13+1],good_filters[filt0_LSB+cross_13
:filt0_LSB]}; // {gf9:gf0}
                        end
                        4'd14:
                        begin // If crossover point is 14th coefficient [bit
112]
                                out_filters[filt0_MSB:filt0_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_14+1],good_filters[filt9_LSB+cross_14
:filt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]   =
{good_filters[filt0_MSB:filt0_LSB+cross_14+1],good_filters[filt1_LSB+cross_14
:filt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_14+1],good_filters[filt0_LSB+cross_14
:filt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]   =
{good_filters[filt1_MSB:filt1_LSB+cross_14+1],good_filters[filt2_LSB+cross_14
:filt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_14+1],good_filters[filt1_LSB+cross_14
:filt1_LSB]}; // {gf2:gf1}
                                out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_14+1],good_filters[filt3_LSB+cross_14
:filt3_LSB]}; // {gf2:gf3}
                                out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_14+1],good_filters[filt2_LSB+cross_14
:filt2_LSB]}; // {gf3:gf2}
                                out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_14+1],good_filters[filt4_LSB+cross_14
:filt4_LSB]}; // {gf3:gf4}
```

```
                                out_filters[filt8_MSB:filt8_LSB]    =
{good_filters[filt4_MSB:filt4_LSB+cross_14+1],good_filters[filt3_LSB+cross_14
:filt3_LSB]}; // {gf4:gf3}
                                out_filters[filt9_MSB:filt9_LSB]    =
{good_filters[filt4_MSB:filt4_LSB+cross_14+1],good_filters[filt5_LSB+cross_14
:filt5_LSB]}; // {gf4:gf5}
                                out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_14+1],good_filters[filt4_LSB+cross_14
:filt4_LSB]}; // {gf5:gf4}
                                out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_14+1],good_filters[filt6_LSB+cross_14
:filt6_LSB]}; // {gf5:gf6}
                                out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_14+1],good_filters[filt5_LSB+cross_14
:filt5_LSB]}; // {gf6:gf5}
                                out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_14+1],good_filters[filt7_LSB+cross_14
:filt7_LSB]}; // {gf6:gf7}
                                out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_14+1],good_filters[filt6_LSB+cross_14
:filt6_LSB]}; // {gf7:gf6}
                                out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_14+1],good_filters[filt8_LSB+cross_14
:filt8_LSB]}; // {gf7:gf8}
                                out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_14+1],good_filters[filt7_LSB+cross_14
:filt7_LSB]}; // {gf8:gf7}
                                out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_14+1],good_filters[filt9_LSB+cross_14
:filt9_LSB]}; // {gf8:gf9}
                                out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_14+1],good_filters[filt8_LSB+cross_14
:filt8_LSB]}; // {gf9:gf8}
                                out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_14+1],good_filters[filt0_LSB+cross_14
:filt0_LSB]}; // {gf9:gf0}
                        end
                        4'd15:
                        begin // If crossover point is 15th coefficient [bit
120]
                                out_filters[filt0_MSB:filt0_LSB]    =
{good_filters[filt0_MSB:filt0_LSB+cross_15+1],good_filters[filt9_LSB+cross_15
:filt9_LSB]}; // {gf0:gf9}
                                out_filters[filt1_MSB:filt1_LSB]    =
{good_filters[filt0_MSB:filt0_LSB+cross_15+1],good_filters[filt1_LSB+cross_15
:filt1_LSB]}; // {gf0:gf1}
                                out_filters[filt2_MSB:filt2_LSB]    =
{good_filters[filt1_MSB:filt1_LSB+cross_15+1],good_filters[filt0_LSB+cross_15
:filt0_LSB]}; // {gf1:gf0}
                                out_filters[filt3_MSB:filt3_LSB]    =
{good_filters[filt1_MSB:filt1_LSB+cross_15+1],good_filters[filt2_LSB+cross_15
:filt2_LSB]}; // {gf1:gf2}
                                out_filters[filt4_MSB:filt4_LSB]    =
{good_filters[filt2_MSB:filt2_LSB+cross_15+1],good_filters[filt1_LSB+cross_15
:filt1_LSB]}; // {gf2:gf1}
```

