

T.C.DOĞUŞ ÜNİVERSİTESİ

**ENGINEERING FACULTY
COMPUTER ENGINEERING**

**MP3 PLAYER-ONLY IMPLEMENTATION USING TMS320C6713 DSP KIT
PROGRAMMED WITH CODE COMPOSER STUDIO**

Graduation Project

**Derya SÖZEN
200431026**

Prof.Dr.İ.Cem GÖKNAR

İstanbul, August 2008

T.C.DOĞUŞ ÜNİVERSİTESİ

**ENGINEERING FACULTY
COMPUTER ENGINEERING**

**MP3 PLAYER-ONLY IMPLEMENTATION USING TMS320C6713 DSP KIT
PROGRAMMED WITH CODE COMPOSER STUDIO**

Graduation Project

**Derya SÖZEN
200431026**

Prof.Dr.İ.Cem GÖKNAR

İstanbul, August 2008

In memories of Prof.Dr.F.Şenel BOYDAĞ

Assoc.Prof.İskender HİKMET

Res.Assis.Mustafa FİDAN

ACKNOWLEDGMENT

This project was spread to a long period of time and accomplished step by step. I started collecting background information on MP3 and DSP topics last autumn. Then I worked on a LabView simulation to see how it works and to understand MP3 decoding algorithm better. For the last 2 months I worked hard on programming the real-time implementation of an MP3 player on TI's TMS320C6713 DSK.

Special thanks to Prof.Dr.İ.Cem Gökner for accepting me as his project student, his belief in me, his effort in finding me help and his appreciation of my work. I also thank to my family and Anıl Esen for their entire support throughout this project. They always pulled me of when I got tired especially in the last period of project submission. All the laboratory assistants were very helpful during my days of laboratory experiments. Other thanks go to Res.Assist.Uğur Çini and Çağrı Güvenel for their leading ideas.

CONTENTS

Acknowledgment	i
List of Figures	iii
List of Tables	iv
Abbreviations	v
1. Introduction	1
2. Overview of Hardware and Software Tools	3
2.1 The ‘C6000 CPU	3
2.2 The General Purpose Registers	4
2.3 Interrupts	4
2.4 Memory Organization for the C6713	5
2.5 EDMA	6
2.6 Serial Ports	7
2.7 The C6713	8
2.8 The Audio Interface Onboard the C6713	9
2.8.1 Properties of the AIC23 Codec	10
2.8.2 Codec interface	11
2.9 The BSL	12
2.10 The CSL	13
2.11 DSP/BIOS RTA and RTDX Features	13
3. Coding C6713	15
3.1 Initialization of C6713 CPU and Codec	15
3.2 MP3 Player	20
3.3 Build, Load, Run	23
4. Conclusion	36
Bibliography	38
Appendix	40
APP-1 Special Purpose registers	40
APP-2 General purpose registers	41
APP-3 L2 Cache registers	42
APP-4 Memory map address ranges of the C6713	43
APP-5 McBSP0 and McBSP1 registers	44
Autobiography	45

* CD Included

LIST OF FIGURES

- Figure 2.1 Functional Block Diagram of the TMS320C6713 DSK
- Figure 2.2 How Interrupts Work?
- Figure 2.3 C6x Peripherals
- Figure 2.4 Block Diagram of the TMS320C6713 DSP Starter Kit (DSK)
- Figure 2.5 Audio connection: The DSK uses two McBSPs to talk with the AIC23 codec, one for control, Another for data
- Figure 2.6 Codec Interface
- Figure 2.7 Support Files
- Figure 2.8 DSP/BIOS RTA
- Figure 2.9 RTDX data flow
- Figure 3.1 Functional blocks associated with MP3 player
- Figure 3.2 Huffman information decoding block as a controller
- Figure 3.3 Block diagram of requantization block
- Figure 3.4 Defining Path
- Figure 3.5 Running a Setup from Command Line
- Figure 3.6 CCS Icons
- Figure 3.7 Device Setup
- Figure 3.8 USB connection
- Figure 3.9 General Diagnostic Test
- Figure 3.10 Advanced Diagnostic Test
- Figure 3.11 Connecting Target DSK
- Figure 3.12 Creating a Project
- Figure 3.13 Adding Files to Project
- Figure 3.14 Text Editor Screen and Project Tree
- Figure 3.15 Build Options
- Figure 3.16 Building Results
- Figure 3.17 Loading .out
- Figure 3.18 Loading Program
- Figure 3.19 Running Program
- Figure 3.20 Halting Program

LIST OF TABLES

Table 2.1 Memory Map for the TMS320C6713

Table 2.2 Sampling Rates

ABBREVIATIONS

3G	3 rd Generation
ADC	Analog-to-Digital Converter
AIC	Analog Interface Chip
ALU	Arithmetic Logic Unit
API	Application Interface
BSL	Board Support Library
C6713	TMS320C6713 DSK
CCS	Code Composer Studio
CD	Compact Disc
CMD	CoMmanD
CODEC	Coder – Decoder
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSL	Chip Support Library
DAC	Digital-to-Analog Converter
DIP	Digital Image Processing
DRAM	Dynamic Random Access Memory
DRR	Data Receive Register
DSK	DSP Starter Kit
DSL	Digital Subscriber Line
DSP	Digital Signal Processing (Processor)
DSP/BIOS	Texas Instruments Real-time Operating System Kernel
DXR	Data Transmit Register
EDMA	Enhanced Direct Memory Access
EMIF	External Memory Interface
EPROM	Erasable Programmable Read-Only Memory
FFT	Fast Fourier Transform
GEL	General Extension Language
GPIO	General Purpose I/O
GUI	Graphical User Interface

HPI	Host Port Interface
IER	Interrupt Enable Register
IFR	Interrupt Flag Register
IMDCT	Inverse Modified Discrete Cosine Transform
IRAM	Intelligent Random Access Memory
ISN	Initial Sequence Number
ISR	Interrupt Service Register
ISTP	Interrupt Service Table Pointer
JTAG	Joint Test Action Group
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
McASP	Multi-Channel Audio Serial Port
McBSP	Multi-Channel Binary Serial Port
MDCT	Modified Discrete Cosine Transform
MIC	Microphone
MP3	MPEG 1 Audio Layer 3
MPEG	Moving Pictures Expert Group
NMI	Non-Maskable Interrupt
PCM	Pulse Code Modulation
PGIE	Programmable Guidance Interface Enable
RTA	Real-Time Analysis
RTDX	Real-Time Data Exchange
SBSRAM	Synchronous Burst Static Random Access Memory
SDRAM	Synchronous Dynamic Random Access Memory
SRAM	Synchronous Random Access Memory
TI	Texas Instruments
VLIW	Very Long Instruction Word
XDS510	USB JTAG Emulator

1. INTRODUCTION

Digital signal processing, and hence the DSPs are playing a major role in today's almost every technology such as telecommunication infrastructure including 3G wireless, cable/DSL modems, digital cameras, digital audio players and etc.

To understand the logic residing in this technology, there is an easy way such as working on a MP3 player from beginning to the end, dealing with its all aspects included. This gives an engineer understanding of digital signal processing and its importance. Regarding from this point of view, I designed an MP3 player-only using C6713 and programming it in CCS.

My sources are mostly, DSP tutorials supported by TI official web site and the books written specifically on DSP technology.

My approach to the subject was from both software and hardware aspects. First of all, I learned the necessary basic information on MP3 format, MP3 player, digital signal processing and DSPs. Then, I worked on a simulation in LabVIEW and my final step was implementing the MP3 player on C6713 by programming it in CCS in both C and assembly languages.

Research was already done on this topic, but my aim was to explain digital signal processing with a specific example in simple terms without oversimplifying it.

There are two basic things this project aims: Understanding digital signal processing with digital signal processors and hands-on implementation of an MP3 player from both software and hardware aspects.

Through this report, I gave some fundamental information about the hardware and software properties I used and introduced all the steps I had to go through during programming the DSP along with specific screen shots and example command lines. Because this is mostly a software project, a CD is included in order to supply necessary codes for the potential user. Explanation of each code resides inside the texts.

In the conclusion section, I tried to summarize the overall concept, succeeded/failed parts and possible future progress of this project. I tried to be as simple as possible in my

explanations without oversimplifying the topic and aimed to approach the concept from a computer engineer point of view and not an electrical or an electronic engineer point of view.

İstanbul, August 2008

Derya SÖZEN

2. OVERVIEW OF HARDWARE AND SOFTWARE TOOLS

The purpose of this part is to introduce you to the main features of the hardware and software tools that have been used in this project. The C6713 is a small external board that connects to the PC through a USB port. The C6713 communicates with the analog world through a TI AIC23 stereo codec on the DSK.

2.1 The 'C6000 CPU

Members of the TMS320C67x family of DSP's all have essentially the same CPU which is also called the DSP core. The CPU has VLIW architecture. The CPU always fetches eight 32-bit instructions at once and there is a 256-bit bus to the internal program memory. Each group of eight instructions is called a fetch packet. The CPU has eight functional units that can operate in parallel and are equally split into two halves, A and B. All eight units do not have to be given instruction words if they are not ready. Therefore, instructions are dispatched to the functional units as execute packets with a variable number of 32-bit instruction words. The eight functional units include:

- Four ALU that can perform fixed and floating-point operations (.L1, .L2, .S1, .S2)
- Two ALU's that perform only fixed-point operations (.D1, .D2)
- Two multipliers that can perform fixed or floating-point multiplications (.M1, .M2)

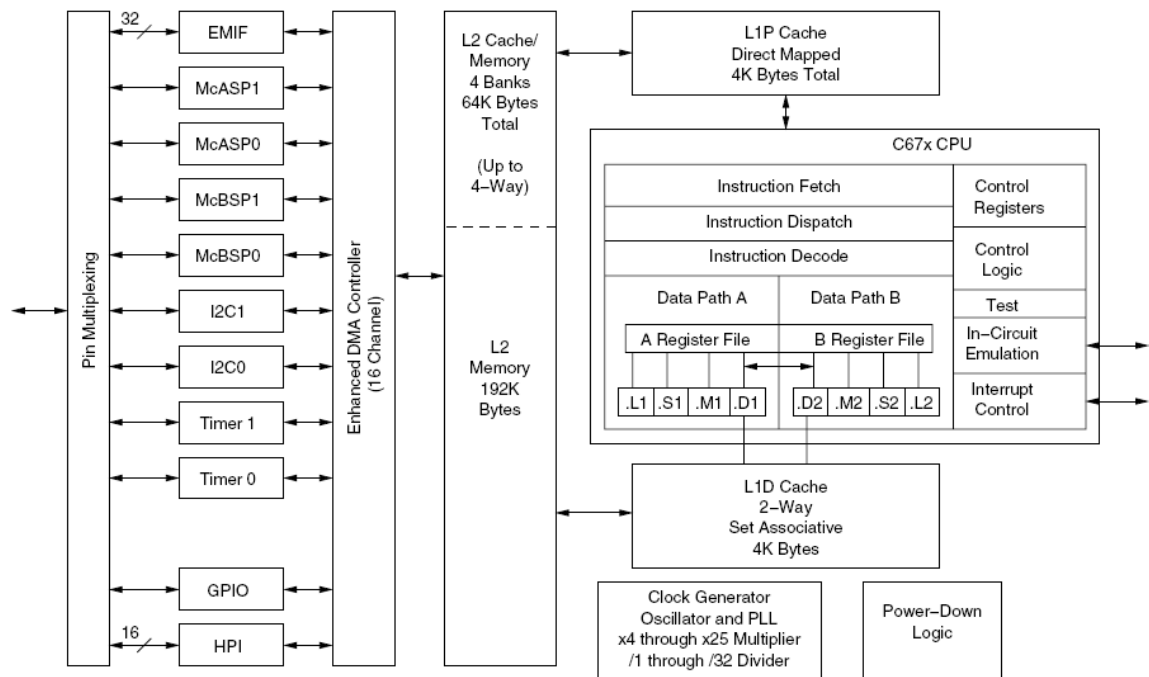


Figure2.1 Functional Block Diagram of the C6713

2.2 The General Purpose Registers¹

The CPU has thirty-two 32-bit general purpose registers split equally between the A and B sides. The CPU has a load/store architecture in which all instructions operate on registers. The data-addressing unit .D1 and .D2 are in charge of all data transfers between the register files and memory. The four functional units on a side can freely share the 16 registers on that side. Each side has a single data bus connected to all the registers on the other side, so the functional units on one side can access data in the registers on the other side. Access to a register on the same side, uses one clock cycle while access to a register on the other side requires a read and writes cycle.

2.3 Interrupts

The 'C6000 CPUs contain a vectored priority interrupt controller. The highest priority interrupt is RESET which is connected to the hardware reset pin and cannot be masked. The next priority interrupt is the NMI which is generally used to alert the CPU of a serious hardware problem like a power failure. Then, there are twelve lower priority maskable

¹ Appendix 1 & Appendix 2 – Special and General Purpose Register Addresses

interrupts INT4–INT15 with INT4 having the highest and INT15 the lowest priority. These maskable interrupts can be selected from up to 32 sources for the 'C6000 family. The sources vary between family members. For the C6713, they include external interrupt pins selected by the GPIO unit, and interrupts from internal peripherals such as timers, McBSP serial ports, McASP serial ports, EDMA channels, and the host port interface. The CPUs have a multiplexer called the interrupt selector that allows the user to select and connect interrupt sources to INT4 through INT15.

