

Audio Processing of Guitar Signals (Real Time Concepts) Using TI DSK TMS320C6713 and DSP Simulink Blocksets

S K Hasnain, Ameer Ishrat Saleem & Aresh Dinshaw Daruwalla
Pakistan Navy Engineering College (NUST), NED University of Engineering & Technology
hasnain@p nec.edu.pk, ameer.ishrat@gmail.com, aresh_d5@yahoo.com

Abstract - This paper focuses on the development of various audio effects on a digital system for guitarists, employed for performances in real time. This would help the guitar player to easily configure/capture various audio effects in advance from a PC based graphical user interface, which is far better than conventional systems that use cumbersome dials, buttons and switches. This new approach would also allow professional users for fine-tuning of specific parameters, which is inaccessible in traditional multi-effects units. The TI C6713 DSK provides adjustment and selection of effects from a built database. The processor used here implemented our designs on an input audio signal; this broadens the versatility of our system by allowing the musician to use the processor for any musical instrument. The paper proceeds with the real-time concepts of controlling the various audio effects via on-board DIP switches on the C6713 DSK. Controlling of Simulink model parameters was done using a MATLAB created GUI. The GUI enables natural and flexible reconfiguration of effects, and storage of set “patches” which can be recalled at the push of a DIP switch. A great deal of research is also being carried out on getting our design to function in the standalone condition.

Keywords: Guitar signals processing, DSP Blocksets, TI TMS320C6713 kit, Model based design, Simulink model, Graphical user interface, standalone DSP TMS320C6713.

I. INTRODUCTION

The DSP courses provided at universities today are theoretical and highly mathematical for students. As important as these profound mathematical foundations are, a hands-on real time experience using hardware is essential for the basic understanding of DSP. The work we adopted here helps achieve its application on the concept of real time audio processing of a guitar, which, serves as an example to our research. In this paper, a number of effects-laden audio signals inputted (generated via an electric guitar) were seen to produce familiar patterns within a particular audio signal after different effects had been implemented on it. Applying these effects directly to an audio signal would have lead to

designing different types of physical filters and also having to tackle the problem of a high Signal-to-Noise ratio which is much lower in the case of digital audio signals. Keeping this in mind, a digital signal processor, such as the C6713 DSK used here, proved appropriate [1]. Each effect is based on the differing values of the same set of parameters. This set of variable parameters with boundaries is established for each effect to ensure straightforward manipulation for users, while allowing professional users advanced algorithm configuration using the blocks provided in Simulink. The C6713 DSK is programmed using the Code Composer Studio Software made available by Texas Instruments. Rather than creating our own algorithms in embedded C. Simulink models of different effects were interfaced with Code Composer Studio and were executed on the C6713 DSK in real-time [1], [2].

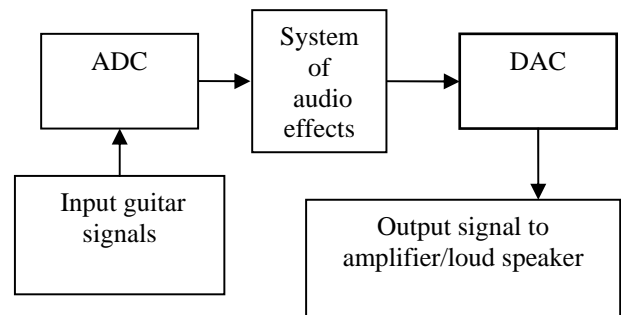


Fig. 1. Typical sequence for a Multi-effects Processor

II. SIMULATION AND UNDERSTANDING OF GUITAR AUDIO SIGNALS USING MATLAB

A “guitar effect” is an analogue or digital circuit that audibly modifies the input guitar audio signal to produce a desired output. There are countless flavors of guitar effects, most of which may be broken down into digital filters, suitable for DSP implementation. Effects can generally be classified into three categories.

- Effects that directly alter the amplitude of the input signal.
- Effects that hold delays in memory.
- Effects that perform frequency filtering.

