Digital Filtering Alternatives for Embedded Designs

Based on a paper presented at ESC-Boston, September 2006

---

1 Abstract

Whether it's filtering out 60 Hz noise sources or looking for a signature in a certain frequency band, signal processing and filtering applications are abundant and easier than ever to implement with the right tools. How does the embedded micro-controller designer decide the best path to take when tackling his filtering problem (with the least amount of pain)? This paper will provide the necessary guidance to select the best approach for a given application. Whether it's using your current MCU, choosing to substitute/add a DSP or an FPGA, or selecting an ASSP solution, there are a number of decisions to be made. Time to implementation, performance, power, cost, and area utilization will be compared to provide a clear and efficient choice for your filtering application needs.

2 Review of Digital Filtering

Filters are useful for separating, extracting, and restoring signals. They can help remove contaminating sources of noise (like 60 Hz) or isolate tones and patterns within a carrier signal. They can also improve a signal such as poorly sampled audio to better represent the original intent (equalizing).

Digital filters are linear systems that in their most straightforward form can be implemented by convolving the discrete samples of an input signal with the filter's impulse response. The impulse response is basically a sequence of numbers that represents the response of the filter to an impulse input. These numbers are also known as the filter coefficients.

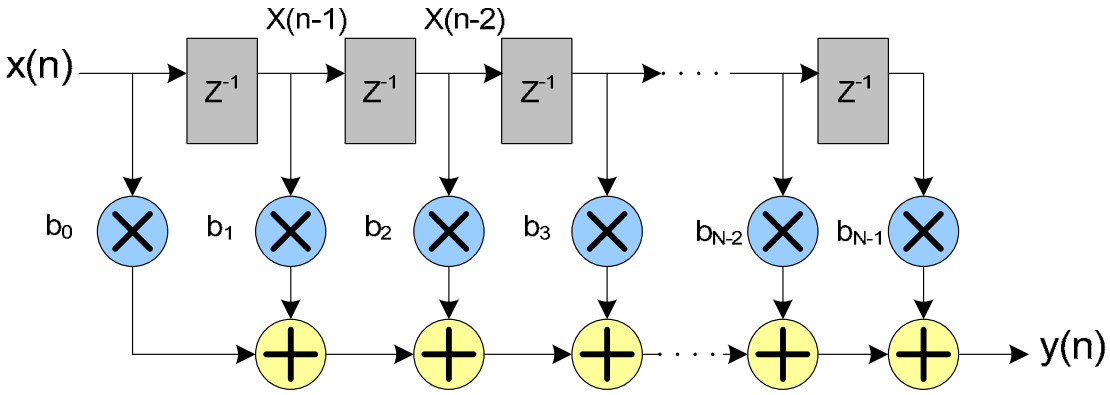
Finite Impulse Response (FIR) filters use a method where each output sample is calculated by first multiplying the samples of the input signal by the coefficients, and then adding them together (accumulating). A characteristic of FIR filters is that their response to an impulse ultimately settles to zero thus is finite in length.

Another approach uses recursion to perform digital filtering. Here previously calculated values in the output as well as values in the input signal are used. These are Infinite Impulse Response (IIR) filters and are defined by a set of recursive coefficients not given by the impulse response but by other means such as a bilinear transform of an analog filter response. Since the output is used in performing these filters, their response to an impulse is non-zero over an infinite length of time, thus is infinite in length.

The basic building blocks used to implement digital filters are multipliers, adders, and storage cells (i.e., RAM). The availability of these resources and how efficiently they are used is a key part of the overall performance of the filter. Here is some additional information on the two types (FIR/IIR) of digital filters.

2.1 FIR Filters

Figure 1: FIR Filter Block Diagram



Source: Quickfilter Technologies, Inc.

Note that the output is a function of the current and previous inputs only (i.e. no feedback). The FIR is therefore a non-recursive filter.

2.1.1 Difference Equation

The difference equation for the FIR filter is as follows:

$$y(n) = \sum_{i=0}^{P} b_i x(n-i)$$

y(n) = output sample at time n

x(n) = input sample at time n

P = order of the filter (# of delay elements) = # of taps -1

b_i = filter coefficients

2.1.2 Transfer Function

The transfer function in the Z domain for the FIR filter is as follows:

$$H(z) = \sum_{i=0}^{P} b_i z^{-i}$$

2.1.3 Key Attributes

The key attributes of an FIR filter are as follows:

- FIR filters have no analog equivalent.
- FIR filters can only be implemented as an “all-zero” filter, since the FIR has no poles.
- FIR filters are therefore inherently stable, since their poles are located at the origin in the z-plane.
- FIR filters have a linear phase response if their filter taps are symmetrical about the center tap position. This results in zero phase distortion in the filter output.
- FIR filters can achieve very high roll-off rates when a large number of taps are used. More taps increase the steepness of the filter roll-off while also increasing calculation time (delay) and for high order filters, limiting bandwidth.
- FIR filters are insensitive to coefficient quantization errors since they are non-recursive.
- FIR filters have no limit cycle issues.
- FIR filters are easy to design and can be implemented using fractional coefficients, which makes fixed point calculations easier.
- FIR filter latency = ((Number of Taps -1)/2) sample periods.

2.1.4 Design Methods

The two FIR filter design techniques that are most commonly used are the Windowed-Sinc and the Parks-McClellan methods, which are summarized below.

Table 1: FIR Filter Design Methods

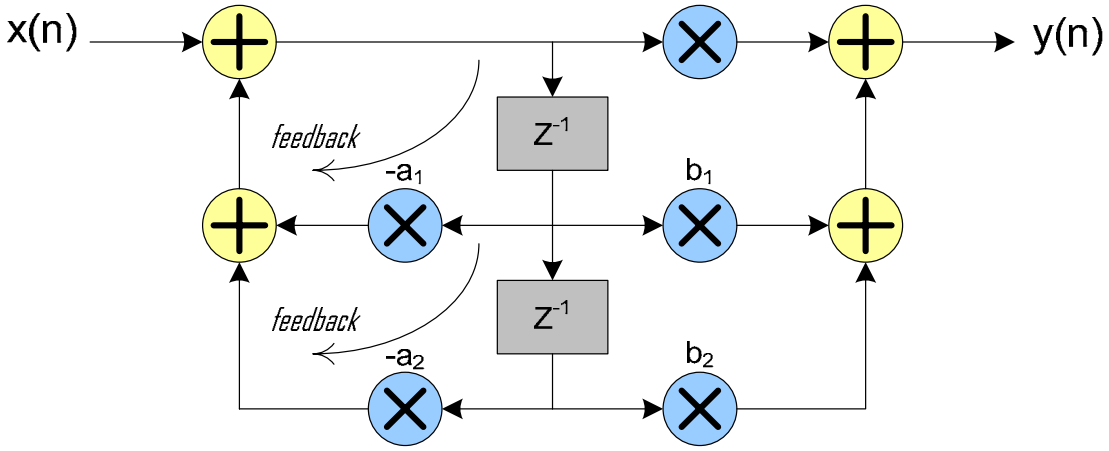
Method	Key Characteristics	Process
Windowed-Sinc	- Low complexity - Low flexibility - Can be calculated “on-the-fly” - Supports adaptive filtering - Good Frequency Domain Characteristics - Sub-optimal design (time domain issues) • Excessive ripple in the passband and stopband • Excessive overshoot in the step response **- Legacy design technique**	The simplest technique is known as Windowed-Sinc. This technique is based on designing a filter using well-known frequency domain transition functions called "windows". The use of windows often involves a choice of the lesser of two evils. Some windows, such as the Rectangular, yield fast roll-off in the frequency domain, but have limited attenuation in the stop-band along with poor group delay characteristics. Other windows like the Blackman, have better stop-band attenuation and group delay, but have a wide transition-band (the band-width between the corner frequency and the frequency attenuation floor). Windowed filters are easy to use and are scalable (give the same results no matter what the corner frequency is). This latter point means that a tunable filter can be designed with the only limitation on corner frequency resolution being the number of bits in the tuning word.