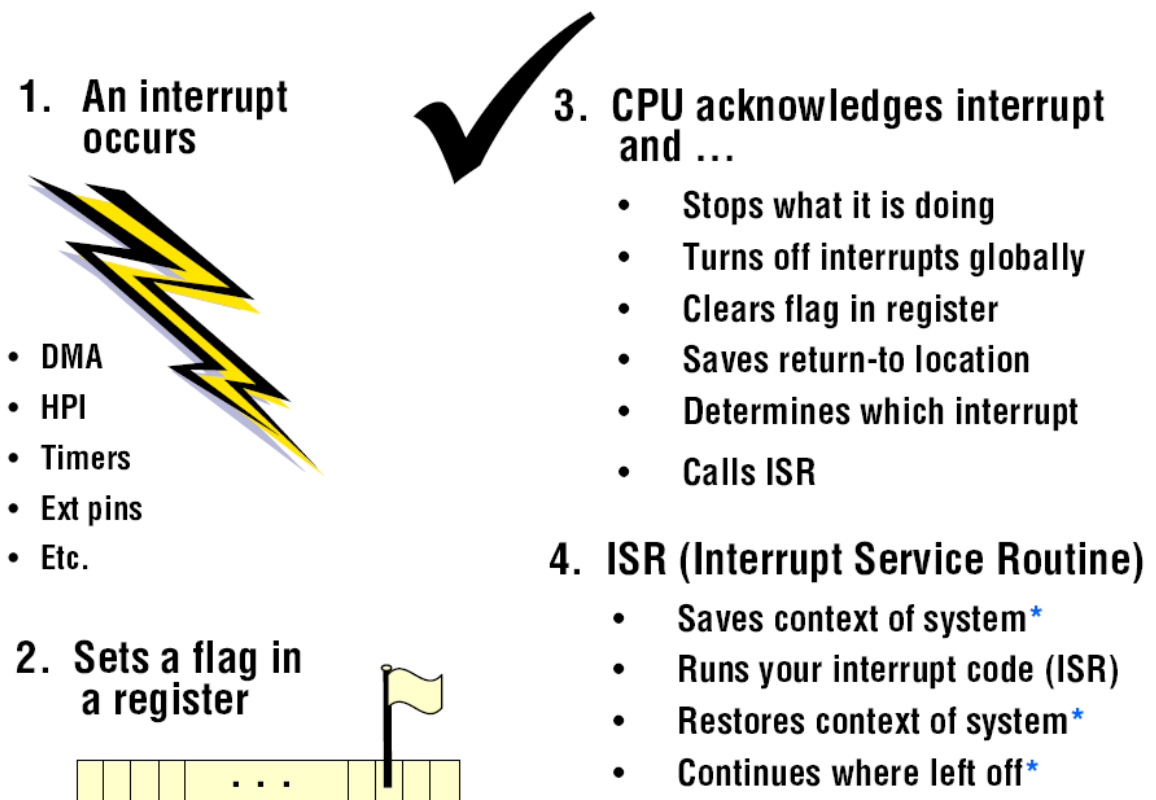


Figure 2.2 How Interrupts Work?

2.4 Memory Organization for the C6713

The C6713 has an L1/L2 memory architecture consisting of a 4K-byte L1P Program Cache² (direct-mapped), a 4K-byte L1D Data Cache (2-way set associative), and an L2 memory with 256K-bytes total. The L2 memory is partitioned into a 64K-byte L2 unified cache/mapped RAM which is up to 4-way set associative, and 192K-bytes of additional L2 mapped RAM. The L1P cache has a 256-bit wide bus to the CPU so the CPU can read a

² Appendix 3 - L2 Cache Registers

fetch packet (eight 32-bit instructions) each cycle. The 'C6713 DSP has a 32-bit EMIF unit that provides a glueless interface to SDRAM, Flash, SBSRAM, SRAM, and EPROM. The DSP has a 512 M-byte total addressable external memory space. Data is byte (8-bit), half-word (16-bit), or word (32-bit) addressable. Table 2.1 shows the default memory map for the TMS320C6713 DSK.

Address	C67x Family Memory Type	C6713 DSK
0x00000000	Internal Memory	Internal Memory
0x00030000	Reserved Space or Peripheral Register	Reserved or Peripheral
0x80000000	EMIF CE0	SDRAM
0x90000000	EMIF CE1	Flash
0x90080000		CPLD
0xA0000000	EMIF CE2	Daughter Card
0xB0000000	EMIF CE3	

Table 2.1 Memory Map for the C6713³

The memory map tells the debugger which areas of memory it can access. Memory maps vary depending on the application. When a memory map is defined and memory mapping is enabled, the debugger checks every memory access against the memory map. The debugger will not attempt to access an area of memory that is protected by the memory map. The debugger compares memory accesses against the memory map in software, not hardware. The debugger cannot prevent your program from attempting to access nonexistent memory.

2.5. EDMA

The C6713 has an EDMA that can transfer data between any locations in the DSP's 32-bit address space independently of the CPU. The EDMA handles all data transfers between the L2 cache/memory controller and the peripherals. These include cache servicing, non-cacheable memory access, user-programmed data transfers, and host access. It can move data to and from any addressable memory spaces including internal memory (L2 SRAM),

³ Appendix 4 – Memory map address ranges of the C6713 device

peripherals, and external memory. The EDMA includes event and interrupt processing registers, an event encoder, a parameter RAM, and address generation hardware. It has 16 independent channels and they can be assigned priorities. Data transfers can be initiated by the CPU or events from the peripherals and some external pins. The user can select how events are mapped to the channels. The EDMA can transfer elements that are 8-bit bytes, 16-bit halfwords, or 32-bit words.

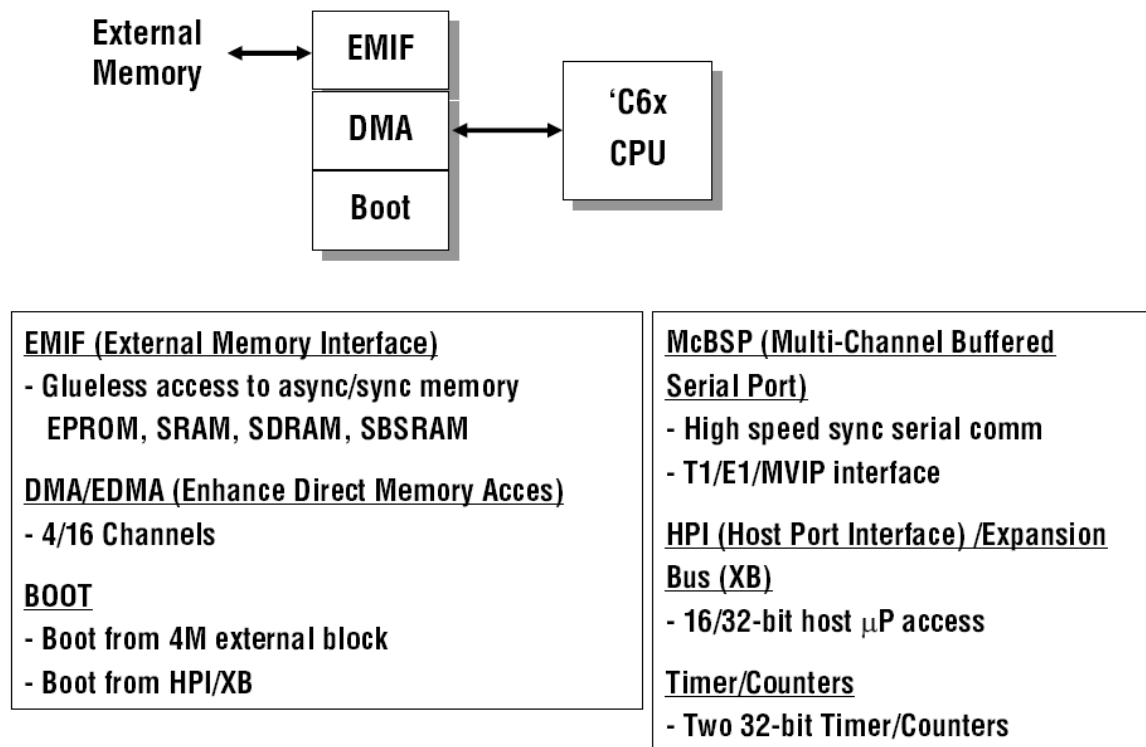


Figure2.3 C6x Peripherals

2.6 Serial Ports

The C6713 contains two bidirectional multichannel buffered serial ports (McBSP0 and McBSP1)⁴. The serial ports operate independently and have identical structures. They can be set to transfer 8, 12, 16, 20, 24, or 32 bit words. The bit clocks and frame synchs can be internal or external and the McBSP includes programmable hardware for generating shift clocks and frame synchs.

⁴ Appendix 5 - McBSP0 and McBSP1 registers

The McASP is a serial port optimized for the needs of multi-channel audio applications. The two McASPs can support two completely independent audio zones simultaneously. Each McASP includes a pool of 16 shift registers that may be configured to operate as either transmit data, receive data, or GPIO.

The C6713 also has a HPI. The HPI provides a 16-bit interface to a host. The host functions as a master and can access the entire memory map of the DSP. Accesses are accomplished by using the EDMA.

2.7 The C6713

The C6713 is a low cost board designed to allow the user to evaluate the capabilities of the C6713 DSP and develop C6713-based products. It demonstrates how the DSP can be interfaced with various kinds of memories and peripherals, and illustrates power, clock, JTAG and parallel peripheral interfaces. The board is approximately 5 inches wide and 8 inches long and is designed to sit on the desktop external to a host PC. It connects to the host PC through a USB port or an XDS510. A simplified block diagram of the DSK is shown in Figure 1.2.

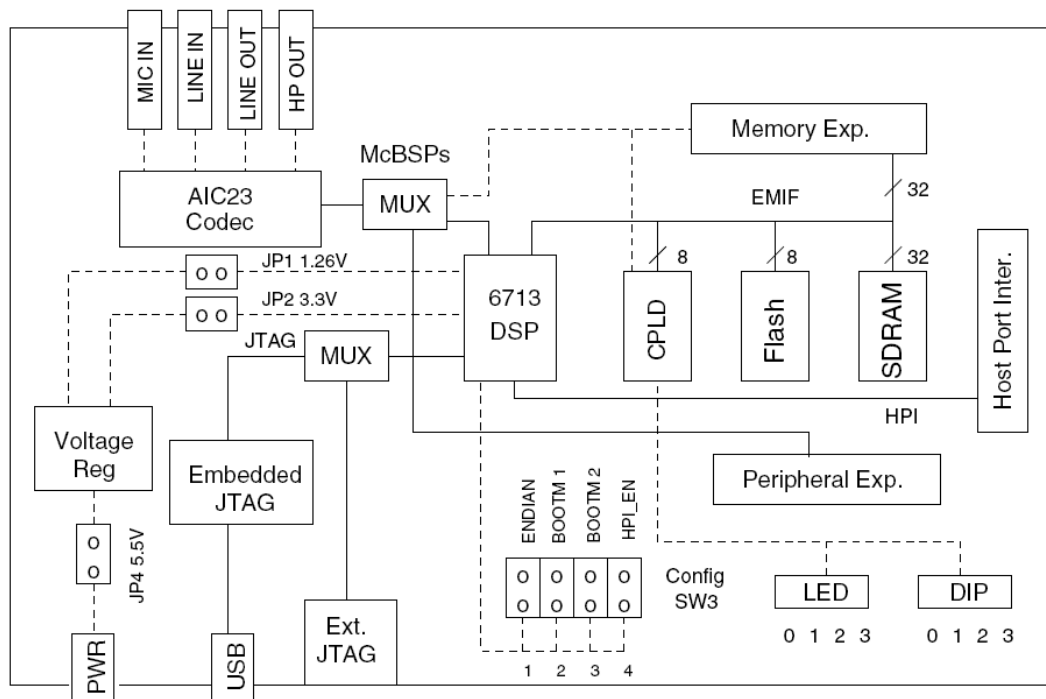


Figure 2.4 Block Diagram of the C6713

The major DSK hardware features are:

- A C6713 DSP operating at 225 MHz.
- An AIC23 stereo codec with Line In, Line Out, MIC, and headphone stereo jacks
- 16 Mbytes of synchronous DRAM (SDRAM)
- 512 Kbytes of non-volatile Flash memory (256 Kbytes usable in default configuration)
- Four user accessible LEDs and DIP switches
- Software board configuration through registers implemented in complex logic device
- Configurable boot options
- Expansion connectors for daughter cards
- JTAG emulation through onboard JTAG emulator with USB host interface or external
- Emulator

2.8 The Audio Interface Onboard the C6713

The C6713 uses a TI AIC23 codec. In the default configuration, the codec is connected to the two serial ports, McBSP0 and McBSP1. McBSP0 is used as a unidirectional channel to control the codec's internal configuration registers. It should be programmed to send a 16-bit control word to the AIC23 in SPI format. The top 7 bits of the control word specify the register to be modified and the lower 9 bits contain the register value. Once the codec is configured, the control channel is normally idle while audio data is being transmitted.

McBSP1 is used as the bi-directional data channel for ADC input and DAC output samples. The codec supports a variety of sample formats. For the experiments in this course, the codec should be configured to use 16-bit samples in two's complement signed format. The codec should be set to operate in master mode so it supplies the frame sync and bit clocks at the correct sample rate to McBSP1. The preferred serial format is DSP mode which is designed specifically to operate with the McBSP ports on TI DSPs.

The codec has a 12 MHz system clock which is the same as the frequency used in many USB systems. The AIC23 can divide down the 12 MHz clock frequency to provide sampling rates of 8000, 16000, 24000, 32000, 44100, 48000, and 96000 Hz.

