**Lab 06 Frame Based DSP Tutorial**

So far we have implemented the DSP algorithms by measuring an analog signal at a constant sample frequency, doing something with that sample and then putting out the result of the signal processing. This was done 1 sample at a time. Next we will investigate doing similar processing except with a contiguous set or frame of data.

First what is a frame? From the book it is a group of consecutive samples. The time between the samples is 1 sample period.

An additional memory managing concept called direct memory access will also be explored in this lab. Direct Memory Access in the digital signal processor is implemented with dedicated hardware. This means that for all practical purposes the CPU won’t have to spend any precious clock cycles moving data from the A/D converter or to the D/A converter. TI calls their version EDMA where E is for enhanced. DMA or EDMA is complicated and requires a lot of setup. The code for EDMA has already been setup for you.

There are some functions in DSP that require frames of data to work at all. One example of this is the FFT or Fast Fourier Transform. Other times we can perform functions that could be done on a sample by sample basis or in a frame of data basis. The various reasons, benefits and drawbacks of frames and sample by sample are well covered in the book section 6.1 so review that now. Frames in WinDSK8 are demonstrated in 6.2. MATLAB examples in 6.3.

Spend real time reviewing the first subsections of 6.4. Here some of the practical problems of implementing frames for certain types of DSP are explained. Also you must understand triple buffering to understand and implement frame buffering. This is only 1 page in the book so I recommend you understand this to make code implementation much easier.

**1. First Assignment**

You will make a program that implements block averaging. The block filtering will use frames of data. The frame size should be 1024. You will use triple buffering for the frames. The block average is done on a 64 sample block. For each block of 64 input samples you should take the average and save this average to the same time correlated 64 output sample. In other words average x(0) to x(63) and save the average value to y(0) to y(63). The next block is x(64) to x(127) and so on until all 1024 data points are used.

Most of the code in section 6.4.2 of the book is usable for the first assignment. It is available in the code/chapter\_06/ccs/Frame directory from the supplied code. IMPORTANT Note: Use vectors.asm for this version of the code. You have always used it before by including it in the project.

**What you need to do:**

(1) set the buffer length as 64

(2) write a function of the block average

(3) implement the function to the ProcessBuffer

Note: There are several samples of function listed in the code (In ProcessBuffer).

**2. Second Assignment:**

**Implement the same function using EDMA.**

Most of the code in section 6.4.3 of the book is usable for the first assignment. It is available in the code/chapter\_06/ccs/Frame\_EDMA\_6748 directory from the supplied code. IMPORTANT Note: Use vectors\_EDMA.asm for this version of the code. You have always used vectors.asm before. This time make sure you don’t use vectors.asm.

So did the EDMA improve performance? I guess the real question is would you know that? Lucky for you CCS has a built in facility for determining just this sort of information. The most important thing for this section is to learn how EDMA works. The EDMA structure has been settled, you don’t need to modify that part.

<http://processors.wiki.ti.com/index.php/Profile_clock_in_CCS>

**3. Assignments that you have to submit:**

**(1) The whole projects for these 2 assignments**

**(2) A txt/word/pdf document that describes how EDMA improve the performance, and what is the difference you hear from these two methods.**