Guitar effects algorithms were simulated using MATLAB. Samples for effects were recorded using a Jackson guitar and a *ZOOM FX processor*, at 44100Hz, 16 bit wav file format. To observe the signal graphically, the *wavread* command in MATLAB was used. This was done by first adding our sample wav format file in MATLAB's work directory, followed by typing the following command in MATLAB's command window $[x,fs,Nbits]=wavread('filename.wav')$. This gives the vector values of the sound file. To see the plot of our input sound file we type $plot(x)$ to obtain the graphs shown in Figure 2 & 3 [3].

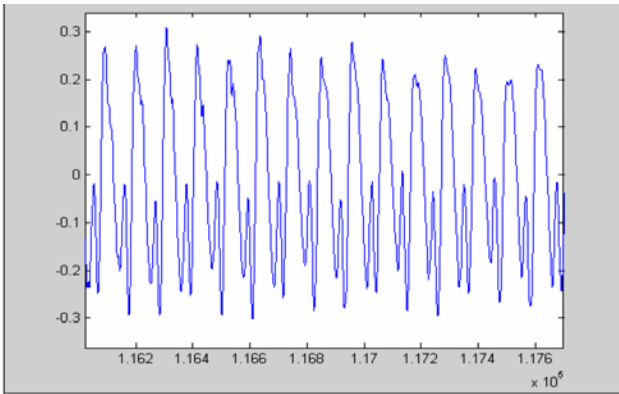


Fig. 2. Clean guitar sound visualized on MATLAB

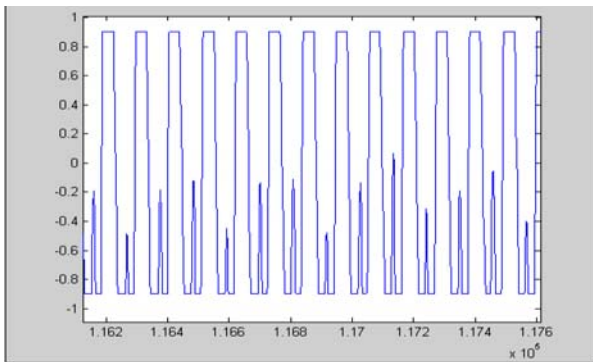


Fig. 3. Distorted guitar sound visualized on MATLAB

The adjustable parameters for this effect are:

- The clipping threshold.
- The amplification co-efficient.

If different thresholds are used for positive and negative clipping then the effect is asymmetrical clipping. Through in experimentation, no major audible differences were noted between the two techniques.

The Delay effect is created by adding delayed, diminished samples to the incoming signal. The following parameters were established to vary the effect:

- Delay period.
- Amplitude of first delay
- Diminishing rate - this determines the amplitude of the next delay as a function of the current sample
- No of delays – this is an optional constraint. Generally this would be determined by the diminishing rate reducing the delay amplitude to zero. However this parameter becomes important when embedding this effect [3],[5].

Varying these parameters can produce a number of different effects. For example, one set of parameter boundaries can produce a traditional *delay* effect as shown in Figure 4; another will produce an *echo* effect as shown in Table 1.

TABLE 1:
Traditional delay effect in different models.

Delay Range (ms)	Modulation	Effect Name
0-20	-	Resonator
0-15	Sinusoidal	Flanging
10-25	Random	Chorus
25-50	-	Slapback
> 50	-	Echo

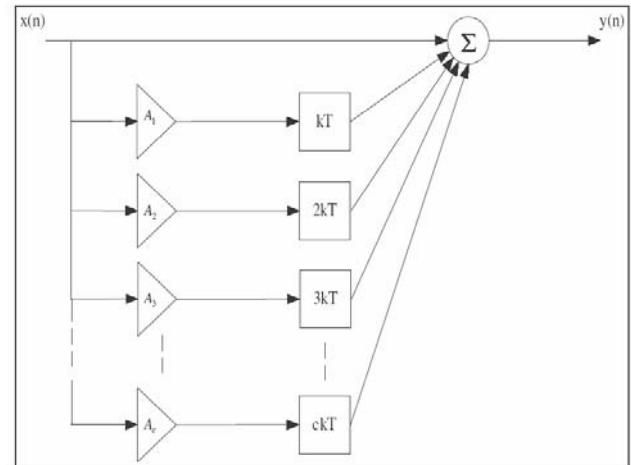


Fig. 4. Block Diagram of Delay Effect

$$y(n) = x(n) + A_1x(n-k) + A_2x(n-2k) + \dots + A_cx(n-ck)$$

Where: c is the number of delays.

k is the number of samples between delays.

