

MICROCONTROLLERS AND PROGRAMMABLE DIGITAL SIGNAL PROCESSORS

LAB MANUAL

Subject Code : BES10
Regulations : IARE-R18
Class : M.Tech. Iyr. I Semester(ES)
Branch : ECE

Prepared by

**Mr. K Chaitanya
Assistant Professor
Department of ECE**



**Department of Electronics & Communication Engineering
INSTITUTE OF AERONAUTICAL ENGINEERING**

**(Autonomous)
Dundigal, Hyderabad – 500 043**



INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal - 500 043, Hyderabad

Vision

To bring forth professionally competent and socially sensitive engineers, capable of working across cultures meeting the global standards ethically.

Mission

To provide students with an extensive and exceptional education that prepares them to excel in their profession, guided by dynamic intellectual community and be able to face the technically complex world with creative leadership qualities.

Further, be instrumental in emanating new knowledge through innovative research that emboldens entrepreneurship and economic development for the benefit of wide spread community.

Quality Policy

Our policy is to nurture and build diligent and dedicated community of engineers providing a professional and unprejudiced environment, thus justifying the purpose of teaching and satisfying the stake holders.

A team of well qualified and experienced professionals ensure quality education with its practical application in all areas of the Institute.

Philosophy

The essence of learning lies in pursuing the truth that liberates one from the darkness of ignorance and Institute of Aeronautical Engineering firmly believes that education is for liberation.

Contained therein is the notion that engineering education includes all fields of science that plays a pivotal role in the development of world-wide community contributing to the progress of civilization. This institute, adhering to the above understanding, is committed to the development of science and technology in congruence with the natural environs. It lays great emphasis on intensive research and education that blends professional skills and high moral standards with a sense of individuality and humanity. We thus promote ties with local communities and encourage transnational interactions in order to be socially accountable. This accelerates the process of transfiguring the students into complete human beings making the learning process relevant to life, instilling in them a sense of courtesy and responsibility.



INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal, Hyderabad - 500 043

Electronics & Communication Engineering

Program Outcomes	
PO1	Apply advanced level knowledge, techniques, skills and modern tools in the field of Embedded Systems and sub areas IoT, Processor technology and Storage technology.
PO2	Function on multidisciplinary environments by working cooperatively, creatively and responsibly as a member of a team.
PO3	Respond to global policy initiatives and meet the emerging challenges with sustainable technological solutions in the field of electronic product designing.
PO4	Demonstrate the importance of embedded technologies and design new innovative products for solving society relevant problems
PO5	Write and present a substantial technical report / document.
PO6	Independently carry out research / investigation and development work to solve practical problems.
PO7	Recognize the need to engage in lifelong learning through continuing education and research.

INDEX

S. No.	List of Experiments	Page No.
1	Blink an LED with software delay, delay generated using the SysTick timer.	
2	System clock real time alteration using the PLL modules.	
3	Control intensity of an LED using PWM implemented in software and hardware	
4	Control an LED using switch by polling method, by interrupt method and flash the LED once.	
5	Take analog readings on rotation of rotary potentiometer connected to an ADC channel.	
6	Temperature indication on an RGB LED.	
7	Evaluate the various sleep modes by putting core in sleep and deep sleep modes.	
8	System reset using watchdog timer in case something goes wrong.	
9	Write a program to interface ADC and DAC with PSOC	
10	To develop an assembly code and C code to compute Euclidian distance between any two points.	
11	To develop assembly and C code for implementation of convolution operation.	
12	To design and implement filters in C to enhance the features of given input sequence/signal	



INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal - 500 043, Hyderabad

Certificate

*This is to Certify that it is a bonafied record of Practical work done by
Sri/Kum. _____ bearing the Roll No.
_____ of _____ Class
_____ Branch in the
_____ laboratory during the Academic year
_____ under our supervision.*

Head of the Department

Lecture In-Charge

External Examiner

Internal Examiner



INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal - 500 043, Hyderabad

ELECTRONICS AND COMMUNICATION ENGINEERING

COURSE OBJECTIVE:

The course should enable the students to:

- I. Demonstrate Keil IDE tool for development of Embedded system.
- II. Program the interfacing of various devices with ARM using Embedded C.
- III. Develop program for implementation of interrupts and serial communications.
- IV. Implementation of digital signal processing algorithms in MATLAB and C.
- V. Understand the real-time operation of digital filters

COURSE OUTCOMES:

Upon the completion of EMBEDDED SYSTEMS LABORATORY the student will be able to:

- A. An ability to design a system, component, or process to meet desired needs within realistic constraints such as economic, environmental, social, political, ethical, health and safety.
 1. Understand Cortex M-3 architecture and basics of programming for ARM7
 2. Design and analyze the 16 -32 bit programming using Cortex M-3 with Various applications
 3. Identity the Embedded C programming in given problem develop suitable application
 4. Understand system peripherals for compact engineering system design and serial bus protocols for system design with communication support.
 5. Understand the basic architecture level and programming using DSP processors
 6. Choose the appropriate programming level for specified I/O devices.
 7. Design and develop various applications using PSOC mixed signal array
 8. Understand and develop Embedded C programming with various applications



INSTITUTE OF AERONAUTICAL ENGINEERING

Dundigal - 500 043, Hyderabad

ELECTRONICS AND COMMUNICATION ENGINEERING

INSTRUCTIONS TO THE STUDENTS

1. Students are required to attend all labs.
2. Students should work individually in the hardware and software laboratories.
3. Students have to bring the lab manual cum observation book, record etc along with them whenever they come for lab work.
4. Should take only the lab manual, calculator (if needed) and a pen or pencil to the work area.
5. Should learn the prelab questions. Read through the lab experiment to familiarize themselves with the components and assembly sequence.
6. Should utilize 3 hours' time properly to perform the experiment and to record the readings. Do the calculations, draw the graphs and take signature from the instructor.
7. If the experiment is not completed in the stipulated time, the pending work has to be carried out in the leisure hours or extended hours.
8. Should submit the completed record book according to the deadlines set up by the instructor.
9. For practical subjects there shall be a continuous evaluation during the semester for 30 sessional marks and 70 end examination marks.
10. Out of 30 internal marks, 15 marks shall be awarded for day-to-day work and 15 marks to be awarded by conducting an internal laboratory test.



INSTITUTE OF AERONAUTICAL ENGINEERING

(Approved by AICTE, New Delhi, Accredited by NBA, New Delhi & Affiliated to JNTU, Hyderabad)

Dundigal, Hyderabad -500 043.

MICROCONTROLLERS AND PROGRAMMABLE DIGITAL SIGNAL PROCESSORS LAB

– Lab Programs List

Submission – 1

Week – 1	Blink an LED with software delay, delay generated using the SysTick timer.
Week – 2	System clock real time alteration using the PLL modules.
Week – 3	Control intensity of an LED using PWM implemented in software and hardware
Week – 4	Control an LED using switch by polling method, by interrupt method and flash the LED once.
Week - 5	Take analog readings on rotation of rotary potentiometer connected to an ADC channel.
Week – 6	Temperature indication on an RGB LED.

Submission – 2

Week – 7	Evaluate the various sleep modes by putting core in sleep and deep sleep modes.
Week – 8	System reset using watchdog timer in case something goes wrong.
Week – 9	Write a program to interface ADC and DAC with PSOC
Week - 10	To develop an assembly code and C code to compute Euclidian distance between any two points.
Week – 11	To develop assembly and C code for implementation of convolution operation.
Week - 12	To design and implement filters in C to enhance the features of given input sequence/signal

**ATTAINMENT OF PROGRAM OUTCOMES
& PROGRAM SPECIFIC OUTCOMES**

Exp. No.	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
1	Blink an LED with software delay, delay generated using the SysTick timer.	PO1, PO2	PSO1
2	System clock real time alteration using the PLL modules.	PO1, PO2	PSO1
3	Control intensity of an LED using PWM implemented in software and hardware	PO1, PO2	PSO1
4	Control an LED using switch by polling method, by interrupt method and flash the LED once.	PO1, PO2	PSO1
5	Take analog readings on rotation of rotary potentiometer connected to an ADC channel.	PO1, PO2, PO3	PSO1, PSO2
6	Temperature indication on an RGB LED.	PO1, PO2, PO3	PSO1
7	Evaluate the various sleep modes by putting core in sleep and deep sleep modes.	PO1, PO2	PSO1
8	System reset using watchdog timer in case something goes wrong.	PO5, PO6	PSO1
9	Write a program to interface ADC and DAC with PSOC	PO5, PO6, PO7	PSO1
10	To develop an assembly code and C code to compute Euclidian distance between any two points.	PO5, PO6	PSO1, PSO2
11	To develop assembly and C code for implementation of convolution operation.	PO5, PO6	PSO2
12	To design and implement filters in C to enhance the features of given input sequence/signal	PO5, PO6, PO7	PSO2

Introduction to C and Assembly Programming

1. Objective

The objective of this lab is to give you a “first foot in the door” exposure to the programming in C and assembly of a program, which when executed by the microcontroller (NXP LPC1768, an ARM Cortex-M3) simply blinks LEDs located on the development board. You also learn how to use the ARM Keil uVision IDE to create projects, build and download them to the board (either Keil MCB1700 or Embest EMLPC1700).

2. Pre-lab Preparation

Optional (but encouraged) You should plan to work on your own computer at home a lot during this semester. You should install the main software (evaluation version) we will use in this course: the Microcontroller Development Kit (MDKARM), which supports software development for and debugging of ARM7, ARM9, Cortex-M, and Cortex-R4 processor-based devices. Download it from ARM's website [1] and install it on your own computer. This is already installed on the computers in the lab. MDK combines the ARM RealView compilation tools with the Keil μ Vision Integrated Development Environment (IDE). The Keil μ Vision IDE includes: Project Management and Device & Tool Configuration, Source Code Editor Optimized for Embedded Systems, Target Debugging and Flash Programming, Accurate Device Simulation (CPU and Peripheral).

3. Lab Description

a. BLINKY 1

Creating a new project

1. First create a new folder called say EE378_S2013 where you plan to work on the labs of this course. Then, inside it, create a new folder called lab1.
2. Launch uVision4, Start->All Programs->Keil uVision4
3. Select Project->New uVision Project... and then select lab1 as the folder to save in; also type blinky1 as the File Name.
4. Then, select NXP (founded by Philips) LPC1768 as CPU inside the window that pops-up. Also, click Yes to Copy “startup_LPCxx.s” to Project Folder.
5. Under File Menu select New.
6. Write your code in the and save it as blinky1.c in the same project folder. This file (as well as all others discussed in this lab) is included in the downloadable archive from the website of this course. The panel on the left side of the uVision IDE is the Project window. The Project window gives you the hierarchy of Target folder and Source Group folder.
7. Right click on the “Source Group 1” and select “Add files to Source Code”.
8. Locate blinky1.c and include it to the group folder.
9. Copy C:\Keil\ARM\Startup\NXP\LPC17xx\system_LPC17xx.c to lab1 directory and add it as a source file to the project too. Open this file and browse it quickly to see what functions are defined inside it.
10. Click Project menu and then select Build Target.

11. Build Output panel should now show that the program is being compiled and linked. This creates blinky1.axf file (in ARM Executable Format), which will be downloaded to the board. To create a .hex file (more popular), select Flash->Configure Flash Tools->Output and check mark Create HEX File.
12. Connect the board to two USB ports of the host computer. One connection is to the port J16 of the board (the one that is a Standard-B plug). The second connection is via the ULINK2 debugger/programmer/emulator. These connections are illustrated in Fig.1.
13. Download the program to the Flash of the microcontroller. Select Flash->Download. This loads the blinky1.axf file. Then press the RESET push-button of the board.
14. Congratulations! You just programmed your first project. You should notice that the LED P1.29 is blinking. If this is not the case, then you should investigate/debug your project to make it work.



