# IMPLEMENTATION AND ANALYSIS OF FIR FILTER USING TMS 320C6713 DSK

Sandeep Kumar
ECE Deptt.
HCTM Kaithal

Munish Verma
ECE Deptt.
HCTM Kaithal

Vijay K.Lamba
ECE Deptt.
HCTM Kaithal

Susheel Kumar
ECE Deptt.
HCTM Kaithal

Avinash Kumar
ECE Deptt.
HCTM Kaithal

## ABSTRACT

In most of the applications, analog signals are produced in response to some physical phenomenon or activity. But it is quite difficult to process that analog signal; here comes the need to convert an analog signal to a digital signal. For this purpose specific digital signal processors (DSP's) are developed. TMS 320C6713 is one of such type of processors that can be used to process or handle the signals in a variety of ways. In the current report, basically the architecture of this processor is studied. Along with the processor architecture, the hardware portion DSK (Digital Starter Kit) and the software portion CCS (Code Composer Studio) is also studied. Digital filters are very commonly found in everyday life and include a variety of applications. Mainly they are used for two major purposes: signal separation and signal restoration. Signal separation is needed when a signal has been contaminated with interference, noise, or other signals. Signal restoration is used when a signal has been distorted in some way. So, various programs have been analyzed in this work to implement efficiently those FIR filter structures on TMS 320C6713 DSK. Characteristics of FIR filters are studied in frequency domain.

**Keywords:** FIR Filter, DSP, DSK, CCS.

## 1. INTRODUCTION

Mostly sensors generate analog signals in response to various phenomena. Signal processing can be carried out either in analog or digital domain. To do processing of analog signals in digital domain, first digital signal is obtained by sampling and followed by quantization (digitization). The digitization can be obtained by analog to digital converter (ADC). The role of digital signal processor (DSP) is the manipulation of digital signals so as to extract desired information. In order to interface DSP with analog world, digital to analog converters (DAC) are used. Figure 1 shows basic components of a DSP system [1].

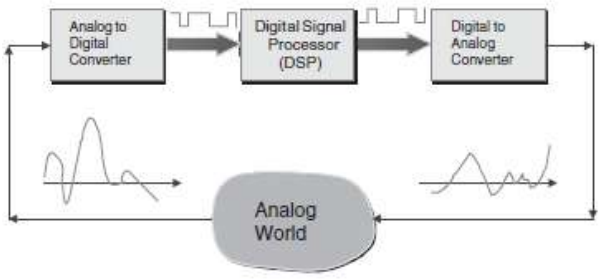


**Fig 1 Basic components of a DSP system**

ADC captures and inputs the signal. The resulting digital representation of the input signal is processed by DSP such as C6x and then output through DAC. Within in the basic DSP system, anti aliasing filter at input to remove erroneous signals and output filter to smooth the processed data is also used [2].

There are various reasons to process the analog signals in the digital domain: The same DSP hardware can be used for various applications by just changing the code. Digital circuits are more stable and tolerant than analog circuits. Many filters and adaptive systems are realizable only by the digital manipulation of signals. Digital signal processing can be carried out on various platforms such as customized very large scale integrated (VLSI) circuits and DSP. A comparative review of both the platforms is as follows:

• DSPs are programmable allowing fair amount of application flexibility which not the case with hardwired digital circuits.

• DSPs are cost effective due to mass production and can be used for various applications whereas VLSI chip is normally built for a signal application.

• Often quite high sampling rates can be obtained by customized chips where in DSP sampling rates are limited due to architecture design and peripheral constraints [1].

Large market shares of DSPs belong to cost-effective real time embedded systems such as cell phones and modems. Real time requires keeping processing pace with some external event [2] or in other words completing the processing within the available time between samples which of course depends upon application. Real time processing depends upon two aspects a) sampling rate b) system latencies (delays) [1].

In the current report DSP processor family TMS320C6X architecture, DSK and various programs implementing FIR filter using Code Composer Studio is studied and analyzed.

## 2. TMS 320C6X (C6X) FAMILY

Digital signal processors such as the TMS320C6x (C6x) family of processors are like fast special-purpose microprocessors with a specialized type of architecture and an instruction set appropriate for signal processing. The C6x notation is used to designate a member of Texas Instruments' (TI) TMS320C6000 family of digital signal processors. Based on a very-long-instruction-word (VLIW) architecture, the C6x is considered to be TI's most powerful processor.

