Agenda

- Representation of signals in time and frequency
- Classification of Filters
- Fast and efficient algorithms
- Validation of filter performance
- Hands-on filter labs using the MSP430
Signal Representation

- Analog systems
  - Everything in continuous domain
  - Analog in, Analog out
  - Post processing difficult
  - Frequency domain analysis difficult
- Digital systems
  - Sampling done to analog signals to convert them to digital using an Analog to Digital Converter (ADC)
  - Conversion back to analog done after processing using a Digital to Analog converter (DAC)
  - Number representations and resolution a key to performance
  - Input/Output easily captured and stored on digital media for post-processing

Frequency Domain Versus Time Domain

- Time domain representation
  - Easy and simple to relate to real-world signals
  - Processing is relatively simpler
  - Easy algorithms that require low CPU overhead
  - Limited capabilities in complex speech or audio applications
- Frequency domain representation
  - Used only for sake of convenience and is truly a Pseudo-domain
  - Conversion from time \(\rightarrow\) frequency must be followed by frequency \(\rightarrow\) time, after processing.
  - Process is expensive and avoided due to CPU overheads
  - Best perspective of signal content leading to efficient system design
  - Absolute necessity for all speech and audio applications
  - Processing frequency domain although expensive, yields the best results
**Agenda**

- Representation of signals in time and frequency
- Classification of Filters
- Fast and efficient algorithms
- Validation of filter performance
- Hands-on filter labs using the MSP430

**Broad Classification of Filters**

- **Analogue Filters**
  - Butterworth
  - Elliptic
  - Chebyshev

- **Digital Filters**
  - Finite Impulse Response (FIR)
  - Infinite Impulse Response (IIR)

**Analogue Filters**

- Mature and well-developed design methodologies available
- Accuracy limited, use components that are subjected to changes over temperature
- Small change in filter specifications leads to complete change in hardware
- Testing and verifications are time-consuming
- Storage and portability a cause for concern
- Inherently expensive to improve accuracy

**Digital Filters**

- Design is simple, borrows all concepts from its analog counterpart
- Modifying characteristics requires small change in software with no hardware changes necessary
- Interface to digital microprocessors is extremely simple
- Extremely accurate - at least a 1,000 times better accuracy than its analog counterpart
- 6dB increase in gain with every bit of increase in resolution on fixed point machines
- Effects of round-off, finite-word lengths and limit cycles a hindrance in fixed point machines
Characteristics of a Filter

- **Low-pass**
  - Removes HF noise, required for most applications
- **High-pass**
  - Seen in few speech and audio applications
- **Band-pass**
  - Medical and communication systems
- **Band-reject**
  - Communication systems
- **Notch**
  - Removal of the 50/60 Hz hum in applications

Digital Filter Basics

- Input sample \(x(n)\) operated by filter \(h=[h(0), h(1), ..., h(M)]\) to give output sample \(y(n)\), every sampling instant, defined by sampling frequency
- Implementation is simple
  - Mathematically a convolution of input vector \(x\) and filter vector \(h\) of order \(M\)
  \[
  y = x \otimes h = \sum_{i=0}^{M-1} h(i) \cdot x(n - i)
  \]
- Current output sample depends on present and previous samples of input and/or output samples
- Filter \(h\) is almost always a real number, and is usually converted into the nearest integer on fixed-point numbers such as microcontrollers
**FIR Filter**

- Impulse Response is finite
  - Time domain representation of the filter $b$ has finite length and equal to the length of vector $h$
- Inherently stable
  - Output depends only on the present and previous samples of the input
  - Output always bounded by input, if input stops, output immediately follows
- Can exhibit linear phase across all frequencies
  - Linear phase property induces symmetry for coefficients $b$, thus reducing CPU overhead
  - Does not introduce phase distortion

\[ y(n) = \sum_{i=0}^{M-1} b(i) \cdot x(n - i) \]

**IIR Filter**

- Recursive in both input and output samples
- Designed by a simple transformation from it Analog filter counterparts
- Better performance than an FIR
  - For same performance the order of an IIR is way smaller than an FIR
- Stability a concern
  - Extremely sensitive to filter coefficients
  - Register-width limitations in fixed point machines can render a stable filter, unstable

\[ y(n) = \sum_{i=0}^{M-1} b(i) \cdot x(n - i) - \sum_{k=1}^{N-1} a(k) \cdot y(n - k) \]
Agenda

- Representation of signals in time and frequency
- Classification of Filters
- Fast and efficient algorithms
- Validation of filter performance
- Hands-on filter labs using the MSP430