Figure 1 Connection of ULINK2 to the board.

Figure 1 Connection of ULINK2 to the board.

Debugging

If your program has errors (discovered by compiler or linker) or you simply want to debug it to verify its operation, then we can use the debugging capabilities of the uVision tools.

1. Click Debug menu option and select Start/Stop Debug Session. A warning about the fact that this is an evaluation version shows up; click OK.
2. Then, a new window appears where we can see the simulation of the program.
3. This window has several different supportive panels/sub-windows where we can monitor changes during the simulation. The left hand side panel, Registers, provides information regarding the Registers of LPC17xx with which we are working.
4. Again, click on the Debug menu option and select Run. The code starts simulating.
5. It is good practice that before going ahead with the actual hardware implementation to perform a debug/simulation session to make sure that our program behaves according to the design requirements.
6. In our example, we use PORT1.
7. Go to Peripherals menu option then select GPIO Fast Interface followed by Port 1.
8. You should get the window shown in Fig.2 below, where you can see LED P1.29 blinking. To actually observe this you should wait until the simulated time (shown on the bottom-right side of the uVision ISE window) is longer than 1 second. Note that in order to actually simulate 1 second of execution time of the program, the simulator must run much longer. This is expected, as typically simulators require much longer computational runtimes (wallclock time) in order to simulate relatively short execution times of the program under investigation!

9. This is a standard method to check that your program works correctly.
10. The debugger is a very powerful tool. This is only a brief exposure to it; we'll revisit it many times later in this course. Once you are done with the simulation/debug of your program, you can stop it by selecting Start/Stop the Debug Session from the Debug menu option. This stops the show and takes us back to the main uVision window.

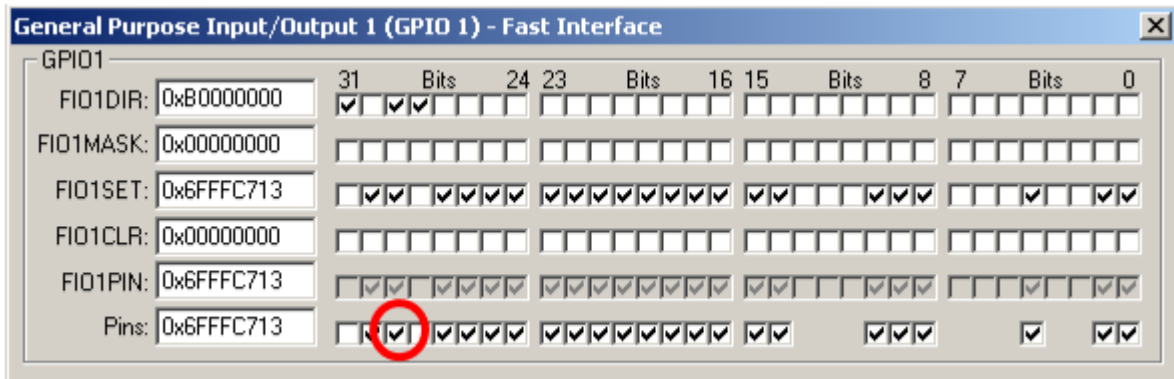


Figure 2 Pin P1.29 is checked on/off every other second or so.

Figure 2 Pin P1.29 is checked on/off every other second or so.

INTRODUCTION TO DSP PROCESSORS

A signal can be defined as a function that conveys information, generally about the state or behavior of a physical system. There are two basic types of signals viz Analog (continuous time signals which are defined along a continuum of times) and Digital (discrete-time).

Remarkably, under reasonable constraints, a continuous time signal can be adequately represented by samples, obtaining discrete time signals. Thus digital signal processing is an ideal choice for anyone who needs the performance advantage of digital manipulation along with today's analog reality.

Hence a processor which is designed to perform the special operations (digital manipulations) on the digital signal within very less time can be called as a Digital signal processor. The difference between a DSP processor, conventional microprocessor and a microcontroller are listed below.

Microprocessor: or General Purpose Processor such as Intel xx86 or Motorola 680xx

Family

Contains - only CPU

-No RAM

-No ROM

-No I/O ports

-No Timer

MICROCONTROLLER

Such as 8051 family Contains - CPU

- RAM

- ROM

- I/O ports

- Timer &

- Interrupt circuitry

Some Micro Controllers also contain A/D, D/A and Flash Memory

DSP PROCESSORS

Such as Texas instruments and Analog Devices Contains

- CPU
- RAM
- ROM
- I/O ports
- Timer

Optimized for

- fast arithmetic
- Extended precision
- Dual operand fetch
- Zero overhead loop
- Circular buffering

The basic features of a DSP Processor are

Feature	Use
Fast-Multiply accumulate	Most DSP algorithms, including filtering, transforms, etc. are multiplication-intensive
Multiple – access memory architecture	Many data-intensive DSP operations require reading a program instruction and multiple data items during each instruction cycle for best performance
Specialized addressing modes	Efficient handling of data arrays and first-in, first-out buffers in memory
Specialized program control	Efficient control of loops for many iterative DSP algorithms. Fast interrupt handling for frequent I/O operations.
On-chip peripherals and I/O interfaces	On-chip peripherals like A/D converters allow for small low cost system designs. Similarly I/O interfaces tailored for common peripherals allow clean interfaces to off-chip I/O devices.

A digital signal processor (DSP) is an integrated circuit designed for high-speed data manipulations, and is used in audio, communications, image manipulation, and other data-acquisition and data-control applications. The microprocessors used in personal computers are optimized for tasks involving data movement and inequality testing. The typical applications requiring such capabilities are word processing, database management, spread sheets, etc. When it comes to mathematical computations the traditional microprocessor are deficient particularly where real-time performance is required. Digital signal processors are microprocessors optimized for basic mathematical calculations such as additions and multiplications.

FIXED VERSUS FLOATING POINT:

Digital Signal Processing can be divided into two categories, fixed point and floating point which refer to the format used to store and manipulate numbers within the devices. Fixed point DSPs usually represent each number with a minimum of 16 bits, although a different length can be used. There are four common ways that these 2^{16} i.e., 65,536 possible bit patterns can represent a number. In unsigned integer, the stored number can take on any integer value from 0 to 65,535, signed integer uses two's complement to include negative numbers from -32,768 to 32,767. With unsigned fraction notation, the 65,536 levels are spread uniformly between 0 and 1 and the signed fraction format allows negative numbers, equally spaced between -1 and 1. The floating point DSPs typically use a minimum of 32 bits to store each value. This results in many more bit patterns than for fixed point, 2^{32} i.e., 4,294,967,296 to be exact. All floating point DSPs can also handle fixed point numbers, a necessity to implement counters, loops, and signals coming from the ADC and going to the DAC. However, this doesn't mean that fixed point math will be carried out as quickly as the floating point operations; it depends on the internal architecture.

C VERSUS ASSEMBLY:

DSPs are programmed in the same languages as other scientific and engineering applications, usually assembly or C. Programs written in assembly can execute faster, while programs written in C are easier to develop and maintain. In traditional applications, such as programs run on PCs and mainframes, C is almost always the first choice. If assembly is used at all, it is restricted to short subroutines that must run with the utmost speed.

HOW FAST ARE DSPS?

The primary reason for using a DSP instead of a traditional microprocessor is speed: the ability to move samples into the device and carry out the needed mathematical operations, and output the processed data. The usual way of specifying the fastness of a DSP is: fixed point systems are often quoted in MIPS (million integer operations per second). Likewise, floating point devices can be specified in MFLOPS (million floating point operations per second).

TMS320 FAMILY:

The Texas Instruments TMS320 family of DSP devices covers a wide range, from a 16-bit fixed-point device to a single-chip parallel-processor device. In the past, DSPs were used only in specialized applications. Now they are in many mass-market consumer products that are continuously entering new market segments. The Texas Instruments TMS320 family of DSP devices and their typical applications are mentioned below.

C1x, C2x, C2xx, C5x, and C54x:

The width of the data bus on these devices is 16 bits. All have modified Harvard architectures. They have been used in toys, hard disk drives, modems, cellular phones, and active car suspensions.

C3x:

The width of the data bus in the C3x series is 32 bits. Because of the reasonable cost and floating-point performance, these are suitable for many applications. These include almost any filters, analyzers, hi-fi systems, voice-mail, imaging, bar-code readers, motor control, 3D graphics, or scientific processing.

C4x:

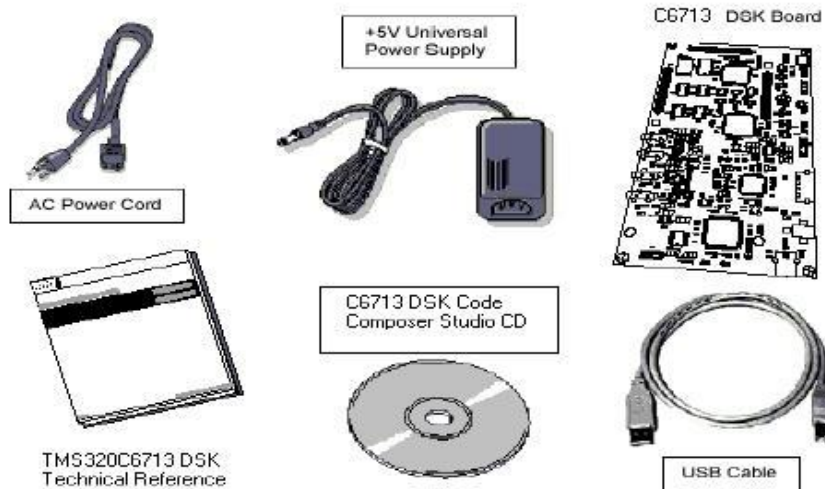
This range is designed for parallel processing. The C4x devices have a 32-bit data bus and are floating-point. They have an optimized on-chip communication channel, which enables a number of them to be put together to form a parallel-processing cluster. The C4x range devices have been used in virtual reality, image recognition, telecom routing, and parallel-processing systems.

C6x:

The C6x devices feature Velocity, an advanced very long instruction word (VLIW) architecture developed by Texas Instruments. Eight functional units, including two multipliers and six arithmetic logic units (ALUs), provide 1600 MIPS of cost-effective performance. The C6x DSPs are optimized for multi-channel, multifunction applications, including wireless base stations, pooled modems, remote-access servers, digital subscriber loop systems, cable modems, and multi-channel telephone systems.

