ColdFire® DSP Library Reference Manual, Rev 0.4

1. Introduction

The ColdFire DSP Library contains digital signal processing algorithms optimized for the ColdFire architecture. These algorithms are implemented directly in assembly for computational efficiency and then encapsulated into a simple C interface. In addition, a large number of predefined and pretested filter configurations are provided in order to reduce the need for a user to design digital filters.

The library is designed to enable embedded sensor applications with basic signal processing functionality, but without the need for a DSP co-processor. By taking advantage of an on-chip multiply-accumulate unit (MAC), a ColdFire microcontroller can efficiently execute DSP algorithms. A typical user may wish to sample an analog sensor (such as an accelerometer) with an analog-to-digital converter (ADC) at a particular rate, filter out unwanted signal components (such as high-frequency noise or other interfering signals), and then use the result in the target application for monitoring, status, data-logging, or control capabilities. The ColdFire DSP Library greatly simplifies the digital filtering part of the process.

2. Acronyms

- DSP – digital signal processing
- MCU – microcontroller unit
- MAC – multiply-accumulate
- EMAC – extended multiply-accumulate
- ADC – analog-to-digital converter
- DAC – digital-to-analog converter
- IIR – infinite impulse response
- FIR – finite impulse response
- SRC – sample rate converter

3. Background DSP Theory

A brief discussion of basic DSP theory will help clarify the features and appropriate applications of the library. Sample rate is a key component to understanding digital frequency and aliasing, and its importance cannot be understated.

3.1. Typical DSP Chain

A typical DSP chain in a sensor system consists of an analog lowpass (anti-aliasing) filter at the front-end, an ADC, one or more digital filters, a DAC, and finally another analog lowpass filter at the back-end. The front-end analog lowpass filter attenuates frequencies above Nyquist before sampling, limiting aliasing effects. The ADC samples and quantizes the signal, the output of which can then be processed by the digital filters. If necessary, a DAC converts the resulting digital signal to an analog signal. Finally, an analog lowpass filter smoothes the output analog signal.

Figure 3-1: Typical DSP Chain
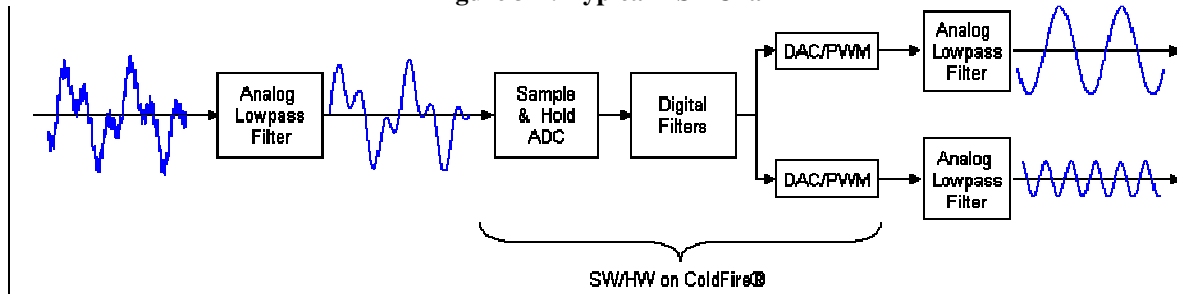
3.2.   Frequency Response

The frequency response of a system describes how that system will respond to the frequency content of the input signal. Given an input signal comprised of one or more frequency components, the response shows how the system modifies each component independently. Frequency is plotted on the x-axis and magnitude on the y-axis, both on logarithmic scales. Because the y-axis units are typically dB, a value of zero indicates the system allows a signal to pass through without any change in magnitude. Negative numbers on the y-axis indicate attenuation or a reduced output signal, while positive numbers indicate amplification. A value of -40 dB corresponds to 100x reduction in amplitude, while +60 dB corresponds to 1000x amplification.

Examining the figure below, the magnitude of the response decreases as frequency increases, demonstrating a lowpass filter. Signals with frequencies between DC and approximately 100 Hz will be modified with a gain of approximately 1. Signal components at a frequency of 10 kHz will be attenuated either by a factor of 0.01 (red response) or 0.0001 (blue response). The blue line corresponds to a Butterworth filter, while the red line is a Chebyshev filter. Both are lowpass filters, but they behave somewhat differently. The Chebyshev filter falls off (decreases in magnitude) faster than the Butterworth, but comes back up at higher frequencies.