Method	Key Characteristics	Process
Parks-McClellan (equiripple)	- High complexity - High flexibility - Must be calculated “off-line” - Does not support adaptive filtering - **Optimal Design** • Minimizes the maximum error between the desired frequency response and the actual frequency response • Fewer Taps • Fast execution • Precise (equal) ripple control in passband and stopband **- Recommended for most applications**	An Equiripple or Remez Exchange (Parks-McClellan) design technique provides an alternative to windowing by allowing the designer to achieve the desired frequency response with the fewest number of coefficients. This is achieved by an iterative process of comparing a selected coefficient set to the actual frequency response specified until the solution is obtained that requires the fewest number of coefficients. The filter is specified as follows: • Filter Type: Low-Pass, Band-Pass, etc. • Sampling Frequency: {Hz} • Passband Frequency: {Hz} • Stopband Frequency: {Hz} • Passband Ripple: {dB} • Stopband Ripple (Attenuation): {dB} • Word-length: {bits} The outputs of the Parks-McClellan algorithm are as follows: • Number of Taps Required • Frequency Response (Linear and Log Scales) • Step Response • Impulse Response • Filter Coefficients • S - and Z - Plane Analysis For equiripple algorithms some values may converge to a false result or not converge at all. Therefore, all coefficient sets must be pre-tested off-line for every corner frequency value.

Source: Quickfilter Technologies, Inc.

2.2 IIR Filters

Figure 2: IIR Filter Block Diagram – Second-order (Bi-quad)



Source: Quickfilter Technologies, Inc.

Note that the output is a function of the current and previous inputs, and the previous outputs (i.e. feedback). The IIR is therefore a recursive filter.

2.2.1 Difference Equation

The difference equation for the IIR filter is as follows:

$$y(n) = \sum_{i=0}^{P} b_i x(n-i) + \sum_{k=1}^{Q} a_k y(n-k)$$

2.2.2 Transfer Function

The transfer function in the Z domain for the IIR filter is as follows:

$$H(z) = \frac{\sum_{i=0}^{P} b_i z^{-i}}{1 - \sum_{k=1}^{Q} a_k z^{-k}}$$

2.2.3 Key Attributes

The key attributes of an IIR filter are as follows:

- IIR filters are typically implemented as a digital equivalent of a classical analog filter such as Butterworth, Chebyshev, Bessel, or Elliptical.
- IIR filters can be implemented as an "all-pole" or "pole-zero" filter. The location of the poles determines the stability in IIR filters. (i.e. must be within the Z plane unit circle to be stable)
- IIR filters usually have poor phase responses that are very non-linear at the edge of the bands.
- IIR filters can achieve high roll-off rates with a minimum number of taps due to feedback.
- IIR filters are high speed due to the small number of MAC cycles that are required.
- IIR filters are difficult to design for high orders (~8 max) due to stability issues that arise from the effects of finite precision math.
- IIR filters are very sensitive to coefficient quantization errors that occur due to using a finite number of bits to represent the filter coefficients.
- IIR filters are very sensitive to rounding effects introduced by finite precision math operations, which could result in limit cycles, which are never-ending low amplitude oscillations in the output value of the filter.
- IIR filter latency = 1 sample period

2.2.4 Design Methods

The three IIR filter design techniques that are most commonly used are the Bilinear Transform, Impulse Invariant, and the Pole-Zero Placement methods, which are summarized below.

Table 2: IIR Filter Design Methods

Method	Key Characteristics	Process
Bilinear Transform	- Achieves good frequency response match between analog template and digital realization. Impulse response may differ slightly **- Recommended for most applications**	The bilinear transform method is used to convert a transfer function H(s) of a linear, time-invariant filter in the continuous-time domain (i.e. an analog filter) to a transfer function H(z) of a linear, shift-invariant filter in the discrete-time domain (i.e. a digital filter). It maps positions on the s-plane to the z-plane. No aliasing effects here.
Impulse Invariant	- Achieves accurate impulse response match with analog template at the sampling points, but poor frequency response match - **Best suited to lowpass filter implementations**	The impulse invariant method is used to map an impulse response H(t) of the analog filter directly into H(z) in the z domain, The resulting digital filter will have an impulse response that matches the analog equivalent (at the points where is it sampled), but may have a less accurate match in the frequency response. Aliasing effects can occur. The impulse invariant method is not suitable for

Method	Key Characteristics	Process
		high pass filter designs and is best used for low pass filters where the passband is a small percentage of the sampling frequency.
Pole-Zero Placement	- Achieves a match of the pole-zero locations of s-plane analog filter with z-plane digital filter - **Impractical for filters requiring more than a handful of poles or zeros**	The pole-zero placement method requires the designer to use a filter design tool to interactively place poles and zeros on the z-plane, and then evaluate the amplitude and phase response of the filter. The pole-zero placement is "tweaked" until the desired filter performance is reached. Filter parameters such as the width of the pass band or ripple in the stop band are not formal inputs to the design procedure. Rather these characteristics are assessed after an initial pass through the design, and then adjusted (indirectly) via an iterative process.

Source: Quickfilter Technologies, Inc.

2.3 FIR or IIR?

The choice between using an FIR filter or an IIR filter depends largely on the inherent advantages of the two filter types, and is summarized below.

Table 3: FIR vs. IIR

Key Concern	Best Filter	Notes
Guaranteed Stability	**FIR**	FIR filters are inherently stable. IIR filter stability is much harder to ensure.
Ease of Design / Design Time	**FIR**	FIR filters, because of their stability, are insensitive to the effects of finite precision math, which reduces complexity and design time. Also, fractional coefficients can be used for fixed point calculations.
Minimum Phase Distortion	**FIR**	FIR filters can have a linear phase response. With IIR filters, phase is difficult to control.
Multirate Design	**FIR**	FIR filters have some computational advantages over IIR filters in multirate (decimation/interpolation) designs. When decimating, only the FIR outputs that are actually used need to be calculated. For an IIR, since unused outputs are fed back and affect the used ones, there is little or no computational savings. For this reason, designs using decimation all use FIR filters
Sharpest Possible Cutoff	**FIR**	With enough taps, an FIR filter can deliver an extremely narrow transition region with stability.
Sharp Cutoff with Fewest Taps	**IIR**	IIR filters can achieve very high roll-off rates with a minimum of coefficients.
Low Latency	**IIR**	IIR filters are low latency due to feedback. Low latency is important in closed-loop control applications.
High Throughput / Minimum Hardware	**IIR**	IIR filters perform fewer MAC operations, and require less data storage (RAM), due to small number of coefficients resulting from lower order designs. However, coefficients and internal data widths are usually much wider to prevent instabilities due to the effects of finite precision math.

Source: Quickfilter Technologies, Inc.

Because of their inherent stability and the extensive development support for their design and implementation, the remainder of this discussion will focus on designing with FIR digital filters.