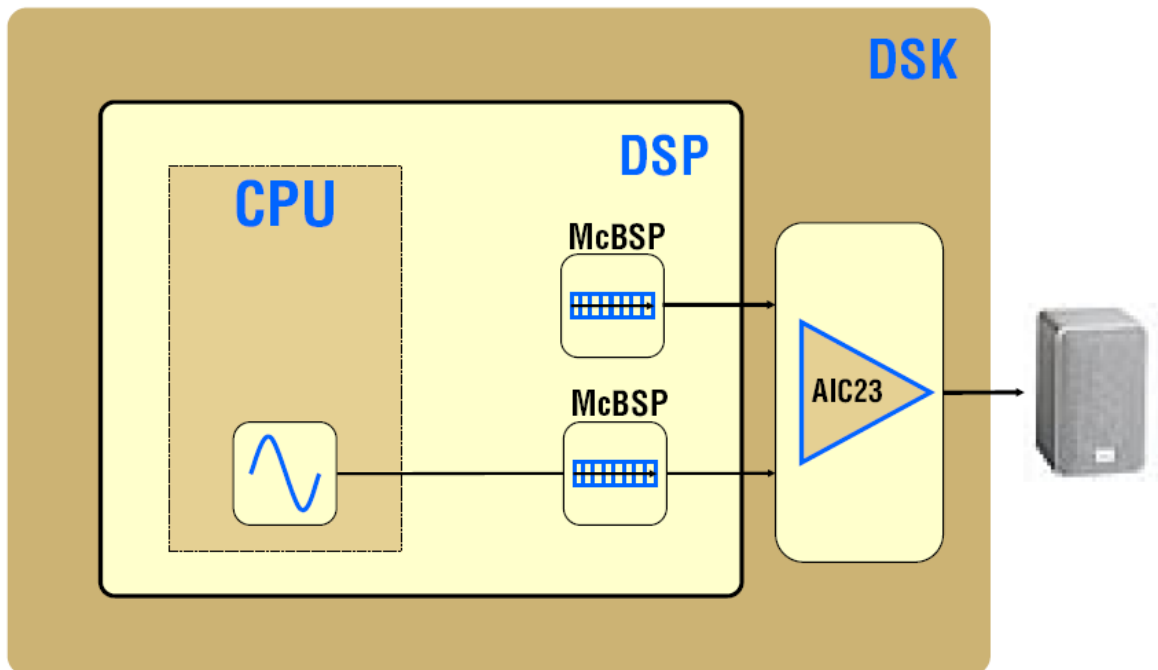


Figure 2.5 Audio connection: The DSK uses two McBSPs to talk with the AIC23 codec, one for control, Another for data

2.8.1 Properties of the AIC23 Codec

The C6713 supplies a 12 MHz clock to the AIC23 codec which is divided down internally in the AIC23 to give the sampling rates shown in the table below. The codec can be set to these sampling rates by using the function `DSK6713_AIC23_setFreq(handle,freq ID)` from the BSL. This function puts the quantity “Value” into AIC23 control register 8.

Some of the AIC23 analog interface properties are:

- The ADC for the line inputs has a full-scale range of 1.0 V RMS.
- The microphone input is a high-impedance, low-capacitance input compatible with a wide range of microphones.

Freq ID	Value	Frequency
DSK6713_AIC23_FREQ_8KHZ	0x06	8000 Hz
DSK6713_AIC23_FREQ_16KHZ	0x2c	16000 Hz
DSK6713_AIC23_FREQ_24KHZ	0x20	24000 Hz
DSK6713_AIC23_FREQ_32KHZ	0x0c	32000 Hz
DSK6713_AIC23_FREQ_44KHZ	0x11	44100 Hz
DSK6713_AIC23_FREQ_48KHZ	0x00	48000 Hz
DSK6713_AIC23_FREQ_96KHZ	0x0e	96000 Hz

Table 2.2 Sampling Rates

- The DAC for the line outputs has a full-scale output voltage range of 1.0 V RMS.
- The stereo headphone outputs are designed to drive 16 or 32-ohm headphones.
- The AIC23 has an analog bypass mode that directly connects the analog line inputs to the analog line outputs.
- The AIC23 has a side tone insertion mode where the microphone input is routed to the line and headphone outputs.

2.8.2 Codec Interface

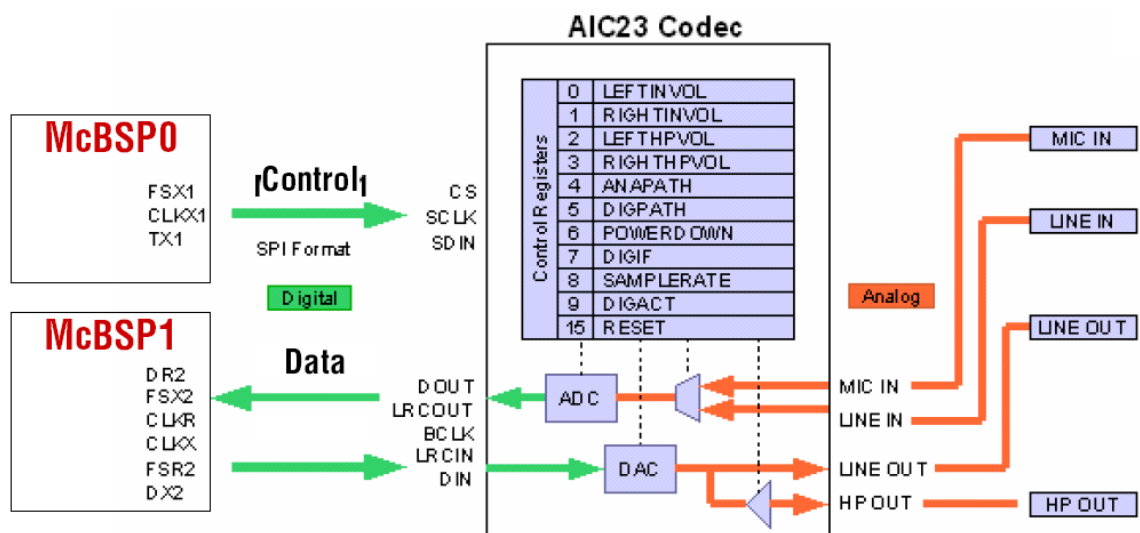


Figure 2.6 Codec Interface

- McBSP0 connected to program AIC23's control registers
- McBSP1 is used to transfer data to ADC and DAC converters
- Programmable frequency: 8K, 16K, 24K, 32K, 44.1K, 48K, 96K
- 24-bit converter, Digital transfer widths: 16-bits, 20-bits, 24-bits, 32-bits

2.9 The BSL

A special Board Support Library (BSL) is supplied with the C6713. The BSL provides C-language functions for configuring and controlling all the on-board devices. The library includes modules for general board initialization, access to the AIC23 codec, reading the DIP switches, controlling the LED's, and programming and erasing the Flash memory. The source code for this library is included in the Code Composer Studio supplied with the DSK and it is set up automatically to use the BSL. The function for configuring the codec in the BSL sets McBSP1 to transmit and receive 16-bit words. The codec sends 16-bit left and right channel input samples to McBSP1 alternately and a program reading these samples from McBSP1's DRR1 would have to somehow figure out which is the right and which is the left channel sample.

I have modified the code configuration function `DSK6713_AIC23_openCodec()` to send and receive data samples from the codec in DSP format using 32-bit words. The first word transmitted by the AIC23 codec is the left channel 16-bit sample and the right channel 16-bit sample is transmitted immediately after the left channel sample. The AIC23 generates single frame sync at the beginning of the left channel sample. Therefore, a 32-bit word received by McBSP1 contains the left sample in the upper 16 bits and the right sample in the lower 16 bits. This solves the channel ambiguity problem. The reverse process takes place when sending samples from the DSP to the codec. The user's program should pack the left channel 16-bit sample in the upper 16 bits of an integer and the right channel 16-bit sample in the lower 16 bits and then write this word to the DXR1 of McBSP1. I have replaced the original BSL codec configuration function with my modified function and renamed the file `dsk6713bsl32.lib`.

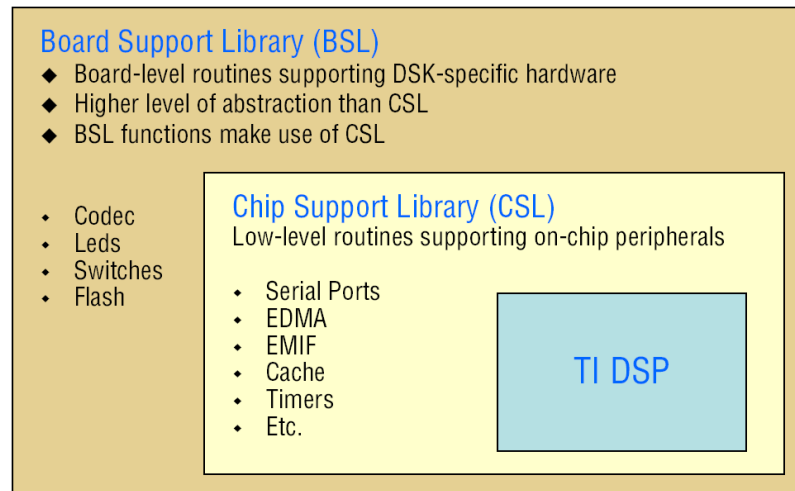


Figure 2.7 Support Files

2.10 The CSL

TI has created a CSL that contains C functions and macros for configuring and interfacing with all the C6713 on-chip peripherals and CPU interrupt controller. This library is loaded onto the PC when the DSK software is installed. Each peripheral is covered by an individual API module. The CSL header files provide a complete symbolic description of all peripheral registers and register fields. The CSL provides a graphical user interface (GUI) that is part of the DSP/BIOS Configuration.

2.11 DSP/BIOS RTA and RTDX Features

The DSP/BIOS RTA facilities utilize the RTDX link to obtain and monitor target data in real-time. I utilized the RTDX link to create my own customized interfaces to the DSP target by using the RTDX API Library. Real-time data exchange (RTDX) transfers data between a host computer and target devices without interfering with the target application. This bi-directional communication path provides for data collection by the host as well as host interaction with the running target application. RTDX also enables host systems to provide data stimulation to the target application and algorithms.

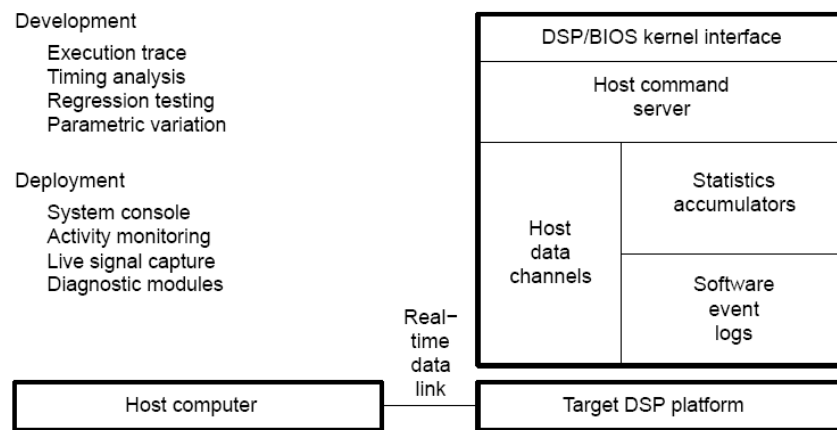


Figure 2.8 DSP/BIOS RTA

Data transfer to the host occurs in real-time while the target application is running. On the host platform, an RTDX host library operates in conjunction with Code Composer Studio IDE. Data visualization and analysis tools communicate with RTDX through COM APIs to obtain the target data and/or to send data to the DSP application. The host library supports two modes of receiving data from a target application: continuous and non-continuous. In continuous mode, the data is simply buffered by the RTDX Host Library and is not written to a log file. Continuous mode should be used when the developer wants to continuously obtain and display the data from a target application, and does not need to store the data in a log file.

