VC5505 FFT Filter Demo Release Notes

The VC5505 FFT Filter Demo is a Code Composer v4.0 project written for the C5505 eZdsp USB Stick development tool. This project demonstrates how to use the Tightly-Coupled FFT Hardware Accelerator of the VC5505 DSP to perform real-time audio filtering on input signals either from the AIC3204 codec or from waveforms stored in memory. The input signal is convolved with the impulse response of a low-pass filter with a 2 KHz cutoff frequency through FFT processing. With FFTs we can perform convolution in time through multiplication in the frequency domain. Remember that convolution in the time domain is equivalent to multiplication in the frequency domain.

The FFT Hardware Accelerator (HWAFFT) is a tightly-coupled coprocessor designed to perform up to two consecutive complex FFT stages in one pass. It implements a Radix-2 Decimation-in-Time (DIT) algorithm. Twiddle Factors are stored in a look up table located within the coprocessor and allow for up to 1024-point FFTs. When compared to calculating FFTs on the CPU with FFT routines from DSPLIB, the HWAFFT is 2.2 – 3.8 times faster and 4 – 6 times more energy efficient depending on the FFT_LENGTH. For limited HWAFFT details, see Section 1.3 of the VC5505 DSP System User's Guide.

The high-level view of the demo is now described. 16-bit stereo samples are captured by the AIC3204 codec at a sampling frequency of 48 KHz and copied to the DSP memory over the Inter-IC Sound (I2S) bus with the DMA. Samples are collected in a ping-pong buffer from the codec. When one buffer is full a DMA interrupt updates the ping-pong buffer and triggers the FFT filter to convolve the new block of samples with the low-pass filter. Filtering is performed in three steps. Calculate the FFT of the new block of samples on the HWAFFT, complex multiply the FFT result of the block of input samples with the FFT result of the filter coefficients, and calculate the IFFT of that product on the HWAFFT. The FFT result of the filter coefficients is computed once during program initialization and stored for reuse.

The Constant-Overlap-and-Add (COLA) method is implemented to recombine separately filtered blocks of samples to produce a continuous output to the codec. The IFFT result of FFT filtering contains "ringing" at the beginning and end of the signal. Also the length of the IFFT result it FFT_LENGTH which is larger than the WINDOW_SIZE, the number of samples that the codec is configured to send/receive as a block of samples to/from the filter. The COLA method overlaps the IFFT results in time and adds the overlapping samples from consecutive windows to each other to cancel the ringing and remove discontinuities from the continuous output signal. The amount of samples that overlap from window to window is calculated as (FFT_LENGTH - WINDOW_SIZE).

The resulting block of filtered and overlapped samples is transferred back to the codec for output over the I2S bus with the DMA. To demonstrate contrast, every 5 seconds an interrupt from the Real-Time Clock (RTC) turns the filter on and off allowing the user to hear the difference between filtered and unfiltered samples. The LED on the eZdsp indicates the status of the filter: when the filter is on, the LED is on; when the filter is off, the LED is off.

This demo allows the user to specify the FFT_LENGTH in the range of 8-points to 1024-points by powers of 2. Seven low-pass, windowed-sinc filters of different filter lengths are provided. These filters were created in MATLAB by windowing an ideal sinc filter (Fs = 48KHz, Fc = 2KHz) with a hamming window. The coefficients are quantized into 16-bit fixed-point, S16Q15 format.

In this demo the user specifies the FFT_LENGTH, FILTER_LENGTH, and SOURCE. The WINDOW_SIZE is calculated as (FFT_LENGTH - FILTER_LENGTH + 1). Always choose an FFT_LENGTH that is greater than 2*FILTER_LENGTH. For example, if FILTER_LENGTH = LPF_63_TAP, then FFT_LENGTH should be between FFT_128PTS and FFT_1024PTS.

The SOURCE selection defines the input signal to the FFT Filter. By default, the SOURCE is defined as SIN1K_5K to read a 1KHz + 5KHz sine wave from memory. Four waveforms are selectable at compile time to demonstrate the performance of the filter. Additionally, samples can be obtained from an external sound source, through the AIC3204 stereo codec. To select the codec as the SOURCE, define SOURCE as FROM_CODEC.

The HWAFFT operates on arrays of complex data (real & imaginary). These arrays store 32-bit words that hold real data in the most significant 16-bits and imaginary data in the least significant 16-bits. Complex = [RE, IM]. 16-bit samples arriving from the codec are placed in memory as 32-bit words, where the 16-bit sample resides in the real part (most significant word) and the imaginary part (least significant word) is zeroed. This is enabled by using 32-bit DMA transfers to copy 16-bit samples from the codec without “pack mode” enabled in the I2S peripheral. These 16-bit samples represent a S16Q15 fixed-point number with a range of [-1, .9999].