INTRODUCTION TO TMS 320 C6713 DSK

The high-performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating point operations per second, the C6713 DSK the most powerful DSK development board



The DSK is USB port interfaced platform that allows efficiently developing and testing applications for the C6713. With extensive host PC and target DSP software support, the DSK provides ease-of-use and capabilities that are attractive to DSP engineers. The 6713 DSP Starter Kit (DSK) is a low-cost platform which lets customers evaluate and develop applications for the Texas Instruments C67X DSP family. The primary features of the DSK are:

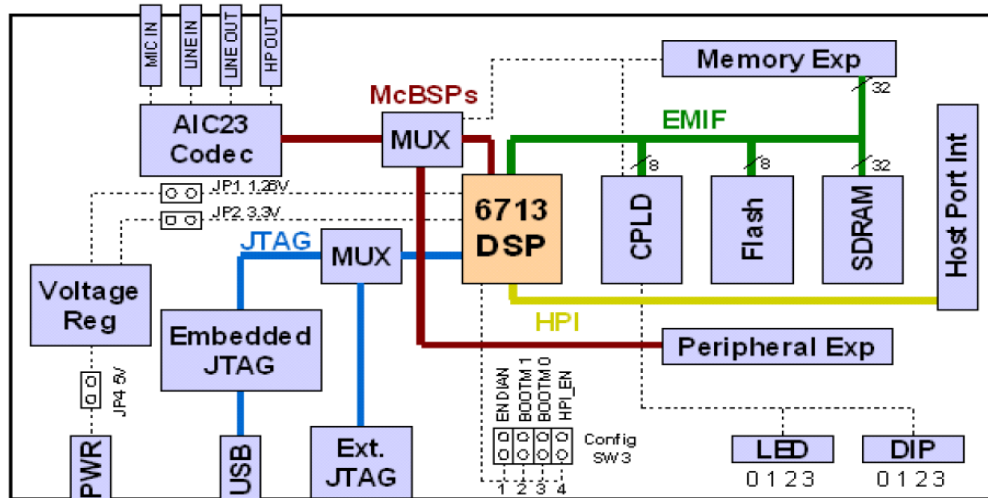
1. 225 MHz TMS320C6713 Floating Point DSP
2. AIC23 Stereo Codec
3. Four Position User DIP Switch and Four User LEDs
4. On-board Flash and SDRAM

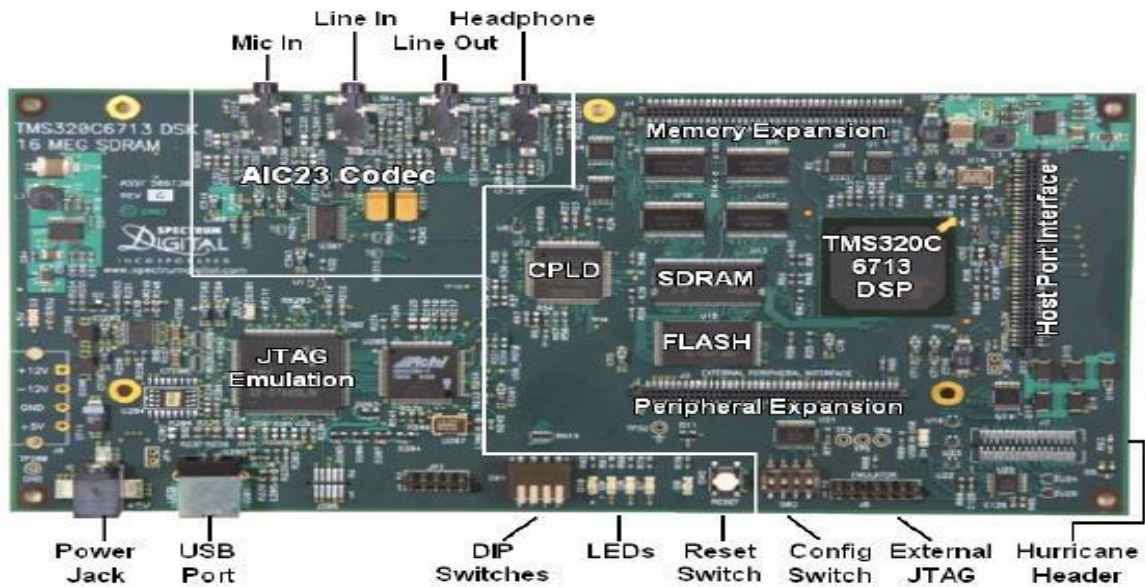
TI's Code Composer Studio development tools are bundled with the 6713DSK providing the user with an industrial-strength integrated development environment for C and assembly programming. Code Composer Studio communicates with the DSP using an on-board JTAG emulator through a USB interface. The TMS320C6713 DSP is the heart of the system. It is a core member of Texas Instruments' C64X line of fixed point DSPs whose distinguishing features are an extremely high

performance 225MHz VLIW DSP core and 256Kbytes of internal memory. On-chip peripherals include a 32-bit external memory interface (EMIF) with integrated SDRAM controller, 2 multi-channel buffered serial ports (McBSPs), two on-board timers and an enhanced DMA controller (EDMA). The 6713 represents the high end of TI's C6700 floating point DSP line both in terms of computational performance and on-chip resources.

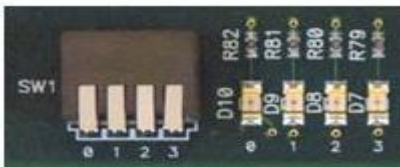
The 6713 has a significant amount of internal memory so many applications will have all code and data on-chip. External accesses are done through the EMIF which can connect to both synchronous and asynchronous memories. The EMIF signals are also brought out to standard TI expansion bus connectors so additional functionality can be added on daughter card modules. DSPs are frequently used in audio processing applications so the DSK includes an on-board codec called the AIC23. Codec stands for coder/decoder, the job of the AIC23 is to code analog input samples into a digital format for the DSP to process, then decode data coming out of the DSP to generate the processed analog output. Digital data is sent to and from the codec on McBSP1.

TMS320C6713 DSK OVERVIEW BLOCK DIAGRAM





The DSK has 4 light emitting diodes (LEDs) and 4 DIP switches that allow users to interact with programs through simple LED displays and user input on the switches. Many of the included examples make use of these user interfaces Options.



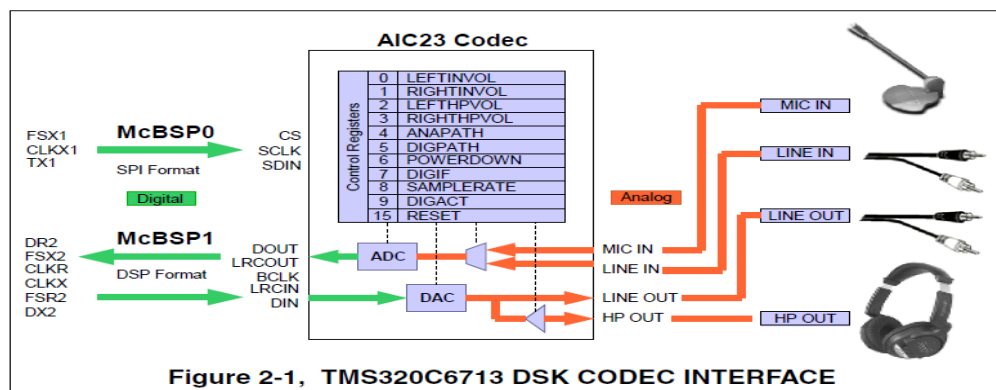
The DSK implements the logic necessary to tie board components together in a programmable logic device called a CPLD. In addition to random glue logic, the CPLD implements a set of 4 software programmable registers that can be used to access the on-board LEDs and DIP switches as well as control the daughter card interface.

AIC23 Codec

The DSK uses a Texas Instruments AIC23 (part #TLV320AIC23) stereo codec for input and output of audio signals. The codec samples analog signals on the microphone or line inputs and converts them into digital data so it can be processed by the DSP. When the DSP is finished with the data it uses the codec to convert the samples back into analog signals on the line and headphone outputs so the user can hear the output.

The codec communicates using two serial channels, one to control the codec's internal configuration registers and one to send and receive digital audio samples. McBSP0 is used as the unidirectional

control channel. 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 should specify the register to be modified and the lower 9 should contain the register value. The control channel is only used when configuring the codec, it is generally idle when audio data is being transmitted, McBSP1 is used as the bi-directional data channel. All audio data flows through the data channel. Many data formats are supported based on the three variables of sample width, clock signal source and serial data format. The DSK examples generally use a 16-bit sample width with the codec in master mode so it generates the frame sync and bit clocks at the correct sample rate without effort on the DSP side. The preferred serial format is DSP mode which is designed specifically to operate with the McBSP ports on TI DSPs.



DSK hardware installation

- Shut down and power off the PC
- Connect the supplied USB port cable to the board
- Connect the other end of the cable to the USB port of PC
- Plug the other end of the power cable into a power outlet
- Plug the power cable into the board
- The user LEDs should flash several times to indicate board is operational
- When you connect your DSK through USB for the first time on a Windows

Loaded PC the new hardware found wizard will come up. **So, Install the drivers** (The CCS CD contains the require drivers for C6713 DSK).

TROUBLESHOOTING DSK CONNECTIVITY

If Code Composer Studio IDE fails to configure your port correctly, perform the following steps: Test the USB port by running DSK Port test from the start menu

Use Start->Programs->Texas Instruments->Code Composer Studio-> Code Composer Studio C6713 DSK Tools -> C6713 DSK Diagnostic Utilities

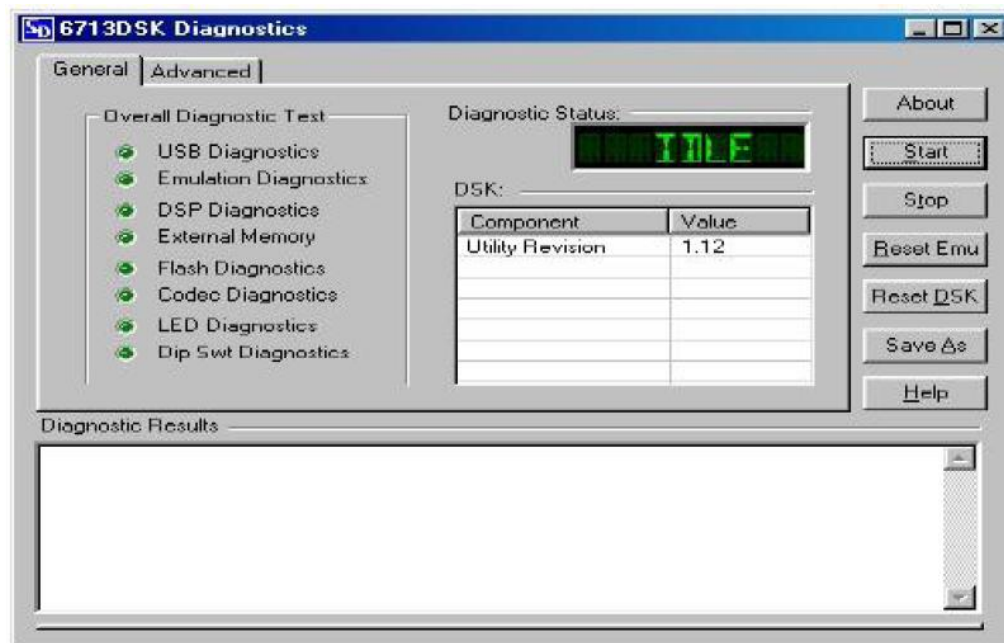
The below Screen will appear

Select 6713 DSK Diagnostic Utility Icon from Desktop, The Screen Look like as below

Select **Start** Option

Utility Program will test the board

After testing Diagnostic Status you will get **PASS**



INTRODUCTION TO CODE COMPOSER STUDIO

Code Composer is the DSP industry's first fully integrated development environment (IDE) with DSP-specific functionality. With a familiar environment liked MS-based C+, Code Composer lets you edit, build, debug, profile and manage projects from a single unified environment. Other unique features include graphical signal analysis, injection/extraction of data signals via file I/O, and multi-processor

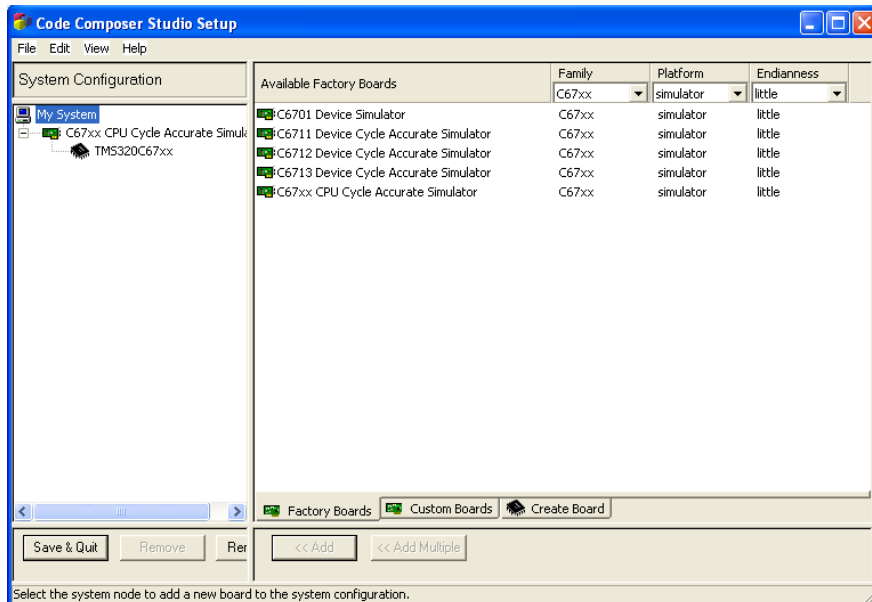
debugging, automated testing and customization via a C-interpretive scripting language and much more.

