Optimized FPGA Implementation of Multi-Rate FIR Filters through Thread Decomposition

Jason Xin Zheng
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
M/S: 303-300
Pasadena, CA 91109-8099
Xin.Zheng@jpl.nasa.gov

Kayla Nguyen
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
M/S: 198-235
Pasadena, CA 91109-8099
Kayla.Nguyen@jpl.nasa.gov

Yutao He
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
M/S: 156-142
Pasadena, CA 91109-8099
Yutao.He@jpl.nasa.gov

Abstract— Multirate (decimation/interpolation) filters are among the essential signal processing components in space-borne instruments where Finite Impulse Response (FIR) filters are often used to minimize nonlinear group delay and finite-precision effects. Cascaded (multi-stage) designs of Multi-Rate FIR (MRFIR) filters are further used for large rate change ratio, in order to lower the required throughput while simultaneously achieving comparable or better performance than single-stage designs. Traditional representation and implementation of MRFIR employ polyphase decomposition of the original filter structure, whose main purpose is to compute only the needed output at the lowest possible sampling rate. In this paper, an alternative representation and implementation technique, called TD-MRFIR (Thread Decomposition MRFIR), is presented. The basic idea is to decompose MRFIR into output computational threads, in contrast to a structural decomposition of the original filter as done in the polyphase decomposition. Each thread represents an instance of the finite convolution required to produce a single output of the MRFIR. The filter is thus viewed as a finite collection of concurrent threads. The technical details of TD-MRFIR will be explained, first showing its applicability to the implementation of downsampling, upsampling, and resampling FIR filters, and then describing a general strategy to optimally allocate the number of filter taps. A particular FPGA design of multi-stage TD-MRFIR for the L-band radar of NASA's SMAP (Soil Moisture Active Passive) instrument is demonstrated; and its implementation results in several targeted FPGA devices are summarized in terms of the functional (bit width, fixed-point error) and performance (time closure, resource usage, and power estimation) parameters.

TABLE OF CONTENTS

1. INTRODUCTION

Multi-rate Finite Impulse Response (MRFIR) filters are ubiquitous in today's Digital Signal Processing (DSP) applications, which can be found on many space-borne instruments. Traditionally, the amount of real-time on-board processing had been limited by the low logic densities of the space-qualified logic devices. In past decade, with the availability of high-density and radiation-tolerant Field Programmable Gate Arrays (FPGAs), reliable and high-order filter designs had become an increasingly common part of real-time on-board data processor.

Most of past research ([1], [2], [3], [4]) have been focusing on the theoretical design aspects of MRFIR filters, such as the optimal filter length and impulse response. Some work in ([5], [6]) have touched on the issues regarding the VLSI (Very Large Scale Integrated Circuit) implementation of particular MRFIR designs. Nevertheless, there has been a little research on the general strategies of implementing arbitrary MRFIR designs at the Register Transfer Level (RTL). In this paper, we introduce a general implementation strategy of arbitrary MRFIR filters that can help achieve the minimum number of multipliers. The novelty of this approach is that it transforms a MRFIR filter design into static scheduling problems through an alternative view of the filter, called Thread Decomposition (TD).

The main assumption of TD-MRFIR is that highly reconfigurable MRFIR designs require use of arbitrary multipliers, and that multiplier implementations are either highly resource-demanding or bound by the number of embedded multipliers on the target VLSI device. Thus the main objective of the TD-MRFIR is to reduce the total multipliers count by facilitating time-multiplexing where possible.

[1] 978-1-4244-3888-4/10/$25.00 ©2010 IEEE.
[2] IEEEAC Paper #1113, Version 0, Updated 20/09/2009.

The rest of the paper is organized as follows: Before introducing the concept of TD, the basic concepts of MRFIR and the traditional way of viewing it as Polyphase Decomposition are presented in Section 2, as well as why Polyphase Decomposition cannot always provide insight to the most efficient implementation. Section 3 explains how the alternative TD view can fill in the efficiency gap for the three general types of MRFIR filters, and how an optimal implementation can be derived by solving the static scheduling problem. Section 4 describes in detail some FPGA-related implementation issues. Section 5 presents both theoretical and empirical comparison between TD and polyphase FIR implementations. A case study of applying the TD-MRFIR methodology to the Soil Moisture Active Passive (SMAP) L-band radar instrument digital filter design is given in Section 6. Conclusions are drawn in Section 7 instrument pre-Phase-A study.

2. MULTI-RATE FIR BASICS

A Finite Impulse Response (FIR) filter with a length of N taps, input samples $x(n)$, output samples $y(n)$, and coefficients $h(n)$ can be expressed by the following finite convolution:

$$y(n) = \sum_{k=0}^{N-1} h(k) * x(n-k). \tag{1}$$

An upsampling of factor L is achieved by inserting $L-1$ zeros between each input samples:

$$y(n) = \begin{cases} x(\frac{n}{L}) & \text{if } L \text{ divides } n; \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

A downsampling of factor M is achieved by taking every M of the input samples:

$$y(n) = x(Mn). \tag{3}$$

A Multi-Rate FIR (MRFIR) is the combination of an FIR with an upsampling stage and/or a downsampling stage. Figure 1 shows a block diagram of generalized MRFIR.