2.4 Tools for FIR Design

Using one of the FIR design methods outlined above, there are numerous tools for generating a set of coefficients given the user understands the type of response he is looking for. Key parameters that a user will need to understand in using any filter design tool are:

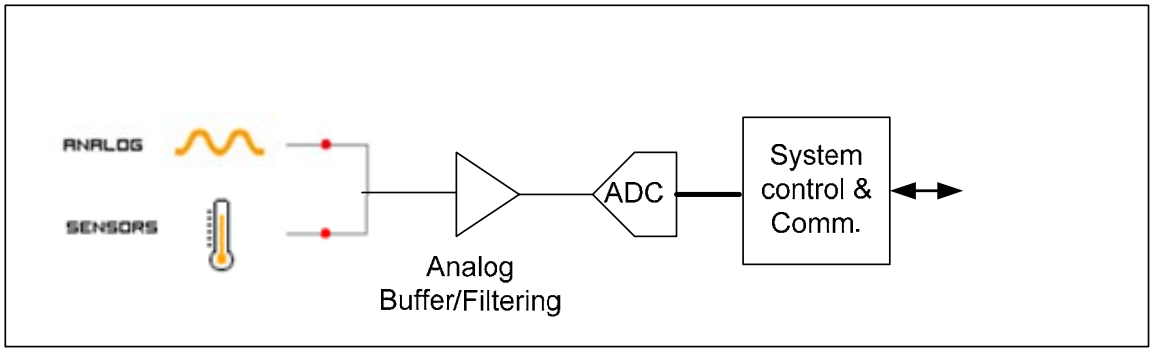
- Filter type: low-pass, high-pass, band-pass, or band-stop
- Sample rate: set at a minimum of 2x the rate of the highest frequency of interest (Nyquist)
- Stop-band Attenuation, pass/stop-band ripple: application dependant

Tools which will generate a set of coefficients include heavyweights like Matlab and Simulink, as well as routines written using Mathcad. But many online, limited use, shareware and freeware tools are available as well. Quickfilter Pro software allows a user to both design AND implement his filter for the ultimate in ease-of-use. For a free demonstration of the Quickfilter Pro software, visit the Quickfilter website at www.quickfiltertech.com.

2.5 An Example Problem

Let's look at an example problem that we need to solve. Currently we have an MCU that is gathering data from an ADC and it is determined that, due to noise sources in our system, some filtering of the sampled data is needed. Specifically we need to get rid of some 60 Hz power line noise and also some higher frequency noise, maybe due to power supply switching. The FIR filter we will design is a Notched Lowpass. It will have a notch around 60 Hz and a passband that goes up to 4 kHz, after which we want to attenuate all frequencies.

Figure 3: Typical Data Acquisition System



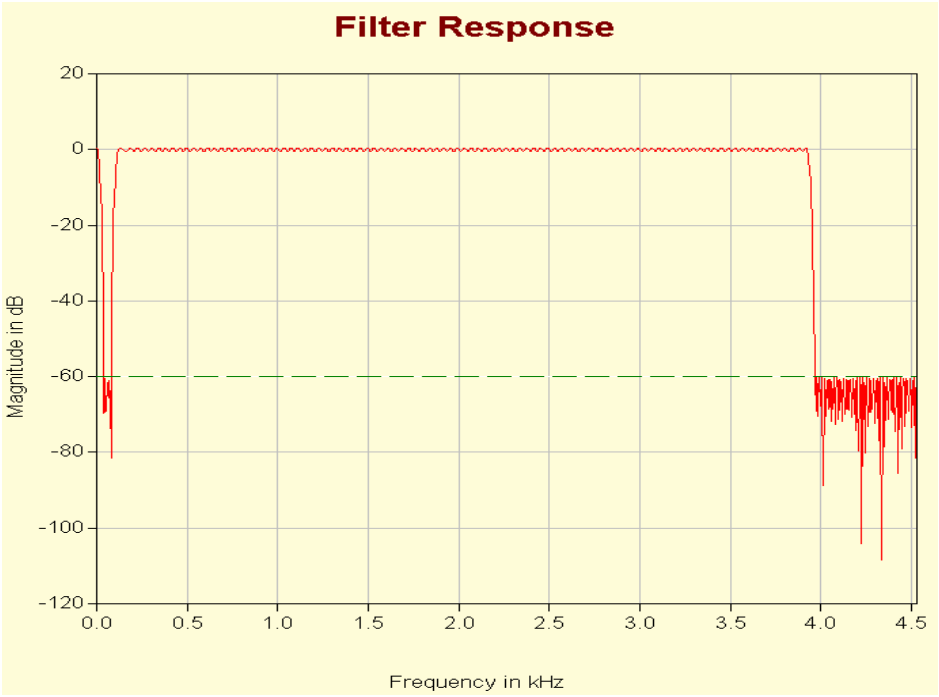
Source: Quickfilter Technologies, Inc.

Some specifications of the Notched Lowpass FIR filter we will design are:

- Notch at 60 Hz : top notch BW = 115 Hz, bottom BW = 40 Hz
- Notch attenuation = 60 dB
- Passband upper frequency = 3.9 kHz
- Stopband lower frequency = 4 kHz
- Passband ripple = 1 dB
- Stopband attenuation = 60 dB
- Sample rate >= 9 ksps

Using software available from Quickfilter Technologies (or any other digital filter design package), we can design this filter and get a set of coefficients for our FIR implementation. The frequency response of our filter is shown below. The number of Taps required to implement this filter is **477**.

Figure 4: Example Problem Frequency Response



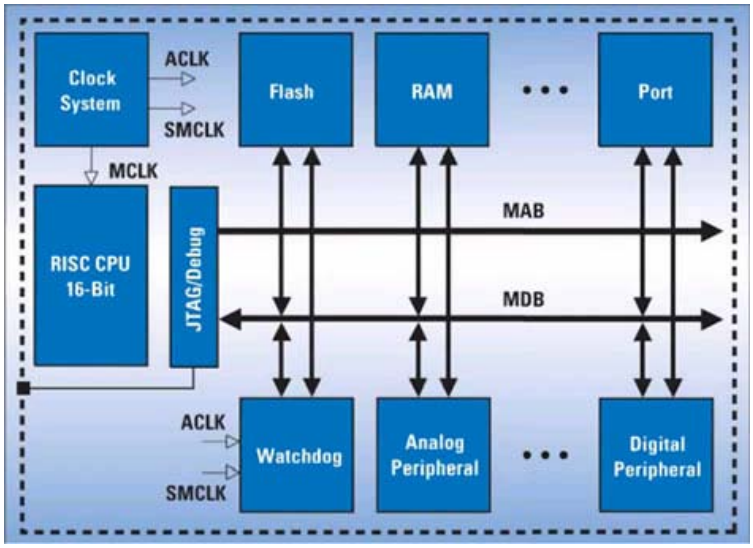
Source: Quickfilter Technologies, Inc.

3 Current Solution – General Purpose MCU

3.1 MCU Architecture

The following diagram shows a typical low cost, low power MCU architecture featuring a Von-Neumann common memory address bus (MAB) and a memory data bus (MDB).

Figure 5: Typical MCU Architecture



Source: Texas Instruments: MSP430

MCUs are very good at data manipulation; moving data from one location to another and testing for inequalities. They are built for cost-effective, low-power system control functions and to achieve this, the CPU pipeline complexity is kept to a minimum.

MCUs are not built for extremely intensive computational applications. Even if the MCU has an embedded multiplier, keeping it supplied with data is a challenge for operations like FIR filtering.

3.2 FIR Implementation

3.2.1 General

Look again at the FIR filter structure in Figure 1. The Z^{-1} blocks represent storage elements. A typical FIR implementation will store input samples in memory (in a circular buffer arrangement) and for each new incoming sample will compute a new

outgoing sample by looping through N-1 (N = # of Taps) stored input samples, performing the appropriate coefficient multiplication and overall accumulation.

Most FIR applications can take advantage of using a Folded FIR design which can be achieved if the coefficient values are symmetric about a midpoint. This is the case for linear phase FIR designs.