CODE COMPOSER FEATURES INCLUDE:

- ❑ IDE
- ❑ Debug IDE
- ❑ Advanced watch windows
- ❑ Integrated editor
- ❑ File I/O, Probe Points, and graphical algorithm scope probes
- ❑ Advanced graphical signal analysis
- ❑ Interactive profiling
- ❑ Automated testing and customization via scripting
- ❑ Visual project management system
- ❑ Compile in the background while editing and debugging
- ❑ Multi-processor debugging
- ❑ Help on the target DSP

TO CREATE A SYSTEM CONFIGURATION USING A STANDARD CONFIGURATION FILES:

- STEP 1:** Start CCS Setup by double clicking on the Setup CCS desktop icon.
- STEP 2:** Select Family ->c67xx
Platform->simulator
Endean's->little
- STEP 3:** Click the Import button (File-> import) to import our selection (c67xx_sim.ccs) to the **System** configuration currently being created in the CCS Setup window.
- STEP 4:** Click the Save and Quit button to save the configuration in the System Registry.
- STEP 5:** Click the Yes button to start the CCS IDE when we exit CCS Setup. The CCS Setup Closes and the CCS IDE automatically opens using the configuration we just created.

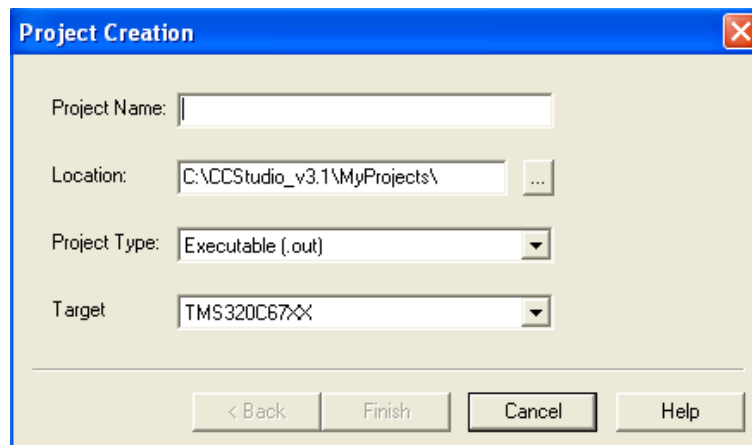


PROCEDURE TO WORK ON CODE COMPOSER STUDIO

STEP 1: Creating a New Project

From the Project menu, choose New. In the Project Name field, type the name we want for our project. Each project we create must have a unique name, and Click Finish. The CCS IDE creates a project file called projectname.pjt. This file stores our project settings and references the various files used by our project.

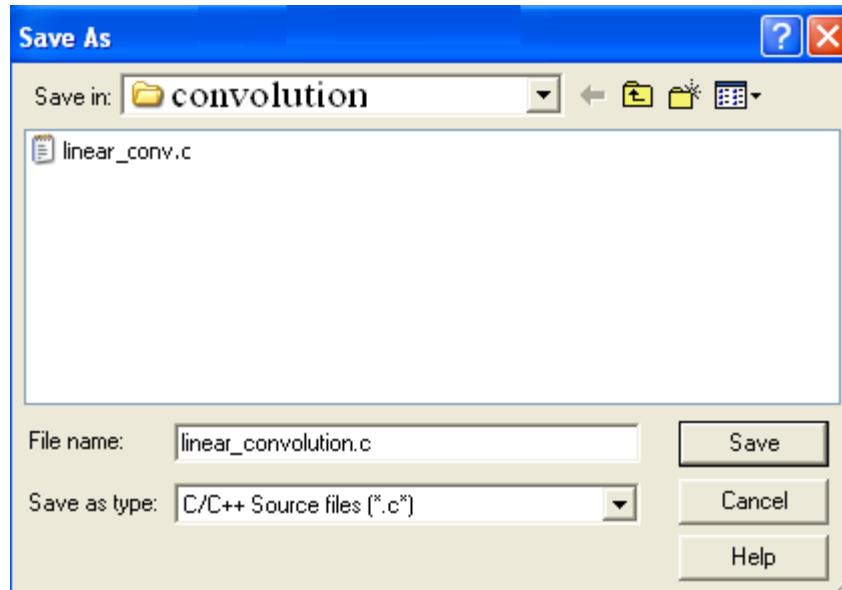
The Project Creation wizard window displays.



STEP 2: Creating a source file

Create a new source file using 'File ->new ->source file' pull down menu and save the source file with .c extension in the current project name directory. Save as type: c/c++ source file (*.c*)

Path: C:\CCStudio_v3.1\ MyProjects\Project Name\



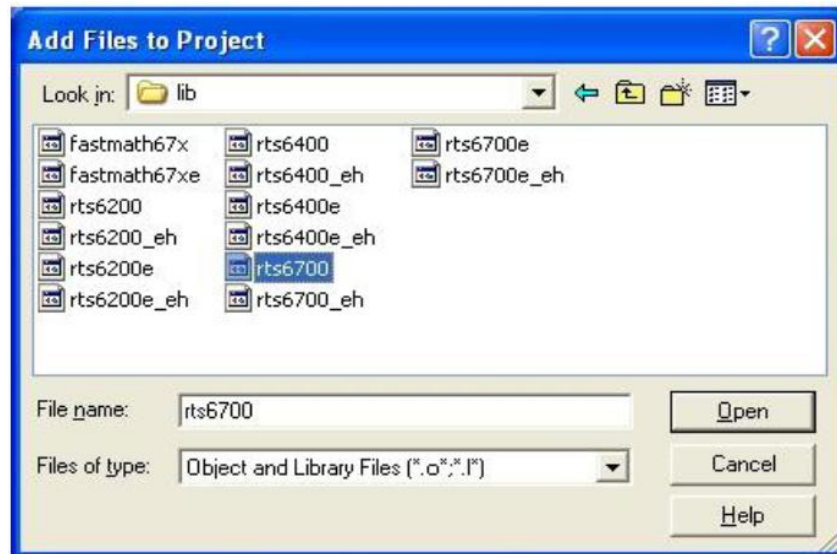
STEP 3: Add files to our project (source file\ library file\ linker file)

SOURCE FILE: Add the source file in the project using 'Project->add files to project' pull down menu. Files of type: c/c++ source file (*.c*)

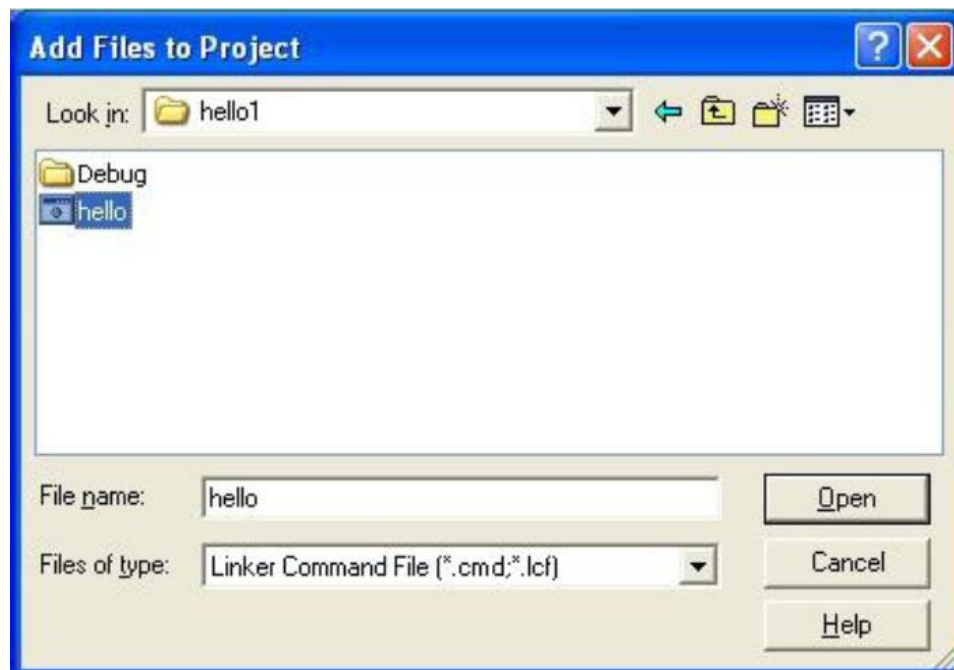
Path: C:\CCStudio_v3.1\ My Projects\Project Name\filename's

LIBRARY FILE: Add the library file in the project using 'Project-> add files to project' pull down menu. Files of type: Object and Library Files (*.o*,*.l*)

Path: C:\CCStudio_v3.1\ C6000\ cgtools\ lib \ rts6700.lib



LINKER FILE: Add the linker file in the project using 'Project-> add files to project' pull down menu. Files of type: Linker command Files (*.cmd;*.lcf*)
Path: C:\CCStudio_v3.1\ tutorial\ dsk6713\ hello1 \ hello.cmd



STEP 4: Building and Running the Program (compile\ Build\ Load Program\ Run)

COMPILE: Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.

BUILD: Build the program using the 'Project-Build' pull down menu or by clicking the shortcut icon on the left side of program window.

LOAD PROGRAM: Load the program in program memory of DSP chip using the 'File-load program' pull down menu. Files of type:(*.out*) Path: C:\CCStudio_v3.1\MyProjects\Project Name\ Debug\ Project ame.out

RUN: Run the program using the 'Debug->Run' pull down menu or by clicking the shortcut icon on the left side of program window.

STEP 5: observe output using graph

Choose View-> Graph-> Time/Frequency. In The Graph Property Dialog, Change The Graph Title, Start Address, And Acquisition Buffer Size, Display Data Size, Dsp Data Type, Auto Scale, And Maximum Y- Value Properties To The Values.