Figure 3-2: Lowpass Filter Frequency Response
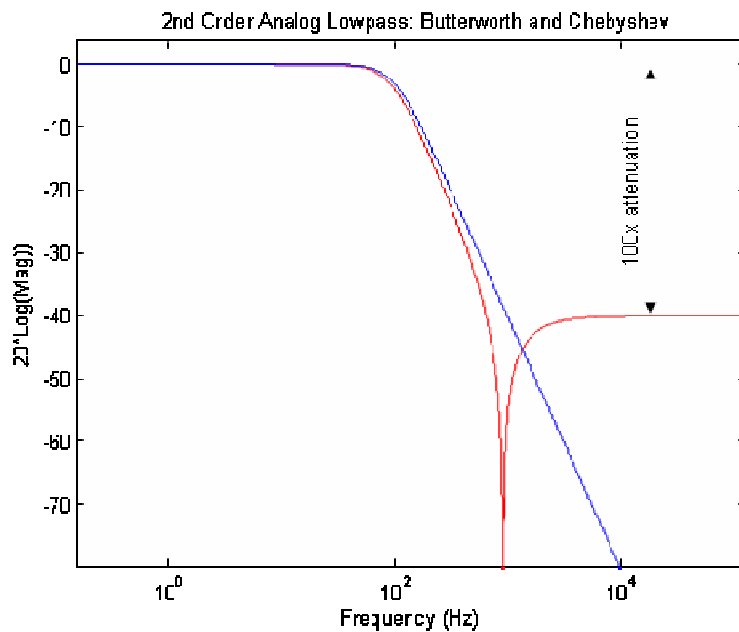
The next figure illustrates the frequency response of a highpass filter. The behavior is opposite to before, attenuating low frequencies and passing high frequencies. Again, the blue and red lines correspond to Butterworth and Chebyshev filters respectively.

Figure 3-3: Highpass Filter Frequency Response
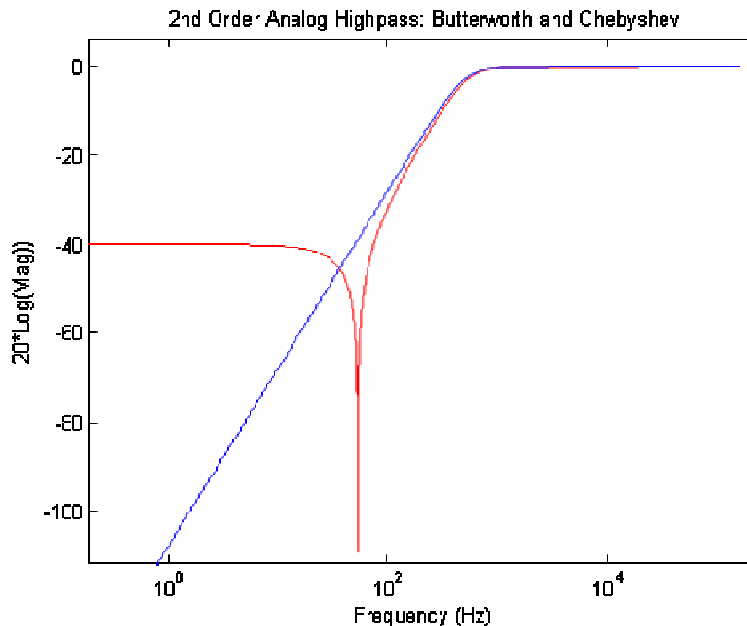

In either case, the range of unattenuated frequencies is called the passband, and the range of attenuated frequencies is called the stopband. By definition, the cutoff frequency separates the two, at -3 dB.

ColdFire® DSP Library Reference Manual, Rev 0.4

3.3.   Sample Rate and Aliasing

The process of sampling an analog signal limits the range of frequencies resolvable in the digital domain. This limit is known as the Nyquist frequency and equals half the sampling frequency. If the original analog signal contains any components above the Nyquist frequency, they will be aliased after sampling. Aliasing causes high frequency signals to appear as low frequency signals, an effect that cannot be undone. It is impossible to determine which components of a sampled signal were present in the original analog signal below Nyquist and which were folded into the resolvable frequency range as a result of aliasing.