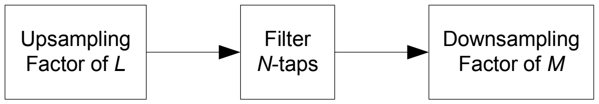


Figure 1. Generalized Multi-Rate FIR Block Diagram

An MRFIR with only a downsampling stage ($M \neq L = 1$) is a Decimation Filter. Similarly, an MRFIR with only an upsampling stage ($L \neq M = 1$) is an Interpolation Filter. Finally, a Resampling Filter has both the upsampling and downsampling stages to achieve a fractional rate change.

Polyphase Decomposition

A naive implementation of a decimation filter consisting of a full FIR followed by a downsampling stage is very inefficient, as most of the computations performed by the FIR stage are thrown away by the downsampling stage. In fact, only $\frac{1}{M}$ of the total computations are useful.

Polyphase Decomposition provides an alternative view of decimation filters, where the downsampling occurs before the FIR stage, and the outputs are viewed as the sum of M sub-filters with length of $\frac{N}{M}$ taps. This approach leads to a more efficient filter design, producing the same results with no wasted computations. Figure 2 is a textbook ([7]) example of the Polyphase Decomposition with the delayed inputs.

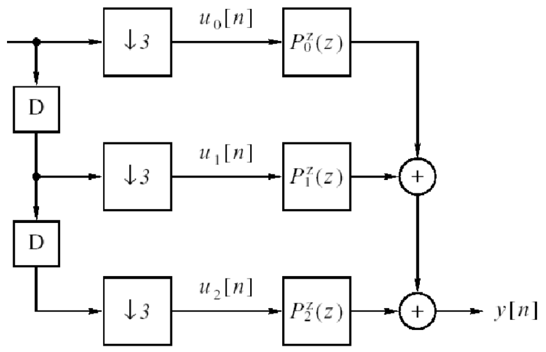


Figure 2. Delayed Input Polyphase Decomposition ([7])

A straightforward implementation of Figure 2 yields three independent filters, and the total number of multipliers is still N if each multiplier runs at one third the input rate. In some cases this is necessary, since the multipliers might not be able to run at the input rate. However, in many cases, e.g. in the later stages of a multi-stage filter bank, the input rate is low enough such that multipliers can be time-multiplexed (shared).

There are two basic ways that multipliers running at faster rates can be shared in the Polyphase Decomposition model. One way is to share multipliers within each sub-filter. The other way is to time-multiplex the individual sub-filters. Using only the first approach, the minimum number of multipliers is M, since each sub-filter must have at least one multiplier. Using the second approach alone, the minimum number of multipliers is the size of each sub-filter $\frac{N}{M}$. One can even combine the two ways to further reduce the total number of multipliers to one (this is requires running the multipliers running at N times faster than the input rate).

However, in the most common cases, the ratio between the fastest sustainable multiplier rate and the input rate is smaller than N but larger than 1. Using the same example from Figure 2, suppose that the input rate is 30MHz, $N = 21$, $M = 3$, and the multipliers can run at 120MHz, it is not obvious how to derive an implementation using less than 3 multipliers (the minimum is 2 multipliers as explained in Section 3). In general, the Polyphase Decomposition model of an MRFIR provides no insight on how to efficiently share multipliers when the optimal number of multipliers is between 1 and MIN(M, $\frac{N}{M}$).

3. Thread Decomposition

Instead of continuing the sub-filter analogy, Thread Decomposition (TD) approaches the MRFIR implementation problem by examining the necessary computations for each output, and transforming them to a static scheduling problem. Each of the MRFIR valid output is considered as the result of a computational thread, whose only goal is to compute a finite convolution using inputs $x(n-N-1)$ to $x(n)$. The constraints of the scheduling problem are the number of available multipliers and the fact that all threads must complete. Since each thread has a fixed run time, and new threads are always spawned in a periodic fashion, solving this static scheduling problem is rather straightforward. To see why this approach leads to the most efficient implementation, we first state the underlying assumptions, followed by a derivation of the minimum number of multipliers.

Assumptions

- All coefficients are arbitrary.
- All inputs are arbitrary.
- A single clock domain at the highest multiplier rate.
- No input/output buffering, i.e. real-time streaming filter.
- The objective is to minimize the total multiplier count.

Note that exploiting symmetry and zeros in the impulse response are not difficult using TD, but the basic assumption of arbitrary coefficients makes the demonstration convenient.

Minimum Number of Multipliers

To derive the minimum number of arbitrary multipliers, first note that the number of *unique* multiplications per output sample is N. The uniqueness of the multiplications are guaranteed by the fact that all inputs and coefficients are arbitrary.

Define the input sample rate as f_{in}, output sample rate as $f_{out} = \frac{f_{in}}{M}$, and the highest multiplier clock rate as f_{mult}. The number of multipliers N_{mult} must satisfy the output throughput requirement $f_{out}N$:

$$\begin{aligned} N_{mult} * f_{mult} &\geq f_{out}N & (4) \\ N_{mult} &\geq \frac{f_{out}N}{f_{mult}} & (5) \\ \min(N_{mult}) &= \lceil \frac{f_{out}}{f_{mult}} N \rceil & (6) \\ &= \lceil \frac{f_{in}N}{f_{mult}M} \rceil & (7) \end{aligned}$$

Notice that there is no factor L in this derivation. This is because the nature of upsampling (inserting zeros) creates no additional computation requirements on the MRFIR. This point will be further demonstrated by the Thread Decomposition Diagrams of the interpolation filters.