EXPERIMENT-1

Aim: Write an Embedded C program to blinky led with software delay, delay generated using systick timer

Apparatus:

S. No	Name of the equipment	Quantity
1	MCB1700 Trainer kit	1
2	5V Adapter	1
3	RS-232 Cable	1
4	Keil software	1

Program:

```
/*-----  
 * Name: LED.c  
 * Purpose: low level LED functions*/  
#include "LPC17xx.H"          /* LPC17xx definitions */  
#include "LED.h"  
  
const unsigned long led_mask[] = { 1UL<<28, 1UL<<29, 1UL<<31, 1UL<< 2,  
                                   1UL<< 3, 1UL<< 4, 1UL<< 5, 1UL<< 6 };  
  
/*-----  
 initialize LED Pins  
 *-----*/  
Void main()  
{  
void LED_Init (void) {  
  
    LPC_SC->PCONP |= (1 << 15);      /* enable power to GPIO & IOCON */  
  
    LPC_GPIO1->FIODIR |= 0xB0000000; /* LEDs on PORT1 are output */  
    LPC_GPIO2->FIODIR |= 0x0000007C; /* LEDs on PORT2 are output */  
}  
  
/*-----  
  
Function that turns on requested LED  
 *-----*/  
void LED_On (unsigned int num) {
```

```

if (num < 3) LPC_GPIO1->FIOPIN |= led_mask[num];
else      LPC_GPIO2->FIOPIN |= led_mask[num];
}

/*-----
Function that turns off requested LED
*-----*/
void LED_Off (unsigned int num) {

if (num < 3) LPC_GPIO1->FIOPIN &= ~led_mask[num];
else      LPC_GPIO2->FIOPIN &= ~led_mask[num];
}

/*-----
Function that outputs value to LEDs
*-----*/
void LED_Out(unsigned int value) {
int i;

for (i = 0; i < LED_NUM; i++) {
if (value & (1<<i)) {
LED_On (i);
} else {
LED_Off(i);
}
}
}
}
}

```

Output: You can see the all led's blinking

EXPERIMENT-2

Aim : Write an embedded C program to take analog readings on rotation of rotary potentiometer connected to an ADC channel.

Apparatus:

S. No	Name of the equipment	Quantity
1	MCB1700 Trainer kit	1
2	5V Adapter	1
3	RS-232 Cable	1
4	Keil software	1

Source Code:

```
/*-----  
* Name: ADC.c  
* Purpose: low level ADC functions  
* Note(s): possible defines select the used ADC interface:  
*   __ADC_IRQ - ADC works in Interrupt mode  
*   - ADC works in polling mode (default) */  
#include "LPC17xx.H" /* LPC17xx definitions */  
#include "ADC.h"  
  
uint16_t AD_last; /* Last converted value */  
uint8_t AD_done = 0; /* AD conversion done flag */  
void main()  
{  
/*-----  
Function that initializes ADC  
*-----*/  
void ADC_Init (void) {  
  
LPC_SC->PCONP |= ((1 << 12) | (1 << 15)); /* enable power to ADC & IOCON */  
  
LPC_PINCON->PINSEL1 &= ~(3 << 18);  
LPC_PINCON->PINSEL1 |= (1 << 18); /* P0.25 is AD0.2 */  
LPC_PINCON->PINMODE1 &= ~(3 << 18);  
LPC_PINCON->PINMODE1 |= (2 << 18); /* P0.25 no pull up/down */  
  
LPC_ADC->ADCR = (1 << 2) | /* select AD0.2 pin */  
                (4 << 8) | /* ADC clock is 25MHz/5 */  
                (1 << 21); /* enable ADC */
```

```

#ifdef __ADC_IRQ
    LPC_ADC->ADINTEN = ( 1 << 8);    /* global enable interrupt */

    NVIC_EnableIRQ(ADC_IRQn);        /* enable ADC Interrupt */
#endif
}

/*-----
start AD Conversion
*-----*/
void ADC_StartCnv (void) {
    LPC_ADC->ADCR &= ~( 7 << 24);    /* stop conversion */
    LPC_ADC->ADCR |= ( 1 << 24);    /* start conversion */
}

/*-----
stop AD Conversion
*-----*/
void ADC_StopCnv (void) {
    LPC_ADC->ADCR &= ~( 7 << 24);    /* stop conversion */
}

/*-----
get converted AD value
*-----*/
uint16_t ADC_GetCnv (void) {
#ifdef __ADC_IRQ
    while (!(LPC_ADC->ADGDR & ( 1UL << 31))); /* Wait for Conversion end */
    AD_last = (LPC_ADC->ADGDR >> 4) & ADC_VALUE_MAX; /* Store converted value */

    AD_done = 1;
#endif

    return(AD_last);
}

/*-----
A/D IRQ: Executed when A/D Conversion is done
*-----*/
#ifdef __ADC_IRQ
void ADC_IRQHandler(void) {
    volatile uint32_t adstat;

```

```
adstat = LPC_ADC->ADSTAT;          /* Read ADC clears interrupt */
AD_last = (LPC_ADC->ADGDR >> 4) & ADC_VALUE_MAX; /* Store converted value */
AD_done = 1;
}
#endif
}
```

Output: You can see the message on LCD.analog readings if required reset the board.

EXPERIMENT-3

Aim : Write an embedded C program to control intensity of a led using pwm implemented in software.

Apparatus:

S. No	Name of the equipment	Quantity
1	MCB1700 Trainer kit	1
2	5V Adapter	1
3	RS-232 Cable	1
4	Keil software	1

Source Code:

```
#include <lpc17xx.h>
#include "pwm.h"
#include "delay.h"

#define CYCLE_TIME 255

/* start the main program */
int main()
{
    int dutyCycle;
    SystemInit();          /* Clock and PLL configuration */
    PWM_Init(CYCLE_TIME); /* Initialize the PWM module and the Cycle time(Ton+Toff) is set
to 255(similar to arduino)*/
    PWM_Start(PWM_1|PWM_2|PWM_3|PWM_4); /* Enable PWM output on PWM_1-PWM_4
(P2_0 - P2_3) */

    while(1)
    {

        for(dutyCycle=0;dutyCycle<CYCLE_TIME;dutyCycle++) /* Increase the Brightness of the Leds
*/
        {
            PWM_SetDutyCycle(PWM_1,dutyCycle); //P2_0
            PWM_SetDutyCycle(PWM_2,dutyCycle); //P2_1
            PWM_SetDutyCycle(PWM_3,dutyCycle); //P2_2
            PWM_SetDutyCycle(PWM_4,dutyCycle); //P2_3
            DELAY_ms(5);
        }

        for(dutyCycle=CYCLE_TIME;dutyCycle>0;dutyCycle--) /* Decrease the Brightness of the Leds
*/
        {
```



```
PWM_SetDutyCycle(PWM_1,dutyCycle); //P2_0
PWM_SetDutyCycle(PWM_2,dutyCycle); //P2_1
PWM_SetDutyCycle(PWM_3,dutyCycle); //P2_2
PWM_SetDutyCycle(PWM_4,dutyCycle); //P2_3
DELAY_ms(5);
    }
}
}
```

Output: In this program control intensity of an led using pwm implemented in software. You can see output on display

EXPERIMENT-4

Aim : To write an embedded C program for Temperature indication on an RGB LED and to Verify the output in the Cortex-M3 kit

Apparatus:

S. No	Name of the equipment	Quantity
1	Cortex-M Trainer kit	1
2	5V Adapter	1
3	RS-232 Cable	1
4	Keil software	1

Source code:

```
#include "adc.h"
#include "lcd.h"
#include "LPC17xx.h"
#include "delay.h"
int main()
{
    int adcValue;
    float temp;
    SystemInit();
    ADC_Init(); /* Initialize the ADC module */

    /*Connect RS, RW, EN and data bus to PORT0.4 to PORT0.7*/
    LCD_SetUp(P2_0,P2_1,P2_2,P_NC,P_NC,P_NC,P_NC,P1_24,P1_25,P1_26,P1_27);
    LCD_Init(2,16);

    while(1)
    {
        adcValue = ADC_GetAdcValue(0); // Read the ADC value of channel zero where the
        temperature sensor(LM35) is connected

        /* Convert the raw ADC value to equivalent temperature with 5v as ADC reference
        Step size of AdC= (5v/1023)=4.887mv = 5mv.
        for every degree celcius the Lm35 provides 10mv voltage change.
        1 step of ADC=5mv=0.5'c, hence the Raw ADC value can be divided by 2 to get
        equivalent temp*/

        temp = adcValue/2.0; // Divide by 2 to get the temp value.
        LCD_GoToLine(0);
        LCD_Printf("ADC0 Value:%4d \nTemp:%f\n\r",adcValue,temp); // Display adc value and
        temp LCD
    }
}
```

```
}
```

RESULT:

Thus the Embedded C program for temperature indication on an RGB LED is written and executed. The output is verified in the CORTEX kit.

EXPERIMENT-5

AIM: Calculate the Euclidian distance between any two points Using DSK Code composer studio

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

THEORY:

Formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

C Code:

```
#include <stdio.h>

#include <math.h>

int main() {

    float x1, y1, x2, y2, gdistance;

    printf("Input x1: ");

    scanf("%f", &x1);

    printf("Input y1: ");

    scanf("%f", &y1);
```

```
printf("Input x2: ");  
  
scanf("%f", &x2);  
  
printf("Input y2: ");  
  
scanf("%f", &y2);  
  
gdistance = ((x2-x1)*(x2-x1))+((y2-y1)*(y2-y1));  
  
printf("Distance between the said points: %.4f", sqrt(gdistance));  
  
printf("\n");  
  
return 0;  
  
}
```

Sample Output:

```
Input x1: 25  
Input y1: 15  
Input x2: 35  
Input y2: 10  
Distance between the said points: 11.1803
```

EXPERIMENT-6

AIM: Verify the linear convolution operation Using DSK Code composer studio

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

THEORY:

Convolution is a formal mathematical operation, just as multiplication, addition, and integration. Addition takes two *numbers* and produces a third *number*, while convolution takes two *signals* and produces a third *signal*. Convolution is used in the mathematics of many fields, such as probability and statistics. In linear systems, convolution is used to describe the relationship between three signals of interest: the input signal, the impulse response, and the output signal.

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k) \quad 0 < n < N - 1$$

In this equation, $x(k)$, $h(n-k)$ and $y(n)$ represent the input to and output from the system at time n . Here we could see that one of the input is shifted in time by a value every time it is multiplied with the other input signal. Linear Convolution is quite often used as a method of implementing filters of various types.

PROGRAM:

```
// Linear convolution program in c language using ccstudio
#include<stdio.h>
int x[15],h[15],y[15];
main()
{
int i,j,m,n;
printf("\n enter the length of first sequence m:");
scanf("%d",&m);
```

```

printf("\n enter the length of second sequence n:");
scanf("%d",&n);
printf("enter values for i/p sequence x(n):\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("enter values for i/p sequence h(n): \n");
for(i=0;i<n; i++)
scanf("%d",&h[i]);
// padding of zeros
for(i=m;i<=m+n-1;i++)
x[i]=0;
for(i=n;i<=m+n-1;i++)
h[i]=0;
/* convolution operation */
for(i=0;i<m+n-1;i++)
{
y[i]=0;
for(j=0;j<=i;j++)
{
y[i]=y[i]+(x[j]*h[i-j]);
}
}
//displaying the o/p
for(i=0;i<m+n-1;i++)
printf("\n the value of output y[%d]=%d\t",i,y[i]);
}

```

PROCEDURE:

- Open Code Composer Studio; make sure the DSP kit is turned on.
- Start a new project using 'Project-new' pull down menu, save it in a separate directory (D: 11951A0xxx) with name **linear convolution**.
- Write the program and save it as linearconv.c

- Add the source files linearconv.c to the project using ‘Project->add files to project’ pull down menu.
- Add the linker command file hello.cmd.
(Path: C:CCstudio_V3.1\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file rts6700.lib.
(Path: C:CCstudio_v3.1\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the ‘Project-compile’ pull down menu
- Build the program using the ‘Project-Build’ pull down menu
- Load the program (linearconv.out) in program memory of DSP chip using the ‘File-load program’ pull down menu.
- Debug-> Run
- To View output graphically Select view ->graph ->time and frequency.