This is a function of the delay period t and the sampling frequency f . For a delay period of 20ms and a sampling frequency of 48 kHz

$$t = 20 * 10^{-3}$$

$$f = 48000$$

$$k = t * f = (20 * 10^{-3}) * 48000 = 960 \text{ samples between each delay}$$

If k were not a whole number it could be rounded to the nearest sample, or the value could be interpolated from the surrounding values.

III. MODEL-BASED DESIGN

In Model-Based Design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing. The model is an executable specification that is continually refined throughout the development process not discussed in detail here for brevity [4]-[5].

A. Building the Simulink Block Diagram

After defining each subsystem mathematically, we undertook the following steps for building a block diagram model in Simulink. Initially block diagrams for each of the subcomponents were built separately. After each subcomponent was modeled individually, they were integrated into the final model of the system. After building the Simulink block diagram, the model was simulated and results were analyzed first individually and later also after integrating all of the models together. Finally, the model was validated so that it accurately represented the physical characteristics of the system. The blockset of our greatest concern was that of TI6000 which allows interfacing of Simulink with the DSP board, the TMS320C6713 DSK.

B. Signal & Parameter Attributes

Signal and parameter attributes can be specified directly in the diagram or in a separate data dictionary. Using the Model Explorer, the user can manage his own data dictionary and quickly repurpose a model by incorporating different data sets. Some of the signal and parameter attributes which can be defined are:

- Data type—single, double, signed or unsigned 8-, 16- or 32-bit integers; Boolean; and fixed-point
- Minimum and maximum range, initial value, and engineering units

C. Generating C/C++ Code

Models that are built in Simulink can be configured and made ready for code generation. Using Real-Time Workshop and Real-Time Workshop Embedded Coder™ products (both available separately), one can generate C/C++ code from the model for real-time simulation, rapid prototyping, and embedded system deployment.

IV. SIMULINK MODELS

Before we proceed to how we implemented our design, we will first discuss our individual audio effects models and

their final integration to produce an effects laden sound controlled by DIP Switches.

A. Audio Distortion

As discussed before, the effect of distortion is obtained by clipping the audio signal to a fixed upper and lower limit and then increasing the gain of the signal to a certain level depending on which distortion type we are setting it to i.e. Fuzz or Overdrive. The model design made is quite self-explanatory as shown in Figure 5.

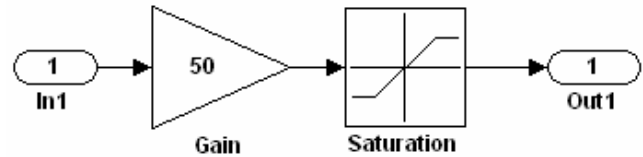


Fig. 5. Audio Distortion Effect

The Gain block is used to set the gain parameter while the saturation block sets the upper and lower limits.

B. Chorus Effect

The chorus effect as shown in Figure 6 allows a single instrument to be modeled into a sound that replicates a group of instruments playing the same part. This is achieved by adding a single delayed signal (echo) to the original input. However, the delay of this echo is varied continuously between a minimum delay and maximum delay at a certain rate. The chorus effect is similar to the flange effect, except the low frequency oscillator produces low frequency random noise rather than a sine wave. The other differences include a much longer delay length, and numerous delays used instead of the singular delay used in the flange effect. Shown below is a single chorus system that is complemented with our Line In audio input. In our final model we have coupled four such systems to get a notable effect.