Need for Fast and Efficient Algorithms

- Multiplication is the core operation for filtering
- Microcontrollers are fixed-point devices, simple in architecture and have limited CPU cycles
- Real-time signal processing broadens the application space
- Reduced CPU utilization increases usability and accommodates better and advanced features
- Less CPU cycles implies lower current consumption with increased battery life, especially for hand-held devices
Fast Multiplication

- Hardware multiplier
  - Simple to use
  - HW multiplier includes a multiply and accumulate (MAC) function for filtering
  - ADC samples are integers, coefficients need to be converted from real numbers to integers, via scaling
  - Not available on all devices
- Absence of a hardware multiplier
  - Booth’s algorithm
    - Based on shift and add operations
    - Fast and efficient
    - Limited to integer-integer multiply
  - Horner’s scheme
    - Also based on shift and add arithmetic
    - Tailor-made for microcontrollers
    - Faster than booth algorithm when used with CSD format

Filtering on the MSP430, the Efficient Way!

- Horner’s algorithm
  - Based on the difference in the bit positions of binary 1s in the multiplier
  - Finite word-length effects does not affect the multiplier
  - Better accuracy compared to the existing methods
  - Scaling of multipliers not needed and easily accommodates real-integer multiplies
  - Multipliers have to be known in advance for it to work
  - Dedicated software routine for each multipliers with increase in code size
Canonical Signed Digit (CSD)

- Maximize efficiency of Horner’s method, elimination of every extra cycle counts!!
- Similar to Booth’s encoding: It increases the speed of execution
- “-1” added to the existing binary set thereby converting it to a ternary set
- Algorithm: Reducing groups of adjacent 1s and representing them using a ternary set
- Leaves the 0s unchanged
- Example

  123 = 011110111, Binary format
  123 = 011110 11 grouped → 011110 00 000 = 011110 00 000
  123 = 10000 01 01 0, Ternary format, 01 01 0 = -1
  123 = 128 - 4 - 1 = 123

Horner’s Method + CSD

- Advantages
  - Further reduction in CPU cycle count
  - No compromise the accuracy of multiplication result
  - Smaller code size for each multiplier

**Fraction**

\[ 0.12345 = 0.00011111111001_{\text{CSD}} = 0.00100000 \bar{1}001_{\text{CSD}} \]

\[
\begin{align*}
X_1 &= X \cdot 2^{-3} + X \\
X_2 &= X_1 \cdot 2^{-1} + X \\
X_3 &= X_2 \cdot 2^{-3} + X \\
X_4 &= X_3 \cdot 2^{-3} + X \\
X_5 &= X_4 \cdot 2^{-3} + X \\
X_6 &= X_5 \cdot 2^{-3} + X \\
\text{Final result} &= X_6 \cdot 2^{-4}
\end{align*}
\]

**Integer**

\[ 441 = 0110111001_{\text{CSD}} = 100 \bar{1}00 \bar{1}001_{\text{CSD}} \]

\[
\begin{align*}
X_1 &= X \cdot 2^3 - X \\
X_2 &= X_1 \cdot 2^2 + X \\
X_3 &= X_2 \cdot 2^3 + X \\
X_4 &= X_3 \cdot 2^1 + X \\
X_5 &= X_4 \cdot 2^3 + X \\
\text{Final result} &= X_5 \cdot 2^0
\end{align*}
\]
Performance Comparison

<table>
<thead>
<tr>
<th>Type</th>
<th>Methods</th>
<th>Instruction Cycles</th>
<th>Code Size (bytes)</th>
<th>Results</th>
<th>Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer-integer multiplication ((81 \times 81))</td>
<td>CLIB(1)</td>
<td>77</td>
<td>56</td>
<td>18081</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Exiting methods</td>
<td>107</td>
<td>54</td>
<td>18081</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>FRUIT</td>
<td>15</td>
<td>32</td>
<td>18081</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Hamming-32</td>
<td>13</td>
<td>32</td>
<td>18081</td>
<td>0</td>
</tr>
<tr>
<td>Integer-Fixed multiplication ((41 \times 61,837))</td>
<td>CLIB</td>
<td>42(2)</td>
<td>822</td>
<td>16110.3375</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Exiting methods</td>
<td>127</td>
<td>54</td>
<td>18081</td>
<td>34.3375</td>
</tr>
<tr>
<td></td>
<td>FRUIT</td>
<td>92</td>
<td>69</td>
<td>18115</td>
<td>0.5375</td>
</tr>
<tr>
<td></td>
<td>Hamming-32</td>
<td>29</td>
<td>69</td>
<td>18115</td>
<td>0.5375</td>
</tr>
</tbody>
</table>

(1) The C library is part of the EIM (Embedded Workbench). Win34 A. Mexico for MSP430 family of devices.
(3) Includes cycles for type conversion from float to integer as part of a requirement of the algorithm used.