OUTPUT AND WAVEFORM:

Enter first sequence length m: 4

Enter second sequence length n: 3

Enter i/p sequence for x (n):

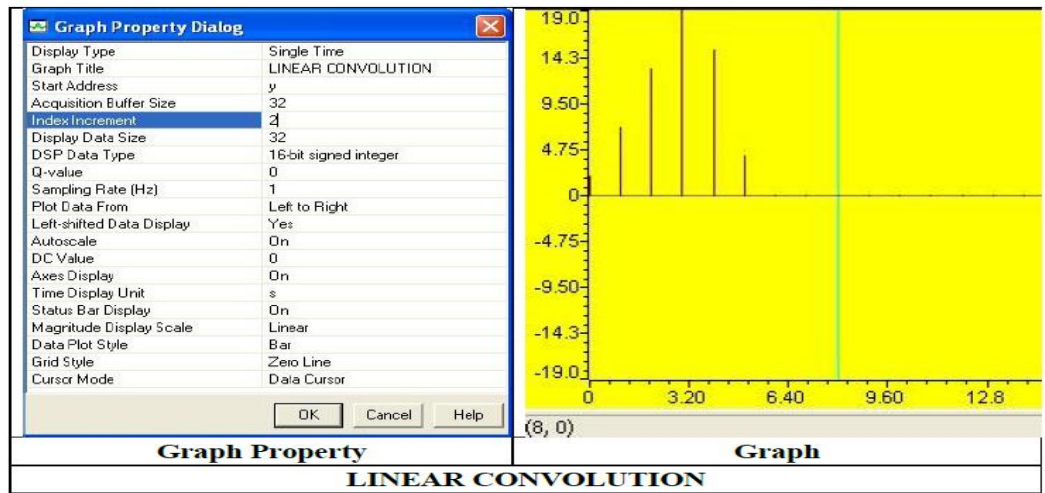
1 2 3 4

Enter i/p sequence for h (n):

2 3 1

Output (Linear Convolution) sequence is:

y[0]=2 y[1]=7 y[2]=13 y[3]=19 y[4]=15 y[5]=4



EMPTY SPACE FOR CALCULATIONS

INSERT GRAPH SHEET (NORMAL)

EXPERIMENT-7

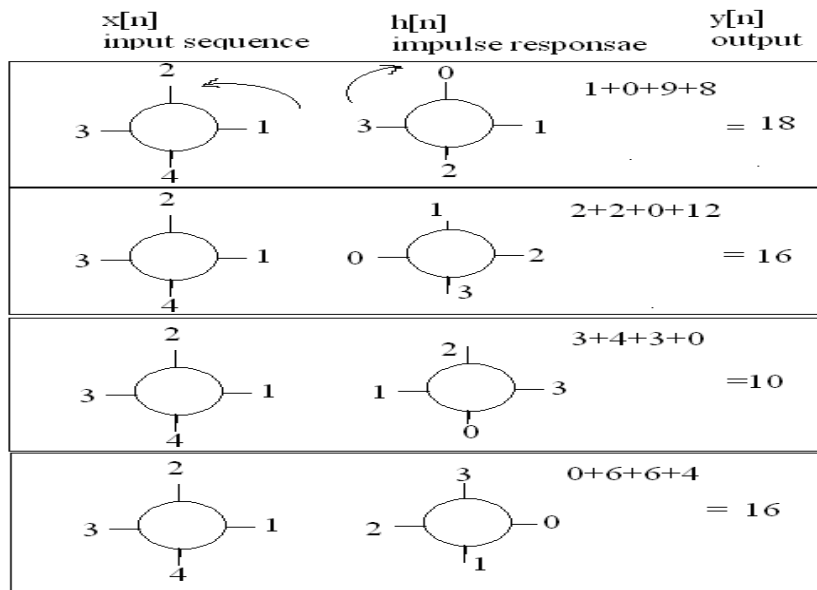
AIM: Verify the circular convolution operation using DSK code composer studio

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

THEORY:

Circular convolution is another way of finding the convolution sum of two input signals. It resembles the linear convolution, except that the sample values of one of the input signals is folded and right shifted before the convolution sum is found. Also note that circular convolution could also be found by taking the DFT of the two input signals and finding the product of the two frequency domain signals. The Inverse DFT of the product would give the output of the signal in the time domain which is the circular convolution output. The two input signals could have been of varying sample lengths. But we take the DFT of higher point, which ever signals levels to. For e.g. If one of the signals is of length 256 and the other spans 51 samples, then we could only take 256 point DFT. So the output of IDFT would be containing 256 samples instead of 306 samples, which follows $N_1 + N_2 - 1$ where N_1 & N_2 are the lengths 256 and 51 respectively of the two inputs. Thus the output which should have been 306 samples long is fitted into 256 samples. The 256 points end up being a distorted version of the correct signal. This process is called circular convolution.



PROGRAM:

```
//program to implement circular convolution
#include<stdio.h>
int m,n,x[30],h[30],y[30],i,j, k,x2[30],a[30];
void main()
{
printf("\n Enter the length of the first sequence\n");
scanf("%d",&m);
printf("\n Enter the length of the second sequence\n");
scanf("%d",&n);
printf("\n Enter the first sequence\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("\n Enter the second sequence\n");
for(j=0;j<n;j++)
scanf("%d",&h[j]);
if(m-n!=0) /*If length of both sequences are not equal*/
{
if(m>n) /* Pad the smaller sequence with zero*/
{
```

```

for(i=n;i<m;i++)
h[i]=0;
n=m;
}
for(i=m;i<n;i++)
x[i]=0;
m=n;
}

y[0]=0;
a[0]=h[0];
for(j=1;j<n;j++) /*folding h(n) to h(-n)*/
a[j]=h[n-j];
/*Circular convolution*/
for(i=0;i<n;i++)
y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
{
y[k]=0;
/*circular shift*/
for(j=1;j<n;j++)
x2[j]=a[j-1];
x2[0]=a[n-1];
for(i=0;i<n;i++)
{
a[i]=x2[i];
y[k]+=x[i]*x2[i];
}
}
//displaying the result
printf("\n The circular convolution is: \n");
for(i=0;i<n;i++)
printf("%d \t",y[i]);
}

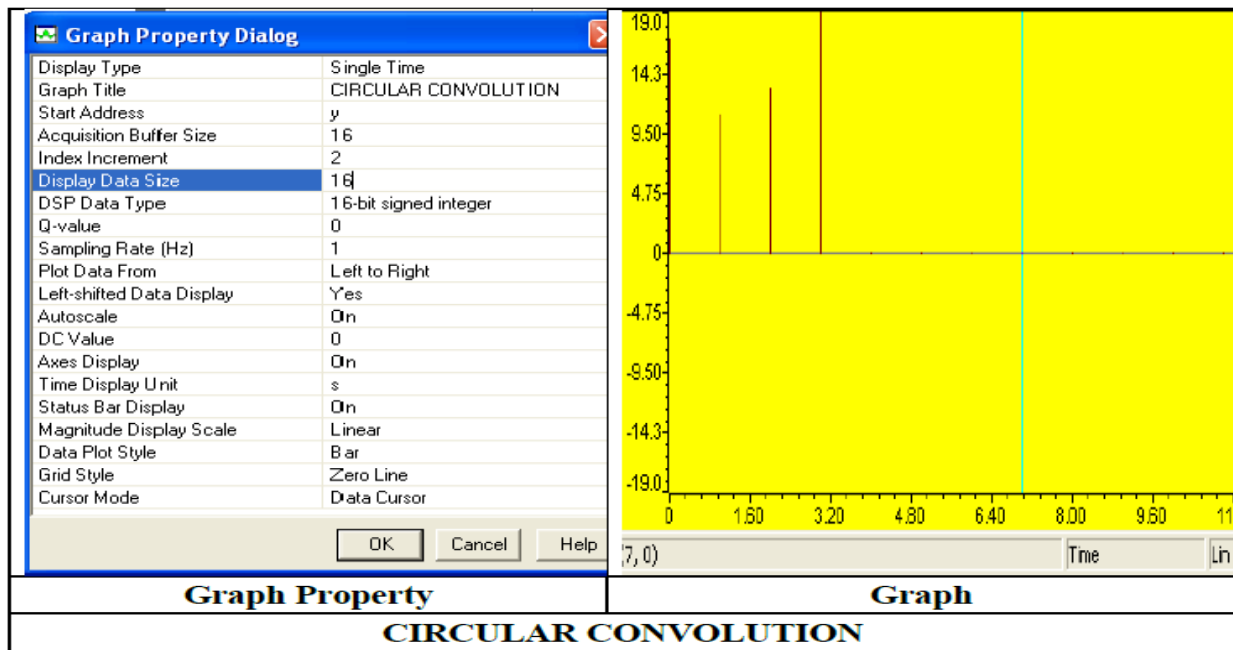
```

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using 'Project-new' pull down menu, save it in a separate directory(D:11951A0xxx) with name **circular convolution**.
- Write the program and save it as circularconv.c
- Add the source files circularconv.c to the project using 'Project->add files to project' pull down menu.
- Add the linker command file hello.cmd.
(Path: C:CCstudio_V3.1\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file rts6700.lib.
(Path: C:CCstudio_v3.1\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the 'Project-compile' pull down menu
- Build the program using the 'Project-Build' pull down menu
- Load the program (circularconv.out) in program memory of DSP chip using the 'File-load program' pull down menu.
- Debug-> Run
- To View output graphically Select view ->graph ->time and frequency.

OUTPUT AND WAVEFORM:

```
Enter the length of the first sequence 4
Enter the length of the second sequence 3
Enter the first sequence
1 2 3 4
Enter the second sequence
1 2 3
The circular convolution is
18  16  10  16
```



EMPTY SPACE FOR CALCULATIONS

INSERT GRAPH SHEET (NORMAL)

EXPERIMENT-8

AIM: To generate a real time fir filter through Rectangular window using TMS320C6713 DSK

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

PROCEDURE:

1. Connect CRO to the LINE OUT sockets.
2. Now switch ON the DSK and bring up Code Composer Studio on PC
3. Create a new project with name sinewave.pjt
4. From File menu->New->DSP/BIOS Configuration->Select dsk6713.cdb and save it as “ firfliter.cdb”
5. Add firfilter.cdb to the current project
6. Create a new source file and save it as firfilter.c
7. Add the source file firfilter.c to the project
8. Add the library file “dsk6713bsl.lib” to the project
(Path: C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib)
9. Copy files “dsk6713.h” and “dsk6713_aic23.h” to the Project folder
(Path: C:\CCStudio_v3.1\C6000\dsk6713\include)
10. Build (F7) and load the program to the DSP Chip (File->Load Program(.out file))
11. Run the program (F5)
12. Observe the waveform that appears on the CRO screen and ccstudio simulator.