Given an example difference equation for a 5 Tap FIR:

$y[n] = b0\ x[n] + b1\ x[n-1] + b2\ x[n-2] + b3\ x[n-3] + b4\ x[n-4]$

Symmetric coefficients require: $b0 = b4$ and $b1 = b3$,
So the equation can be rewritten:

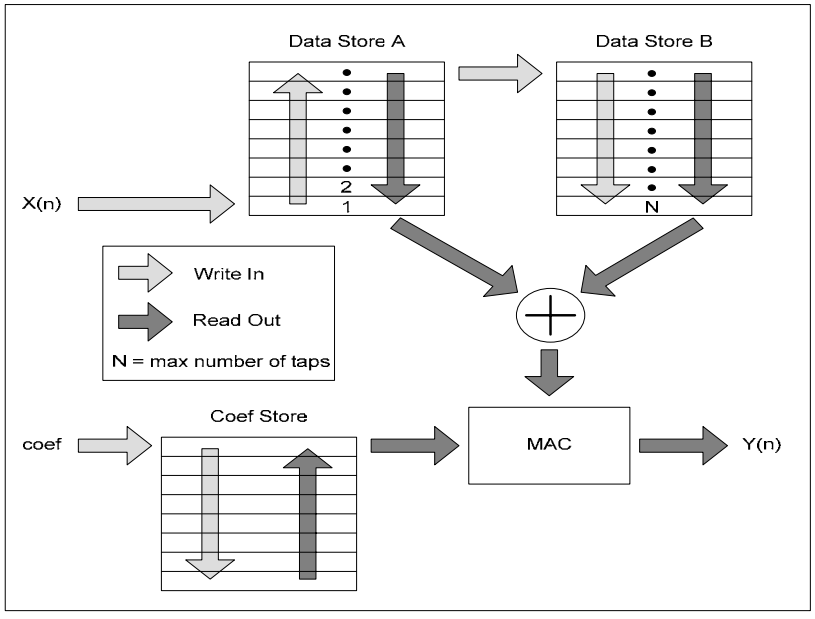
$y[n] = b0\ [x[n] + x[n-4]] + b1[x[n-1] + x[n-3]] + b2\ x[n-2]$

It can be seen that the **Folded FIR** performs an initial addition of the input samples before the multiply, thus reducing the # of multiplies to:

Even Tap Case: Multiplies = Number of Taps/2
Odd Tap Case: Multiplies = (Number of Taps + 1)/2

Figure 6:

Folded FIR Filter : High-Level Conceptual Diagram



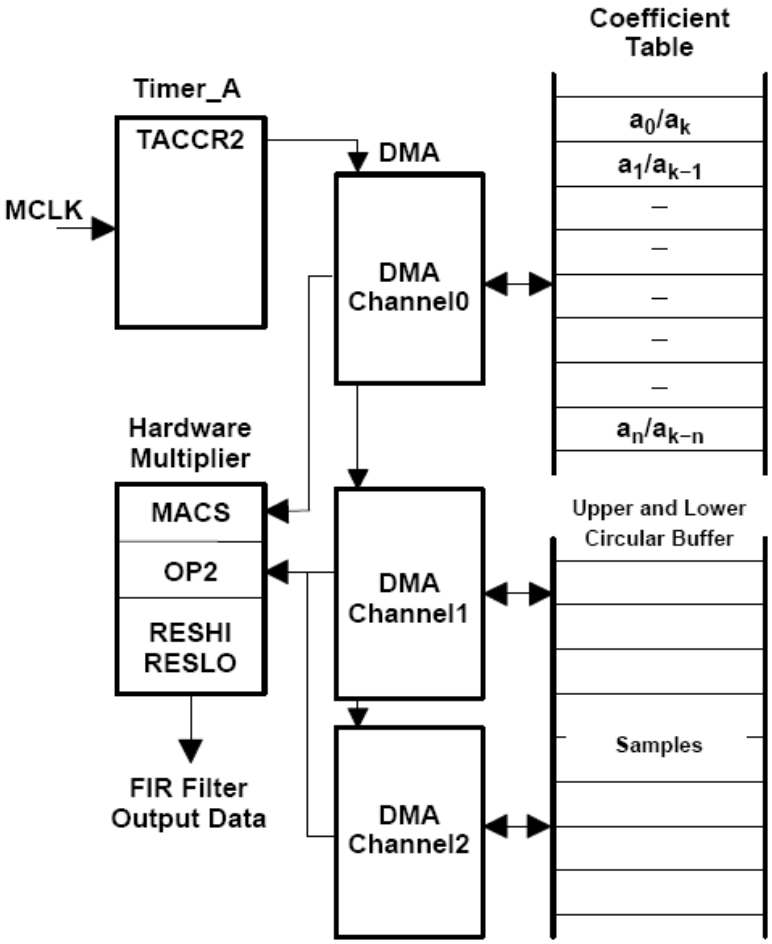
Source: Quickfilter Technologies, Inc.

A **Folded FIR** is the preferred implementation since it uses fewer multiplies, thus reducing the number of loops through the MAC sub-block.

3.2.2 MCU Implementation

Here we'll look at an FIR implementation using an MSP430F16x device (a low cost MCU from Texas Instruments). This device has an integrated hardware multiplier and DMA peripheral to improve the efficiency of the FIR calculations.

Figure 7: Folded FIR Implementation Using the MSP430F169



Source: Digital FIR Filter Design Using the MSP430F16x (Texas Instruments)

Steps to performing the FIR:

- Get input sample from integrated ADC or external interface and store in memory
- Loop N-1 times:
  - Move (2) input samples from memory to MAC
  - Move coefficient from memory to MAC
  - Perform MAC operation
- Retrieve output from MAC and store in memory
- Send output sample to interface

For higher order FIR filters (larger N), the required amount of data movement and the computation time can be large, so the rate of incoming samples will be rather low. The MSP430F169 can perform a DMA-MAC-Memory combination autonomously without any CPU intervention and thus performs the FIR function efficiently for an MCU. See the following table for example performance numbers.

Table 4: MSP430F169 Folded FIR Filter Benchmarks (8MHz CPU clock rate)

Number of Taps, N	Number of CPU Clocks	Maximum Sampling Rate (kHz) (100% CPU loading)
32	310	25
50	410	19
100	730	11

Source: Digital FIR Filter Design Using the MSP430F16x (Texas Instruments)

It is obvious from the above numbers that for a reasonably sized FIR filter (>= 50 Taps), the most efficient MCU approach is only going to allow sample rates in the 10s of kHz. Often, this is a limiting factor and thus other alternative approaches must be investigated.

The table below shows some normalized comparison numbers for the MSP430 and other MCU families. The majority of these other MCUs have **longer** cycle times for the comparable FIR operation, so they will produce **slower** sample times. Again, for many FIR filter applications, an MCU alone won't be able to handle your FIR filtering needs. **Given our example problem, there is no possible way our example FIR of 477 Taps, even after being folded where only 239 multiplies are needed in the loop, can be implemented with an MCU**.

Table 5: Folded FIR Operation Results Normalized to MSP430

MICROCONTROLLER	TOTAL CODE SIZE		TOTAL INSTRUCTION CYCLE COUNT	
	UNOPTIMIZED	FULLY OPTIMIZED	UNOPTIMIZED	FULLY OPTIMIZED
MSP430FG4619	1.00	1.00	1.00	1.00
MSP430F149	1.006	1.006	1.04	1.04
PIC24FJ128GA	1.61	1.52	1.12	1.13
PIC18F242	2.02	1.99	2.17	1.68
8051	2.08	2.04	2.92	2.97
H8/300H	1.41	1.38	2.52	2.51
MaxQ20	1.56	1.47	1.56	1.54
ARM7TDMI	1.52	1.52	0.33	0.31
HCS12	1.91	1.90	9.23	9.54
ATmega8	1.33	1.35	3.23	3.25