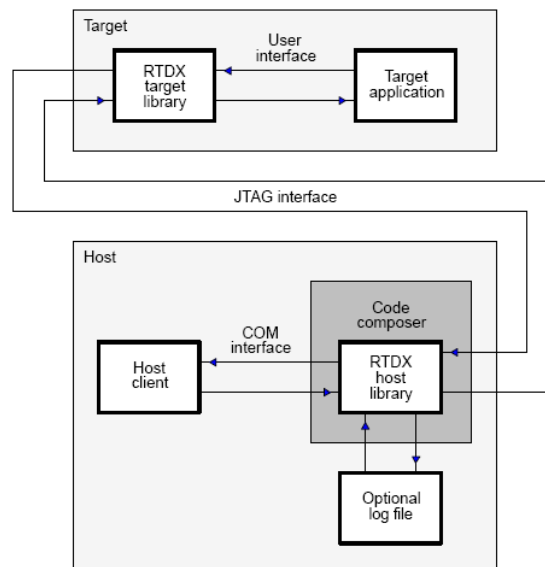


Figure 2.9 RTDX data flow

3. CODING C6713

3.1 Initialization of C6713 CPU and Codec⁵

Before executing the code that performs our desired signal processing algorithm, the DSK and DSP have to be initialized. This is partially taken care of when we start Code Composer. The version of CCS supplied with the C6713 has been configured to automatically load the GEL file, DSK6713.gel, in the directory C:\CCStudio_v3.1\cc\gel is automatically called. It defines a memory map, creates some GEL functions for the GEL menu, sets some CPLD registers and initializes the EMIF for the memory on the C6713.

```

/*****
/* Program: dskstart32.c
/*
/* The codec can be set to the sampling rates shown in the
/* table below by using the function
/* DSK6713_AIC23_setFreq(handle, freq ID)
/*
/* freq ID      Value Frequency
/* DSK6713_AIC23_FREQ_8KHZ, 0x06, 8000 Hz
/* DSK6713_AIC23_FREQ_16KHZ, 0x2c, 16000 Hz
/* DSK6713_AIC23_FREQ_24KHZ, 0x20, 24000 Hz
/* DSK6713_AIC23_FREQ_32KHZ, 0x0c, 32000 Hz
/* DSK6713_AIC23_FREQ_44KHZ, 0x11, 44100 Hz
/* DSK6713_AIC23_FREQ_48KHZ, 0x00, 48000 Hz
/* DSK6713_AIC23_FREQ_96KHZ, 0x0e, 96000 Hz
/*
*****/
#include <stdio.h>
#include <stdlib.h>
#include <dsk6713.h>
#include <dsk6713_aic23.h>
#include <intr.h>
#include <math.h>

/* Codec configuration settings
/* See dsk6713_aic23.h and the TLV320AIC23 Stereo Audio CODEC Data Manual
/* for a detailed description of the bits in each of the 10 AIC23 control
/* registers in the following configuration structure.
DSK6713_AIC23_Config config = { \
0x0017, /* 0 DSK6713_AIC23_LEFTINVOL Left line input channel volume */ \
0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */ \
0x00d8, /* 2 DSK6713_AIC23_LEFTHPVOL Left channel headphone volume */ \

```

⁵ CD included


```

0x00d8,      /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume*/
0x0011,      /* 4 DSK6713_AIC23_ANAPATH Analog audio path control */\
0x0000,      /* 5 DSK6713_AIC23_DIGPATH Digital audio path control */\
0x0000,      /* 6 DSK6713_AIC23_POWERDOWN Power down control */\
0x0043,      /* 7 DSK6713_AIC23_DIGIF Digital audio interface format */\
0x0081,      /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control (48 kHz) */\
0x0001      /* 9 DSK6713_AIC23_DIGACT Digital interface activation */\
};
void main(void){
DSK6713_AIC23_CodecHandle hCodec;
Uint32 sample_pair = 0;
/* Initialize the interrupt system */
intr_reset();
/* dsk6713_init() must be called before other BSL functions */
DSK6713_init(); /* In the BSL library */
/* Start the codec */
hCodec = DSK6713_AIC23_openCodec(0, &config);
/* Change the sampling rate to 16 kHz */
DSK6713_AIC23_setFreq(hCodec, DSK6713_AIC23_FREQ_16KHZ);
/* Read left and right channel samples from the ADC and loop them back out to the DAC.
for(;;){
    while(!DSK6713_AIC23_read(hCodec, &sample_pair));
    while(!DSK6713_AIC23_write(hCodec, sample_pair));
}
}
}

```

The program C:\c6713dsk\dskstart32.c can be used as a starting point for writing C6713 applications. It contains the code necessary to initialize the DSK board, initialize the C6713 McBSPs and initialize the AIC23 codec.

The program dskstart32.c uses functions from the UMD modified DSK BSL, C:\c6713dsk\dsk6713bsl32.lib, to continue the initialization. The file c6713dsk.hlp also contains detailed information about the DSK hardware. The modified library, its header files, and sources are in the directories: (CD included)

C:\c6713dsk\dsk6713bsl32\lib

C:\c6713dsk\dsk6713bsl32\include

C:\c6713dsk\dsk6713bsl32\sources

The program dskstart32.c first initializes the board support library by calling DSK6713_init() whose source code is in the BSL file dsk6713.c. This initializes the chip's

PLL, configures the EMIF based on the DSK version and sets the CPLD registers to a default state.

Next `dskstart32.c` initializes the interrupt controller registers and installs the default interrupt service routines by calling the function `intr_reset()` in the UMD added file `intr.c`. This clears GIE and PGIE, disables all interrupts except RESET in IER, clears the flags in the IFR for the maskable interrupts INT4 - INT15, resets the interrupt multiplexers, initializes the ISTP and sets up the Interrupt Service Routine Jump Table. The object modules `intr.obj` and `intr_.obj` were added to BSL library so we should not include `intr.c` and `intr_.asm` in your project. Functions included in `intr.c` are:

`intr_reset()` Reset interrupt regs to defaults

`intr_init()` Initialize ISTP

`ints_isn()` Assign ISN to CPU interrupt

`intr_get_cpu_intr()` Return CPU int. assigned to ISN

`intr_map()` Place ISN in int. mux. register

`intr_hook()` Hook ISR to interrupt

A set of macro functions for setting and clearing bits in the IER and IFR are available.

Next the codec is started by calling the function `DSK6713_AIC23_openCodec()`. This function configures serial port McBSP0 to act as a unidirectional control channel in the SPI mode transmitting 16-bit words, then configures the AIC23 stereo codec to operate in the DSP mode with 16-bit data words with a sampling rate of 48 kHz and then McBSP1 is configured to send data samples to the codec or receive data samples from the codec in the DSP format using 32-bit words.

The first word transmitted by the AIC23 is the left channel sample. The right channel sample is transmitted immediately after the left sample. The AIC23 generates a single frame sync at the beginning of the left channel sample. Therefore, a 32-bit word received by McBSP1 contains the left sample in the upper 16 bits and the right sample in the lower

16 bits. The 16-bit samples are in 2's complement format. Words transmitted from McBSP1 to AIC23 must have the same format. The codec and McBSP1 are configured so that the codec generates the frame syncs and shift clocks.

```

/*****
/* File dsk6713.cmd */
/* This linker command file can be used as the starting point for linking */
/* programs for the TMS320C6713 DSK. This CMD file assumes everything */
/* fits into internal RAM and is that is not true, it maps some sections to the */
/* external SDRAM. */
*****/
-c
-heap 0x1000
-stack 0x400
-lrts6700.lib
-lcsl6713.lib
MEMORY
{
    IRAM:      origin = 0x0,          len = 0x40000      /* 256 Kbytes */
    SDRAM:     origin = 0x80000000, len = 0x1000000      /* 16 Mbytes SDRAM */
    FLASH:     origin = 0x90000000, len = 0x40000      /* 256 Kbytes */
}
SECTIONS
{
    .vec:      load = 0x00000000 /* Interrupt vectors included by using intr_reset() */
    .text:     load = IRAM      /* Executable code */
    .const:    load = IRAM      /* Initialized constants */
    .bss:      load = IRAM      /* Global and static variables */
    .data:     load = IRAM      /* Data from .asm programs */
    .cinit:    load = IRAM      /* Tables for initializing variables and constants */
    .stack:    load = IRAM      /* Stack for local variables */
    .far:      load = IRAM      /* Global and static variables declared far */
    .sysmem:   load = IRAM      /* Used by malloc, etc. (heap) */
    .cio:      load = IRAM      /* Used for C I/O functions */
    .csldata   load = IRAM
    .switch    load = IRAM
}

```

Linker command files (dsk6713.cmd) are used to define how relocatable program sections are mapped into the physical system memory. They can also contain assembler options and lists of object programs to be included in the output modules. Additional object modules can also be included on the linker command line. The modules are loaded in the order in which they appear in the command line list of .cmd and .obj files. Command files are

convenient for saving definitions and operations that will be ordinarily used when linking programs for a particular project.

The `-c` line in `dsk6713.cmd` tells the linker to use the auto-initialization feature of C programs. The TI C compiler builds a table containing the data required to initialize all variables initialized in the C program. Code is included in the executable module to load the data values in the table into the variables when the program starts. The `-heap` and `-stack` lines allocate memory for the heap and stack. The number after these commands is the allocated memory size in bytes. The `-lrts6700.lib` line tells the linker to search the C run-time library `rts6700.lib` for unresolved references. This library provides the standard functions the C compiler expects. The line `-lcs16713.lib` tells the linker to search the CSL `cs16713.lib`. CCS has been set to know the path to these libraries. They are almost always used by C programs. Including these lines in the linker command file automatically includes them in the linker search path without any further effort on our part.

The MEMORY portion of the command file is used to define the physical memory layout.

For example, the line `"IRAM : origin = 0x0, len = 0x40000 "` defines the internal program memory to be a region called IRAM which starts at byte address `0x00000000`, and has a length of `0x00040000` bytes which is 256 Kbytes.

The C compiler puts data and program code into named sections. Named sections can also be created by the programmer in assembly source code. The SECTIONS portion of the linker command file tells the linker how to place sections into defined memory regions. The standard conventions are to place program instructions in the `.text` section, initialized constants in the `.const` section, global and static variables in the `.bss` section, initialization tables for variables and constants in the `.cinit` section, local variables in the `.stack` section, and buffers for C I/O functions in the `.cio` section. Data from assembly programs can be put in the `.data` section. C does not use the `.data` section.

3.2 MP3 Player⁶

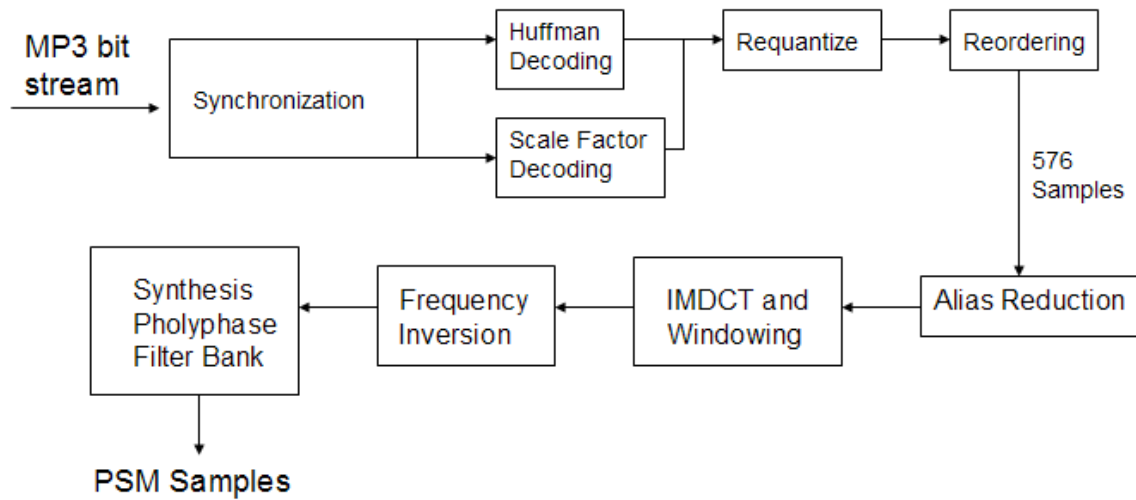


Figure 3.1 Functional blocks associated with MP3 player

The first block is the *Synchro-nization block*⁷. This block serves the purpose of receiving the incoming bit stream, extracting certain information from it and passing the extracted information to the succeeding blocks. This information consists of the Header Information, the CRC Information, and the Side Information. The Header Information specifies the type of the MP3 file, the bit rate used for transmission, the sampling frequency, and the nature of the audio.

The *Scale Factor Decoding block*⁸ decodes the scale factors to allow the reconstruction of the original audio signal. Scale factors are used to mask out the quantization noise during encoding by boosting the sound frequencies that are more perceptible to human ears.

The Huffman decoding is the most critical block in the decoding process. The *Huffman Decoding block*⁹ consists of two components: Huffman Information Decoding and Huffman Decoding. Huffman Information Decoding uses the Side Information to set up the fields for Huffman Decoding. It acts as a controller and controls the decoding process by providing information on Huffman table selection, codeword region, and how many frequency lines are decoded. This decoding is illustrated in Figure 2.3.

⁶ CD included

⁷ Appendix-1 for source code

⁸ Appendix-2 for source code

⁹ Appendix-3 for source code

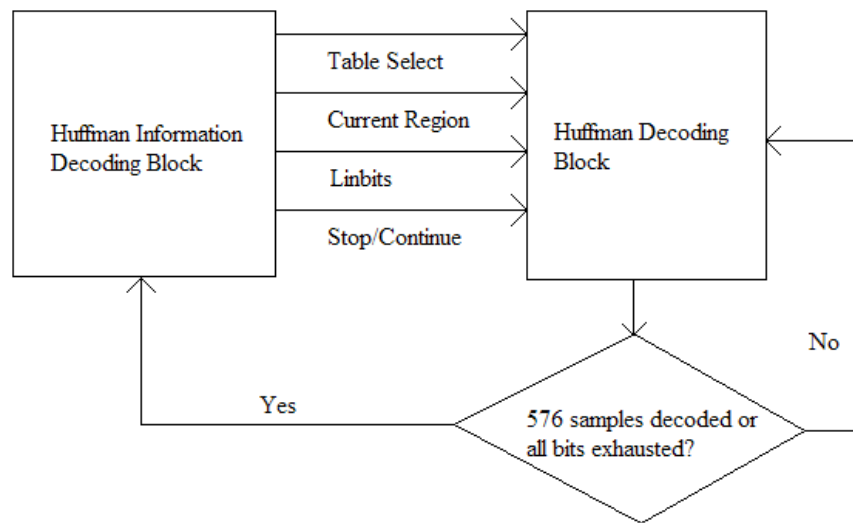


Figure 3.2 Huffman information decoding block as a controller

The MP3 encoder incorporates a quantizer block that quantizer the frequency lines so that they can be Huffman coded. The output of the quantizer is multiplied by the scale factors to suppress the quantization noise. The function of the *requantizer block*¹⁰ is to combine the outputs of the Huffman Decoder and Scale Factor Decoder blocks, generating the original frequency lines.