// c program for generation of fir filter using c6713 DSK

```
#include "firfiltercfg.h"
```

```
#include "dsk6713.h"
```

```
#include "dsk6713_aic23.h"
```

```
#include "stdio.h"
```

```
Float filter_coeff[ ]={-0.020203,-0.016567,0.009656,0.027335,0.011411,-0.023194,-  
0.033672,0.000000,0.043293,0.038657,-0.025105,-0.082004,-  
0.041842,0.115971,0.303048,  
0.386435,0.303048,0.115971,-0.041842,-0.082004,-  
0.025105,0.038657,0.043293,0.000000,-0.033672,-  
0.023194,0.011411,0.027335,0.009656,-0.016567,-0.020203};//FIR Low pass
```

Rectangular Filter pass band range 0-1500Hz

```
DSK6713_AIC23_Config
```

```
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001}
```

```
;
```

```
void main()
```

```
{
```

```
DSK6713_AIC23_CodecHandle hCodec;
```

```
Uint32 l_input, r_input,l_output, r_output;
```

```
DSK6713_init();
```

```
hCodec = DSK6713_AIC23_openCodec(0, &config);
```

```
DSK6713_AIC23_setFreq(hCodec, 1);
```

```
while(1)
```

```
{
```

```
while(!DSK6713_AIC23_read(hCodec, &l_input));
```

```
while(!DSK6713_AIC23_read(hCodec, &r_input));
```

```
l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
```

```
r_output=l_output;
```

```
while(!DSK6713_AIC23_write(hCodec, l_output));
```

```
while(!DSK6713_AIC23_write(hCodec, r_output));
```

```
}
```



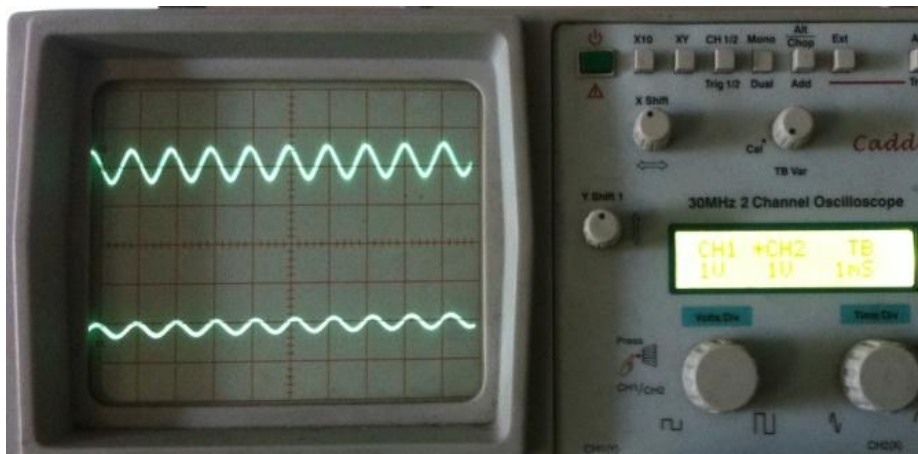
```

DSK6713_AIC23_closeCodec(hCodec);
}
signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int in_buffer[100];
in_buffer[0] = x;
for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];
for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
return(output);
}

```

WAVEFORM:

- a) Waveforms of input and output from cro



- b) Function generator (input signal frequency at 1 KHz)



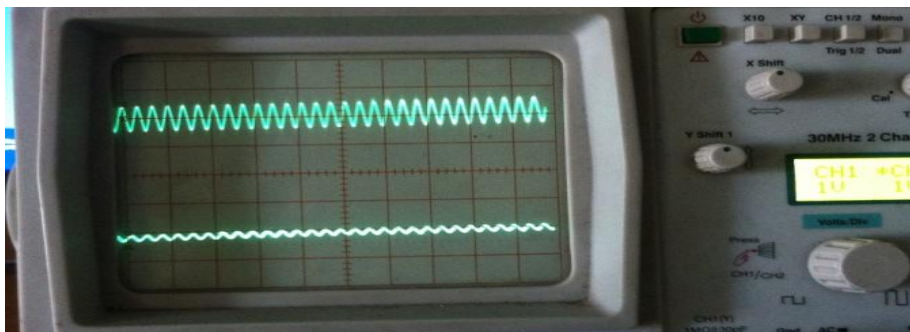
c) C6713 DSK



d) Function generator (input signal frequency at 1500Hz)



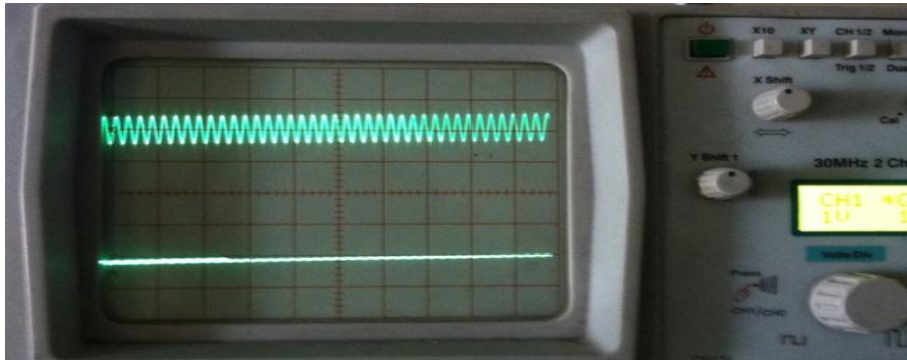
e) Waveforms of input and output from CRO (output signal attenuates)



d) Function generator (input signal frequency at 2000Hz)



a) Waveforms of input and output from CRO (output signal fully attenuated)



INSERT GRAPH SHEET (NORMAL)

EXPERIMENT-9

AIM:

To generate a real time fir filter through Kaiser Window using TMS320C6713 DSK and Sample sound using a microphone and display sound levels on LEDs

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

PROCEDURE:

1. Connect CRO to the LINE OUT sockets.
2. Now switch ON the DSK and bring up Code Composer Studio on PC
3. Create a new project with name sinewave.pjt
4. From File menu->New->DSP/BIOS Configuration->Select dsk6713.cdb and save it as “firfliter_kaiser.cdb”
5. Add firfilter.cdb to the current project
6. Create a new source file and save it as firfilter_kaiser.c
7. Add the source file firfilter_kaiser.c to the project
8. Add the library file “dsk6713bsl.lib” to the project
(Path: C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib)
9. Copy files “dsk6713.h” and “dsk6713_aic23.h” to the Project folder
(Path: C:\CCStudio_v3.1\C6000\dsk6713\include)
10. Build (F7) and load the program to the DSP Chip (File->Load Program (.out file))
11. Run the program (F5)
12. Observe the waveform that appears on the CRO screen and cstudio simulator.

// c program for generation of fir filter using c6713 DSK

```
#include "firfilter_kaisercfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
#include "stdio.h"

float filter_coeff[ ]={0.000000,-0.000138,-0.000611,-0.001345,-0.001607, 0.000000, 0.004714,
0.012033,0.018287,0.016731,0.000000,-0.035687,-0.086763,-0.141588,-0.184011,0.800005, -
0.184011,-0.141588,-0.086763,-0.035687,0.000000,0.016731,0.018287,0.012033,0.004714, -
0.000000,-0.001607,-0.001345,-0.000611,-0.000138,0.000000};//FIR High pass Kaiser filter pass
band range 800Hz-3.5KHz

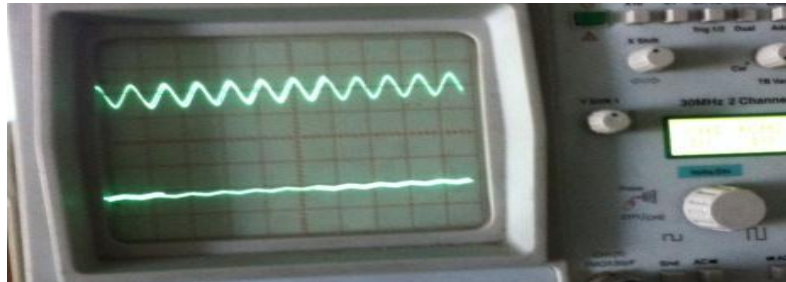
void main()
{
    DSK6713_AIC23_CodecHandle hCodec;
    Uint32 l_input, r_input,l_output, r_output;
    DSK6713_init();
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, 1);
    while(1)
    {
        while(!DSK6713_AIC23_read(hCodec, &l_input));
        while(!DSK6713_AIC23_read(hCodec, &r_input));
        l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
        r_output=l_output;
        while(!DSK6713_AIC23_write(hCodec, l_output));
        while(!DSK6713_AIC23_write(hCodec, r_output));
    }
    DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
    int i=0;
    signed long output=0;
```

```
static short int in_buffer[100];
in_buffer[0] = x;
for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];
for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

WAVEFORM:

- a) Waveforms of input and output from CRO



- b) function generator (input signal frequency at 500Hz)



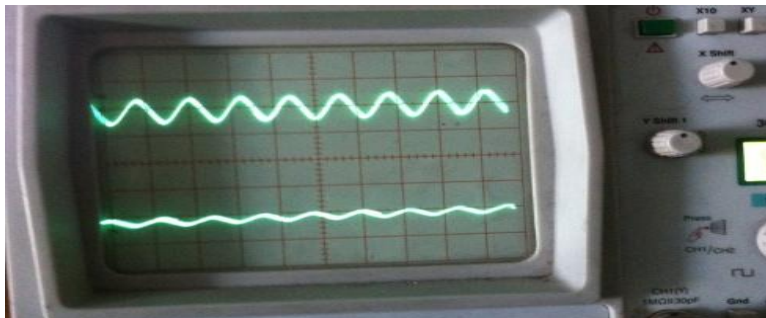
c) C6713 DSK



d) Function generator (input signal frequency at 800Hz)



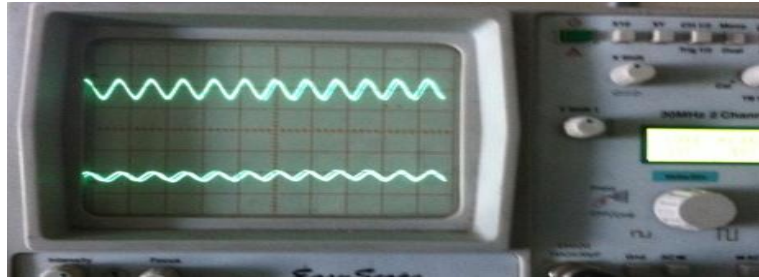
e) Waveforms of input and output from CRO



d) Function generator (input signal frequency at 1.1kHz)



e) Waveforms of input and output from CRO



Insert Graph Sheet (Normal)

EXPERIMENT-10

AIM:

To generate a real time iir filter through Butterworth approximation using TMS320C6713 DSK

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

PROCEDURE:

1. Connect CRO to the LINE OUT socket.
2. Now switch ON the DSK and bring up Code Composer Studio on PC
3. Create a new project with name sinewave.pjt
4. From File menu->New->DSP/BIOS Configuration->Select dsk6713.cdb and save it as “iirfliter.cdb”
5. Add firfilter.cdb to the current project
6. Create a new source file and save it as iirfilter.c
7. Add the source file iirfilter.c to the project
8. Add the library file “dsk6713bsl.lib” to the project
(Path: C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib)
9. Copy files “dsk6713.h” and “dsk6713_aic23.h” to the Project folder
(Path: C:\CCStudio_v3.1\C6000\dsk6713\include)
10. Build (F7) and load the program to the DSP Chip (File->Load Program(.out file))
11. Run the program (F5)
12. Observe the waveform that appears on the CRO screen and cecstudio simulator.

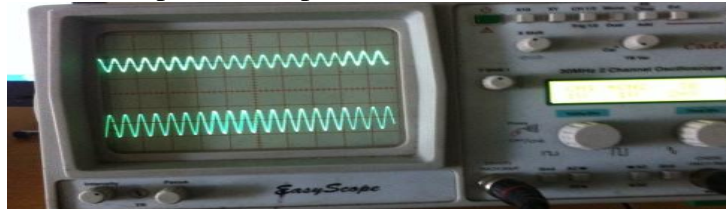
// c program for generation of iir filter using c6713 DSK