- Detailed information on web
  - Efficient Multiplication and Division Using MSP430 (slaa329)

Agenda

- Representation of signals in time and frequency
- Classification of Filters
- Fast and efficient algorithms
- Validation of filter performance
- Hands-on filter labs using the MSP430
Fourier Analysis

- Quick way to convert time domain representation to frequency
- Frequency domain truly reflects signal content
- Only method to validate filter performance
- Complex number representation
  - Magnitude is the energy composition at each frequency
  - Phase is the shift of the time domain signal at each frequency
- Comparison of filter input and filter output Fourier representations validates filter performance
- Not required once filter performance is validated

Discrete Fourier Transform (DFT)

- Fourier representation of time signal after sampling
- Discrete in time and frequency
- Block processing of data, where N represents block size
- Frequency resolution depends on sampling frequency and N
- Not an efficient way of Fourier representation
- Utilization of CPU is very high due to the complexity of math involved

\[ X(k) = \sum_{n=0}^{N-1} x(n) e^{-j \frac{2\pi kn}{N}}, \quad k = 0, 1, \ldots, N-1 \]
Fast Fourier Transform (FFT)

- Efficient form of the DFT
  - Conventional DFT requires $O(N^2)$ complex MAC
  - FFT requires only $O(N/2 \times \log_2 N)$ complex MAC
- Most commonly used in DSPs and microcontrollers
- Bit reversal required
  - Decimation in time reverses time samples
  - Decimation in frequency reverses frequency samples
- Several variants available
  - Radix 2: Simple to use, most widely used, $N$ is a power of 2
  - Radix 4: More efficient, MACs reduced by 25%, $N$ is a power of 4 and hence less number of stages
- Block processing of data unlike filtering
  - Choice of $N$ must be realistic
  - Higher $N \rightarrow$ larger computation, higher memory requirements, larger delay in processing
  - Choose $N$ wisely to not-jeopardize real-time operation

Agenda

- Representation of signals in time and frequency
- Classification of Filters
- Fast and efficient algorithms
- Validation of filter performance
- Hands-on filter labs using the MSP430
Lab 1: Low-pass Filter

• Specifications
  – Design a low-pass filter that has a 3-dB cut-off frequency at 700 Hz.

• Input signal characteristics
  – Signal is a multi-tone signal with the following frequencies:
    • f1=100Hz, f2=500Hz, f3=800Hz
    • Stop-band attenuation should be ~ 50dB, pass-band ripple ~ 0.1dB

• Things to ponder
  – Choice of sampling frequency?
  – Choice of pass-band and stop-band edge frequencies?
    • Remove the high frequency signal 800Hz
  – Choice of order, do you have filter length restrictions?
    • What order does the tool give you?
    • Does the free version give errors? If so, WHY?
    • What stop band attenuation and pass-band ripple does the tool report? DOES IT MEET YOUR REQUIREMENTS?

Lab 1: Low-pass Filter- Coefficient Generation

• Step 1: Find lab folder (LAB-3) from the ATC2008 CD.
• Step 2: Copy entire lab contents to desktop.
• Step 3: Open lab folder and on the root select →FIR_filter\ScopeFIR401\ScopeFIR.exe
• Step 4: Double click on executable to see windows similar to this and click Run and then Start ScopeFIR
• Step 5: Select the default filter design algorithm as shown and click OK
Lab 1: Coefficient Generation (Contd.)

- Fill all parameters as shown
- Click on Estimate first, how many taps do you get?
- It is < 32, click Design

Lab 1: Filter performance in Theory

- What order does the tool give you?
- What stop band attenuation and pass-band ripple does the tool report?
- DOES IT MEET YOUR REQUIREMENTS?
Lab 1: Exporting Coefficients

- From the file menu, select **Export Coefficients**
- Choose **Text Decimal** and use **Browse** button to save as a **LPF.txt (text file)** in a known location in local hard drive.