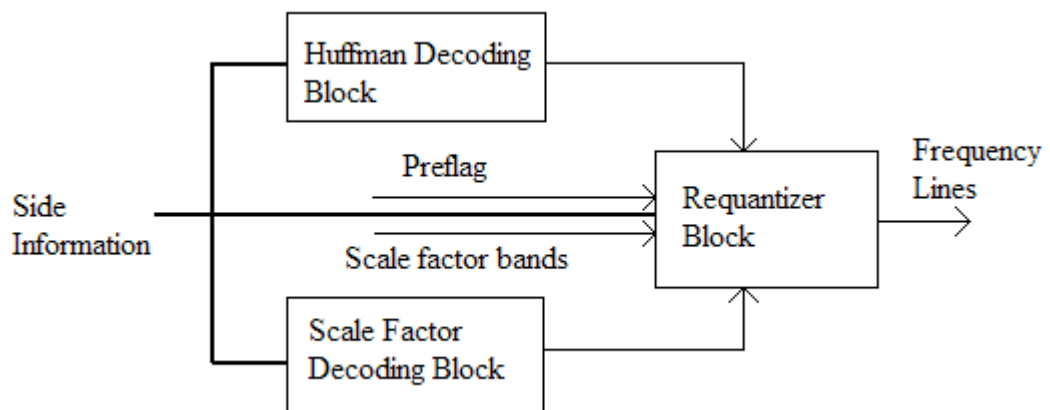


Figure 3.3 Block diagram of requantization block

Huffman coding gives better results if its inputs are ordered in an increasing order or have similar values. This is the reason the frequency lines are ordered in increasing order of frequency during encoding as values closer in frequency have similar values. The main

¹⁰ Appendix-4 for source code

task of the reordering block is to search for short windows to reorder the frequency lines. The output of the requantizer, for short windows, gives 18 samples in a sub-band. These samples are not dependent on the window used. The reordering block simply picks up the samples and reorders them in groups of six for each window, thereby generating them as they were before reordering.

During the encoding process, the PCM samples are filtered into subbands using bandpass filters. However, due to the non-ideal nature of the bandpass filters, aliasing effects occur. To minimize aliasing artifacts, windowing is done after the MDCT block. That is why during the decoding process, an alias reduction block is used to generate the frequency lines similar to those generated by the MDCT in the encoder. This block adds the alias components to each frequency line to produce the original frequency lines.

The Inverse MDCT ($IMDCT^{11}$) block is responsible for generating samples which serve as the input to the Polyphase filter. The IMDCT takes in 18 input values and generates 36 output values per subband in each granule. The reason for generating twice as many output values is that the IMDCT contains a 50% overlap. This means that only 18 out of 36 values are unique, and the remaining 18 values are generated by a data copying operation.

Finally, a Polyphase filter bank is used to transform the 32 subbands, each with 18 time samples from every granule, to 18 bands of 32 PCM samples. PCM is a standard format of storing digital data in uncompressed format, CD audio being the prime example. PCM samples are defined depending on the sampling frequency and bitrate. A higher sampling frequency implies that higher frequencies are present and a higher bitrate produces a better resolution. Generally, CD audio uses 16 bits at 44.1 kHz.

¹¹ Appendix-5 for source code

3.3 Build, Load, Run¹²

1. Copy the directory, digfil, on CD to C:

2. Create the folder C:\c6713dsk

3. Copy the following files from the CD folder c6713dsk to C:\c6713dsk

convoll.sa, convolve.sa, dsk6713_RTDX.cmd, dskstart32.c, edma_sines.c

4. Create the folder C:\c6713dsk\dsk6713bsl32

a. Copy the folder C:\CCStudio_v3.1\C6000\dsk6713\lib to C:\c6713dsk\dsk6713bsl32 and then rename dsk6713bsl.lib in C:\c6713dsk\dsk6713bsl32\lib to dsk6713bsl32.lib.

b. Copy the folder C:\CCStudio_v3.1\C6000\dsk6713\include to C:\c6713dsk\dsk6713bsl32

c. Copy DSKintr.h and regs.h on CD disk in the directory c6713dsk\dsk6713bsl32\include to C:\c6713dsk\dsk6713bsl32\include

d. Create the folder C:\c6713dsk\dsk6713bsl32\sources

e. Copy the files dsk6713_opencodec.c, DSKintr.c, and intvecs.asm on this disk in the directory c6713dsk\dsk6713bsl32\sources to C:\c6713dsk\dsk6713bsl32\sources

f. Copy the files dsk6713_opencodec.obj, DSKintr.obj, and intvecs.obj on CD in the directory c6713dsk\dsk6713bsl32\lib to C:\c6713dsk\dsk6713bsl32\lib

5. To add the path to your path variable in Windows XP, click on "start," then "Settings," and then "Control Panel." Double click on the "System" icon and then on the "Advanced" tab. Then click on "Environment Variables" near the bottom of the window. Select the variable "Path" and click on the "Edit" tab. Add the path C:\CCStudio_v3.1\C6000\cgtools\bin to the end of the path variable. It must be preceded by a semi-colon (;). Then click on the "OK" tabs to accept the edited path variable.

¹² CD included

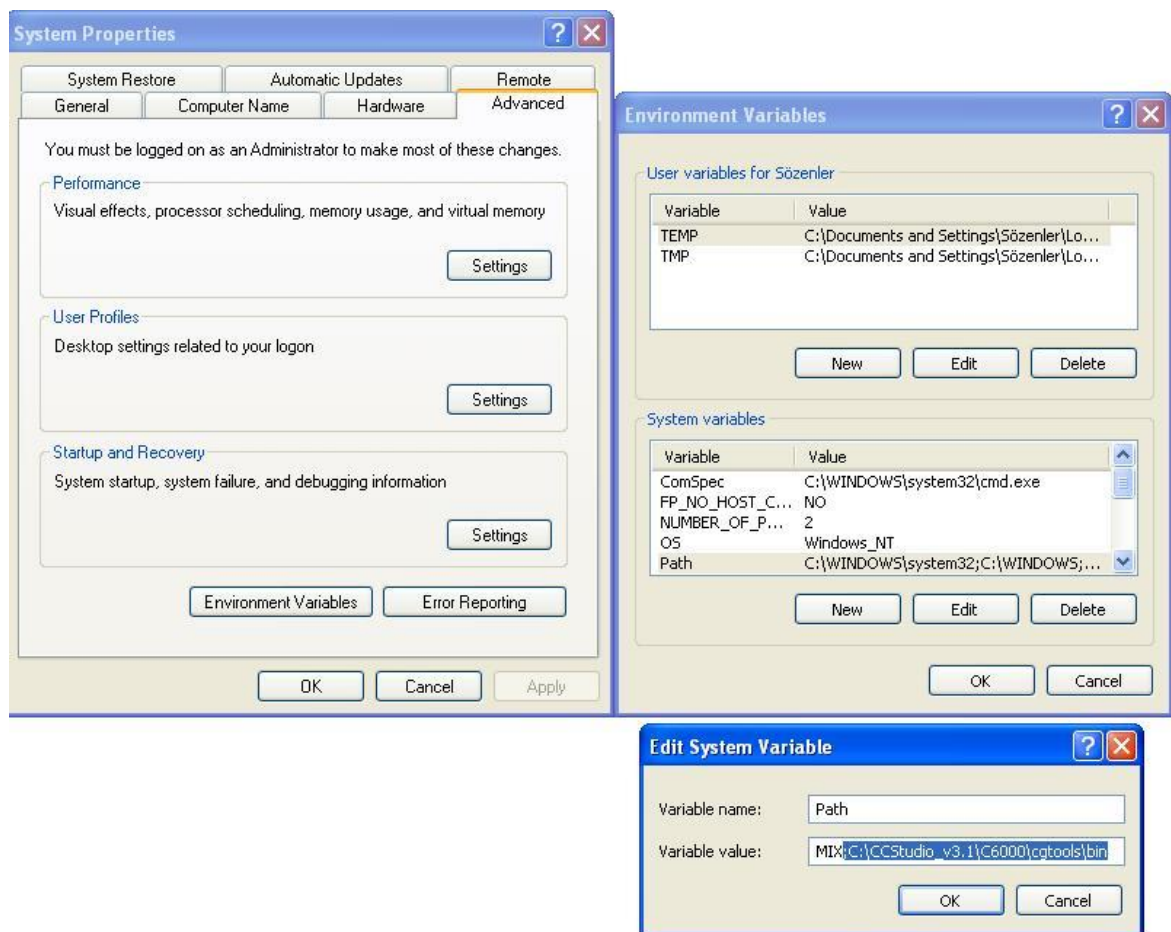
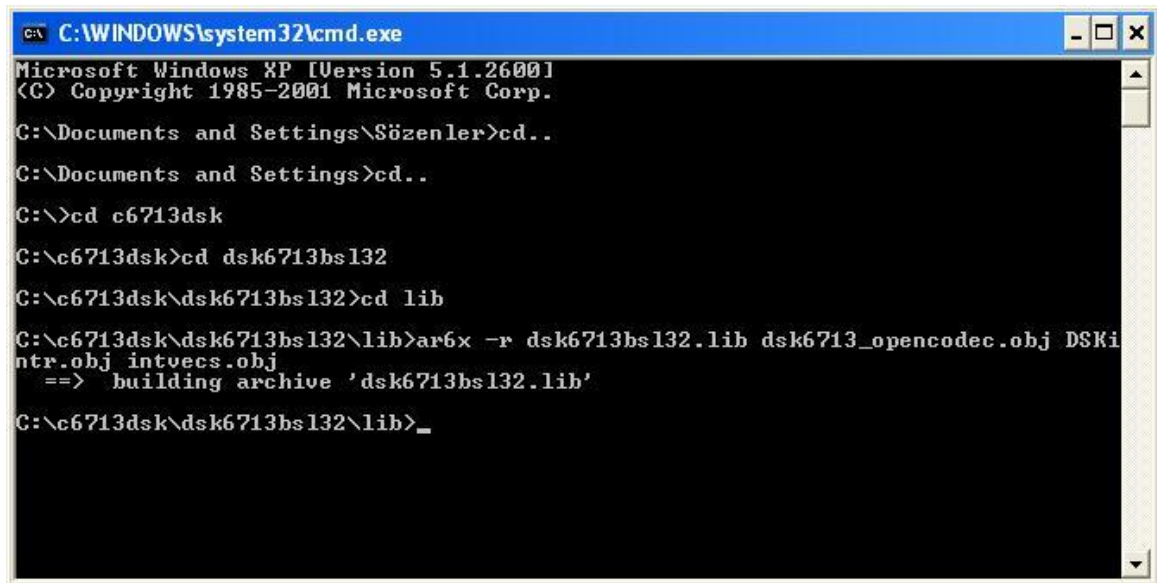


Figure 3.4 Defining Path

Open up a command prompt window and navigate to the directory `C:\c6713dsk\dsk6713bsl32\lib`. Assuming you edited the path variable, run the following command to modify the BSL: `ar6x -r dsk6713bsl32.lib dsk6713_opencodec.obj DSKintr.obj intvecs.obj`



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Süzenler>cd..
C:\Documents and Settings>cd..
C:\>cd c6713dsk
C:\c6713dsk>cd dsk6713bsl32
C:\c6713dsk\dsk6713bsl32>cd lib
C:\c6713dsk\dsk6713bsl32\lib>ar6x -r dsk6713bsl32.lib dsk6713_opencodec.obj DSKi
ntr.obj intvecs.obj
==> building archive 'dsk6713bsl32.lib'
C:\c6713dsk\dsk6713bsl32\lib>_
```

Figure 3.5 Running a Setup from Command Line

The -r option causes ar6x to replace any module in the library having the same name as one in the list by the one in the list. If a name in the list is not found in the library, the file in the list is added to the library. After completing all these steps, the c6713dsk directory tree should look like:

```
c6713dsk
  convol1.sa
  convolve.sa
  dsk6713_RTDX.cmd
  dskstart32.c
  edma_sines.c
  dsk6713bsl32
    include
      dsk6713.h
      dsk6713_aic23.h
      dsk6713_dip.h
      dsk6713_flash.h
      dsk6713_led.h
      DSKintr.h
      regs.h
      IO.h
    lib
      dsk6713bsl32.lib
      dsk6713bsl.zip (Contains sources for original BSL library)
      DSKintr.obj
      intvecs.obj
      dsk6713_opencodec.obj
```

sources