Thread Decomposition Diagrams

To see how an implementation with $\min(N_{mult})$ can be achieved, Thread Decomposition Diagrams can be used to illustrate the static scheduling problem. A TD diagram shows a snapshot of the concurrently running threads, each representing a finite impulse convolution that produces an output value. Each thread begins when the first input sample is available, and finishes when the output is computed. For example, a simple N-tap filter with no rate change (M=L=1), N concurrent computation threads are active at any time.

The simple filter example ($N = 5$) is illustrated in Table 1. Each input *x* is multiplied with the corresponding coefficient *h* in the same column, and accumulated with the previous multiplication in the same thread. For convenience, the time scale is chosen to be $\frac{1}{f_{in}}$. Notice that when $f_{mult} = f_{in}$, the number of concurrent threads is exactly the minimum number of multipliers.

Table 1. TD Diagram for $N = 5$, $M = L = 1$

	x0	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*	*x7*	*x8*
Thread 0	*h4*	*h3*	*h2*	*h1*	*h0*				
Thread 1		*h4*	*h3*	*h2*	*h1*	*h0*			
Thread 2			*h4*	*h3*	*h2*	*h1*	*h0*		
Thread 3				*h4*	*h3*	*h2*	*h1*	*h0*	
Thread 4					*h4*	*h3*	*h2*	*h1*	*h0*
Outputs					*y0*	*y1*	*y2*	*y3*	*y4*

Now suppose the multipliers can run at twice the input rate, then the minimum number of multipliers is 3. A straightforward solution of the multiplier scheduling can be obtained by dividing the active portions of each column into three parts, with two multipliers active at every clock cycle, and the third multiplier active every other clock cycle. The control logic of such scheduling algorithm can be implemented using a simple counter.

Decimation Filters—TD diagrams can be constructed for decimation filters in a similar fashion. Table 2 shows the TD diagram for $M = 2$ and $N = 6$. Notice that the number of concurrent threads is $\frac{N}{M} = 3$. Again, the multiplier requirement is also 3 when $f_{in} = f_{mult}$.

Now suppose the multipliers can run at twice the input rate, then the multiplier requirement is reduced to 2. To solve the multiplier scheduling problem, one only need to divide each column by 2 using a simple counter logic.

Interpolating Filters—An interpolating filter inserts $L-1$ zeros between every input samples. The TD diagram can be derived in a similar fashion. The only difference is that the

Table 2. TD Diagram for $N = 6$, $M = 2$, $L = 1$

	x0	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*	*x7*	*x8*	*x9*
Thread 0	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*				
Thread 1			*h5*	*h4*	*h3*	*h2*	*h1*	*h0*		
Thread 2					*h5*	*h4*	*h3*	*h2*	*h1*	*h0*
Outputs						*y0*		*y1*		*y2*

columns whose inputs are zero need not to be computed. Table 3 shows the diagram for $L = 2$ and $N = 4$. Although the zero-columns can be effectively deleted, they are shown here for illustration purpose.

Notice that unlike decimation filters, the interpolation filters do not change the multiplier requirement. This point is demonstrated by the fact that only a single column per input time requires real computation, hence the number of multiplications performed per input time is still equal to the number of concurrent threads in the diagram.

Table 3. TD Diagram for $N = 4$, $M = 1$, $L = 2$

	time0		time1		time2		time3	
	x0	0	*x1*	0	*x2*	0	*x3*	0
Thread 0	*h3*	*h2*	*h1*	*h0*				
Thread 1		*h3*	*h2*	*h1*	*h0*			
Thread 2			*h3*	*h2*	*h1*	*h0*		
Thread 3				*h3*	*h2*	*h1*	*h0*	
Outputs				*y0*	*y1*	*y2*	*y3*	

To solve the multiplier scheduling problem when $f_{in} < f_{mult}$, one simply needs to divide each non-zero column by $\frac{f_{mult}}{f_{in}}$.

Re-Sampling FIR—The re-sampling FIR is a combination of the decimation filter and the interpolation filter to achieve a fractional rate change (3/2, 2/3, etc). The TD diagram of such a filter can derived from Table 3 and Table 2 with some minor tweaking of the time scale.

When f_{mult} Is Not Divisible by f_{in}

So far, all the discussions have an hidden assumption, that f_{in} divides f_{mult}. When such is not the case, a sensible approach is to lower f_{mult} to f'_{mult} such that it is divisible by f_{in}. This approach yields minimum number of multipliers for f'_{mult} instead of f_{mult}, but does not require additional rate-changing logic.

Optimality of Thread Decomposition

TD's optimality refers to the fact that it yields implementations that facilitates the highest level of multiplier sharing, or lowest number of multipliers. This can be shown by first observing that the number of concurrent threads in a TD diagram is exactly $\frac{N}{M}$. Secondly, the solution to the multiplier scheduling problem is given by dividing the active portion of each column by the factor $\frac{f_{mult}}{f_{in}}$, where the height of the active portion is the number of concurrent threads. Finally, notice that the minimum required number of multipliers, given in Equation 7, is essentially $\frac{\frac{N}{M}}{\frac{f_{mult}}{f_{in}}}$.