The following picture illustrates the effect of aliasing. The original analog signal contains sinusoidal components at 2 Hz and 100 Hz, therefore sampling at any frequency less than 200 Hz causes aliasing. For example, if the signal is sampled at 110 Hz, the 100 Hz component will alias to 10 Hz. Likewise, if the signal is sampled at 105 Hz, the 100 Hz component will alias to 5 Hz. In both cases, the 2 Hz component is unaffected.

Figure 3-4: Aliasing
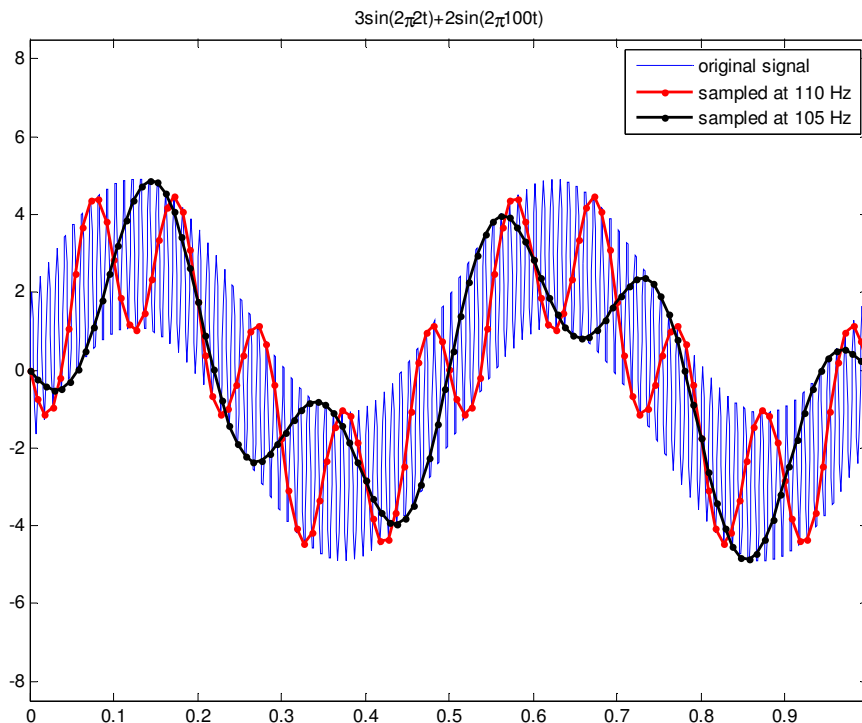
The exact nature of how signals are aliased is beyond the scope of this paper, but it is sufficient in many applications to simply understand which signals will alias, i.e., anything above the Nyquist frequency.

Two solutions exist to the aliasing problem: sample at least twice as fast as the highest possible frequency, or attenuate signals above the Nyquist frequency before sampling. The first solution is computationally expensive and often unnecessary overkill, so the

second solution is usually preferred. An analog anti-aliasing filter reduces or removes signals above the Nyquist frequency before sampling. It is important to note that this must occur in the analog domain before sampling. Going back to the above example, this would mean filtering out the 100 Hz component from the analog signal before sampling at anything less than 200 Hz.