```
dsk6713_opencodec.c
DSKintr.c
intvecs.asm
convol1.sa
convolve.sa
dsk6713_RTDX.cmd
dskstart32.c
edma_sines.c
bootloader.c
fft.c
player.c
synth_full.c
```

CCS on the PC is used to generate programs for the C6713, load them into the DSP memory, run them, and monitor program execution.



Figure 3.6 CCS Icons

C6713 DSK CCStudio v3.1 icon starts up the CCS run time machine for C6713 only. *6713 DSK Diagnostics Utility v3.1* is the test center for checking connection and testing every main elements on board. *Setup CCStudio v3.1* sets up the DSK board that will be used and loads the necessary device drivers¹³. *CCStudio 3.1* can be used as a text editor independent of the DSK board used.

¹³ Special software modules called device drivers, are used to communicate with the target. Each driver file defines a specific target configuration.

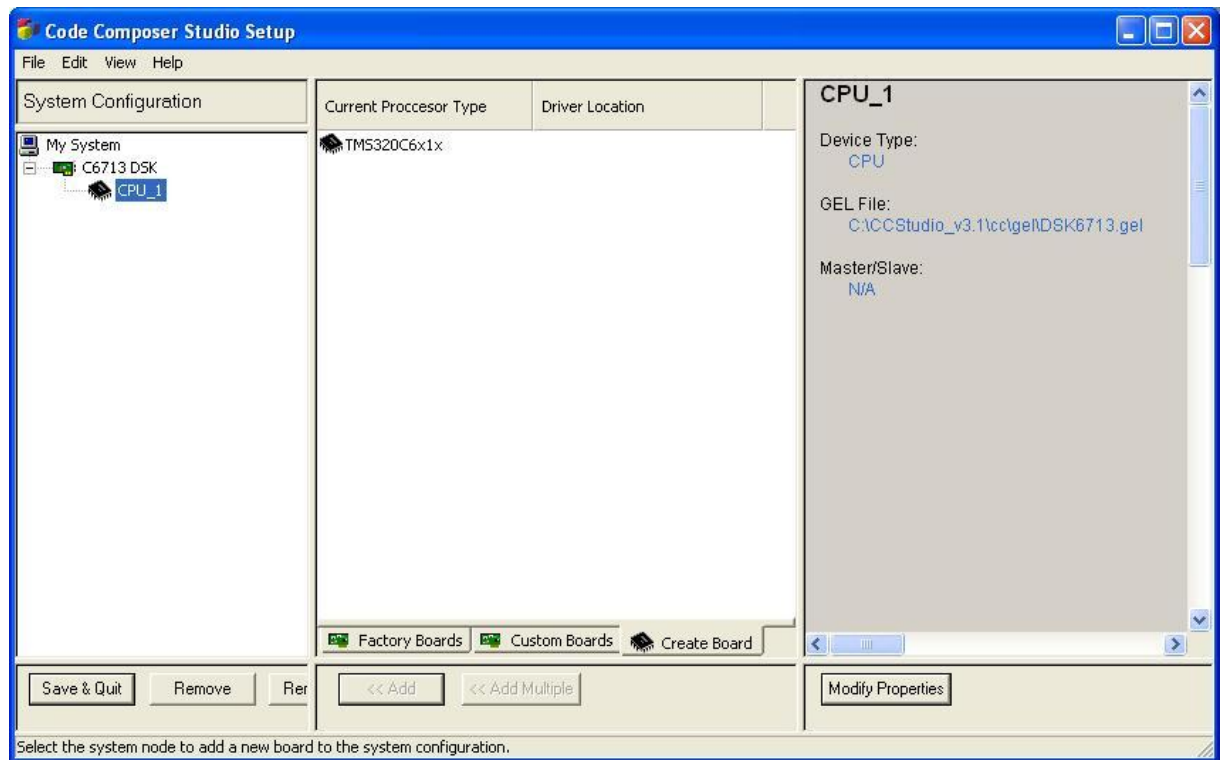


Figure 3.7 Device Setup

When we start diagnostic test, it first checks the USB connection.

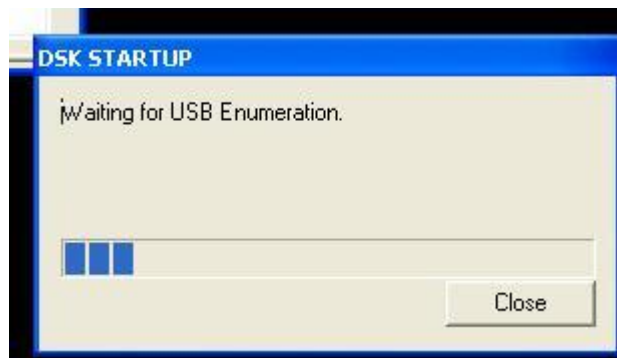


Figure 3.8 USB connection

Then we run the overall diagnostic test:

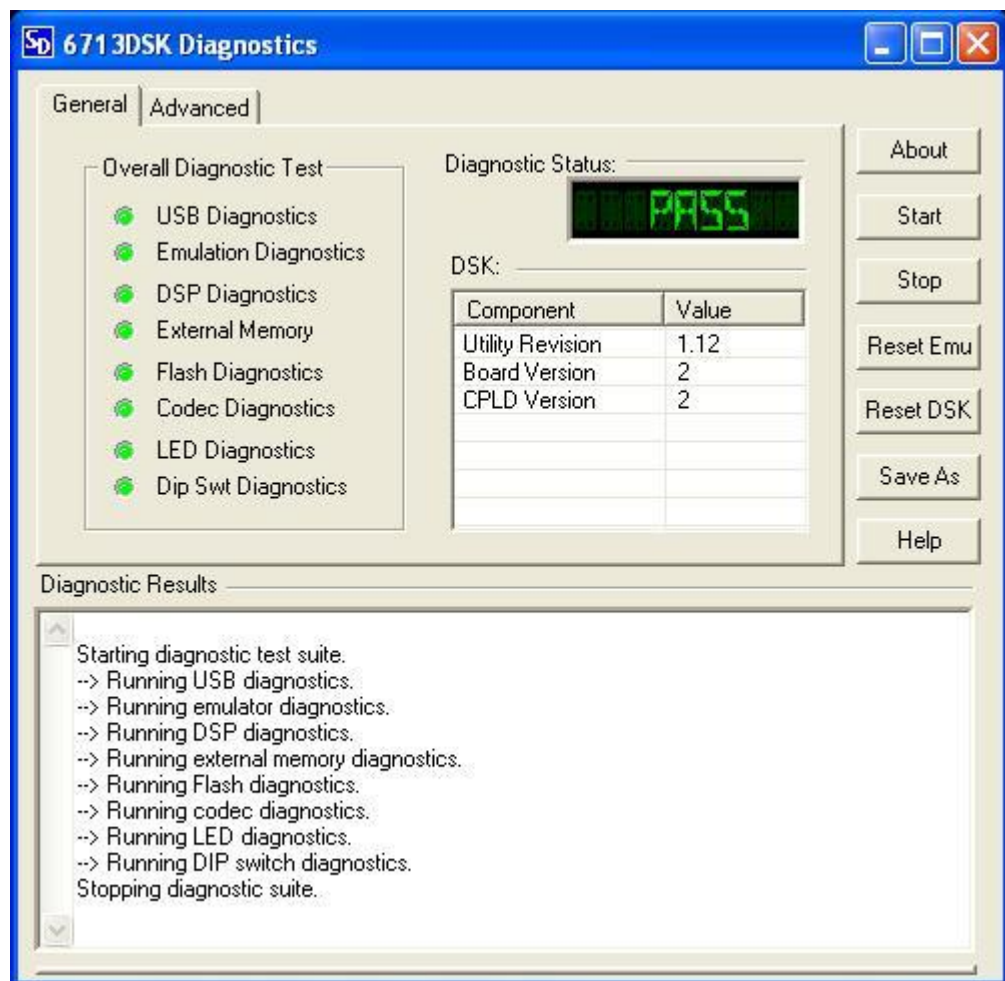


Figure 3.9 General Diagnostic Test

In Advanced tab we can check other peripherals such as LEDs, Memory, Codec and DSP.

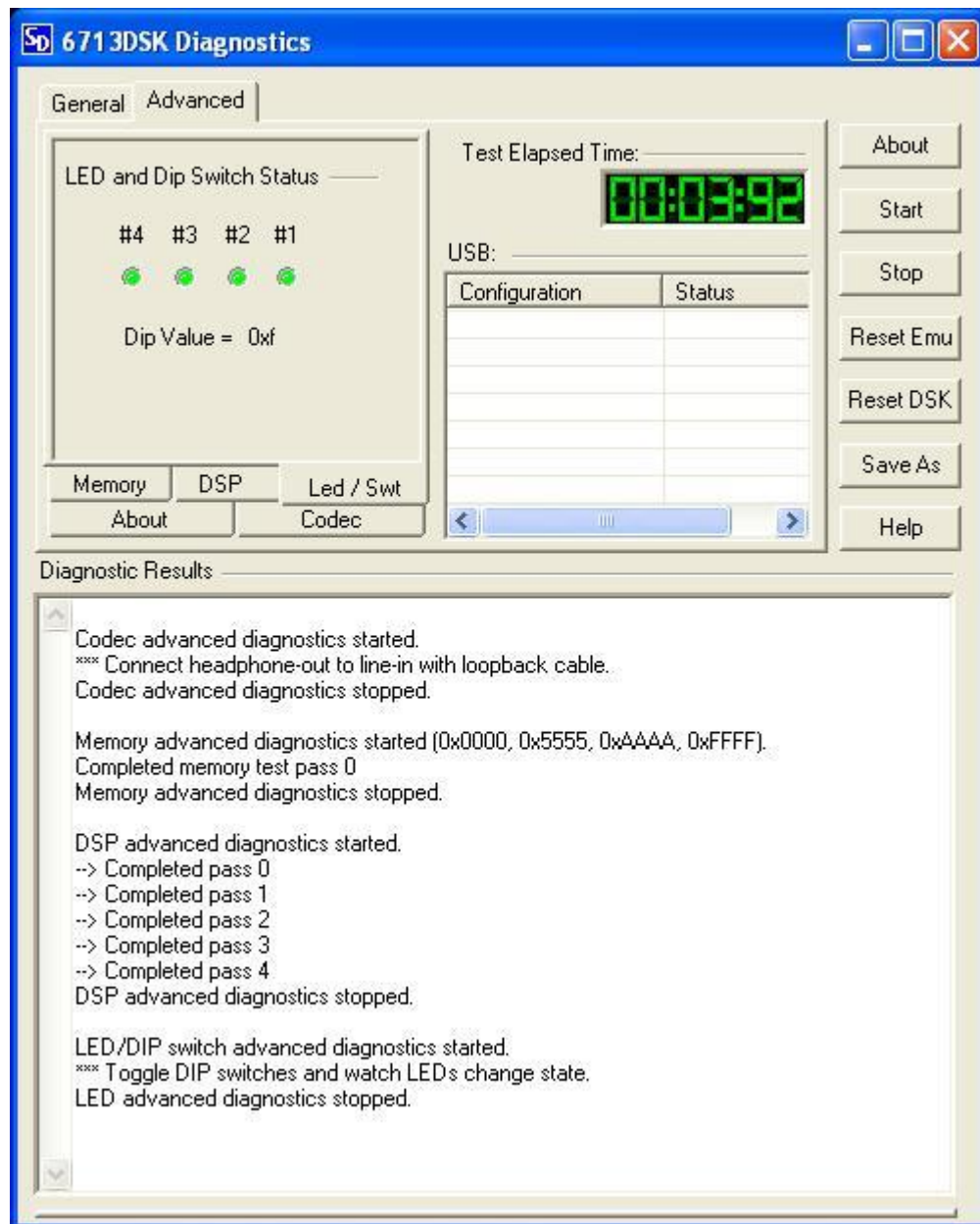


Figure 3.10 Advanced Diagnostic Test

After starting *C6713 DSK CCStudio v3.1* we have to connect target from Debug menu in order to start a project.

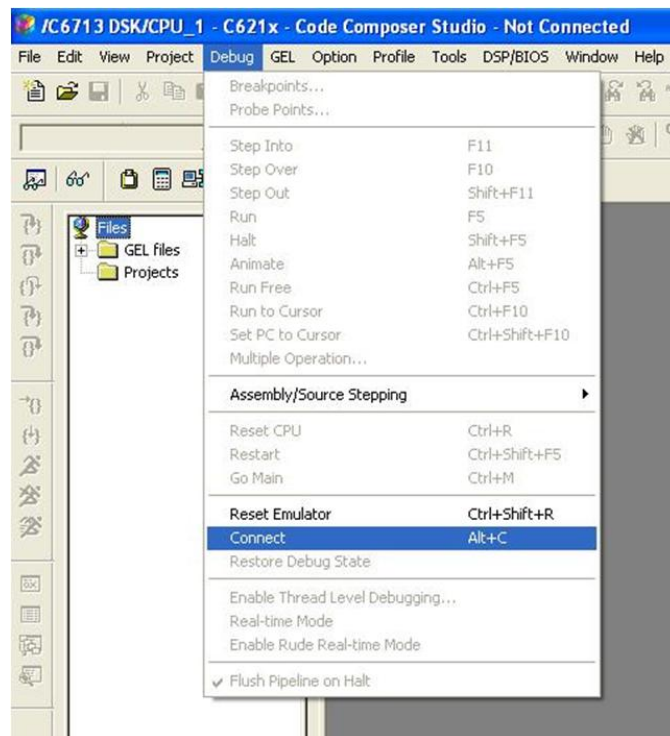


Figure 3.11 Connecting Target DSK

From Project → New window, I specified the project name, location, project type and target.

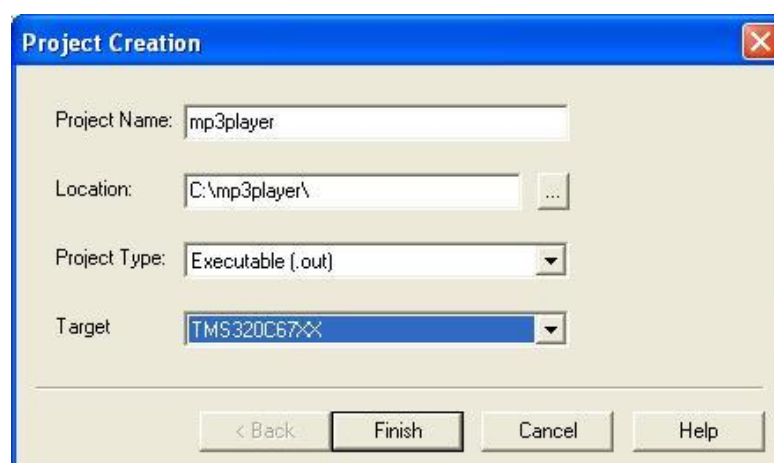


Figure 3.12 Creating a Project