Table 4. SMAP Pre-Phase-A Filter Stages

Stage	f_{in}	MCA	N/M	N_{mult}
1	240MHz	0.25	12/4	12
2	60MHz	1	15/5	3
3	12MHz	5	25/5	1
4	2.4	25	50/2	1

Multiplier Clock Advantage

The term $\frac{f_{mult}}{f_{in}}$ can also be understood as the Multiplier Clock Advantage (MCA) of the filter implementation. For a filter design with fixed $\frac{N}{M}$ ratio, the faster the multiplier runs relative to the input rate, the fewer multipliers are required. On the other hand, the concept of MCA still applies even in the cases of $f_{mult} < f_{in}$, where $\frac{f_{mult}}{f_{in}} < 1$ and the total number of multipliers is more than $\frac{N}{M}$. This particular case is demonstrated in the first stage filter of the SMAP radar instrument pre-phase-A study (see Section 6), where the input rate is four times the multiplier rate.

The MCA also has an implication for the multi-stage decimation filter designs. In a multi-stage decimation filter, the data rate is successively reduced through each of the decimation filter. Assuming that the entire design runs in a single clock domain, the MCA becomes greater with every stage in the data flow. This suggests that an implementation-friendly multi-stage design should allocate more filter taps to the later stages, where the high MCA helps reduce the total multiplier count. An example from the SMAP Pre-Phase-A study (Figure 5) is given in Table 4.

4. FPGA IMPLEMENTATION

Although the TD model uses the notion of threads and scheduling, the RTL implementation of an MRFIR consists only multipliers, accumulators, multiplexers, coefficient banks, and counter logic. This section provides discussion on some of the FPGA-specific issues when implementing an MRFIR.

Coefficient Banks

Coefficient banks store the constant coefficients for the arbitrary multipliers. There are multiple ways to create coefficient banks, but some are more efficient on FPGA platforms as they take advantage of the underlying FPGA fabrics. In this paper, we describe two approaches: SRAM look-up tables (LUT), and rotating coefficient banks (RCB).

SRAM Look-up Tables—SRAM look-up tables utilize internal SRAM in the FPGA fabric to store the coefficients, and can be directly inferred from HDL source codes. For SRAM-based FPGA platforms, the approach is especially useful, since the LUT can be synthesized to utilize the Configuration RAM (CRAM). For larger LUTs, embedded block RAM (BRAM) can be utilized.

In addition to lowering register (flip-flop) usage, SRAM-based LUTs also have an advantage when Single-Event Upset (SEU) mitigation is considered. Scrubbing of the FPGA CRAM is a frequently used technique to remove SEU effects on the FPGA, and the contents of the CRAM-based LUTs are automatically scrubbed with the entire FPGA. The BRAM-based LUTs can also be scrubbed with the entire FPGA in theory. However, this requires the scrubbing logic to selectively scrub only the BRAM blocks used for LUTs, since BRAM blocks are often used as data buffers (and should not be scrubbed).

SRAM-based LUTs have a key disadvantage: each LUT serves up to two multipliers only, since most SRAM blocks have only two read ports. Nevertheless, this issue can be mitigated by breaking up coefficient banks and storing only the necessary coefficients for the multipliers served by the LUT. For example, for a 16-tap filter with 4 shared multipliers, each multiplier only need to access 4 of the 16 coefficients, hence each LUT only needs to store 8 instead of 16 coefficients.

The following is an example of the SRAM LUT inference in Verilog:

```
parameter e0 = 'habcd;
parameter e1 = 'hcdef;
...
parameter eE = 'hdead;

always @ (posedge clk or negedge arst_n)
    if (arst_n!==1'b1)
       coe[15:0] <= 16'd0;
    else case (coe_cnt[3:0])
          4'b0000: coe[15:0] <= e0;
          4'b0001: coe[15:0] <= e1;
          4'b0010: coe[15:0] <= e2;
            ...
          4'b1110: coe[15:0] <= eE;
          default: coe[15:0] <= eE;
         endcase
```

Rotating Coefficient Banks— A Rotating Coefficient Bank (RCB) utilizes register resources to store the coefficients. A key difference between RCB and other coefficient storage is that the coefficients are not static in RCB; they are rotated (shifted to the next coefficient register) upon every valid input to the filter. By rotating the coefficients by a fixed amount, the multiplier no longer needs a multiplexer to choose which coefficient to use.

Another advantage of the RCB is that some FPGA platforms offer register cells with built-in Triple Modular Redundancy (TMR). Combined with the rotating nature of the RCB, the coefficients are always protected against SEU, and single-bit errors are automatically corrected upon the next valid input.

Finally, the RCB has no fan-out limit, as each coefficient entry may be accessed by more than one multiplier during each clock period. The only down-side of the RCB is the high register usage. The following is an example written in Verilog:

```
always @ (posedge clk or negedge arst_n)
     if (arst_n!==1'b1)
       begin
          eff0[15:0] <= e0;
          eff1[15:0] <= e1;
          eff2[15:0] <= e2;
           ...
          effB[15:0] <= eB;
       end
     else if (inValid)
       begin
          eff0[15:0] <= eff1[15:0];
          eff1[15:0] <= eff2[15:0];
          eff2[15:0] <= eff3[15:0];
           ...
          effB[15:0] <= eff0[15:0];
    end
```

One-Hot Multiplexers

While the use of SRAM LUT or RCB eliminates the need for the multiplexers that feed coefficient values to the multiplier, multiplexers are still required to feed the accumulators and the final output stage. For SRAM-based FPGAs, the nature of LUT-based logic synthesis could often result in sub-optimal implementation of large multiplexers. One way to mitigate the large multiplexer issue is to use one-hot enable signals to implement the multiplexer. The one-hot enable signals are logically ANDed with the inputs, then ORed together to produce the result:

```
assign SigOut[31:0] =
      (accum1[31:0] & {32{valid1}}) |
      (accum2[31:0] & {32{valid2}}) |
      (accum3[31:0] & {32{valid3}}) |
      (accum4[31:0] & {32{valid4}}) ;
```

When implemented with 4-input LUTs alone, the above 32-bit 4:1 multiplexer requires only 64 LUT cells. On the other hand, the same multiplexer using binary selectors would require 96 LUT cells.