Lab 1: Generation of MSP430 FIR filter code

- Once coefficients are ready, it is time to generate the MSP430 code
  - Open `xx\FIR_filter\Utility\FIR_filter_coeff.dat`, paste the coefficients you just created and saved in file LPF.txt, Save `FIR_filter_coeff.dat`
  - Double-click on the MSP430 code generation tool `xx\FIR_filter\Utility\FIR_filter_codegen.exe`
  - Enter all the information as shown in this snap shot

- Choose a multi-tone signal frequencies to generate simulated data
  - Open `xx\FIR_filter\Utility\Sine_data_gen.exe`
  - Select up to 3 frequencies that are realistic This generates the multi-tone signal data
Lab 1: Summary of files

- **FIR_filter_wrapper.c**
  - Wrapper file that performs function calls for filtering, FFT and LCD display
- **FIR_filter.asm**
  - MSP430 assembly code generated by the tool FIR_filter_codegen.exe
  - Responsible for efficient digital filtering
  - Copy this file to the lab directory → \FIR_filter\FIR_for_MSP430
- **FFT_430.asm**
  - MSP430 assembly code that performs an FFT in time
  - Responsible for generation of data for LCD
- **sine_data.dat**
  - Simulated multi-tone sine wave data file
  - Generated by the tool Sine_data_gen.exe
  - Copy this file to the lab directory → \FIR_filter\FIR_for_MSP430

Detailed information on web

*Efficient MSP430 Code Synthesis for an FIR Filter (slaa357)*

Lab 1: MSP430 Code Execution

- Connect the ATC2008 target board to the MSP430 USB FET via the USB cable and connect 14-pin JTAG cable to JTAG header on board
- Open CCE
  - Start menu → Programs → Texas Instruments Inc. → Code Composer Essentials V3 → Code Composer Essentials V3
- In window Workspace Launcher click OK to have the default location for Workspace or choose /Lab-3/FIR_filter
- Open Project
  - From Project menu select → Open Existing Project
  - **Select Root Directory** using Browse button to Desktop. Choose lab folder → /Lab-3/FIR_filter
  - Check box for FIR_for_MSP430 (if not already checked) directory and click Finish
Lab 1: MSP430 Code Execution (Contd.)

- Ensure power selection switch is JTAG
- Ensure jumper is present on JP2
- Select from the menu Run → Debug Active Project
- Advanced RUN would start execution on the target

Lab 1: Software Flow

- FFT of input data evaluated
- Input data is filtered
- FFT of filtered data evaluated
- LCD configuration complete
- FFT of input data displayed
- Switches control the LCD contents
Lab 1: Viewing the output

- Viewed on the LCD screen
- Output is the FFT of the signal
- By default the spectrum of the simulated input signal is shown
- Press S2 to see filtered spectrum
- Press S1 to change view and back to unfiltered input spectrum

Lab 1: Results Summary

Real-time MSP430 Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling frequency</td>
<td>2000 Hz</td>
</tr>
<tr>
<td>Filter length</td>
<td>27</td>
</tr>
<tr>
<td>Example CPU frequency</td>
<td>8 MHz</td>
</tr>
<tr>
<td>Cycles available between samples</td>
<td>4000</td>
</tr>
<tr>
<td>Filter execution cycles</td>
<td>831</td>
</tr>
<tr>
<td>% CPU Utilization</td>
<td>21 %</td>
</tr>
<tr>
<td>Code size in bytes</td>
<td>1404</td>
</tr>
</tbody>
</table>
Lab 2: Design a High-pass Filter

• Specifications
  – Signal is a multi-tone signal with the following frequencies:
    • f_1=200Hz, f_2=1200Hz, f_3=1800Hz
    • Remove the low frequency signal 200Hz
    • Stop-band attenuation should be > 50dB, pass band ripple < 0.01dB
  – Things to ponder
    • Choice of sampling frequency?
    • Choice of pass-band and stop-band edge frequencies?
    • Choice of order, do you have filter length restrictions?
      • What order does the tool give you?
      • Does the free version give errors? If so, WHY?
      • What stop band attenuation and pass-band ripple does the tool report? DOES IT MEET YOUR REQUIREMENTS?