I created all the source files separately before creating a project. So, after creating the project I add files from Project → Add Files to Project... menu. All file types are included in this manner and I added all source, header, linker command and library files from here.

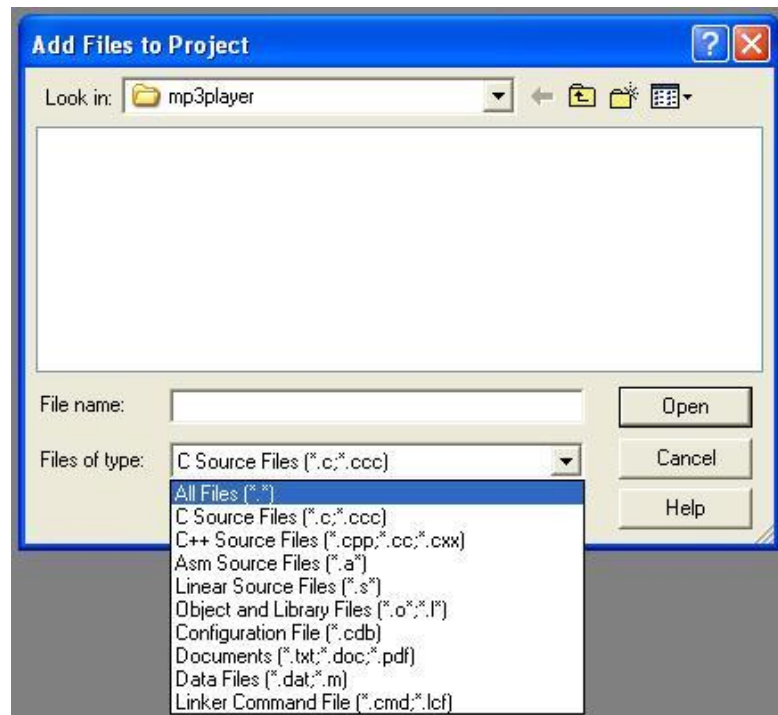


Figure 3.13 Adding Files to Project

As can be seen, after adding the files there is still a chance of updating codes with CCS text editor.

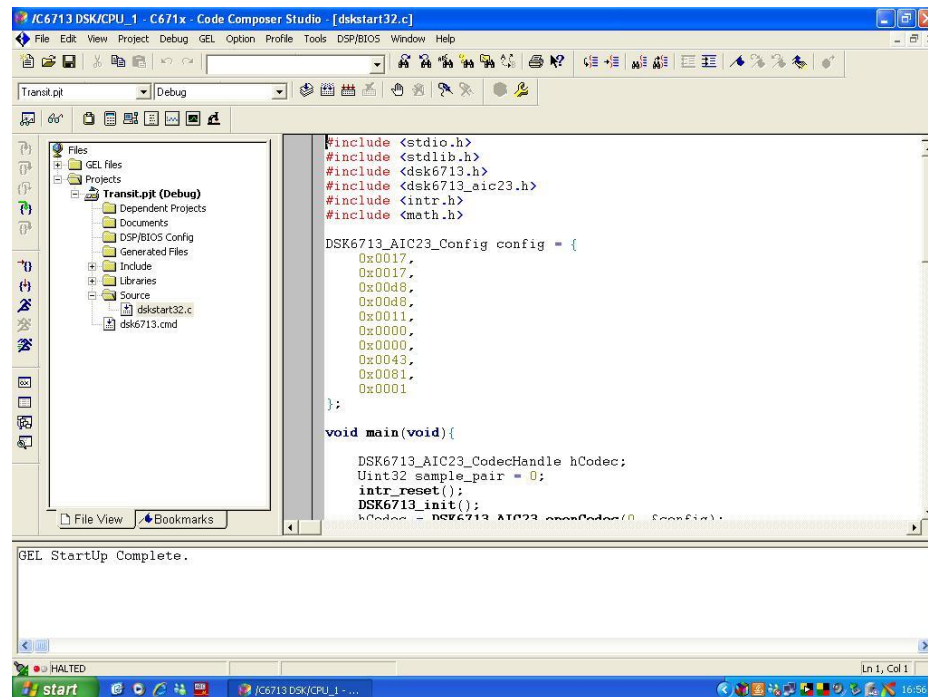


Figure 3.14 Text Editor Screen and Project Tree

After adding files to the project, we have to make build options before building the project because CCS will create the .out file accordingly. From Project → Build Options I entered the following:

Compiler → Basic

Target Version: 671x (-mv6710)

Generate Debug Info: Full Symbolic Debug (-g)

Opt Speed vs Size: Speed Most Critical (no ms)

Opt Level: None

Program Level Opt: None

Compiler → Preprocessor

Include Search Path (-i): ;c:\c6713dsk\dsk6713bsl32\include

Define Symbols (-d): CHIP_6713

Preprocessing: None

Compiler → Files

Asm Directory: "a directory in my workspace"

Obj Directory: "a directory in my workspace"

Linker → Basic

Output Filename (-o): MP3player.out (mostly the project name)

Map Filename (-m): MP3player.map (mostly the project name)

Autoinit Model: Run-time auto initialization

Make sure to add to the project the linker command file: c:\c6713dsk\dsk6713.cmd and the library c:\c6713dsk\dsk6713bsl32\lib\dsk6713bsl32.lib

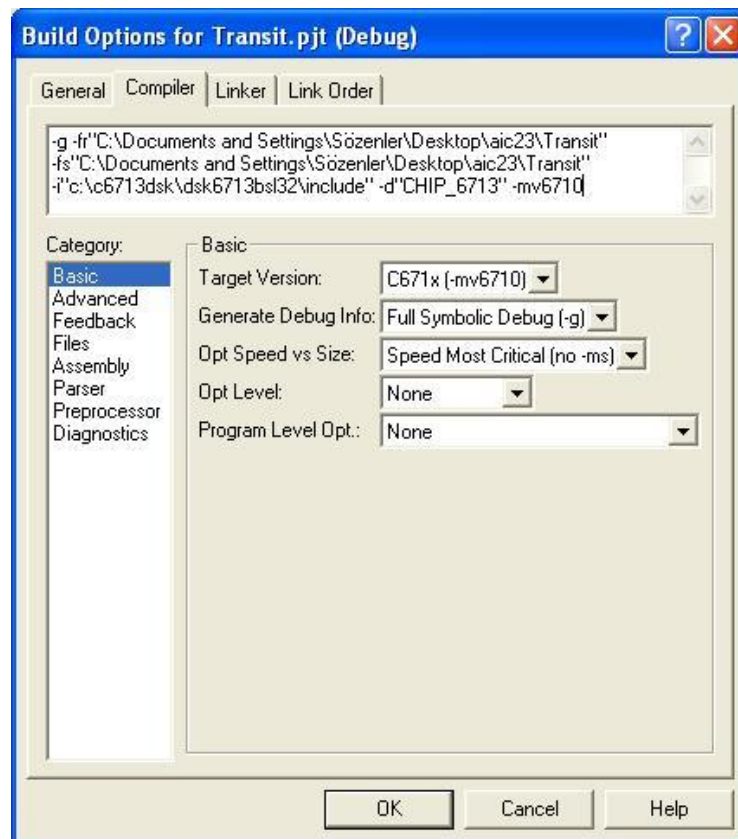


Figure 3.15 Build Options

After completing options, I have rebuilt the project from Project → Rebuild menu and the results were error free.