Texas Instruments introduced the first - generation TMS32010 DSP in 1982, the TMS320C25 in 1986 [4], and the TMS320C50 in 1991. Several versions of each of these processors — C1x, C2x, and C5x — are available with different features, such as faster execution speed. These 16 - bit processors are all fixed - point processors and are code compatible [5].

The TMS320C30 floating - point processor was introduced in the late 1980s. The C31, the C32, and the more recent C33 are all members of the C3x family of floating - point processors [6, 7]. The C4x floating - point processors, introduced subsequently, are code compatible with the C3x processors and are based on the modified Harvard architecture [8].

The TMS320C6201 (C62x), announced in 1997, is the first member of the C6x family of fixed - point digital signal processors. Unlike the previous fixed - point processors, C1x, C2x, and C5x, the C62x is based on a VLIW architecture, still using separate memory spaces for instructions and data, as with the Harvard architecture. The VLIW architecture has simpler instructions, but more are needed for a task than with a conventional DSP architecture.

Generally, a fixed - point processor is better for devices that use batteries, such as cellular phones, since it uses less power than does an equivalent floating - point processor. It is necessary to scale the data. And a floating - point processor is generally more expensive since it has more "real estate" or is a larger chip because of additional circuitry necessary to handle integers well as floating - point arithmetic. Several factors, such as cost, power consumption, and speed, come into play when choosing a specific DSP. The C6x processors are particularly useful for applications requiring intensive computations [9]. So we choose to analyze TMS320C6713 Digital Signal Processor which is floating point and good for real time applications.

## 2.1 TMS 320C6713

The TMS320C6713 Digital Signal Processor is the floating-point processor. The main application of this processor is the real time processing of digital signals. It is used to implement FIR/IIR filters.

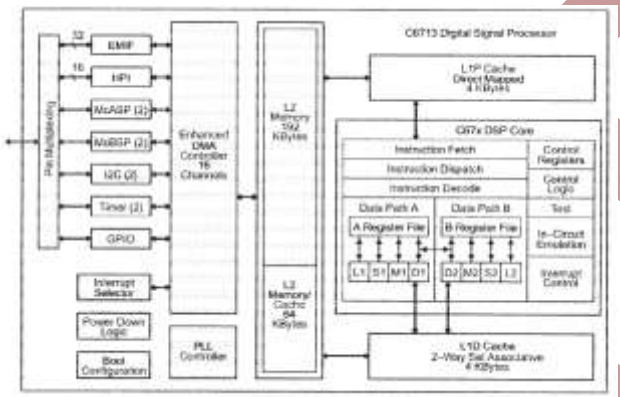


**Fig 2 Functional Block Diagram of TMS320C6713**

The TMS320C6713 onboard the DSK is a floating - point processor based on the VLIW architecture [10-12]. Internal memory includes a two - level cache architecture with 4 kB of level 1 program cache (L1P), 4 kB of level 1 data cache (L1D), and 256 kB of level 2 memory shared between program and data space. It has glue less (direct) interface to both synchronous memories (SDRAM and SBSRAM) and asynchronous memories (SRAM and EPROM). Synchronous memory requires clocking but provides a compromise between static SRAM and dynamic DRAM, with SRAM being faster but more expensive than DRAM.

On - chip peripherals include two McBSPs, two timers, a host port interface (HPI), and a 32 - bit EMIF. It requires 3.3 V for I/O and 1.26 V for the core (internal). Internal buses include a 32 - bit program address bus, a 256 - bit program data bus to accommodate eight 32 - Bit instructions, two 32 - bit data address buses, two 64 - bit data buses and two 64 - bit store data buses. With a 32 - bit address bus, the total memory space is $2^{32}$ = 4 GB, including four external memory spaces: CE0, CE1, CE2, and CE3.

Independent memory banks on the C6x allow for two memory accesses within one instruction cycle. Two independent memory banks can be accessed using two independent buses. Since internal memory is organized into memory banks, two loads or two stores of instructions can be performed in parallel. Separate buses for program, data, and direct memory access (DMA) allow the C6x to perform concurrent program fetches, data read and write, and DMA operations. The C6x has a byte - addressable memory space. Internal memory is organized as separate program and data memory spaces, with two 32 - bit internal ports to access internal memory [5].