Source: MSP430 Competitive Benchmarking

4 Alternative FIR Filter Implementation Methods

4.1 DSP Devices

4.1.1 Overview and Architecture

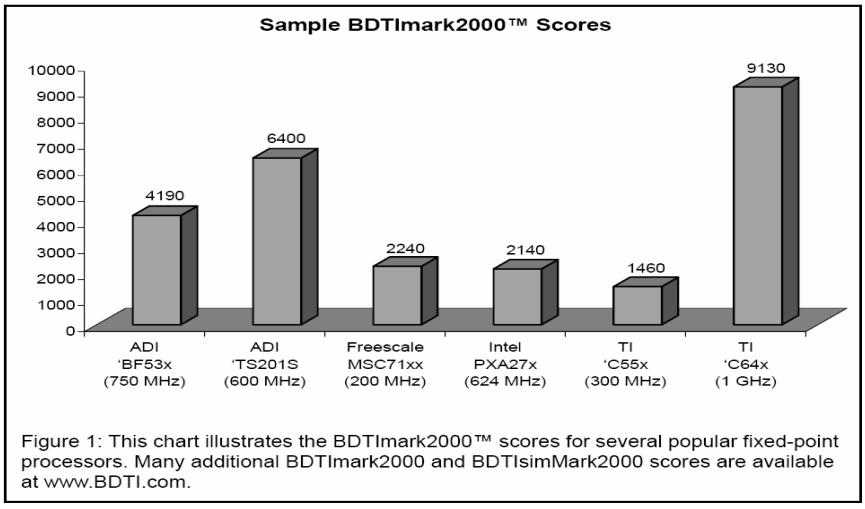
DSP devices are special purpose microprocessors specifically designed to perform digital signal processing. They use special architectures with cache and pipelining stages to accelerate repetitive (looped), numerically intensive calculations. They also employ high-speed I/O with data written directly into memory (circular buffers). They use the latest hardware performance techniques and high clock rates (vs. MCUs) to achieve their main purpose: As fast as possible in real time – get the data in, perform the calculation, and get the data out. All else is secondary.

Among different vendors DSPs, architectures can vary greatly, and the reliance on MIPS (million integer operations / second) and MFLOPS (million floating point operations / second) is not always the best way to identify which one will perform your FIR filtering the fastest. For example, the TI TMS320C6202 at 300 MHz and executing up to 8 instructions per clock cycle, has a MIPS rating of 2400. Freescale's DSP56852 at 120 MHz and 1 instruction per cycle, has a MIPS rating of 120, 20 times less than the TI part. Using BDTI's Real Block FIR Filter Benchmark, the Freescale part is only 6 times slower than the TI part. The discrepancy between MIPS and BDTI's rating is a result of the real amount of work that is actually being done each instruction cycle.

4.1.2 Performance Comparison

Berkeley Design Technology, Inc offers some very good 3rd party benchmarks of the most popular DSP devices. Their analysis includes a very thorough FIR operation run-down. As with the previous example, you can see that the difference in clock speed isn't always proportional to better FIR performance (higher score = faster).

Figure 8: Folded FIR Implementation Using the MSP430F169



Source: BDTI.com

4.1.3 FIR Filtering Implementation

The actual steps involved in performing the FIR in a DSP are very similar to those outlined in the MCU section above. Differences occur in the memory usage where the DSP treats the memory as a circular buffer with pointers that automatically update. Due to a high amount of parallelism and pipelining, the whole - fetch data, perform MAC, return data - process is usually done in one clock cycle.

So how fast is this for our example FIR filter?
If we assume:

- 477 Tap folded FIR design
- 24-bit sample data, 32-bit coefficients
- 160 MIPS with 1 MAC operation per clock cycle and a 32 x 32 multiplier (an average DSP)
- Some overhead : priming 1st loop and completing last loop – generally negligible for large # of taps
- Max Sample Rate = 150M / (478/2) = 628 ksps

We are well under this sample rate, so our example filter is definitely achievable with this average DSP.

4.1.4 Power and Cost

A general survey of power and cost (also from BDTI) is available in the following table. The example FIR from above would be in line with what the TI TMS320C55x could achieve, so for our example FIR filter, cost = $4, and power = 62mW.

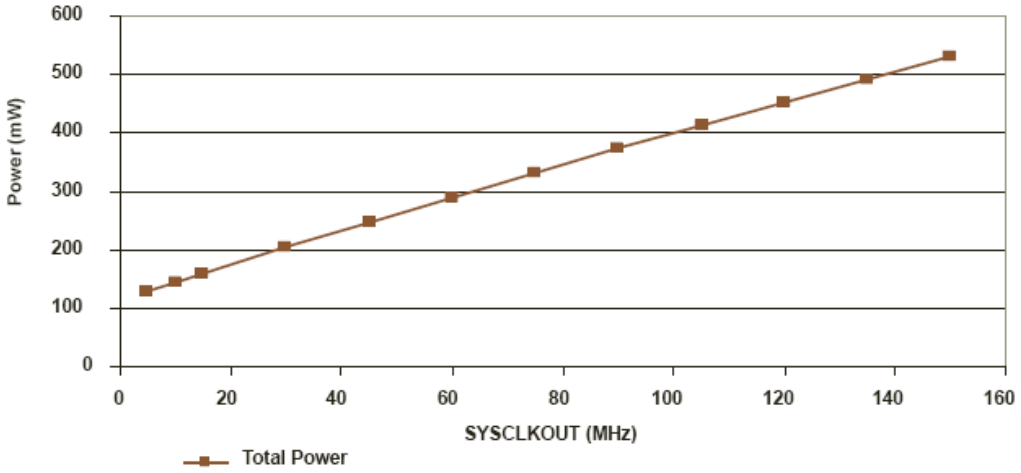
Table 6: BDTI Speed, Power, Cost Comparison – Fixed Point DSPs

Processor Family	Clock Rate (min-max)	BDTImark2000* Score (min-max)	Power (min-max)	Cost (min-max)
ADI ADSP-BF53x (Blackfin)	300-750 MHz	1680-4190	24-644 mW	$5-$32
ADI ADSP-TS201S (TigerSHARC)	500-600 MHz	5330-6400	2583-3907 mW	$186-$205
Freescale MSC71xx (SC1400)	200-300 MHz	2240-3370	222-333 mW	$13-$35
Intel PXA27x (Xscale)	312-624 MHz	1070-2140	n/a	n/a
TI TMS320C55x	160-300 MHz	780-1460	62-204 mW	$4-$17
TI TMS320C64x	400-1000 MHz	3650-9130	654-1303 mW	$15-$208

* From BDTI DSP Kernel Benchmark Results; higher Score = faster; www.bdti.com

In general, floating point DSPs cost more and use much more power for comparable BDTI scores.
Dynamic power is typically going to increase linearly with clock speed as shown below.

Figure 9: Power Consumption Vs. Clock Frequency



4.1.5 Tools Needed

4.1.5.1 Assembly or C?

Compilers are always being improved to produce more efficient DSP machine code, but for many speed critical or complex operations, assembly is still a preferred choice even if used just for that particular function. This added benefit is only available if the code developer has enough knowledge in the functioning of the hardware in order to take advantage of its "shortcuts" and speed enhancements. If new to DSP, then coding in C is probably the safe bet. A rule-of-thumb from a couple of years ago stated that complex function subroutines written in assembly are probably 1.5 – 3 times faster than the comparable high-level program.

Tools for development and implementation of DSPs are very similar to MCU tools. Integrated development environments with debuggers and compilers are available as well as a large base of sample code and routines. The time required to implement and debug these designs can range into many weeks, especially if complex algorithms are employed and mixing of C and assembly routines is needed.