Design and Verification Flow

The TD-MRFIR technique is an RTL-oriented approach to allow the most control over the final VLSI implementation.

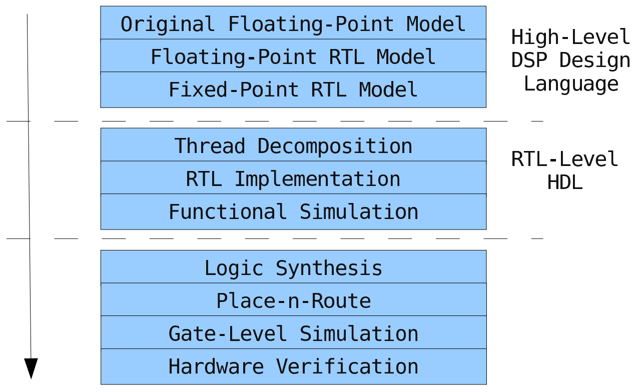


Figure 3. DSP-RTL Design Flow

Hence the design and verification flow follows that of a typical RTL-based flow: RTL design, functional simulation, logic synthesis, place-n-route, etc. To interface with a typical DSP design flow from a high-level language such as Matlab, bit-true fixed-point models proves very useful in a system-level simulation. In addition, very often the floating-point golden models used by algorithm designers have subtle phase differences when compared to the RTL designs. To reconcile the phase differences between the golden model and RTL design, sometimes it is necessary to construct a floating-point model of the RTL design.

Figure 3 shows the complete design and verification flow. The floating-point RTL model is compared against the floating-point golden model to determine whether all phase differences are resolved. The fixed-point RTL model is compared to the floating-point models to assess the level of quantization errors. Finally, the result of the functional simulation must match that of the fixed-point model bit-by-bit.

5. Comparison between TD and Non-TD Approaches

In this section, we attempt to show the advantages of the TD approach over polyphase decomposition by comparing benchmarks such as resoure utilization. An analytical comparison based on theoretical limits of multiplier sharing is presented first, followed by experimental results.

Limits of Multiplier Sharing

As mentioned in Section 2, multi-rate filters inspired by polyphase decomposition share multipliers either within each sub-filter, or by time-multiplexing the sub-filter. Neither approach is absolutely superior than the other. For the sake of simplicity, only the first approach is compared against TD-FIR.

Figure 4 shows the relationship between the theoretic limits of the multiplier count and increasing MCA ($\frac{f_{mult}}{f_{in}}$) for three approaches: polyphase FIR, TD-FIR, and plain FIR without multiplier sharing. The y-axis represents the number of multipliers. The x-axis represents MCA. For obvious reasons the

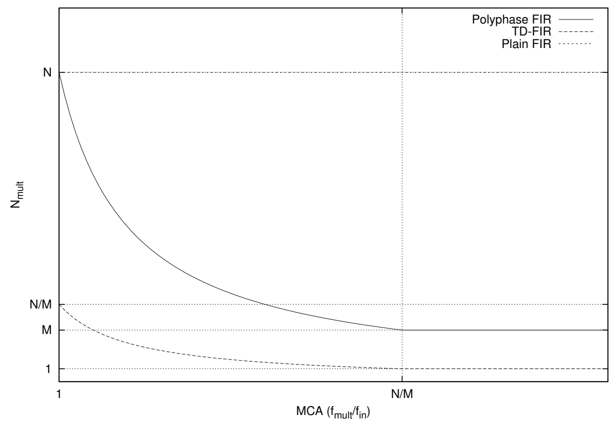


Figure 4. Multiplier Count vs. Multiplier Clock Advantage (MCA)

plain FIR's multiplier count is not affected by the MCA. The polyphase FIR cannot have less than M number of multipliers, since each sub-fitler must have at least 1 multiplier. On the other hand, TD-FIR always has an absolute limit of 1 multiplier. Both TD and polyphase approach their absolute limit when MCA= $\frac{N}{M}$ as a function of x^{-1}. In theory, Figure 4 shows that TD-FIR is always better at sharing multipliers than polyphase FIR.

Testing Methodology

To measure the comparative advantage of TD-FIR over polyphase decomposition, a performance test is devised. The test is conducted by implementing three different FIR design samples in both TD-FIR and polyphase decomposition. The performance measurements include resource usage numbers such multiplier and flip-flop count, as well as dynamic power.

The three FIR design samples are taken from the pre-Phase-A study of Soil Moisture Active Passive (SMAP) L-Band Radar Digital Filter (see Section 6). The first filter is N= 15, M= 5. The second filter is N= 25, M= 5. The third filter is N= 50,M= 2.