Before computing the FFT/IFFT on the HWAFFT, the windowed input signal must be extended to length FFT_LENGTH by zero-padding the last (FFT_LENGTH – WINDOW_SIZE) samples. This zero-padded input is then bit-reversed to facilitate an in-place DIT computation. Constraints are placed on the address of the bit-reversed buffer. This buffer must be aligned in RAM such that (2+log2(FFT_LENGTH)) zeros appear in the least significant bits of this address. This buffer’s address is specified in the c5505.cmd file.

The HWAFFT function calls require 4 parameters: a pointer to the 32-bit input buffer, a pointer to a 32-bit scratch buffer, an FFT flag that selects either FFT or IFFT operation, and a scale flag that when set to 1, scales the output of every other butterfly by a factor of ½ to avoid overflow. The return value is a flag that specifies whether the FFT/IFFT result is stored in the input buffer (0) or in the scratch buffer (1).

The HWAFFT function calls are stored in the hwafft.asm file. These C-callable functions are written in assembly and contain coprocessor instructions to communicate with the HWAFFT coprocessor. These functions are also stored in the ROM of the VC5505 DSP at the following addresses:

```
_hwafft_br        = 0x00ff7342
_hwafft_8pts      = 0x00ff7356
_hwafft_16pts     = 0x00ff7445
_hwafft_32pts     = 0x00ff759b
_hwafft_64pts     = 0x00ff78a4
_hwafft_128pts    = 0x00ff7a39
_hwafft_256pts    = 0x00ff7c4a
_hwafft_512pts    = 0x00ff7e48
_hwafft_1024pts = 0x00ff80c2
```

To call the hwafft functions from ROM instead of from RAM, uncomment the hwafft lines at the bottom of the c5505.cmd file and exclude hwafft.asm from the build (Right click on hwafft.asm -> Exclude File(s) from Build). Now when the project is rebuilt, the hwafft functions will not be located RAM and the function calls will reference the ROM instead. Based on the FFT_LENGTH defined in configuration.h, a function pointer named *Hwafft_Func points to the appropriate hwafft function for that FFT_LENGTH. (E.g. if FFT_LENGTH == FFT_1024PTS, then *Hwafft_Func points to &hwafft_1024pts).

DESCRIPTION OF IMPORTANT FILES:

main.c – device initialization, foreground while loop that calls FFT Filter and COLA after DMA ISR
configuration.h – #defines for FFT_LENGTH, filter selection, source selection, and debug mode
configuration.c – Hwafft_Func function pointer, Array declarations, filter coefficients, sine waveforms
codec_routines.asm – configures AIC3204 codec
dma_routines.c/h – Setup DMA to transfer between DSP and codec, DMA ISR, Ping-Pong Buffering
filter_routines.c/h – FFT Filter routines – ZeroPad, FFT Coeffs, FFT_Filter, CPLX_Mul, COLA
hwafft.asm/h – C-callable assembly functions that talk to HWAFFT coprocessor, Actual FFT routines
i2s_routines.c/h – configures I2S peripheral
rtc_routines.c/h – configures RTC peripheral, RTC ISR (interrupts every 5 seconds)
c5505.cmd – Specifies array to memory mapping, function pointers to hwafft functions stored in ROM

BUILD INSTRUCTIONS:

Instructions to import, build, and run the demo are now described.

1) Unzip the CCS4 project and save it to your host PC.
2) Open Code Composer 4.0 and select your workspace.
3) Open the Import Project window (File -> Import…).
4) Select “Existing CCS/CCE Eclipse Project”, click Next, browse to the directory where the project is saved, make sure the project titled “VC5505 FFT Filter Demo” is selected, click Finish.
5) Build the project (Project -> Build Active Project).
6) Ignore warnings about entry-point symbol other than “_c_int00” and DSP/BIOS versions
7) Connect to the C5505 eZdsp USB Stick (Target -> Debug Active Project).
8) Load the object file to the device: Open the Load Program(Target -> Load Program), browse to <project directory>\project\Debug\VC5505_FFT_Filter_Demo.out”, click Open, then OK.
9) Once the project is loaded to the device, Run the project (Target -> Run)

The default configuration for the project (as specified in configuration.h) specifies a 1024-point FFT, a 511-tap Low-Pass Filter (Fc = 2KHz, Fs = 48KHz), and the SOURCE set to read a 1KHz+5KHz Sine wave from memory(SIN1K_5K). To test the project, plug-in speakers to the HP_OUT jack and run the program. When the LED is off, the unfiltered input signal is heard. It contains both a 1KHz tone and a 5KHz tone. When the LED is on, the signal is sent through the FFT Filter and the 5KHz tone is filtered by the 2KHz-cutoff low-pass filter.