4.2 FPGA Devices

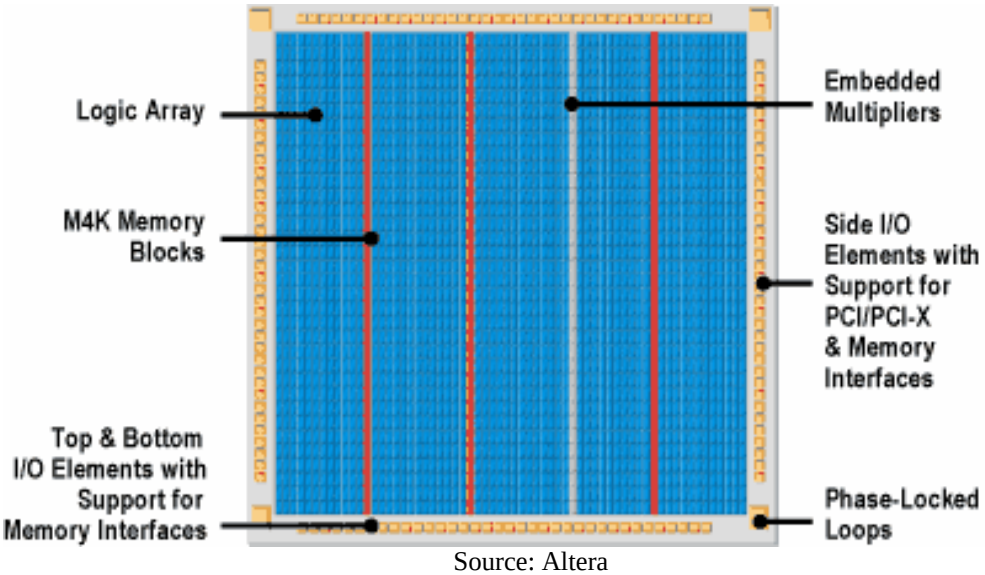
4.2.1 Overview and Architecture

FPGAs by their nature are fully programmable digital logic devices that contain large arrays of logic clusters as well as more specific functional blocks such as mulipliers and RAMs. The available hardware resources vary widely depending on vendor, FPGA family, and size of the part chosen.

Available hardware resources include:

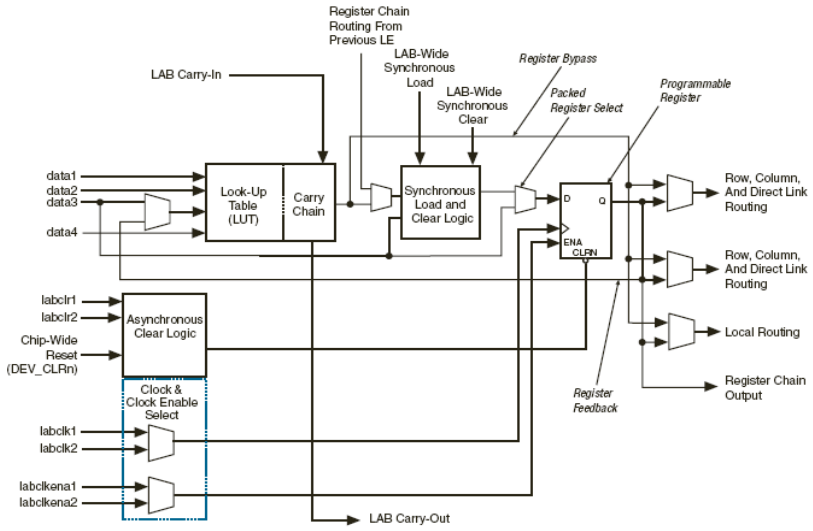
- Configurable general purpose logic blocks (LUT, Slice, LE, LC, ALM, CLB, …)
- Configurable block RAM (SRAM with different R/W port and FIFO combinations)
- Embedded multipliers (usually 18 x 18 bit)
- PLLs and/or DLLs
- Fully programmable I/O cells
- Embedded processing hard cores for very large devices

Figure 10: FPGA Example Architecture – Altera Cyclone II



Source: Altera

Figure 11: FPGA Example Logic Element – Altera LE



Source: Altera

4.2.2 FIR Filter Implementation

Resources needed for a typical FIR filter implementation include:

- RAM storage for both data and coefficients. RAM blocks are typically dual port (2 R/W ports) with each port at most 36 bits wide. The total size is usually 4 kbits or 16 kbits. Assuming again a folded FIR design, the amount of storage for coefficients is half that needed for the input data samples.
- Multipliers are typically 18 x 18 bits with the option to break them up into four 9 x 9 bit blocks. To do a 24 x 32 multiply in an FPGA would take four 18 x 18 multipliers. Multiplier speeds are typically about 150 MHz to 250 MHz depending on the cost/speed of your part. When combining multipliers, the speed can be almost cut in half!
- Adders are needed for the accumulation which (for the output of a 24 x 32 multiply) can be anywhere from 40 to 60 bits depending on how much rounding precision you are willing to live with. Adders are built from the standard logic elements in the FPGA and can be the speed bottleneck. 60-100 MHz for a large adder is typical.

- For our example FIR problem lets assume:
  - 477 Tap folded FIR
  - 24-bit data, 32-bit coefficients, 10 kHz sample rate
  - 4 kbit, dual port RAMS with 32-bit ports
  - 75 MHz 24x32 multiplier (four 18 x 18 multipliers)
  - 75 MHz 62-bit accumulator (24x32 + 8 bits for accumulation)

The implementation will require:

- Multipliers:
  - Need 239 multiplies @ 10 kHz = 2.5 MHz
  - This will take one 24 x 32 multiplier (4 Multiplier Blocks)
- Adders
  - 1 accumulator @ 2.5 MHz
  - 64 x 3 = 192 Logic Cells
- RAM
  - Data RAM = 24-bits x 477 samples = 12 kbits, and due to the folded design, the RAM blocks need to be distributed evenly, so we need 4 RAM Blocks.
  - Coeff RAM = 32-bits x 239 coeffs = 8 kbits = 2 RAM Blocks
- Control Overhead = ~250 Logic Cells
- PLL = 1 for clock multiplication
- **Total Utilization = 4 Multipliers, 6 RAMS, ~450 Logic Cells, 1 PLL**

4.2.3 Part Comparisons

Table 7: FPGA Vendor Family Comparison

- Xilinx
  - Virtex 5 (65nm, 20-200K LC's, 324-1760 pin packages)
  - Virtex 4 (90nm, 14-200K LC's, 363-1513 pin packages, $200-$5K)
  - Virtex-II Pro (.13um, 6-100K LC's, 256-1696 pin packages, $125-$2K)
  - Spartan 3 (90nm, 1.7-75K LC's, 100-1156 pin packages, $10-$250)
  - Spartan IIE (.18um, 1.7-16K LC's, 144-676 pin packages, $12-???)
- Altera
  - Statix II (90nm, 15-180K LE's, 484-1508 pin packages, $280-$6K)
  - Stratix (.13um, 10-80K LE's, 672-1508 pin packages $200-$10K)
  - Cyclone II (90nm, 4-68K LE's, 144-896 pin packages, $12-$300)
  - Cyclone (.13um, 3-20K LE's, 100-400 pin packages, $12-$100)
- Lattice
  - XP (90nm, "non-volatile", 3-20K LUT's, 100-484 pin packages, $13-$70)
  - SC (90nm, 15-115K LUT's, 256-1704 pin packages, $????)
  - ECP (90nm, 6-68K LUT's, 144-900 pin packages, $18-$75)

Source: Quickfilter Technologies, Inc.

Table 8: Altera Cyclone II Selection Guide

Table 1. Cyclone II FPGA Overview

Device	EP2C5	EP2C8	EP2C20	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	18,752	33,216	50,528	68,416
M4K RAM Blocks (4 kbits + 512 Parity Bits)	26	36	52	105	129	250
Total RAM Bits	119,808	165,888	239,616	483,840	594,432	1,152,000
Embedded 18x18 Multipliers	13	18	26	35	86	150
PLLs	2	2	4	4	4	4
Maximum User I/O Pins	158	182	315	475	450	622
Differential Channels	58	77	132	205	193	262

Package Size (mm x mm)	EP2C5	EP2C8	EP2C20	EP2C35	EP2C50	EP2C70
144-Pin TQFP (1) (22 x 22)	89	85				
208-Pin PQFP (2) (30.6 x 30.6)	142	138				
240-Pin PQFP (32 x 32) NEW (3)			142			
256-Pin FineLine BGA® (4) (17 x 17)	158	182	152			
484-Pin Ultra FineLine BGA (19 x 19) NEW (4)				322	294	
484-Pin FineLine BGA (23 x 23)			315	322	294	
672-Pin FineLine BGA (27 x 27)				475	450	422
896-Pin FineLine BGA (31 x 31)						622

Source: Altera

Our example FIR filter with 4 RAM Blocks, 6 Multipliers, 1 PLL, and 450 LE's will fit into the smaller Cyclone II part above. Cost of this part is about $12.