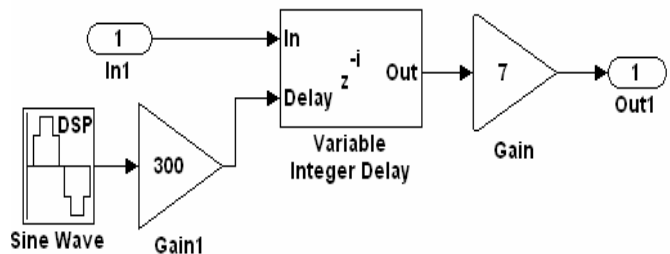


Fig. 6. Audio Chorus Effect System

C. Tremolo

In technical terms, tremolo is the *amplitude modulation* of the signal. Tremolo results when the amplitude of an audio signal is repetitiously varied as show in Figure 7.

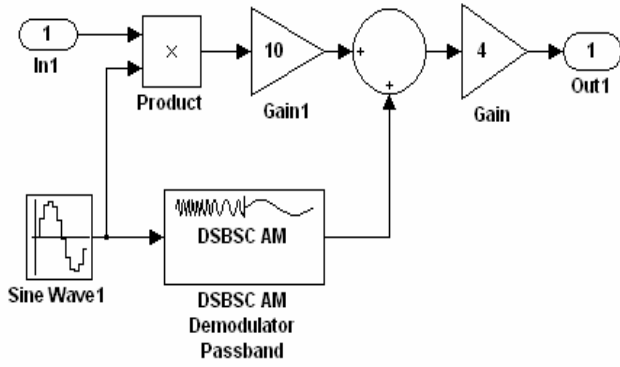


Fig. 7. Tremolo Effects Model

D. Reverberation

Reverberation is the persistence of sound in a particular space after the original sound is removed. This is shown by the delay & feedback gain in Figure 8.

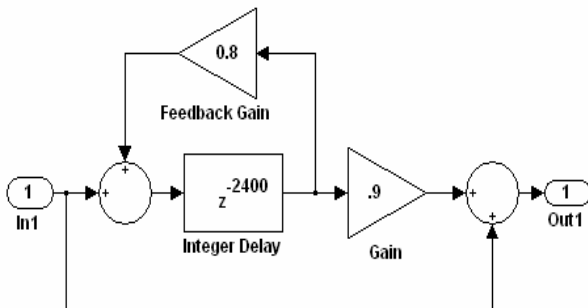


Fig. 8. Audio Reverberation Model

E. Combined Audio Effect Model

The combined effect model is shown in Figure 9. On having created this model, it is first *Test Run* in Simulink to check its correctness. If it executes properly, the value of time T (at the bottom of the screen) will start increasing from 0. Having tested the model, CCS can now be connected by pressing *Ctrl+B*. The in-built feature in Simulink in the form of *Real-time Workshop* will automatically generate all the required C source files, Linker files and Include files corresponding to our Simulink design. After completing this, CCS is launched and the C6713 is connected. The last step in the process is building the *executable .out* file after which control is transferred to CCS and our program is run on the

DSK in real-time. Before, all this is carried out, it is imperative to take notice of which version of MATLAB is compatible with the Code Composer Studio being used. For Code Composer Studio version 3.1, *MATLAB 2006b* or *MATLAB 2007a* may be used (we used MATLAB 2006b). If a non-compatible version of MATLAB is used, the project will not build, regardless of whether it has the Real-Time Workshop utility or not.