The TD-FIR implementations are performed by hand-coding based on TD diagrams of each filter. The polyphase implementations are performed by an automated HDL generator software called Xilinx AccelDSP. The AccelDSP converts high-level descriptions of FIR designs (in Matlab language) to Verilog/VHDL descriptions, which can then by synthesized like the hand-coded designs. Both the TD-FIR and polyphase implementations are synthesized and place-n-routed by the same Xilinx tool-flow. The tool-flow target is Xilinx VirtexII-3000.

The term comparative advantage is used to emphasize the fact that results are normalized by the maximum throughput rate of the filter implementation. The reason for the normalization is because different implementations of the same FIR design can often differ in the $\frac{f_{mult}}{f_{in}}$ ratio, making direct comparisons of resource usage and dynamic power difficult. By dividing each number by the throughput (input) rate of each implementation, architectural differences are removed. For example, an FIR implementation that uses 10 multipliers and can stream at a maximum input rate of 10MHz has a normalized multiplier usage of $10/10 = 1$ multiplier/MHz.

Results

The results of performance test are presented in Tables 5,6, and 7. Both the absolute and normalized numbers are presented. The term LUTs refers to the number of 4-input Look-Up Tables (LUTs, the basic logic elements of Xilinx Virtex FPGAs). The normalized resource and power consumption indicate an obvious comparative advantage of TD-FIR over polyphase FIR implementations in all aspects.

Table 5. TD vs. Polyphase (N=15,M=5)

Parameters	**TD-FIR**	**Polyphase**
Input Rate	167MHz	100.3MHz
Multipliers	3	5
Flip-Flops	259	529
LUTs	264	329
Dynamic Power	22mW	22mW
Normalized MHz^{-1}		
Multipliers	0.02	0.05
Flip-Flops	1.55	5.27
LUTs	1.58	3.28
Dynamic Power	0.13mW	0.22mW

Table 6. TD vs. Polyphase (N=25,M=5)

Parameters	**TD-FIR**	**Polyphase**
Input Rate	34MHz	32MHz
Multipliers	1	5
Flip-Flops	241	1063
LUTs	244	896
Dynamic Power	16mW	30mW
Normalized MHz^{-1}		
Multipliers	0.03	0.16
Flip-Flops	7.09	33.22
LUTs	7.18	28
Dynamic Power	0.47mW	0.94mW

6. APPLICATION

A Reusable IP Core in the ISAAC framework

The TD-MRFIR IP core described in the previous sections has been implemented as a reusable and parameterizable IP core in the ISAAC [8] framework. ISAAC (Instrument ShAred Artifact for Computing) is a technology currently under development at JPL. It is aimed to provide a highly capable, highly reusable, modular, and integrated FPGA-based common instrument control and computing platform that can be shared by multiple Earth Science and Planetary Exploration instruments. Please refer to [8] for more details on ISAAC.

Table 7. TD vs. Polyphase (N=50,M=2)

Parameters	**TD-FIR**	**Polyphase**
Input Rate	6.4MHz	3.8MHz
Multipliers	1	2
Flip-Flops	856	1035
LUTs	1086	661
Dynamic Power	37mW	26mW
Normalized MHz^{-1}		
Multipliers	0.16	0.53
Flip-Flops	133.75	272.37
LUTs	169.69	173.95
Dynamic Power	5.78mW	6.84mW

Soil Moisture Active Passive (SMAP) L-Band Radar Digital Filter

The TD-MRFIR technique has been successfully applied and demonstrated in the pre-Phase-A design of SMAP L-Band radar on-board processor's 240MHz 4-stage decimation filter. This multi-stage filter accepts 4x60MHz (de-muxed from 240MHz) digital input data, and successively filters and decimates the data rate to 1.2MHz. The four decimation stages are illustrated in Figure 5 (taken from [9]). The internal multipliers of all four stages run at a single clock rate of 60MHz.

The implemented FPGA target contains 3 complex filters to resolve three 1MHz sub-bands within a 5MHz bandwidth. To implement all three complex filters, a Quadrature Demodulation stage is used for each sub-band, and the multi-stage filter is instantiated six times. For detailed algorithmic discussions, please refer to [10].

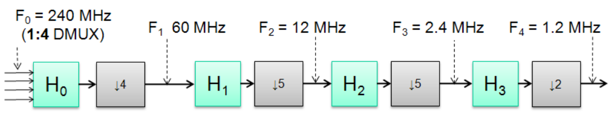


Figure 5. SMAP Digital Processor Filter Design ([9])

1st Stage: $N = 12$, $M = 4$, $L = 1$

The first stage of the filter is a 12-tap filter with the decimation factor of 4. This filter is the only stage where $f_{mult} < f_{in}$. Since the input data rate is four times of the multiplier clock, the input is de-multiplexed to 4 simultaneous streams before entering the filter. The output of this filter is 60MHz digital data. Table 8 shows TD diagram in the multiplier time scale (60MHz), where three concurrent threads are running. However, since the multipliers are running at $\frac{1}{4}$ of the input rate (60MHz), the minimum number of multipliers is greater than the ratio N/M (Equation 7 yields 12 multipliers).

For every clock period, there are 4 parallel inputs into the filter (time *clk0* has inputs *x0*, *x1*, *x2*, and *x3*). These inputs must each be multiplied by a coefficient concurrently.

To derive a solution to the multiplier scheduling problem, one must consider the active portions of 4 adjacent columns (e.g. *clk2*) at a time. The solution is that 12 arbitrary multipliers are needed to compute the 12 unique multiplications per clock period.