Now, to implement a real time application with the above described processor, we require the TMS3206713DSK (Digital Starter Kit); that includes the TMS320C6713 Digital Signal Processor. For simulation purpose, this kit can be connected to PC with the help of software known as Code Composer Studio.

## 2.2 TMS 320C6713 DSK

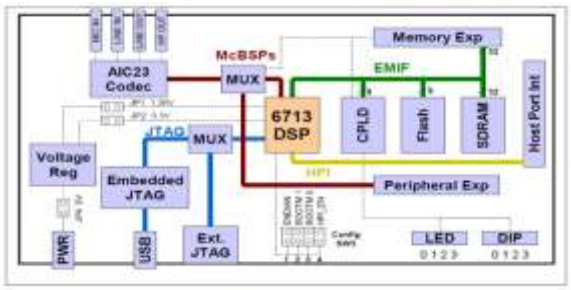


**Fig 3 Block Diagram of TMS320C6713 DSK**

The DSK comes with a full complement of on-board devices that suit a wide variety of application environments. Key features include: Texas Instruments TMS320C6713 DSP operating at 225 MHz, An AIC23 stereo codec, 16 Mbytes of synchronous DRAM, 512 Kbytes of non-volatile Flash memory, 4 user accessible LEDs and DIP switches, Software board configuration through registers implemented in CPLD, Configurable boot options, Standard expansion connectors for daughter card use, JTAG emulation through on-board JTAG emulator with USB host, Single voltage power supply (+5V) [4].

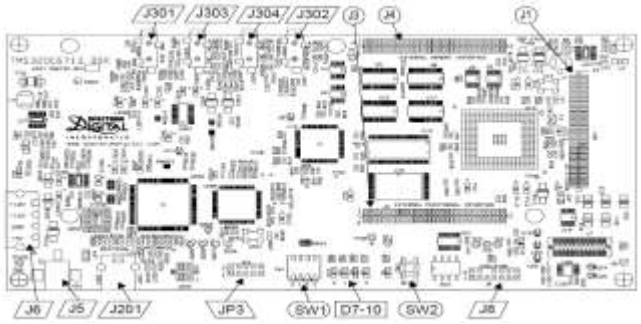


**Fig4 Board diagram of TMS320C6713 DSK**

The TMS320C6713DSK is connected to PC with the help of USB cable and the **Code Composer Studio** software interface the PC with the C6713DSK.

## 2.3 CODE COMPOSER STUDIO

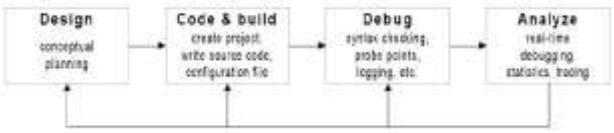


**Fig 5 Code Composer Studio IDE Development Flow**

The CCS provides an integrated development environment (IDE) as in fig 5 to incorporate the software tools. CCS includes tools for code generation, such as a C compiler, an assembler, and a linker. Once the generated machine code is

loaded and run on the target, the IDE also offers some analysis tools with graphical capabilities to visualize processes running on the DSPs. CCS extends the basic code generation tools with a set of debugging and real time-analysis capabilities.

CCS works with a project paradigm. Essentially, within CCS it is necessary to create a project for each executable program that is to be created. A project stores all the basic information to build the executable file ("project". out).

CCS provides an IDE to incorporate the software tools. CCS includes tools for code generation, such as a C compiler, an assembler, and a linker. It has graphical capabilities and supports real-time debugging. It provides an easy-to-use software tool to build and debug programs. The C compiler compiles a C source program with extension .c to produce an assembly source file with extension *.asm*. The assembler assembles an *.asm* source file to produce a machine language object file with extension *.obj*. The linker combines object files and object libraries as input to produce an executable file with extension *.out*. This executable file represents a linked common object file format (COFF), popular in Unix-based systems and adopted by several makers of digital signal processors. This executable file can be loaded and run directly on the C6713 processor. A linear optimizer optimizes this source file to create an assembly file with extension .asm.