```
[Linking...] "C:\CCStudio_v3.1\C6000\cgtools\bin\cl6x" -@"Debug.lkf"  
<Linking>  
  
Build Complete,  
  0 Errors, 0 Warnings, 0 Remarks.
```

Figure 3.16 Building Results

After clear build, CCS created the .out file and I loaded this from File → Load Program menu to DSP in order to run on it.

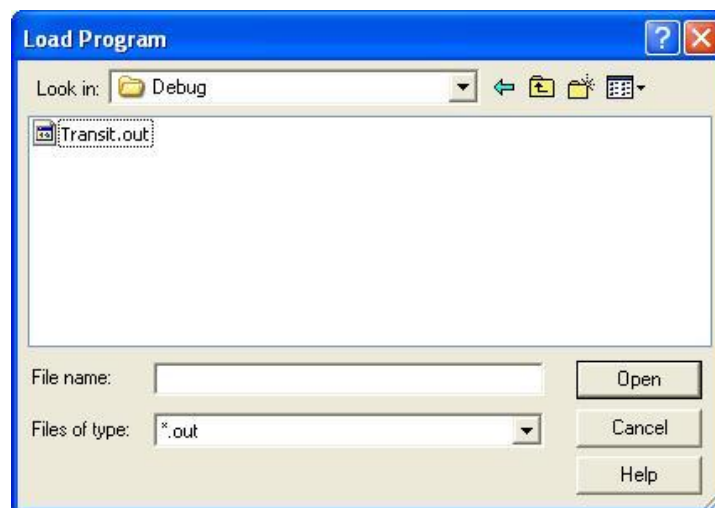


Figure 3.17 Loading .out

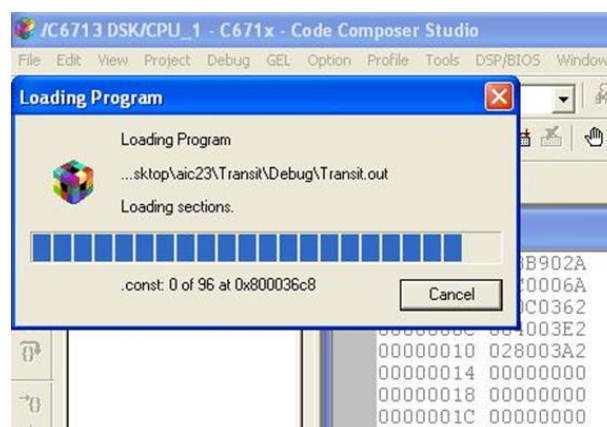


Figure 3.18 Loading Program



Figure 3.19 Running Program

While running the program on DSP, I could have halted it whenever I wanted and see the memory pointer where it stopped.

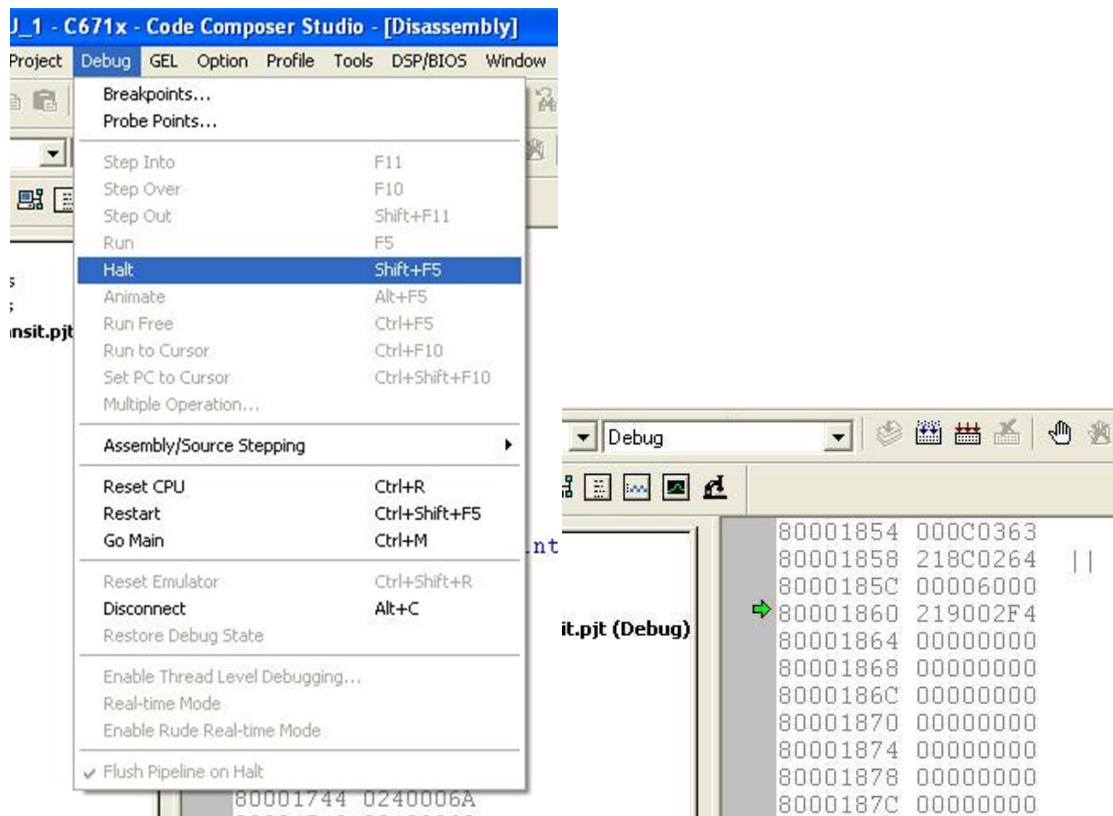


Figure 3.20 Halting Programs, Memory State When Halted

4. CONCLUSION

In this project my aim was to design an MP3 player by using C6713 high performance, floating-point DSK and programming it using Code Composer Studio.

Sound data (.MP3) comes from AIC23 audio codec's line-in jack, passes through ADC and is sent to the C6713 CPU via McBSP1 port through its internal buffer registers. After MP3 data stream is decoded in DSP it is sent back to DAC through McBSP1 port again and the result comes out from the line-out jack of the codec. Headphone output is weaker than the line-out output because it affects human ear directly. But still, when we connect a speaker to the line out the output is weak if the speaker does not have any external power supply.

A CD is included with this documentation which contains the header, library and source files. The header files for this project were downloaded from the Texas Instruments official web site. The C source codes, except FFT and Synth_full files, were developed within the project. FFT and Synth_full C files are ready-to-use packets easily found on internet and need not to be programmed unless a special application is needed. To write the codes some already developed applications were searched and found some pseudo-codes which lead me a way [38].

The most time consuming part of the project was debugging. Code Composer Studio is very helpful in this regard and notifies one when an error occurs. For a computer engineer, working with different text editors gives a practical understanding of programming environments. Especially working on a project with more than one source file was very challenging and this difficulty was overcome with the help of "Project" and "Build" utilities of CCS.

A prototype of an MP3 player with two manageable disadvantages was built with this project:

- i) Data is fed through audio codec's line-in
- ii) A host memory is used

The designed MP3 player can also be used as a sound filtering device in addition. The sound streams coming from the line-in jack of the audio codec can be directed to DSP again with the help of McBSP and filtered with an algorithm other than MP3 decoding.

For future work, one can program the JTAG emulator, EMIF, the correct memory mapping and connect a 16 Mbytes memory (for TMS320C6713 max. is 16 Mbytes) in order to have a hand-held device. This necessitates up a lot of architectural and programming effort.

BIBLIOGRAPHY

BOOK SOURCES

A.Tretter, Steven, (2008), Communication System Design Using DSP Algorithms with Laboratory Experiments for the TMS320C6713 DSK, University of Maryland

Kehtarnavaz, Nasser, (2005), **Real-Time Digital Signal Processing: Based on the TMS320C6000, UK: Elsevier, Inc.**

Kehtarnavaz, Nasser, and Kim, Namjin, (2005), Digital Signal Processing System-Level Design Using LabVIEW, UK: Elseveir, Inc. + CR-ROM including all lab files discussed throughout the book

Mano, M.Morris, (Third Edition), Computer System Architecture, LA: Prentice-Hall International, Inc.

Tanenbaum, Andrew S., (2006 Fifth Edition), Structured Computer Organization, NJ: Prentice-Hall International, Inc.

Wakerly, John F., (2001 Third Edition Updated), Digital Design Principles and Practices, LA: Prentice Hall International, Inc.

INTERNET SOURCES

<http://www.ti.com>

<http://www.dspvillage.com>

<http://www.dsprelated.com>

http://sourceforge.net/project/showfiles.php?group_id=196745

TUTORIAL SOURCES

Texas Instruments Tutorial, “TMS320C6713 Floating-Point Digital Signal Processor”, (December 2001 – Revised November 2005), SPRS186L

Texas Instruments Tutorial, “TMS320 DSP Product Family Glossary”, (February 1998), SPRU258A

Texas Instruments Tutorial, “Code Composer Studio Development Tools v3.2 Getting Started Guide”, (March 2006), SPRU509G

Texas Instruments Tutorial, “TMS320C6000 Instruction Set Simulator Technical Reference Manual”, (April 2007), SPRS600I

Texas Instruments Tutorial, “How to Begin Development Today With the TMS320C6713 Floating-Point DSP”, (October 2002), SPRA809A

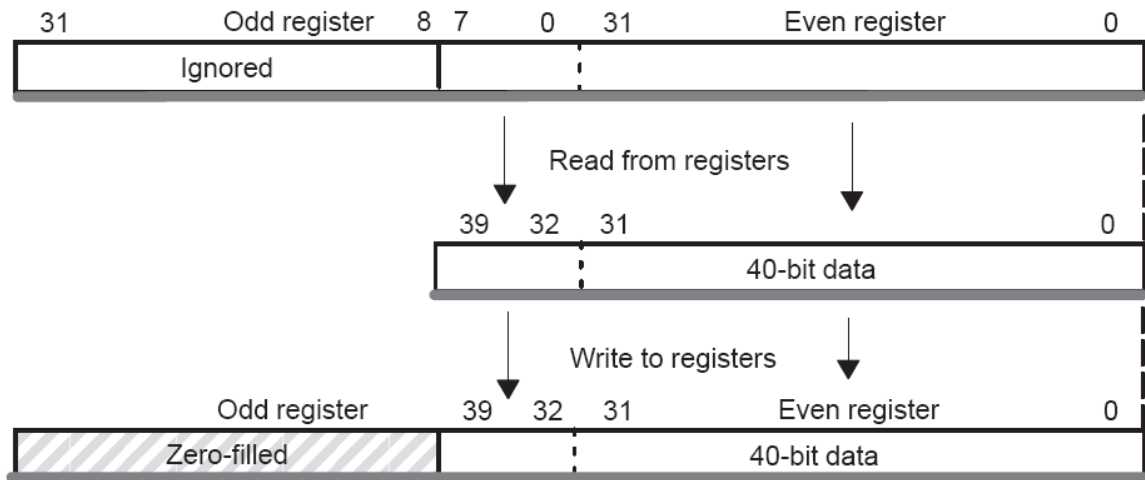
Texas Instruments Tutorial, “TMS320C6713 Hardware Designer’s Resource Guide”, (July 2004), SPRAA33

APPENDIX

APP-1 Special Purpose registers

Abbreviation	Register Name	Description
AMR	Addressing mode register	Specifies whether to use linear or circular addressing for each of eight registers; also contains sizes for circular addressing
CSR	Control status register	Contains the global interrupt enable bit, cache control bits, and other miscellaneous control and status bits
IFR	Interrupt flag register	Displays status of interrupts
ISR	Interrupt set register	Allows manually setting pending interrupts
ICR	Interrupt clear register	Allows manually clearing pending interrupts
IER	Interrupt enable register	Allows enabling/disabling of individual interrupts
ISTP	Interrupt service table pointer	Points to the beginning of the interrupt service table
IRP	Interrupt return pointer	Contains the address to be used to return from a maskable interrupt
NRP	Nonmaskable interrupt return pointer	Contains the address to be used to return from a nonmaskable interrupt
PCE1	Program counter, E1 phase	Contains the address of the fetch packet that is in the E1 pipeline stage

APP-2 General purpose registers



A0, A1, B0, B1 and B2 used as conditional registers

A4 – A7 and B4 – B7 used for circular addressing

A0 – A9 and B0 – B9 (except B3) are temporary registers

Any of A10 – A15 and B10 – B15 used are saved and later restored before returning from sub-routine.

A 40-bit data value can be contained across a register pair.

Similarly 64-bit values can be kept.

APP-3 L2 Cache registers

HEX ADDRESS RANGE	ACRONYM	REGISTER NAME
0184 0000	CCFG	Cache configuration register
0184 4000	L2WBAR	L2 writeback base address register
0184 4004	L2WWC	L2 writeback word count register
0184 4010	L2WIBAR	L2 writeback-invalidate base address register
0184 4014	L2WIWC	L2 writeback-invalidate word count register
0184 4020	L1PIBAR	L1P invalidate base address register
0184 4024	L1PIWC	L1P invalidate word count register
0184 4030	L1DWIBAR	L1D writeback-invalidate base address register
0184 4034	L1DWIWC	L1D writeback-invalidate word count register
0184 5000	L2WB	L2 writeback all register
0184 5004	L2WBINV	L2 writeback-invalidate all register
0184 8200	MAR0	Controls CE0 range 8000 0000 – 80FF FFFF
0184 8204	MAR1	Controls CE0 range 8100 0000 – 81FF FFFF
0184 8208	MAR2	Controls CE0 range 8200 0000 – 82FF FFFF
0184 820C	MAR3	Controls CE0 range 8300 0000 – 83FF FFFF
0184 8240	MAR4	Controls CE1 range 9000 0000 – 90FF FFFF
0184 8244	MAR5	Controls CE1 range 9100 0000 – 91FF FFFF
0184 8248	MAR6	Controls CE1 range 9200 0000 – 92FF FFFF
0184 824C	MAR7	Controls CE1 range 9300 0000 – 93FF FFFF
0184 8280	MAR8	Controls CE2 range A000 0000 – A0FF FFFF
0184 8284	MAR9	Controls CE2 range A100 0000 – A1FF FFFF
0184 8288	MAR10	Controls CE2 range A200 0000 – A2FF FFFF
0184 828C	MAR11	Controls CE2 range A300 0000 – A3FF FFFF
0184 82C0	MAR12	Controls CE3 range B000 0000 – B0FF FFFF
0184 82C4	MAR13	Controls CE3 range B100 0000 – B1FF FFFF
0184 82C8	MAR14	Controls CE3 range B200 0000 – B2FF FFFF
0184 82CC	MAR15	Controls CE3 range B300 0000 – B3FF FFFF
0184 82D0 – 0185 FFFF	–	Reserved

APP-4 Memory map address ranges of the C6713

MEMORY BLOCK DESCRIPTION	BLOCK SIZE (BYTES)	HEX ADDRESS RANGE
Internal RAM (L2)	192K	0000 0000 – 0002 FFFF
Internal RAM/Cache	64K	0003 0000 – 0003 FFFF
Reserved	24M – 256K	0004 0000 – 017F FFFF
External Memory Interface (EMIF) Registers	256K	0180 0000 – 0183 FFFF
L2 Registers	128K	0184 0000 – 0185 FFFF
Reserved	128K	0186 0000 – 0187 FFFF
HPI Registers	256K	0188 0000 – 018B FFFF
McBSP 0 Registers	256K	018C 0000 – 018F FFFF
McBSP 1 Registers	256K	0190 0000 – 0193 FFFF
Timer 0 Registers	256K	0194 0000 – 0197 FFFF
Timer 1 Registers	256K	0198 0000 – 019B FFFF
Interrupt Selector Registers	512	019C 0000 – 019C 01FF
Device Configuration Registers	4	019C 0200 – 019C 0203
Reserved	256K – 516	019C 0204 – 019F FFFF
EDMA RAM and EDMA Registers	256K	01A0 0000 – 01A3 FFFF
Reserved	768K	01A4 0000 – 01AF FFFF
GPIO Registers	16K	01B0 0000 – 01B0 3FFF
Reserved	240K	01B0 4000 – 01B3 FFFF
I2C0 Registers	16K	01B4 0000 – 01B4 3FFF
I2C1 Registers	16K	01B4 4000 – 01B4 7FFF
Reserved	16K	01B4 8000 – 01B4 BFFF
McASP0 Registers	16K	01B4 C000 – 01B4 FFFF
McASP1 Registers	16K	01B5 0000 – 01B5 3FFF
Reserved	160K	01B5 4000 – 01B7 BFFF
PLL Registers	8K	01B7 C000 – 01B7 DFFF
Reserved	264K	01B7 E000 – 01BB FFFF
Emulation Registers	256K	01BC 0000 – 01BF FFFF
Reserved	4M	01C0 0000 – 01FF FFFF
QDMA Registers	52	0200 0000 – 0200 0033
Reserved	16M – 52	0200 0034 – 02FF FFFF
Reserved	720M	0300 0000 – 2FFF FFFF
McBSP0 Data Port	64M	3000 0000 – 33FF FFFF
McBSP1 Data Port	64M	3400 0000 – 37FF FFFF
Reserved	64M	3800 0000 – 3BFF FFFF
McASP0 Data Port	1M	3C00 0000 – 3C0F FFFF
McASP1 Data Port	1M	3C10 0000 – 3C1F FFFF
Reserved	1G + 62M	3C20 0000 – 7FFF FFFF
EMIF CE0†	256M	8000 0000 – 8FFF FFFF
EMIF CE1†	256M	9000 0000 – 9FFF FFFF
EMIF CE2†	256M	A000 0000 – AFFF FFFF
EMIF CE3†	256M	B000 0000 – BFFF FFFF
Reserved	1G	C000 0000 – FFFF FFFF

APP-5 McBSP0 and McBSP1 registers

HEX ADDRESS RANGE		ACRONYM	REGISTER DESCRIPTION
McBSP0	McBSP1		
018C 0000	0190 0000	DRRx	McBSPx data receive register via Configuration Bus The CPU and EDMA controller can only read this register; they cannot write to it.
3000 0000 – 33FF FFFF	3400 0000 – 37FF FFFF	DRRx	McBSPx data receive register via Peripheral Data Bus
018C 0004	0190 0004	DXRx	McBSPx data transmit register via Configuration Bus
3000 0000 – 33FF FFFF	3400 0000 – 37FF FFFF	DXRx	McBSPx data transmit register via Peripheral Data Bus
018C 0008	0190 0008	SPCRx	McBSPx serial port control register
018C 000C	0190 000C	RCRx	McBSPx receive control register
018C 0010	0190 0010	XCRx	McBSPx transmit control register
018C 0014	0190 0014	SRGRx	McBSPx sample rate generator register
018C 0018	0190 0018	MCRx	McBSPx multichannel control register
018C 001C	0190 001C	RCERx	McBSPx receive channel enable register
018C 0020	0190 0020	XCERx	McBSPx transmit channel enable register
018C 0024	0190 0024	PCRx	McBSPx pin control register
018C 0028 – 018F FFFF	0190 0028 – 0193 FFFF	–	Reserved

AUTOBIOGRAPHY

I was born in February 9, 1987 in Istanbul. I took my high school education in Eyuboglu High School and enter the Computer Engineering department of Doğuř University for my undergraduate education. My tendency in this area is through programming the brains of embedded systems other than interface programming. I have special interest on working computer architecture; microprocessors and microcontrollers and programming them in Assembly. So far, I worked on Intel MCS-51 family (model 8051) and Motorola HC08 family (model GP32). I am going to have my master degree in Istanbul Technical University, Physics Engineering Department.