Figure 3-5: Anti-aliasing Filter
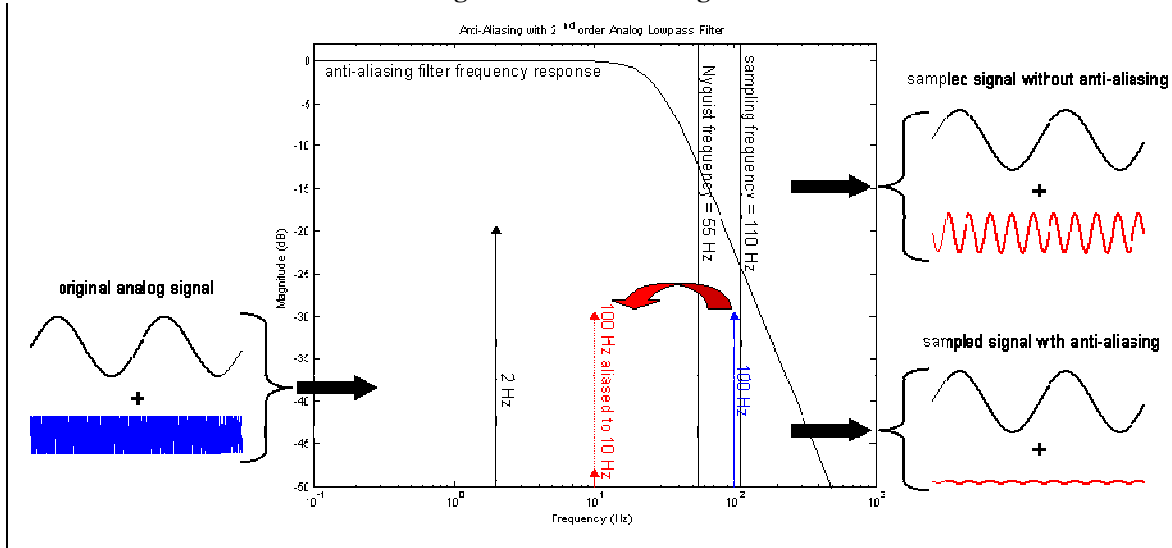
3.4.   Digital Frequency

Digital signals have no concept of time – they are just a sequence of numbers. Likewise, digital systems, such as the filters in the ColdFire DSP Library, process this sequence of numbers (samples). Time and frequency are relative to the sampling rate, governed by the following important relationship:

Equation 3-6: Relationship between Analog and Digital Frequency

$$f_{digtal} = \frac{f_{ana\log}}{f_s/2} = \frac{f_{ana\log}}{f_{nyquist}}$$

In other words, digital frequency is normalized by the Nyquist frequency and ranges from zero to one[1].

3.5.   Analog vs. Digital Filters

An analog filter is realized by discrete circuit components such as amplifiers, resistors, inductors and capacitors. Its frequency response is a function of these component values. Tuning or adjusting an analog filter response requires replacing circuit components. In

---

[1] Some references may use digital frequencies that range from zero to π or 0.5, the upper limit corresponding to the Nyquist frequency in both cases.

addition, parameter variations, temperature and other possibly time-varying sources can significantly affect the frequency response of the analog filter.

A digital filter, on the other hand, is realized by data registers and an ALU. Tuning or adjusting the filter is as simple as modifying register values. Frequency response is a function of coefficient quantization and does not vary with time.

A significant advantage to digital filters is a consequence of normalized digital frequency: the same filter can be used in very different applications because frequency is relative. For example, consider sampling an analog signal at several different rates and then processing with the same digital lowpass filter, defined by the filter cutoff at 0.2 (digital frequency). Depending on the sample rate, the signal may fall in the passband or the stopband of the filter.

Table 3-1: Effects of Lowpass Digital Filter

Analog Frequency	Sample Rate	Digital Frequency	Filter Cutoff	Filter Effect
200 Hz	500 Hz	200/(500/2)=0.8	0.2	signal above cutoff, attenuated
200 Hz	1 kHz	200/(1000/2)=0.4	0.2	signal above cutoff, attenuated
200 Hz	4 kHz	200/(4000/2)=0.1	0.2	signal below cutoff, passed through

3.6.   IIR vs. FIR Filters

The current ColdFire DSP Library contains only IIR filters because of their significant computational advantage over FIR filters. The major difference between FIR and IIR filters is that the latter includes feedback terms.

Equation 3-1: IIR vs. FIR Filter Equations

$$IIR: y_n = \sum_{i=0}^{N} b_i x_{n-i} + \sum_{i=1}^{N} a_i y_{n-i}$$

$$FIR: y_n = \sum_{i=0}^{N} b_i x_{n-i}$$

The presence of feedback terms enables an IIR filter to achieve higher frequency isolation with fewer operations and fewer coefficients (lower order N) than an FIR filter. In general, IIR filters provide a higher rate of attenuation/unit frequency than FIR filters, but at the cost of potential instability or high sensitivity of filter response to small changes in filter coefficients. In the ColdFire DSP Library, this stability or sensitivity issue has been eliminated by our pre-testing of every filter, running on ColdFire hardware (not simulators).