Real-time analysis can be performed using real-time data exchange (RTDX). RTDX allows for data exchange between the host PC and the target DSK, as well as analysis in real time without stopping the target. Key statistics and performance can be monitored in real time. Through the joint team action group (JTAG), communication with on-chip emulation support occurs to control and monitor program execution.

## 3. RESULTS AND DISCUSSIONS

### 3.1 MOVING AVERAGE FILTER

The moving average filter is widely used in DSP, mainly because it is the easiest digital filter to understand and use. In spite of its simplicity, the moving average filter is optimal for a common task: reducing random noise while retaining a sharp step response. This makes it the premier filter for time domain encoded signals. As the name implies, the moving average filter operates by averaging a number of points from the input signal to produce each point in the output signal. In equation form, this is written:

$$y[i] = \frac{1}{M}\sum_{j=0}^{m-1} x[i+j] \qquad (1)$$

Where *x* [n ] is the input signal, *y* [ n] is the output signal, and M is the number of points in the average. For example, in a 5 point moving average filter, point 80 in the output signal is given by:

$$y\,[80] = \frac{x[81]+x[82]+x[83]+x[84]+x[85]}{5} \qquad (2)$$

As an alternative, the group of points from the input signal can be chosen symmetrically around the output point:

$$y\,[80] = \frac{x[78]+x[79]+x[80]+x[81]+x[82]}{5} \qquad (3)$$

This corresponds to changing the summation in Eq. 2 from: j= 0 to M-1, to: j= - (M-1) /2 to (M-1) /2. For instance, in an 11 point moving average filter, the index, j, can run from 0 to 11 (one side averaging) or -5 to 5 (symmetrical averaging). Symmetrical averaging requires that M be an odd number. Programming is slightly easier with the points on only one side; however, this produces a relative shift between the input and output signals. We can easily recognize that the moving average filter is a convolution using a very simple filter kernel. For example, a 5 point filter has the filter kernel: ….0, 0, 1/5, 1/5, 1/5, 1/5, 1/5, 0, 0…. . That is, the moving average filter is a convolution of the input signal with a rectangular pulse having an area of one.

As far as implementation is concerned, at the *n*th sampling instant we could either:

**1.** Multiply *N* past input samples individually by 1 */N* and sum the *N* products,

**2. S**um *N* past input samples and multiply the sum by 1 */N*, or

**3.** Maintain a moving average by adding a new input sample (multiplied by 1/ *N*) to and subtracting the $(n - N + 1)$th input sample (multiplied by 1/ *N* ) from a running total.

In this report, we used first option, even though it is not the most computationally efficient. The value of *N* defined near the start of the source file determines the number of previous input samples to be averaged. Source file average.c is stored in folder average, which also contains project file average.pjt. Then we build the project as **average** and run the program.

Several different methods exist by which the characteristics of the five point moving average filter may be demonstrated. A test file mefsin.wav, stored in folder average, was containing a recording of speech corrupted by the addition of a sinusoidal tone. We can listen to this file using Gold-Wave, Windows Media Player, or similar. Then, we connected the PC soundcard output to the LINE IN socket on the DSK and listen to the filtered test signal (LINE OUT or HEADPHONE). We found that the sinusoidal tone has been blocked and that the voice sounds muffled.

To analyze the frequency response of the filter, we use a signal generator and an oscilloscope to measure its gain at different individual frequencies. We identified the distinct notches in the magnitude frequency response at 1600 Hz and at 3200 Hz. The magnitude frequency response of the filter is illustrated in Figure 6.

### 3.2 MOVING AVERAGE FILTER WITH INTERNALLY GENERATED PSEUDORANDOM NOISE AS INPUT

In another example, we tried to assess the magnitude frequency response of a filter by using wideband noise as an input signal. The rest of the procedure remains the same for this example also. A pseudorandom binary sequence (PRBS) is generated within the program and used as an input to the filter in lieu of samples read from the ADC. The filtered noise is analyzed on a spectrum analyzer and whereas the frequency content of the PRBS input is uniform across all frequencies, the frequency content of the filtered noise will reflect the frequency response of the filter. Goldwave player can also be used as another option for a dedicated spectrum analyzer. Figure 7 shows the output captured using the oscilloscope and figure 8 using Goldwave. One can easily compare the three figures 6, 7 & 8.