In terms of power usage, most FPGA's are not optimized for dynamic or static power use and thus power for our example application will probably be in the range of 50-100 mW. The dynamic power is related to the clock frequency and resource usage in the part. If the FPGA is used for other housekeeping chores other than our filter, that number will increase.

4.2.4 Tools Needed

Each FPGA vendor offers a suite of tools for synthesis and implementation of the FPGA design. The actual functional coding is done in a Hardware Description Language (HDL) and then is synthesized into a low-level representation that is optimized to use the specific logic cells and functional blocks for each FPGA family. The development, simulation, implementation and validation time required for an FPGA can easily be months.

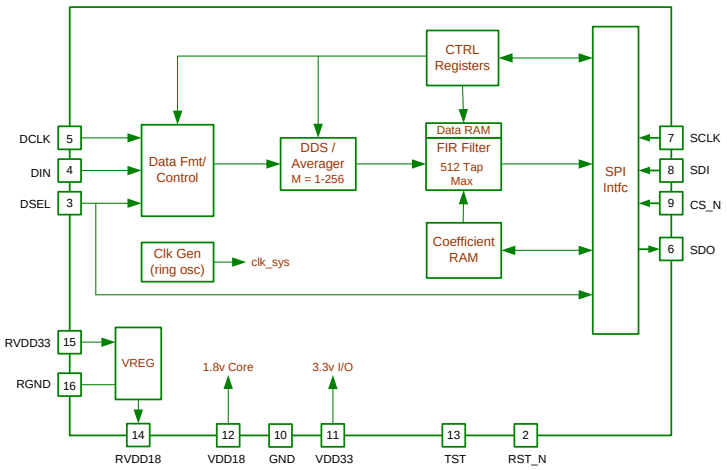
4.3 ASSP – Application Specific Standard Product (SavFIRe™)

Now we will look at using an ASSP device. This is a device that is already customized for FIR filtering and has been developed, along with a formidable software package, to ease both the filter design and the hardware implementation.

Quickfilter's SavFIRe™ ASSP is a tiny (3x3 mm – 16 pin QFN pkg) single-channel FIR filtering device that offers these features:

- Supports up to 512 taps - folded FIR, or 256 taps – non-folded
- Supports data sample rates of 10 sps – 500 ksps
- Supports data size of 12-24 bits, coefficient size of 32 bits
- Simple SPI configuration interface
- Simple Serial configurable data interface
- Supports insertion inline between ADC and MCU – near seamless integration
- Supports filter coprocessor mode
- Has programmable average and downsampling ability
- Requires only a 3.3v power source using its on-chip 1.8v regulator

Figure 12: Quickfilter SavFIRe™ ASSP



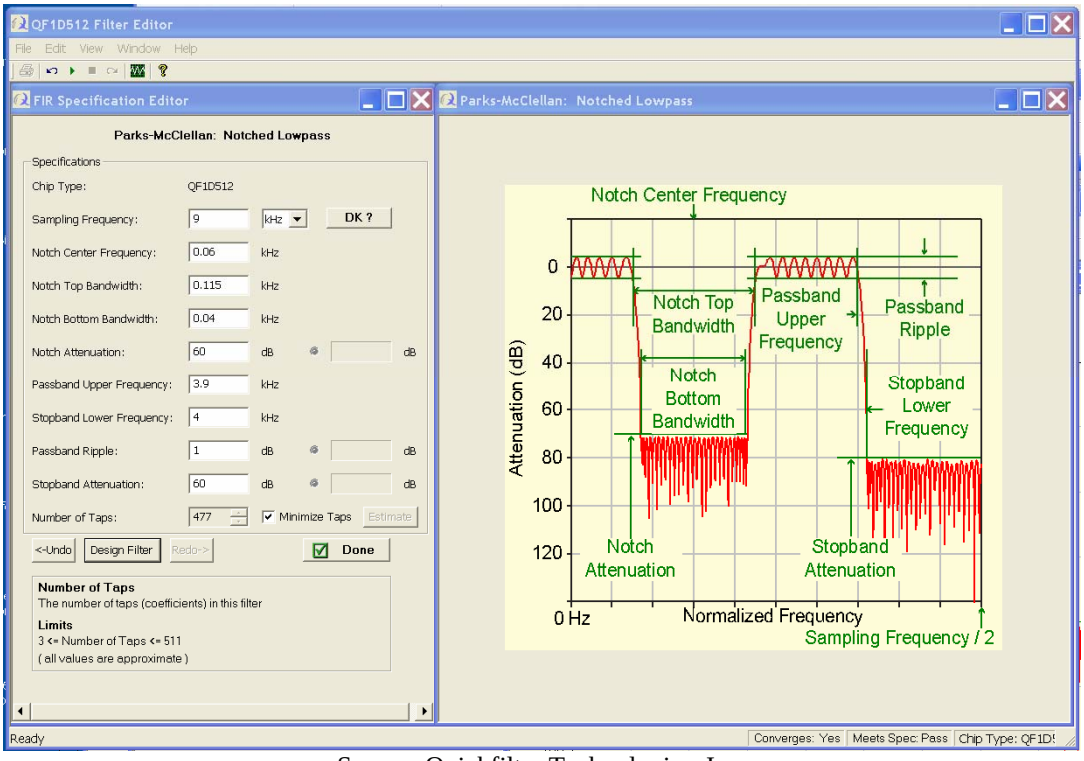
Source: Quickfilter Technologies, Inc.

4.3.1 Filter Design

Using Quickfilter's SavFIRe™ Development Kit, we can easily design our example FIR filter, implement it on the SavFIRe™ (QF1D512) device and view the results of actual data moving through the part in just a few minutes!

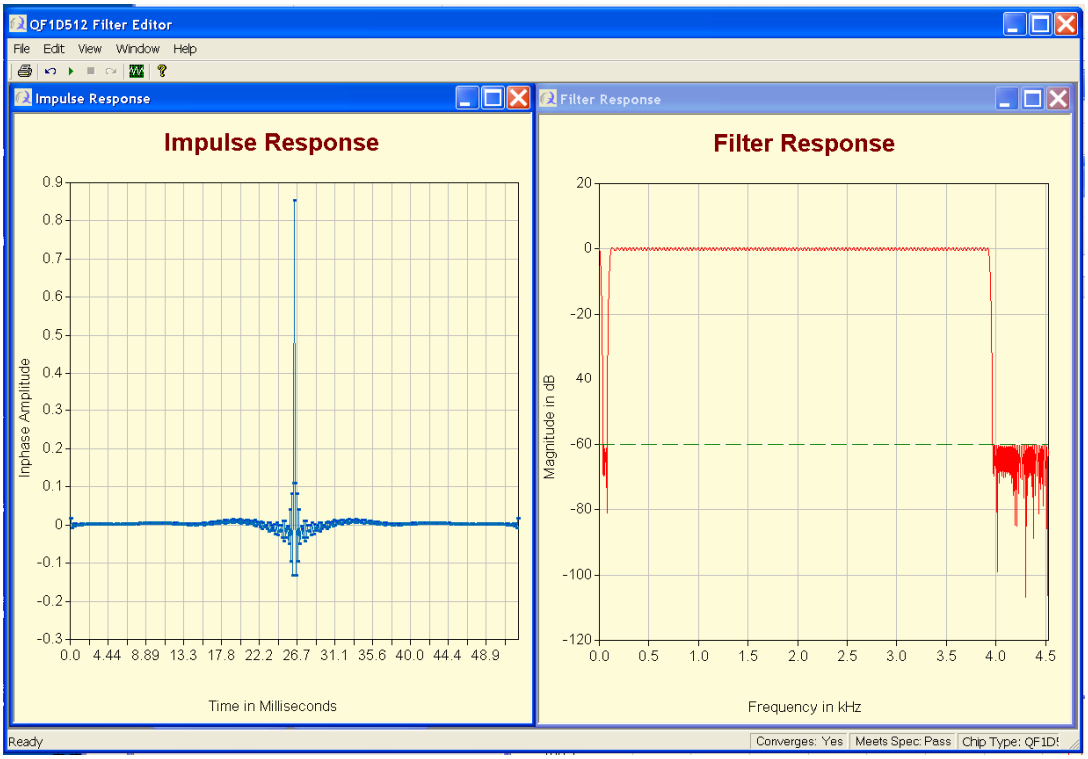
The Quickfilter Pro software offers and easy fill-in-the blank approach for designing FIR filters. Entering in the parameters from our example FIR, we see the following:

Figure 13: Quickfilter Pro Software – Filter Specification Editor



Source: Quickfilter Technologies, Inc.