Lab 2: Project File Updates

• Follow similar steps from Lab 1
  – Generate coefficients
    • Save the generated coefficients as HPF.txt
  – Use tool FIR_filter_codegen.exe
    • Paste HPF.txt contents to FIR_filter_coeff.dat and use tool
    • FIR_filter.asm is generated for HPF
  – Generate simulated multi-tone using Sine_data_gen.exe
    • sine_data.dat generated
    • Copy and paste into the directory FIR_for_MSP430 overwriting all files
      • The files FIR_filter.asm and sine_data.dat are automatically overwritten for the new filter
  – CCE environment
    • From Run menu click Debug Active Project and hit Advanced Run
    • See results on LCD screen
Lab 2: Results Summary

- Once the filter coefficients are generated follow all the necessary steps to execute your code and see results

![Unfiltered and Filtered Spectrum](image)

**Real-time MSP430 Performance**
- Sampling frequency = 4000Hz
- Filter length = 21
- Example CPU frequency = 4 MHz
- Cycles available between samples = 1000
- Filter execution cycles = 693
- % CPU Utilization = 69%
- Code size in bytes = 1208

Lab 3: Band-pass Filter

- Specifications
  - Signal is a multi-tone signal with the following frequencies:
    - $f_1=200\text{Hz}$, $f_2=1200\text{Hz}$, $f_3=1800\text{Hz}$
    - Remove the frequency signals 200Hz and 1800Hz
    - Stop-band attenuation should be > 50dB, pass band ripple < 0.01dB
  - Choice of sampling frequency? ________
  - Choice of lower band-pass and upper band-pass edge frequencies?
    - What order does the tool give you?
    - What stop band attenuation and pass-band ripple does the tool report? **DOES IT MEET YOUR REQUIREMENTS?**
Lab 3: Project File Updates

- Follow similar steps from Lab 1 and 2
  - Generate coefficients
    - Save the generated coefficients as BPF.txt
  - Use tool FIR_filter_codegen.exe
    - Paste BPF.txt contents to FIR_filter_coeff.dat and use tool
    - FIR_filter.asm is generated for HPF
  - Generate simulated multi-tone using Sine_data_gen.exe
    - sine_data.dat generated
    - Copy and paste into the directory FIR_for_MSP430 overwriting all files
      - The files FIR_filter.asm and sine_data.dat are automatically overwritten for the new filter
- CCE environment
  - From Run menu click Debug Active Project and hit Advanced Run
  - See results on LCD screen

Lab 3: Results Summary

Real-time MSP430 Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampling frequency</td>
<td>4000Hz</td>
</tr>
<tr>
<td>Filter length</td>
<td>32</td>
</tr>
<tr>
<td>Example CPU frequency</td>
<td>20 MHz</td>
</tr>
<tr>
<td>Cycles available between samples</td>
<td>5000</td>
</tr>
<tr>
<td>Filter execution cycles</td>
<td>1016</td>
</tr>
<tr>
<td>% CPU Utilization</td>
<td>20.4 %</td>
</tr>
<tr>
<td>Code size in bytes</td>
<td>1768</td>
</tr>
</tbody>
</table>

ATC 2008
MSP430 Advanced Technical Conference
Lab 4: Design Your Own Digital Filter

• It takes less than 5 minutes, but you have 10 minutes to design your own filter.

START!!

• Did it work?!

Lab 5: Digital IIR Filters, Time Permitting!

• Wave Digital Filters are alternative to conventional IIR filters
• Lattice-type structure
• Tailor-made for fixed-point low-end micro-controllers
• Extremely stable over non-linear operating conditions
• The coefficients have excellent dynamic range
• Little effect from register-width limitations
• Perform as well as the conventional IIR filters
• Lattice structure most widely used
• Tough to design, but works great!

Detailed information on web
Wave Digital Filtering Using the MSP430 (slaa331)
Lab 5A: IIR Low-pass Filter

- Filter specifications
  - Sampling frequency = 16000 Hz
  - Pass-band edge frequency = 3400 Hz
  - Stop-band edge frequency = 4500 Hz
  - Pass-band ripple = 0.5 dB
  - Stop-band attenuation = 50 dB
  - Filter type = LWDF Chebyshev
  - Order = 9
- Demo
  - Close all projects in CCE
  - Right click on the active project and select Delete
  - In the next window select **DO NOT DELETE CONTENTS**
  - From Project menu select **Open Existing Project**
  - Select Root Directory using Browse button to Desktop. Choose lab folder **IIR_filter**
  - Check box only for IIR_for_LPF directory and click Finish
  - Debug Active Project and hit **Advanced Run**

ATC 2008

Lab 5A: Results Summary

- Signal is a multi-tone signal with the following frequencies:
  - \( f_1=5000\text{Hz}, f_2=1600\text{Hz}, f_3=6500\text{Hz} \)

Real-time MSP430 Performance

- Sampling frequency = 16000Hz
- Filter length = 9
- Example CPU frequency = 8 MHz
- Cycles available between samples = 500
- Filter execution cycles = 320
- % CPU Utilization = 64 %
- Code size in bytes = 546