4. Software Architecture

The core component of the ColdFire DSP Library is a group of DSP algorithms implemented in assembly for optimal computational performance. In order to make these

assembly functions more user-friendly, custom data structures and initialization functions are included. As a result, the assembly functions are C-callable with a minimum number of arguments. The user must only initialize a data structure through the use of its associated initialization routine, requiring no intimate knowledge of the data structure implementation or the assembly code. With the assembly algorithms implemented as functions rather than macros, the instruction code is not replicated in memory even if called multiple times.

4.1.   Supported Platforms

ColdFire ISA_A platforms with an on-board MAC are supported. The library was developed and tested using M52221DEMO hardware evaluation board and Freescale CodeWarrior 6.4 Integrated Development Environment (IDE). EMAC platforms are supported as long as the assembler consistently uses the same accumulator (typically ACC0). This was tested with CodeWarrior 6.4 on MCF5227x.

4.2.   Data Types

The ColdFire DSP Library implements a 16-bit datapath, since most ADCs utilized in sensor applications quantize analog data to 12 bits or less. In addition, the ColdFire MAC is optimized for 16-bit multiply-accumulate operations. Longer or shorter word lengths may be used, but must first be cast to a signed 16-bit integer.

It is important to remember that signed 16-bit integers use twos-complement format. Any other data types, including floating-point or unsigned integers, are not compatible with the library. Since sensors and ADCs often operate only on positive voltages, producing an unsigned integer result, it may be required to add an offset to convert to signed twos-complement format.

The DSP algorithms included in this library are linear systems, meaning that superposition and scaling properties apply. The latter property, scaling, allows any fixed-point scale factor to propagate through the system. That is, if an input is scaled by a certain constant value, the output will also be scaled by that value. The system requires no knowledge of the scaling constant.

4.3.   Data Structures

In order to make DSP functions configurable across a variety of applications, they require the ability to parse multiple parameters. For example, an IIR filter algorithm evaluates an equation whose form does not change for different types of filters such as lowpass or highpass or even different frequency cutoffs, as long as the order does not change. The filter coefficients govern this behavior and are therefore supplied to the IIR function as parameters, enabling the same instruction code to execute multiple filters with different frequency responses. Note, however, that different order IIR filters are implemented separately, so a 3rd order IIR filter does not use the same assembly code as a 4th order IIR filter.

Each assembly routine uses its own custom data structure in order to maximize data memory efficiency. It parses the data structure elements internally since it is aware of element sizes, offsets, and relative order. An important consequence of this assembly-level parsing is that data structure definitions cannot be modified in any way, including element order and size.

In general, DSP algorithms utilize state, which means that data structures must be maintained outside the scope of the assembly functions. For example, a 3rd order IIR filter uses three previous input and output values to compute the next output. Consequently, the data structure associated with a 3rd order IIR filter includes an input/output buffer to save these previous values for the subsequent assembly function call.

Although every data structure is different, two common elements exist – input address and output value. The input address points to the location of the input data sample, while the output value contains the actual output data. The use of a pointer for the input allows various DSP functions to be chained together in numerous series or parallel configurations. As a result, the output of a 3rd order IIR filter can be cascaded to the input of a 4th order IIR filter or even another instance of a 3rd order IIR filter.

4.4.   Initialization

The most visible component of the library is the set of data structure initialization functions. These functions allow a user to configure DSP algorithms by setting options such as filter coefficients and fixed-point scale factors. In addition, they load the input pointer entry and clear any buffer entries. Because every data structure is implemented differently, each has a separate initialization function that performs specific operations. Note that initialization should occur only once for each instance of a data structure, prior to executing the assembly algorithm.

4.5.   Algorithm Execution

The actual DSP algorithms are implemented in optimized assembly, but have a simple C-callable interface. They accept just a single argument, a pointer to the appropriate type of data structure, and return void. Algorithms preserve core register state (D0-D7/A0-A7) but do not preserve MAC state (ACCx/MACSR). In general, one call to a DSP algorithm produces one new output value. While the initialization routine executes once, the assembly function usually executes in a loop. It parses the elements in the data structure as necessary, evaluates the algorithm, and then places the result back into the data structure.