2nd Stage: $N = 15$, $M = 5$, $L = 1$

The second stage is a 15-tap filter with a decimation factor of 5 and input rate of 60MHz. With $f_{mult} = f_{in}$, Equation 7 yields that the minimum number of multipliers is 3. Scheduling of the multipliers is straightforward from the TD diagram Table 9. The output of this stage is 12MHz digital data.

3rd Stage: $N = 25$, $M = 5$, $L = 1$

The third stage is a 25-tap filter with a decimation factor of 5 and input rate of 12MHz. Due to a large MCA $(60/12 = 5)$, this stage only needs a single multiplier.

Table 10 shows the TD diagram of the 3rd stage filter with the input time scale (each column is 5 multiplier clock periods). Since the number of concurrent threads is 5, a single multiplier running every clock cycle can perform all the required multiplications.

4th Stage: $N = 50$, $M = 2$, $L = 1$

The fourth and last stage is a 50-tap with a decimation factor of 2 and input rate of 2.4MHz. Again the high MCA allows for a single multiplier to be shared by 25 concurrent threads. Due to the large size, the TD diagram will not be shown in this paper.

Functional Verification

The filter design is simulated using synthetic radar data inputs, and the result of the RTL functional simulation is compared to a Matlab floating-point simulation. The relative errors are presented in Figure 6 with all three sub-bands. Additional verifications based on meaningful radar performance parameters can be found in [10].

Resource Usage

The resource usage for the 240MHz design is listed in table 11. The XQR2V series is a flight-qualified FPGA part based on the Xilinx Virtex-2 FPGA architecture. The XC5V series is a commercial-grade FPGA based on the Xilinx Virtex-5 FPGA architecture. The resource usage includes all 6 instances of the multi-stage filters and 3 quadrature demodulators.

Timing

The timing goal of the pre-Phase A design is to study the feasibility of running a 60MHz fully contained in a Virtex2 series FPGA. The final design is timed by the Xilinx-provided tools to run at up to 85.4MHz on a XQR2V3000-4, and up to 120MHz on the commercial XC5VFX130T-1.

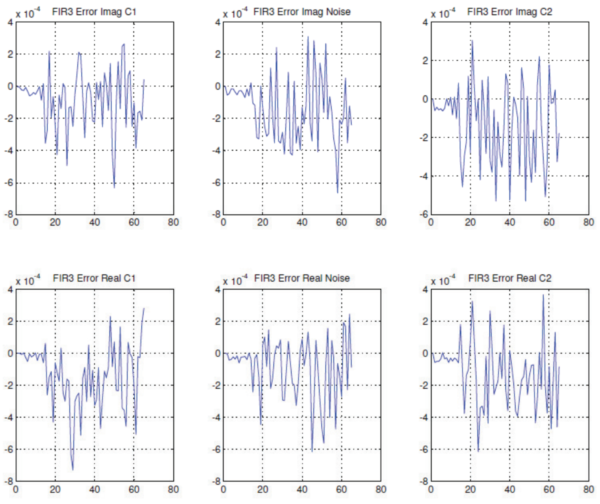


Figure 6. RTL Simulation vs. Floating-Point

Table 11. SMAP 240MHz Design Resource Usage

FPGA	XC5VFX130T-1		XQR2V3000-4	
	Used/Total	**%**	**Used/Total**	**%**
# Mult	68/120	57%	68/96	71%
# Slices	3,864/20,480	19%	8,278/14,336	58%
# Flip Flops	9,887/81,920	12%	9,680/28,672	33%
# 4-input LUTs	11,199/81,920	13%	14,064/28,672	49%

Power Consumption

Power consumption estimation was made using Xilinx XPower tool with industrial temperature range. Again this includes all six instances of the filters.

Table 12. SMAP 240MHz Design Power Consumption Estimation

FPGA	XC5VFX130T-1	XQR2V3000-4
Dynamic	0.10W	0.89W
Quiescent	1.78W	0.38W
Total	1.88W	1.26W

7. Conclusions

In this paper we have presented a systematic and general strategy to implement a wide variety of Multi-rate FIR designs. We have also showed that the strategy yield results with the minimum number of multipliers when the inputs and coefficients are arbitrary. A case study of application for implementation of digital filter design in meeting the requirements of SMAP pre-Phase-A design is also given.

Table 8. TD Diagram for 1st Stage SMAP Filter

	clk0				*clk1*				*clk2*				*clk3*				*clk4*			
	x0	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*	*x7*	*x8*	*x9*	*x10*	*x11*	*x12*	*x13*	*x14*	*x15*	*x16*	*x17*	*x18*	*x19*
Thrd 0	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*								
Thrd 1					*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*				
Thrd 2									*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*
Outputs												*y0*				*y1*				*y2*

Table 9. TD Diagram for 2nd Stage SMAP Filter

	x0	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*	*x7*	*x8*	*x9*	*x10*	*x11*	*x12*	*x13*	*x14*	*x15*	*x16*	*x17*	*x18*	*x19*	*x20*	*x21*	*x22*	*x23*	*x24*
Thrd 0	*h14*	*h13*	*h12*	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*										
Thrd 1						*h14*	*h13*	*h12*	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*					
Thrd 2											*h14*	*h13*	*h12*	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*
Outputs															*y0*					*y1*					*y2*