- In file IIR_LPF.asm, change the last add instruction to sub.w R13, &output. **WHAT HAPPENS TO OUTPUT??**
Lab 5B: IIR Band-pass Filter

- Filter specifications
  - Sampling frequency = 8000 Hz
  - Lower stop-band edge frequency = 700 Hz
  - Lower pass-band edge frequency = 950 Hz
  - Lower pass-band ripple = 0.5 dB
  - Lower stop-band attenuation = 50 dB
  - Higher stop-band edge frequency = 1500 Hz
  - Higher pass-band edge frequency = 1850 Hz
  - Higher pass-band ripple = 0.5 dB
  - Higher stop-band attenuation = 50 dB
  - Filter type = LWDF Elliptical
  - Order = 14

- Demo
  - Close all projects in CCE
  - Right click on the active project and select Delete
  - In the next window select DO NOT DELETE CONTENTS
  - From Project menu select Open Existing Project
  - Select Root Directory using Browse button to Desktop. Choose lab folder IIR_filter
  - Check box only for IIR_for_BPF directory and click Finish
  - Debug Active Project and hit Advanced Run

Lab 5B: Results Summary

- Signal is a multi-tone signal with the following frequencies:
  - f_1=500Hz, f_2=1300Hz, f_3=3850Hz

Real-time MSP430 Performance

- Sampling frequency = 8000Hz
- Filter length = 9
- Example CPU frequency = 8 MHz
- Cycles available between samples = 1000
- Filter execution cycles = 501
- % CPU Utilization = 50.1 %
- Code size in bytes = 858
Lab 5C: IIR Notch Filter

- Filter specifications
  - Sampling frequency = 400 Hz
  - Notch frequency = 60 Hz
  - Filter type = IIR
  - Order = 2

- Demo
  - Close all projects in CCE
  - Right click on the active project and select Delete
  - In the next window select DO NOT DELETE CONTENTS
  - From Project menu select Open Existing Project
  - Select Root Directory using Browse button to Desktop. Choose lab folder → IIR_filter
  - Check box only for IIR_for_Notch directory and click Finish
  - Debug Active Project and hit Advanced Run

Lab 5C: Results Summary

- Signal is a multi-tone signal with the following frequencies:
  - \( f_1 = 50 \text{Hz}, f_2 = 60 \text{Hz}, f_3 = 70 \text{Hz} \)

Real-time MSP430 Performance

- Sampling frequency = 400Hz
- Filter length = 2
- Example CPU frequency = 1 MHz
- Cycles available between samples = 2500
- Filter execution cycles = 131
- \% CPU Utilization = 5.24 \%
- Code size in bytes = 260
Quick step summary:
From specifications to design

- Identify the type of filter necessary
  - Spectral analysis of the input signal has it all
  - Low-pass, high-pass, band-pass, band-stop, notch
- What sampling frequency works for you?
  - Application specific → Realistic selection can make all the difference
    - Heart rate, max of 2kHz
    - Speech or voice, max of 16kHz
    - Fancy audio, max of 40kHz
  - MSP430 can do it all
- How good should your filter be?
  - Higher the order, better the performance
  - Choose IIR over FIR, if ultimate performance is needed
  - Set order based on CPU bandwidth available for filtering, approximately 30-35 cycles for each increase in order
- MSP430 takes care of you from here
  - Efficient MSP430 RISC architecture to boost your performance and reduce power consumption
  - The tools available online auto-generates efficient MSP430 code in seconds
  - Horner and CSD – A pair fostering efficient solutions
  - LWDF eliminates the possibility of instability of IIR filters
  - Implementation of all types of filters on the MSP430 show real-time operation possible.
  - Final cost reduced with no external circuitry needed

Conclusion

- Filtering on MSP430
  - Efficient MSP430 RISC architecture to boost your performance and reduce power consumption
  - Software efficiency key to low-cost-low-power solution
  - Extremely simple and efficient with easy steps to final design
  - Code size is large when Horner’s algorithm is used
  - Horner and CSD – A pair fostering efficient solutions
  - Performance close to Floating point implementation
  - LWDF eliminates the possibility of instability of IIR filters
  - Approximately 30-35 cycles with every increase in the order
  - Integer-real multiplication no longer a CPU overhead
  - Implementation of all types of filters on the MSP430 show real-time operation possible.
  - Final cost is reduced with no external circuitry needed
Thank you