```
                              out_filters[filt5_MSB:filt5_LSB]   =
{good_filters[filt2_MSB:filt2_LSB+cross_15+1],good_filters[filt3_LSB+cross_15
:filt3_LSB]}; // {gf2:gf3}
                              out_filters[filt6_MSB:filt6_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_15+1],good_filters[filt2_LSB+cross_15
:filt2_LSB]}; // {gf3:gf2}
                              out_filters[filt7_MSB:filt7_LSB]   =
{good_filters[filt3_MSB:filt3_LSB+cross_15+1],good_filters[filt4_LSB+cross_15
:filt4_LSB]}; // {gf3:gf4}
                              out_filters[filt8_MSB:filt8_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_15+1],good_filters[filt3_LSB+cross_15
:filt3_LSB]}; // {gf4:gf3}
                              out_filters[filt9_MSB:filt9_LSB]   =
{good_filters[filt4_MSB:filt4_LSB+cross_15+1],good_filters[filt5_LSB+cross_15
:filt5_LSB]}; // {gf4:gf5}
                              out_filters[filt10_MSB:filt10_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_15+1],good_filters[filt4_LSB+cross_15
:filt4_LSB]}; // {gf5:gf4}
                              out_filters[filt11_MSB:filt11_LSB] =
{good_filters[filt5_MSB:filt5_LSB+cross_15+1],good_filters[filt6_LSB+cross_15
:filt6_LSB]}; // {gf5:gf6}
                              out_filters[filt12_MSB:filt12_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_15+1],good_filters[filt5_LSB+cross_15
:filt5_LSB]}; // {gf6:gf5}
                              out_filters[filt13_MSB:filt13_LSB] =
{good_filters[filt6_MSB:filt6_LSB+cross_15+1],good_filters[filt7_LSB+cross_15
:filt7_LSB]}; // {gf6:gf7}
                              out_filters[filt14_MSB:filt14_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_15+1],good_filters[filt6_LSB+cross_15
:filt6_LSB]}; // {gf7:gf6}
                              out_filters[filt15_MSB:filt15_LSB] =
{good_filters[filt7_MSB:filt7_LSB+cross_15+1],good_filters[filt8_LSB+cross_15
:filt8_LSB]}; // {gf7:gf8}
                              out_filters[filt16_MSB:filt16_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_15+1],good_filters[filt7_LSB+cross_15
:filt7_LSB]}; // {gf8:gf7}
                              out_filters[filt17_MSB:filt17_LSB] =
{good_filters[filt8_MSB:filt8_LSB+cross_15+1],good_filters[filt9_LSB+cross_15
:filt9_LSB]}; // {gf8:gf9}
                              out_filters[filt18_MSB:filt18_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_15+1],good_filters[filt8_LSB+cross_15
:filt8_LSB]}; // {gf9:gf8}
                              out_filters[filt19_MSB:filt19_LSB] =
{good_filters[filt9_MSB:filt9_LSB+cross_15+1],good_filters[filt0_LSB+cross_15
:filt0_LSB]}; // {gf9:gf0}
                        end
                  endcase

                  CALC_IS_DONE = 1;
            end
            // In DONE stage, wait for output transaction to occur
            else if ( current_state == STATE_CALC ) begin
                  req_rdy                 = 0;
                  resp_val          = 1;
            end
      end
```

```
endmodule

      `endif /* MENG_PROJECT_CROSS */
```

Register Model

```
//=============================================================================
// Verilog_Register
//=============================================================================
// Register with a parametrized number of bits. Triggers on positive edge
// of the clock.


`ifndef MENG_PROJECT_REG
`define MENG_PROJECT_REG

module Verilog_Reg
#(
      // Parameter for number of bits
      parameter nbits = 16
)(
      input                         clk,
      // Data signals
      input             [nbits-1:0] in,
      output            [nbits-1:0] out
);
      // Sequential logic

      reg         [nbits-1:0] temp_wire;
      always @( posedge clk )
      begin
            temp_wire <= in;
      end
      assign out = temp_wire;
endmodule


      `endif /* MENG_PROJECT_REG */
```

Resettable Register Model

```
//=============================================================================
// Verilog_Register_Reset
//=============================================================================
// Register with a parametrized number of bits. Triggers on positive edge
// of the clock.
//
// This version has a reset function that allows for setting register to
// known value.


`ifndef MENG_PROJECT_REG_RST
`define MENG_PROJECT_REG_RST

module Verilog_RegRst
#(
```

```
      // Parameter for number of bits and reset value
      parameter nbits = 16,
      parameter reset_val = 0
)(
      input                         clk,
      input                         reset,
      // Data signals
      input             [nbits-1:0] in,
      output            [nbits-1:0] out
);

      // Sequential logic
      reg         [nbits-1:0] temp_wire;
      always @( posedge clk )
      begin
            if ( reset ) begin
                  temp_wire <= reset_val;
            end
            else begin
                  temp_wire <= in;
            end
      end
      assign out = temp_wire;
endmodule


      `endif /* MENG_PROJECT_REG_RST */
```

Incrementer Model

```
//============================================================================
// Verilog_Incr
//============================================================================
// Combinational adder that increments by a parametrized amount.


`ifndef MENG_PROJECT_INCR
`define MENG_PROJECT_INCR

module Verilog_Incr
#(
      // Parameter for number of bits
      parameter nbits = 16,
      parameter incr_step = 1
)(
      // Data signals
      input   [nbits-1:0] in,
      output  [nbits-1:0] out
);

      // Combinational logic

      reg         [nbits-1:0] temp_wire;
      always @(*)
      begin
            temp_wire = in + incr_step;
```

```
	end

	assign out = temp_wire;

endmodule


	`endif /* MENG_PROJECT_INCR */
```

Multiplexer Model

```
//============================================================================
// Verilog_Mux_2
//============================================================================
// Multiplexer with a parametrized number of bits and two inputs


`ifndef MENG_PROJECT_MUX2
`define MENG_PROJECT_MUX2

module Verilog_Mux_2
#(
	// Parameter for number of bits
	parameter nbits = 16

)(
	// Data signals
	input		[nbits-1:0] in_a,
	input		[nbits-1:0] in_b,
	input			    sel,
	output		[nbits-1:0] out
);

	// Combinational logic
	reg		[nbits-1:0] temp_wire;

	always @(*)
	begin
		if ( ~sel )
			temp_wire = in_a;
		else
			temp_wire = in_b;
	end

	assign out = temp_wire;

endmodule


	`endif /* MENG_PROJECT_MUX2 */
```