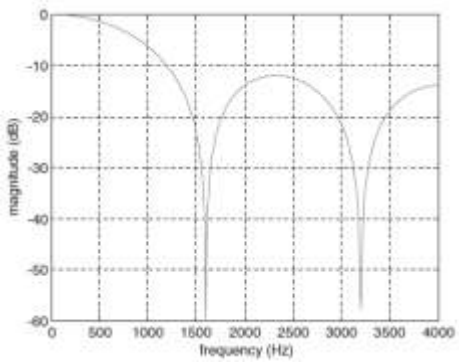


**Fig. 6 Magnitude frequency response of five point moving average filter**

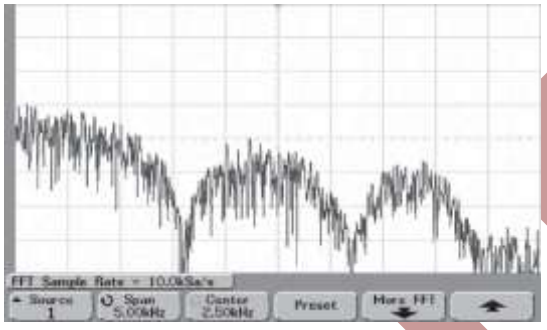

**Fig 7 Magnitude frequency response of five point moving average filter on oscilloscope**

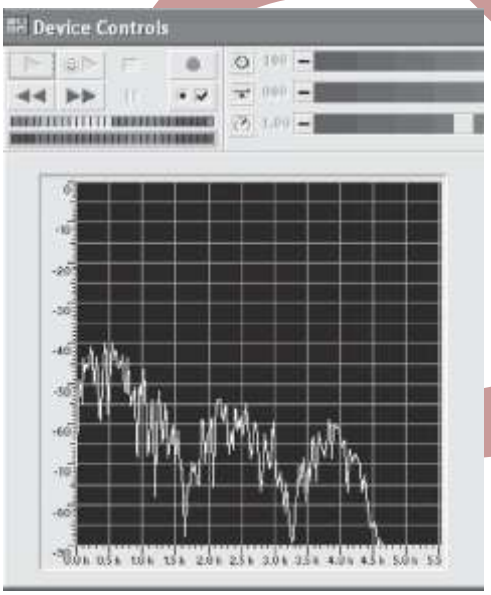


**Fig 8 Magnitude frequency response of five point moving average filter using Goldwave**

## 3.3 IDENTIFICATION OF MOVING AVERAGE FILTER FREQUENCY RESPONSE USING A SECOND DSK

In this program, we tried to analyze the characteristics of the moving average filter. For this, we used two DSKs connected as shown in Figure 9. On these two DSKs, we run two different programs. The program identifies the characteristics of the system connected between points A and B in figure 9, including the codec DAC between point A and the LINE OUT socket and the codec ADC between the LINE IN socket and point B. In broad terms, it identifies the system connected between LINE OUT and LINE IN sockets.

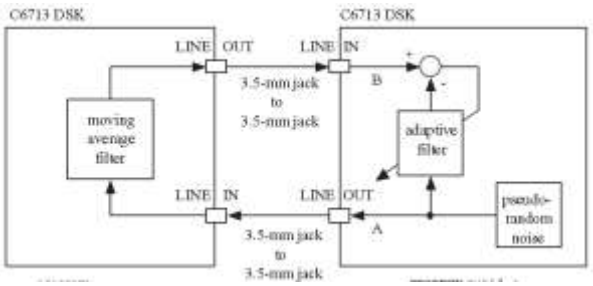


**Fig. 9 Connection diagram to identify characteristics of the moving average filter**

Figure 10 shows the graph exported from Code Composer as a text file and imported to MATLAB; plotted on the same axes as the magnitude frequency response of the five point moving average filter. The discrepancy between figure 6 & 10 at frequencies greater than 3.5 kHz is due to the characteristics of the anti aliasing and reconstruction filters in the AIC23 codec.