Table 10. TD Diagram for 3rd Stage SMAP Filter

	x0	*x1*	*x2*	*x3*	*x4*	*x5*	*x6*	*x7*	*x8*	*x9*	*x10*	*x11*	*x12*	*x13*	*x14*	*x15*	*x16*	*x17*	*x18*	*x19*	*x20*	*x21*	*x22*	*x23*	*x24*	*x25*	*x26*	*x27*	*x28*	*x29*
Thrd 0	*h24*	*h23*	*h22*	*h21*	*h20*	*h19*	*h18*	*h17*	*h16*	*h15*	*h14*	*h13*	*h12*	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*					
Thrd 1						*h24*	*h23*	*h22*	*h21*	*h20*	*h19*	*h18*	*h17*	*h16*	*h15*	*h14*	*h13*	*h12*	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*	*h4*	*h3*	*h2*	*h1*	*h0*
Thrd 2											*h24*	*h23*	*h22*	*h21*	*h20*	*h19*	*h18*	*h17*	*h16*	*h15*	*h14*	*h13*	*h12*	*h11*	*h10*	*h9*	*h8*	*h7*	*h6*	*h5*
Thrd 3																*h24*	*h23*	*h22*	*h21*	*h20*	*h19*	*h18*	*h17*	*h16*	*h15*	*h14*	*h13*	*h12*	*h11*	*h10*
Thrd 4																					*h24*	*h23*	*h22*	*h21*	*h20*	*h19*	*h18*	*h17*	*h16*	*h15*
Outputs																									*y0*					*y1*

ACKNOWLEDGMENTS

We would like to thank Charles Le and Ian Tan of Jet Propulsion Laboratory for the numerous technical discussions on the digital filter design.

We would also like to thank Sunant Katanyoutanant and Martin Le of Jet Propulsion Laboratory for their inputs on the one-hot-style multiplexers.

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

[1] S. Smith, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, vol. 6, pp. 4–19, July 1989.

[2] R. Crochiere and L. Rabiner, "Interpolation and decimation of digital signals - a tutorial review," *Proceedings of the IEEE*, vol. 69, pp. 300–331, March 1981.

[3] P. Vaidyanathan, "Multirate digital filters, filter banks, polyphase networks, and aplications: A tutorial," *Proceedings of the IEEE*, vol. 78, pp. 56–93, January 1990.

[4] A. Groth and H. Gockler, "Efficient minimum group delay block processing approach to fractional sample rate conversion," in *IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS*. IEEE; 1999, 2001, pp. 189–192.

[5] T. Denk, C. Nicol, P. Larsson, and K. Azadet, "Reconfigurable hardware for efficient implementation of programmable FIR filters," in *IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS SPEECH AND SIGNAL PROCESSING*, vol. 5. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEE), 1998.

[6] D. Andreas, "VLSI implementation of a one-stage 64: 1 FIR decimator," in *89th Convention of the Audio Engineering Society, J. Audio Eng. Soc.(Abstracts)*, vol. 38, 1990, p. 872.

[7] B. Porat, *A Course in Digital Signal Processing*. New York, NY, USA: John Wiley & Sons, Inc., 1996.

[8] Y. He, C. Le, J. Zheng, K. Nguyen, and D. Bekker, "Isaac - a case of highly-reusable, highly-capable computing and control platform for radar applications," in *Radar Conference, 2009 IEEE*, May 2009, pp. 1–4.

[9] C. Le, "Smap onboard processor digital filter algorithm development," JPL Internal Design Documentation, Febuary 2009.

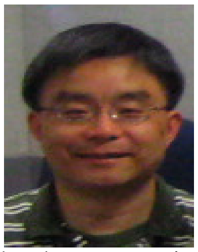
[10] C. Le, M. Spencer, L. Veilleux, S. Chan, Y. He, J. Zheng, and K. Nguyen, "Smap's radar obp algorithm development," in *Radar Conference, 2009 IEEE*, May

2009, pp. 1–4.

BIOGRAPHY

Jason Xin Zheng is a member of the Technical Staff at Jet Propulsion Laboratory. He received a B.S. degree with honor in Electrical Engineering and Computer Science from the University of California, Berkeley in 2004. His research interests include embedded systems, fault-tolerant systems, and DSP algorithm implementations.

Kayla Nguyen received the B.S. degree in Electrical Engineering and the M.S. degree in Electrical & Computer Engineering from the University of California, Irvine in 2007 and 2008, respectively. She is currently an Electronics Engineer at the Jet Propulsion Laboratory in Pasadena, CA. Her research interest is in FPGA implementation of DSP applications.



Yutao He is a Senior Member of Technical Staff at Jet Propulsion Laboratory. He obtained a Bachelor degree in Electrical Engineering from Tsinghua University, Beijing, China, and a Ph.D. degree in Computer Science from UCLA. Since starting at JPL in 2004, he has worked on several flight and technology development projects. Currently he is the PI of two research tasks that are developing advanced avionics technology for future flight spacecraft and instrument missions. His research interests include FPGA-based reconfigurable computing, advanced fault-tolerant computer and avionics architecture, real-time embedded systems, high-performance Linux-based embedded systems for multimedia/network applications, software engineering, and systems engineering of complex systems design. He is members of IEEE Computer Society and ACM. He is also a visiting faculty at Computer Science Department of UCLA and Electrical/Computer Engineering Department of California State University at Los Angeles.