4.6.   Putting It All Together

A typical real-time sensor application will first initialize a data structure, enable a timer-based interrupt handler, and fall into an infinite-wait loop or background process. The DSP work then occurs in the timer-based interrupt handler – periodically sampling an ADC and calling the assembly function. The rate of the interrupt defines the sample rate and therefore Nyquist frequency.

5. Directory Structure

The files included in the library are organized as follows:

Table 5-1: Directory Structure

Directory	Filename	Description
headers	dsp_library.h	top-level header file
	dsp_library_defines.h	pre-processor defines
	dsp_library_structures.h	data structure definitions
	dsp_library_init.h	initialization function prototypes
	dsp_library_asm_macros.h	assembly-level macros
	dsp_library_asm_functions.h	assembly function prototypes
	dsp_library_c_functions.h	miscellaneous C function prototypes
filters	iir_filters.h	external declarations for IIR filter configuration parameters
	iir_filters.c	definitions of IIR filter configuration parameters
functions	dsp_library_init.c	initialization functions
	dsp_library_c_functions.c	miscellaneous C functions
	iir2_asm.s	2nd order IIR filter assembly code
	iir3_asm.s	3rd order IIR filter assembly code
	iir4_asm.s	4th order IIR filter assembly code
	iir5_asm.s	5th order IIR filter assembly code
	iir6_asm.s	6th order IIR filter assembly code

6. DSP Routines

The following sections describe the DSP algorithms included in the library, their associated data structures, and function calls.

6.1. IIR Filters: 2nd-6th Orders

Separate assembly routines exist for each order of IIR filter between two and six inclusive. Consequently, separate data structures and initialization routines exist as well, although the general form of each is essentially the same. Each IIR filter order evaluates the following equation where *N* is the order.

Equation 6-1: Nth Order IIR Filter

$$y_n = 2^{-B} \sum_{i=0}^{N} b_i x_{n-i} + 2^{-A} \sum_{i=1}^{N} a_i y_{n-i}$$

Table 6-1: Nth Order IIR Data Structure

Name	Type	Offset	Description, N={2,3,4,5,6}
output	int16	0	Filter output data
diff_sf	uint8	2	Difference between numerator and denominator coefficient scale factors

ColdFire® DSP Library Reference Manual, Rev 0.4

Name	Type	Offset	Description, N={2,3,4,5,6}
			$diff\_sf = B - A$
den_sf	uint8	3	Denominator coefficient scale factor $den\_sf = A$
input	int16 *	4	Pointer to filter input data
flags	uint32	8	Not used
coef	int32 *	12	Pointer to coefficient array $*coef[2(N+1)] = \{b_N,...,b_0,a_N,a_1,a_0\}$
order	uint32	16	Filter order, N
buffer[2N]	int16	20	Buffer to hold previous input and output values $buffer[2N] = \{x_{n-1}, y_{n-1}, x_{n-2}, y_{n-2},...,x_{n-N}, y_{n-N}\}$

There exist two types of routines that accept an IIR data structure argument, one to initialize the data structure and another to evaluate the IIR filter equation in assembly. Although these functions are implemented separately for each order, the prototypes all use the same format.

Table 6-2: Nth Order IIR Function Calls

Purpose	Prototype, N={2,3,4,5,6}
Data Structure Initialization	void iirN_init(IIRN_STRUCT *x, int16 *input, int16 *coef, uint8 num_sf, uint8 den_sf, uint8 order)
Assembly Algorithm	void iirN_asm(IIRN_STRUCT *x)

The following table delineates IIR order-specific names for data structures, initialization functions, and assembly routines.

Table 6-3: IIR Order-Specific Structures and Functions

Order	Data Structure	Initialization	Assembly Algorithm
2	IIR2_STRUCT	iir2_init	iir2_asm
3	IIR3_STRUCT	iir3_init	iir3_asm
4	IIR4_STRUCT	iir4_init	iir4_asm
5	IIR5_STRUCT	iir5_init	iir5_asm
6	IIR6_STRUCT	iir6_init	iir6_asm

7. IIR Filter Configurations

The ColdFire DSP Library includes a large set of IIR filter configurations that span a wide range of applications. These predefined configurations allow a user to quickly select a specific frequency response by making three simple decisions.