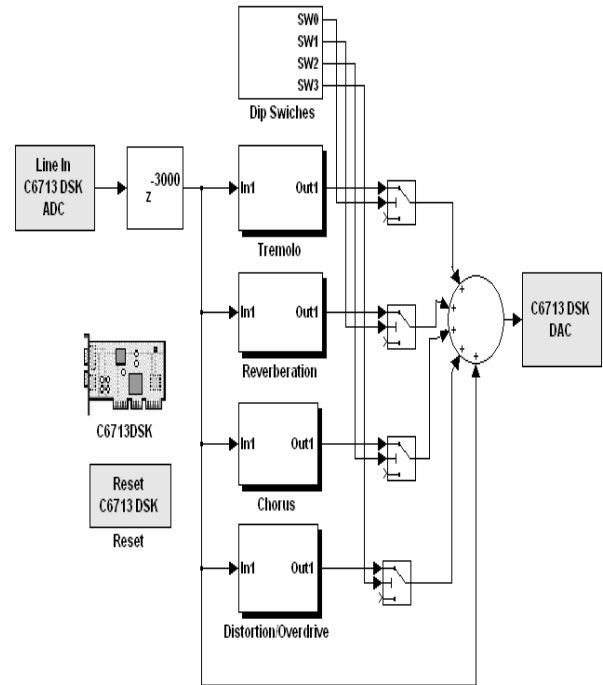


Fig. 9. Final Combined Audio Effect Simulink Model

V. GRAPHICAL USER INTERFACE

The pedals available in the markets today have the capability to store various effects, but the musician would be required to make changes within the effect using manual knobs. Our effects unit is able to accomplish this by using a software configured DSP, which, provides the user with the flexibility and ease in varying within the effects. For someone like a guitarist who has no knowledge about the internals of the effects system, it is important to develop a tool which would make the effects unit of use to them; something which could make it easy for the common man to manage. Using this GUI he can set the desired extents of each effect; burn it on the DSK using the FlashBurn utility and then take the stored effects wherever he desires [7],[8].

A. Creating the GUI

On the main screen of MATLAB click *GUIDE -- Create New GUI -- Blank GUI (default)* – select the check box *Save on startup as:* and give the name of the FIG file and click *OK*. By default, the first time while saving or running the GUI, GUIDE stores the GUI in two files. The FIG-file

and the M-file, both were saved in the *work* directory of MATLAB. They correspond to the tasks of laying out and programming the GUI. When we lay the GUI in the Layout Editor, the work is stored in the FIG-file and when we program the GUI; our work is stored in the corresponding M-file.

B. Slider/Edit Box Callback

The GUI uses sliders as well as edit boxes to specify block parameters since these components enable the selection of values. It does this via separate callbacks created for each GUI template. When a user changes any of the values in the GUI, the callback executes the following steps:

- Calls `model_open` to ensure that the Simulink model is open so that simulation parameters can be set.
- Gets the new slider/edit box value.
- Sets the value in edit box to match the slider movement and vice versa.
- Sets the appropriate block parameter to the new value (`set_param`).

Some of the more important functions needed to link the GUI to Simulink are listed below:

C. Opening Function

The opening function is the first callback in every GUI M-file. It is executed just before the GUI is made visible to the user, but after all the components have been created, i.e., after the components' CreateFcn callbacks, if any, have been run.

The opening function may also be used to perform initialization tasks before the user has access to the GUI. Below is the prototype of our opening function:

```
function filename_OpeningFcn(hObject,  
    eventdata, handles, varargin)  
model_open(handles)  
handles.output = hObject;  
guidata(hObject, handles);
```

D. Set_param

Sets Simulink system and block parameters

Syntax

```
set_param('obj', 'parameter1', value1,  
    'parameter2', value2, ...),
```

where 'obj' is a system or block path, sets the specified parameters to the specified values. Value strings are case sensitive. Case is ignored for parameter names. Any parameters that correspond to dialog box entries have string values. Model and block parameters are listed in Model and Block Parameters.

E. Open System

While running the GUI our Simulink model automatically opens due to the use of the open system function.

Syntax:

```
open_system('.mdl filename')
```

VI. STANDALONE DSP TMS320C6713 DSK

The C6713DSK is one of the most resourceful kits available today for Digital Signal Processing, but its practical applications are greatly minimized if it is not made to operate in the standalone state. The Code Composer Studio comes with a burning application called 'FlashBurn'. It is provided on the CCS CD and needs to be separately installed. It is this utility that is employed to burn our program on to the DSP's flash memory. The flash memory has a default capacity of 256Kb. The DSP does also have an SDRAM with a capacity of 16MB. The Flash on the C6713 DSK is arranged as a 256Kb x 16 device, but the software and board are configured to use CE1 (the chip enable space the Flash is in) in 8-bit mode to match other C6x DSKs. To use the full 512Kb, we used the following steps:

1. Create a new GEL file that sets CE1 up for 16-bit mode instead of 8-bit.
2. Create a new copy of the FlashBurn algorithm in `c:\ti\bin\utilities\flashburn\c6000\dsk6713` to write to all 16-bits of Flash instead of only the lower 8 bits.
3. Using the BSL, we modified the BSL to work with CE1 in 16-bit mode. The affected portions were all in `dsk6713.c`. We needed to modify the EMIF setup to configure CE1 as 16-bits, and the CPLD register address offsets in `DSK6713_rget()` and `DSK6713_rset()` which had to be multiplied by two because the registers appeared to be 16-bits apart rather than 8.
4. Recompile the application and built a new hex file using hex6x options for 16-bit wide memory rather than 8-bit.
5. Place application in Flash using FlashBurn .
6. Next we power off the board and set the *boot mode config switches (SW3)* for 16-bit mode instead of 8-bit mode.

A. Hex Conversion Utility

The executable *.OUT file* needs to be converted from *COFF to a hex file format* that can be loaded in to flash. The COFF-to-hex converter file *hex6x.exe* is included in CCS in the directory it is saved in. To invoke hex6x.exe within DOS,

an appropriate path needs to be created. Now, two details need to be kept in mind. Firstly, before we run the hex6x utility on our project .OUT file, we need to create a *command linker file* in CCS. This can be done by simply opening a new source file in the CCS main window and then saving it as a command linker file with extension .cmd. The contents of this linker file are shown below:

```

** ===== app_hex.cmd =====
** hex6x command file
*/
`.Debug\app.out          /* input COFF file
*/
-map .\Hex\apphex.map /* generate hex.map
map file*/-a
/* ASCII HEX format */
-image                      /*
set image mode */
-zero                      /*
reset address origin to 0 */
-memwidth 8                /* 8-bit
wide ROM */
ROMS
{
FLASH: org = 0x90000000, len = 0x40000,
romwidth = 8, files = {.\Hex\app.hex}
}
SECTIONS /* list of COFF sections to be
ROMed */
{
.boot_code
.bios
.sysinit
.gblinit
.trcdata
.rtdx_text
.text
.cinit
.pinit
.const
.switch
.hwi_vec }

```

The only adjustment to be made to the above created file is to replace the name *app* with that of our .OUT file we are setting out to build. After having done this, we then copy *hex6x* in to our project file. Having done this we access DOS and from our respective project folder we type the following for hex conversion:

```
Hex6x appname_hex.cmd
```

This will produce the project specific hex file in the project folder.

B. Configuring the Flashburn Utility

The figure above shows the .cdd file used by flashburn. The remaining fields should be filled as shown below as shown in Figure 10. Within CCS, select tools FlashBurn to invoke the flashburn utility. Then select *File* → *New* to configure the FlashBurn utility and create *appname.cdd*, with the following fields.

1. Conversion .cmd file
2. File to burn: our project hex file
3. FBTC program file: present in our CCS directory

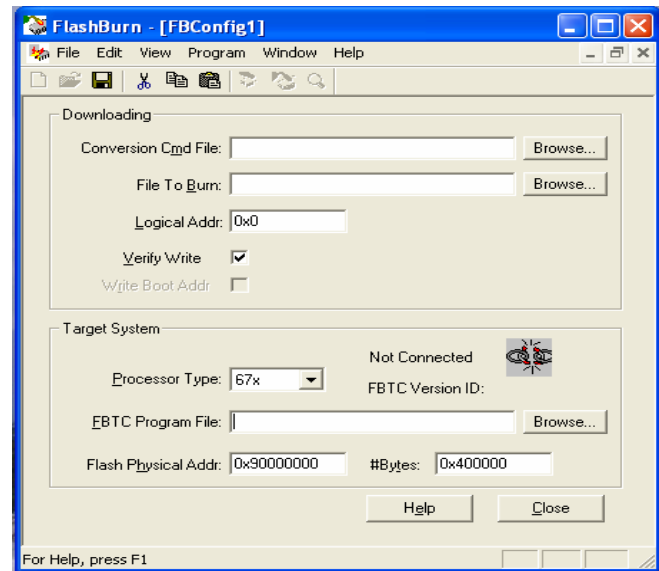


Fig. 10. FlashBurn utility (.cdd) configuration

C. Programming the Flash

Within the Flashburn utility, select *Program* → *Erase Flash*. This erases any program stored in the flash memory. Still, within the FlashBurn Utility select *Program* → *Program Flash*. This loads the Hex file created initially in to the flash memory [4].