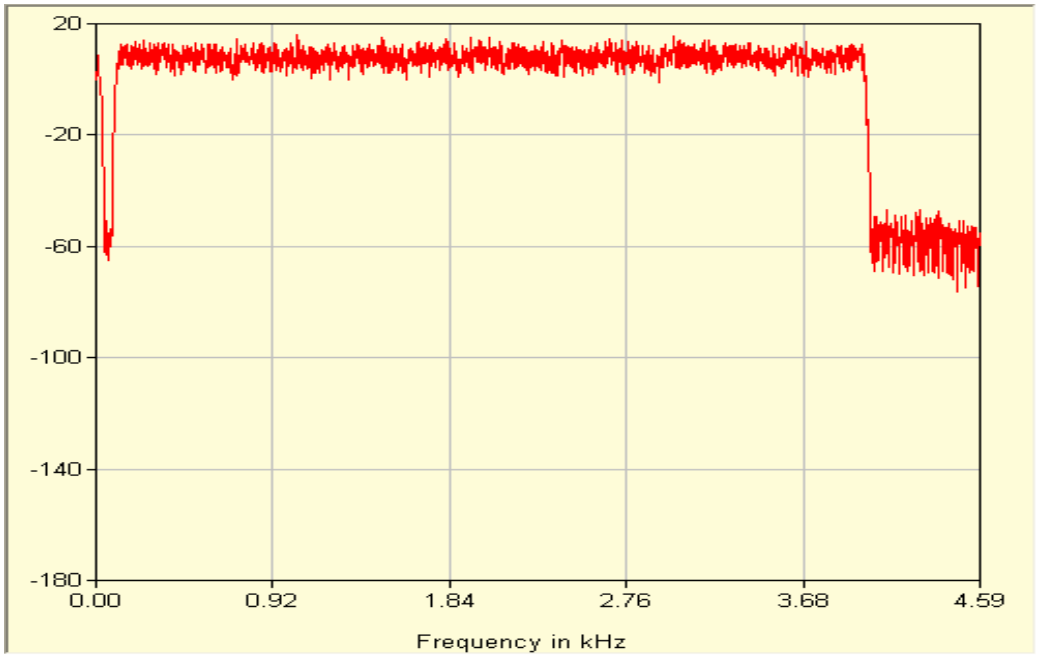
Figure 14: Quickfilter Pro Software – Computed Impulse and Frequency Response



Source: Quickfilter Technologies, Inc.

A list of the coefficients can also be written out to a file. These coefficients along with a few setup values are all that needs to be loaded into the SavFIRe™ device.

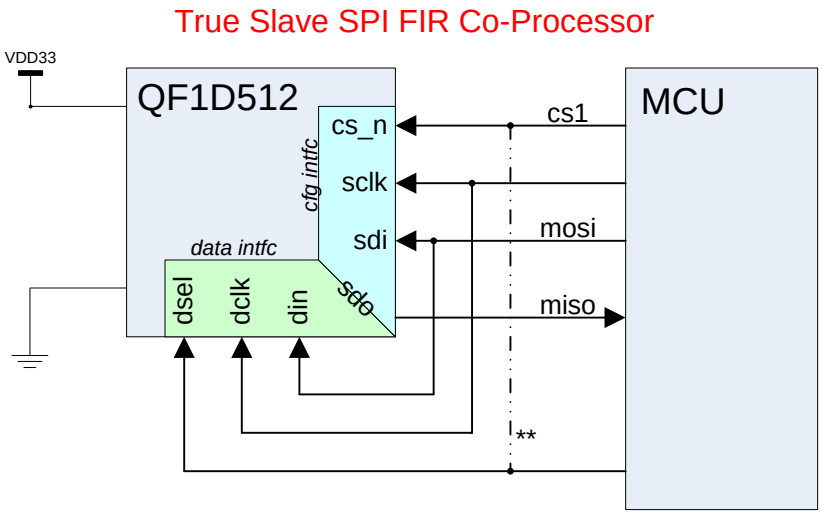
Figure 15: SavFIRe™ Device Output for Example FIR (White Noise Input)



Source: Quickfilter Technologies, Inc.

The easiest way to use the SavFIRe™ device is as a coprocessor to your MCU as shown below.

Figure 16: SavFIRe™ as an MCU Co-processor



** Possible to use a single CS

Source: Quickfilter Technologies, Inc.

The connections utilize an existing SPI port for configuration (CS_N, SCLK, SDI, SDO). Bit-banging the device via GPIO can also be done if an SPI peripheral is unavailable on your MCU.

Once configured (a few registers and all coefficients have been loaded), the FILT_EN bit needs to be set, at which time the device is ready to start filtering data. This is accomplished by clocking the data to be filtered into the DIN port. Concurrently, the filtered output data is sent out the SDO port. It's as simple as that!

5 Conclusion

Table 9: Example FIR Filter Implementation Comparison Summary

Implementation	Performance	Power Used	$ Cost	Physical Area Used	Dev Time Needed
MCU	**Unable to realize above example**: 100 taps max at example sample rate				
DSP	Average DSP can perform # of calculations possible within the sample time	~ 62 mW	~ $4	Smallest Pkg (TMS320C55x) : 179-pin BGA (12x12mm)	Weeks - Months
FPGA	Can perform # of calculations possible within the sample time. Possible waist of resources.	~ 50-100 mW	~ $12	Smallest Pkg (Cyclone II) : 256-pin FineLineBGA (17x17mm)	Weeks - Months
ASSP (SavFIRe™)	Can perform # of calculations possible within the sample time.	< 3 mW	~ $2.50	16-pin QFN (3x3mm). VERY SMALL	Minutes to design and implement with QF Development Kit. Days - Weeks to incorporate into your design

Source: Quickfilter Technologies, Inc.

In conclusion, The SavFIRe™ ASSP solution from Quickfilter provides the most cost and power efficient solution for our example FIR problem. It is also very, very small (3x3mm) and thus could even be inconspicuously be put down on a board where filtering isn't needed yet, but is planned for the future! Finally, the Quickfilter software and development kit make designing and implementing the users filter quite simple, especially compared with the alternative solutions.

6 References

Analog to Digital Filter Migration, white paper, compiled by Paul Highley, Quickfilter Technologies, Inc.

Quickfilter vs. FPGA and DSP Solutions, presentation by Jeramy Leonard, Quickfilter Technologies, Inc.

Digital Signal Processing, A practical Approach, Ifeachor and Jervis, Addison-Wesley, 1993, ISBN 0-201-54413-X.

MSP430 Competitive Benchmarking, Application Report, Greg Morton, Kripasagar Venkat, SLAA2025B – Revised July 2006

Digital FIR Filter Design Using the MSP430F16x, application report, Murugavel Raju, SLAA228 – November 2004

www.dspguru.com

Evaluating DSP Processor Performance, white paper, Berkeley Design Technology, Inc.

Microprocessors vs. DSPs, white paper, Berkeley Design Technology, Inc.

The BDTImark200: A Summary Measure of DSP Speed, white paper, Berkeley Design Technology, Inc.