Table 7-1: IIR Filter Decisions

Parameter	Options	Implications
Shape	lowpass, highpass, bandpass, notch	Shape of frequency response, i.e., are high frequencies passed through or attenuated
Order	2,3,4,5,6	Rolloff steepness. Higher orders roll off faster

Parameter	Options	Implications
		but require more MCU bandwidth
Cutoff	varies by order, most cover 0.20-0.80 range in 0.05 increments	Digital cutoff frequency (-3dB). Related to analog frequency by sample rate, $$f_{digtal} = \frac{f_{ana\log}}{f_s/2} = \frac{f_{ana\log}}{f_{nyquist}}$$

The following table identifies all filter configurations included in the library. Each combination of filter shape, order and cutoff constitutes a single filter configuration. Each configuration comes with four parameters – filter coefficients array, numerator scale factor, denominator scale factor, and filter order. The configurations below are all characterized as Butterworth IIR filters.

Table 7-2: IIR Filter Configurations

Shape	Order	Minimum Cutoff	Cutoff Increment	Maximum Cutoff
lowpass	2	0.20	0.05	0.85
	3	0.20	0.05	0.85
	4	0.25	0.05	0.80
	5	0.25	0.05	0.80
	6	0.25	0.05	0.75
highpass	2	0.20	0.05	0.80
	3	0.20	0.05	0.80
	4	0.25	0.05	0.75
	5	0.25	0.05	0.75
	6	0.25	0.05	0.75
bandpass	4	0.20	0.05	0.80
notch	4	0.20	0.05	0.80

A straightforward naming convention identifies the order, shape, and frequency cutoff of each filter: butter[ORDER]_[SHAPE]_[CUTOFF(S)]. Rather than listing the name of every filter parameter included in the library, the following table demonstrates several examples. For each of four parameters that constitute a single filter configuration, the base name is appended by the parameter name.

Table 7-3: Naming Convention Examples

Parameter Name	Description
butter2_lp_0_20_coef	array of coefficients for a 2nd order Butterworth lowpass filter, digital cutoff frequency is 0.20
butter3_hp_0_75_num_sf	numerator coefficients scale factor for a 3rd order Butterworth highpass filter, digital cutoff frequency is 0.75
butter4_bp_0_20_0_25_den_sf	denominator coefficients scale factor for a 4th order Butterworth bandpass filter, digital cutoff frequencies are 0.20 and 0.25

Parameter Name	Description
butter4_nt_0_30_0_35_order	order of a 4th order Butterworth notch filter, digital cutoff frequencies are 0.30 and 0.35

Accumulator saturation can cause the filter response to become nonlinear, therefore input magnitudes of 12 bits or less are recommended for 4-6th order filters and 13 bits or less for 2-3rd order filters. This is not a strict requirement since occasional spikes in the input data will be tolerated. A large persistent input (i.e. a step with 15 bits of magnitude), on the other hand, will see a large steady-state error in the output.

8. Hardware Validation

All filter definitions have been tested in hardware for the recommended input ranges. They have been validated against a floating-point model to have minimal fixed-point errors, in both the RMS and absolute maximum measures. Filter definitions that are highly susceptible to fixed-point errors are intentionally excluded from the library. Hence, very low or very high digital frequency cutoffs, especially for higher order filters, are excluded.

9. Performance and Memory

The following table outlines the execution latencies and memory footprints for each assembly-level algorithm included in the library. Execution time is measured at the parent function level, including latencies required to jump into and out of the subroutine as well as overhead from saving and restoring processor state. The instruction code is instantiated in SRAM for minimal memory latency. Performance was measured with a M52221DEMO board. Note that execution times may vary on an EMAC platform.

Table 9-1: Algorithm Performance and Memory

Assembly Algorithm	Execution Time (cycles)	Assembly Code Size (Bytes)	Data Structure Size (Bytes)
iir2_asm	126	142	28
iir3_asm	135	154	32
iir4_asm	139	164	36
iir5_asm	148	174	40
iir6_asm	149	176	44

10. References

1. A. Oppenheim, R. Schafer, and J. Buck, Discrete-Time Signal Processing, 2nd Edition, Prentice Hall, 1999.
2. Freescale Semiconductor, ColdFire Family Programmer's Reference Manual, Rev. 3, 2005.