VII. CONCLUSION

This developed system provides a unique and dynamic environment for guitarists a novice approach. A PC based GUI provides intricate customization, circumventing the need to trawl through oceans of manuals. The presence of a built-in *codec* on the C6713 kit allows the addition of other standard equipment such as synthesizers. This paper 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.

VIII. FURTHER WORK

Our hex file was of 789kB, while the flash memory could only hold 256kB. What we then attempted to do was to save our program in the larger *external memory*. For this we also found a sample file, of which we created a hex File. This file was the same size as our program's size and was successfully burnt on to the DSP. On checking, we found that this program was working perfectly in the standalone state. On observing this, we suspected that our program may also work on standalone if we followed the same procedure we had for the sample file. We first started with converting our program's output file in to the Hex format. This did not go too smoothly and we faced a number of errors. These were mainly to do with the linker file we had created earlier and we had to make some changes to it before our program got converted. Having converted our program to the *.hex* format, we proceeded to what we hoped would be the final step of burning our Simulink file on to the DSK. FlashBurn utility executed perfectly, but again for some reason our file did not get programmed on to the DSK. After spending days checking and cross checking, we realized that when we converted our program file in to the *.hex* format we got a warning message. This message did not pop up when we converted our sample program (the one that worked). The warning message is '*the bootload section could not be found in the .OUT file*'. This, we believe is an error in our program's address allocation. Our first approach was to open this *.OUT* file and try to match it to our sample file's *.OUT* file, but *.OUT* files cannot be opened. We searched for softwares to open this file and eventually stumbled across one but the language in which it opened our *.OUT* file was one we could not decipher. The next notion that we got was to build the sample program that worked and in the end, simply replace its *.c* file, but we ran in to a number of errors. The matter of whether the C6713 can operate in standalone is a disputed issue, but we have proved that it does have this capacity. The working project *.hex* file can be obtained from the following website:

<http://www.-s.ti.com/sc/psheets/spra999a1/spra999a1.zip>

We hope the information provided will be helpful to our fellow colleagues and if we are unsuccessful in our endeavors, perhaps, they can use the information provided to get it done right.

REFERENCES

- [1] S K Hasnain, Nighat Jamil, "Implementation of Digital Signal Processing Real Time Concepts Using Code Composer Studio 3.1, TI DSK TMS320C6713 and DSP Simulink Blocksets", IC4 Ist International Multi-topic Conference, Pakistan Navy Engineering College (NUST), Karachi, Pakistan, Conference, page(s) 119-125, 12-13 Nov, 2007.
- [2] S K Hasnain, Nighat Jamil, "Real Time Concepts Using TI DSK TMS320C6713, Target for TI 6000 and DSP Simulink Blocksets in Matlab", Technology Forces, A journal of Engineering and Sciences Pakistan Air Force, Karachi Institute of Economics & Technology, Korangi Creek, Pages(s) 1-14, Karachi. June 2008
- [3] The Mathworks Inc. Matlab Users Guide, 2008.
- [4] DSP Blockset (For use with Simulink) Users Guide Mathworks Inc., 2008
- [5] The Mathworks Inc. Simulink, Simulation and Model Based Design, 2008.
- [6] Code Composer Studio IDE Getting started Users Guide, 2005.
- [7] Gannot and Arvin: A Simulink and Texas Instruments C6713 based Signal Processing laboratory, 2006 Texas Instruments Inc., TMS320C6713 DSK User's Guide, 2005.
- [8] Rulph Chassaing, "DSP Applications Using C and the TMS320C6x DSK", The Mathworks Inc. Simulink, Simulation and Model Based Design, 2004.