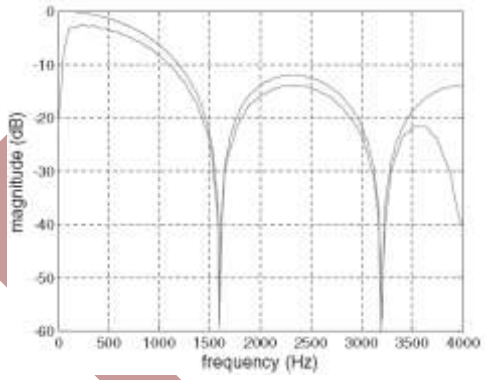


**Fig 10 Magnitude frequency response of five point moving average filter using two DSK**

## 3.4 FIR FILTER WITH MOVING AVERAGE, BANDSTOP, AND BANDPASS CHARACTERISTICS

Next we analyzed different filter structures using the different procedures. Coefficient file ave5f.cof is generated. Using that file, program implements the same five point moving average filter implemented by figure 6. The number of filter coefficients is specified by the value of the constant *N* and the coefficients are specified as the initial values in an *N* element array, h, of type float. We build the project as **fir,** run the program and verify that it implements a five point moving average filter.

To implement a Bandstop filter which is centered around 2700 Hz we changed the coefficient file again. Build and run this project as **fir**. Input a sinusoidal signal and vary the input frequency slightly below and above 2700 Hz and verified that the output is a minimum at 2700 Hz. The values of the coefficients for this filter were calculated using MATLAB's filter design and analysis tool, FDA tool.

Same way to design a Band Pass filter centered at 1750 hz, we again changed the coefficient file which is again generated using MATLAB. Again, the output is verified.

***Generating Filter Coefficient (.cof) Files Using MATLAB***

If the number of filter coefficients is small, a coefficient (.cof) file can be edited by hand. For larger numbers of coefficients the MATLAB function dsk_fir67 () can be used. This function

expects to be passed a MATLAB vector of coefficient values and prompts the user for an output filename.

## 4. CONCLUSIONS

This present report provides a unique and dynamic environment for sound engineers to experiment in without having to physically construct any filters. A PC based GUI provides intricate customization, circumventing the need to trawl through oceans of manuals when programming an effects system manually. The presence of a built-in *codec* on the C6713 kit allows much more flexibility. This work incorporated a vast range of software, hardware, digital signal processing and embedded systems and this is what made this work an exciting and challenging venture.

We have analyzed codes for real time implementation of FIR/Moving Average Filter. We have analyzed the characteristics of designed filter: Effect of internally generated Pseudorandom Noise; FIR Filter with Moving Average, Band stop and Band pass characteristics; Effects on voice or music using three FIR Low pass Filters; Implementation of four different FIR filters: Low pass, High pass, Band pass, and Band stop.

## 5. REFERENCES

[1] Steven W. Smith 1999. The Scientist and Engineer's Guide to Digital Signal processing.

[2] Gene Frantz 2000. Digital Signal Processor Trends. IEEE.

[3] Berkeley Design Technology 2006. The Evolution of DSP Processors.

[4] R. Chassaing and D. W. Horning 1990. Digital Signal Processing with the TMS320C25.

[5] R. Chassaing 2008.Digital Signal Processing and Applications with the C6713 and C6416 DSK.

[6] R. Chassaing 1999. Digital Signal Processing Laboratory Experiments Using C and the TMS320C31 DSK.

[7] R. Chassaing 1992. Digital Signal Processing with C and the TMS320C30.

[8] R. Chassaing and P. Martin 1995. Parallel processing with the TMS320C40. ASEE.

[9] R. Chassaing and R. Ayers 1996. Digital signal processing with the SHARC. ASEE.

[10] SPRU189F 2000. TMS320C6000 CPU and Instruction Set. Texas Instruments.

[11] SPRU190D 2001. TMS320C6000 Peripherals. Texas Instruments.

[12] SPRU198G 2002. TMS320C6000 Programmer's Guide. Texas Instruments.

[13] Dallas. TX 1999. TMS320C62X/C67X, Programmers' Guide. Texas Instruments.

[14] SPRU301C 2000. TMS320C6000 Code Composer Studio Tutorials. Texas Instruments.

[15] Dallas. TX 2005. Code Composer Studio IDE Getting Started Guide. Texas Instruments.

[16] T. W. Parks and J. H. McClellan 1972. Chebychev approximation for nonrecursive digital filter with linear phase. IEEE.

[17] J. H. McClellan and T. W. Parks 1973. A unified approach to the design of optimum linear phase digital filters. IEEE.