```
#include "iirfiltercfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
#include "stdio.h"
const signed int filter_coeff [ ] = {15241,15241,15241,32761,10161,7877};
//IIR_BUTERWORTH_LP FILTER pass band range 0-8kHz
DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001}
;
void main()
{
    DSK6713_AIC23_CodecHandle hCodec;
    Uint32 l_input, r_input,l_output, r_output;
    DSK6713_init();
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, 3);
    while(1)
    {
        while(!DSK6713_AIC23_read(hCodec, &l_input));
        while(!DSK6713_AIC23_read(hCodec, &r_input));
        l_output=IIR_FILTER(&filter_coeff ,l_input);
        r_output=l_output;
        while(!DSK6713_AIC23_write(hCodec, l_output));
        while(!DSK6713_AIC23_write(hCodec, r_output));
    }
    DSK6713_AIC23_closeCodec(hCodec);
}
signed int IIR_FILTER(const signed int * h, signed int x1)
{
    static signed int x[6] = {0,0,0,0,0,0};
    static signed int y[6] = {0,0,0,0,0,0};
```

```
int temp=0;
temp = (short int)x1;
x[0] = (signed int) temp;
temp = ( (int)h[0] * x[0]);
temp += ( (int)h[1] * x[1]);
temp += ( (int)h[1] * x[1]);
temp += ( (int)h[2] * x[2]);
temp -= ( (int)h[4] * y[1]);
temp -= ( (int)h[4] * y[1]);
temp -= ( (int)h[5] * y[2]);
temp >>=15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = temp;
y[2] = y[1];
y[1] = y[0];
x[2] = x[1];
x[1] = x[0];
return (temp<<2);
}
```

RESULT:

a) Waveforms of input and output from CRO



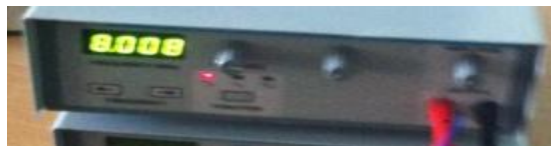
b) Function generator (input signal frequency at 1KHz)



b) C6713 DSK



c) function generator (input signal frequency at 8kHz)



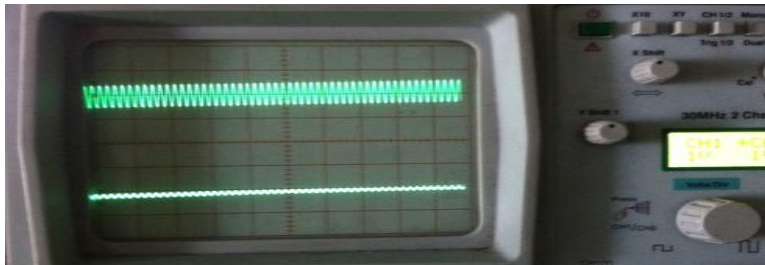
d) Waveforms of input and output from CRO



e) function generator (input signal frequency at 10kHz)



g) Waveforms of input and output from CRO (output signal attenuated)



Insert Graph Sheet (Normal)

EXPERIMENT-11

AIM: Verify the convolution operation Using DSK Code composer studio

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

THEORY:

In this program the Discrete Fourier Transform (DFT) of a sequence $x[n]$ is generated by using the formula,

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi j k n / N}$$

Where, $X(k)$ is DFT of sequence $x[n]$ $n=0$

N represents the sequence length and it is calculated by using the command 'length'. The DFT of any sequence is the powerful computational tool for performing frequency analysis of discrete-time signals.

PROGRAM:

```
#include  
  
<stdio.h>  
  
#include  
  
<math.h> int  
  
N,k,n,i;
```

```

float
pi=3.1416,sumre=0,sumim=0,out_real[8]={0.0},out_imag[8]={0.0}

; int x[32];

void main(void)

{

printf("enter the length of the sequence\n");

scanf("%d",&N);

printf("\nenter the sequence\n");
for(i=0;i<N;i++)

scanf("%d",&x[i]);

for(k=0;k<N;k++)

{

sumre=0;

sumim=0;

for(n=0;n<N

;n++)

{

sumre=sumre+x[n]*cos(2*pi*k*

n/N); sumim=sumim-

x[n]*sin(2*pi*k*n/N);

}

out_real[k]=sum

re;

out_imag[k]=su

```

```

mim;

printf("DFT of the sequence:\n");

printf("x[%d]=\t%f\t+\t%f\n",k,out_real[k],out_ima
g[k]);

}

}

```

PROCEDURE:

- Open code composer studio, make sure the dsp kit is turned on.
- Start a new project using ‘project-new ‘ pull down menu, save it in a separate directory(d:11951a0xxx) with name **dft**.
- Write the program and save it as dft.c
- Add the source files dft.c to the project using ‘project->add files to project’ pull down menu.
- Add the linker command file hello.cmd.
(path: c:ccstudio_v3.1\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file rts6700.lib.
(path: c:ccstudio_v3.1\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the ‘project-compile’ pull down menu
- Build the program using the ‘project-build’ pull down menu
- Load the program (dft.out) in program memory of dsp chip using the ‘file-load program’ pull down menu.
- Debug-> run
- To view output graphically select view ->graph ->time and frequency.

OUTPUT AND WAVEFORM:

enter the length of the sequence

4

enter the sequence

1 2 3 4

DFT of the sequence:

$$x[0] = 10.000000 + 0.000000i$$

DFT of the sequence:

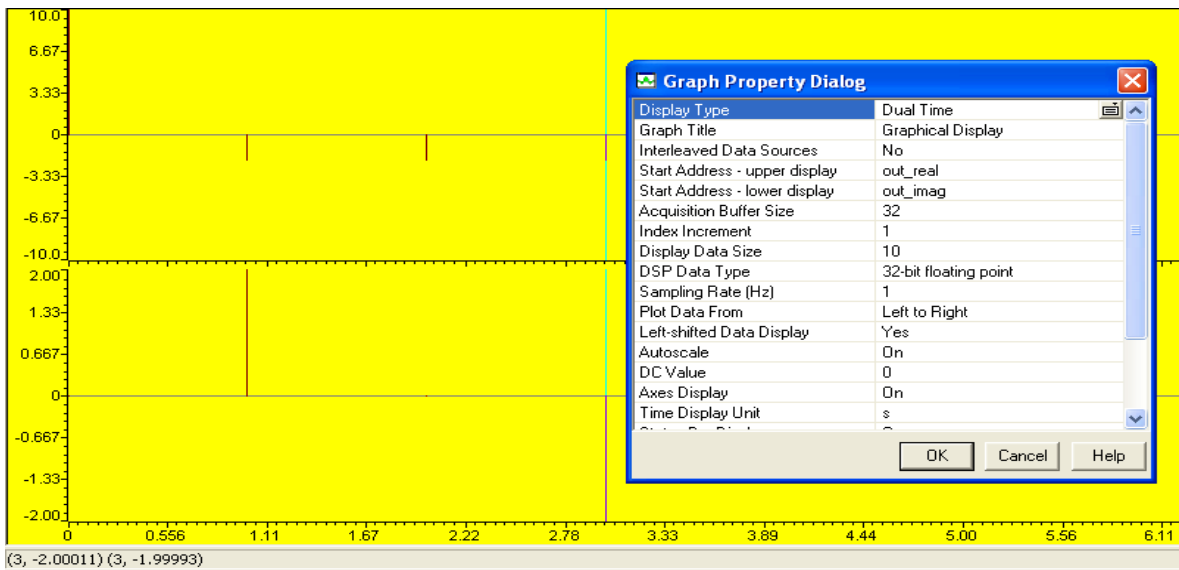
$$x[1] = -1.999963 + 2.000022i$$

DFT of the sequence:

$$x[2] = -2.000000 + 0.000059i$$

DFT of the sequence:

$$x[3] = -2.000108 + -1.999934i$$



Empty Space for

Calculations Insert Graph

Sheet (Normal)

EXPERIMENT-12

AIM: To design and implement filters in C to enhance the features of given input sequence/signal

EQUIPMENTS:

S. No	Name of the equipment	Quantity
1	TMS 320C6713 Kit	1
2	RS232 Serial Cable	1
3	Power Cord	1
4	Operating System – Windows XP	1
5	Software – CCStudio_v3.1	1

PROCEDURE:

1. Connect CRO to the LINE OUT socket.
2. Now switch ON the DSK and bring up Code Composer Studio on PC
3. Create a new project with name sinewave.pjt
4. From File menu->New->DSP/BIOS Configuration->Select dsk6713.cdb and save it as “iirfliter.cdb”
5. Add firfilter.cdb to the current project
6. Create a new source file and save it as iirfilter.c
7. Add the source file iirfilter.c to the project
8. Add the library file “dsk6713bsl.lib” to the project
(Path: C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib)
9. Copy files “dsk6713.h” and “dsk6713_aic23.h” to the Project folder
(Path: C:\CCStudio_v3.1\C6000\dsk6713\include)
10. Build (F7) and load the program to the DSP Chip (File->Load Program(.out file))
11. Run the program (F5)
12. Observe the waveform that appears on the CRO screen and cccstudio simulator.

// c program for generation of iir filter using c6713 DSK

```
#include "iirfiltercfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
#include "stdio.h"
const signed int filter_coeff [ ] = {15241,15241,15241,32761,10161,7877};
//IIR_BUTERWORTH_LP FILTER pass band range 0-8kHz
DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001}
;
void main()
{
    DSK6713_AIC23_CodecHandle hCodec;
    Uint32 l_input, r_input,l_output, r_output;
    DSK6713_init();
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, 3);
    while(1)
    {
        while(!DSK6713_AIC23_read(hCodec, &l_input));
        while(!DSK6713_AIC23_read(hCodec, &r_input));
        l_output=IIR_FILTER(&filter_coeff ,l_input);
        r_output=l_output;
        while(!DSK6713_AIC23_write(hCodec, l_output));
        while(!DSK6713_AIC23_write(hCodec, r_output));
    }
    DSK6713_AIC23_closeCodec(hCodec);
}
signed int IIR_FILTER(const signed int * h, signed int x1)
{
    static signed int x[6] = {0,0,0,0,0,0};
    static signed int y[6] = {0,0,0,0,0,0};
```

```

int temp=0;
temp = (short int)x1;
x[0] = (signed int) temp;
temp = ( (int)h[0] * x[0]);
temp += ( (int)h[1] * x[1]);
temp += ( (int)h[1] * x[1]);
temp += ( (int)h[2] * x[2]);
temp -= ( (int)h[4] * y[1]);
temp -= ( (int)h[4] * y[1]);
temp -= ( (int)h[5] * y[2]);
temp >>=15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = temp;
y[2] = y[1];
y[1] = y[0];
x[2] = x[1];
x[1] = x[0];
return (temp<<2);
}

```

RESULT:

a) Waveforms of input and output from CRO



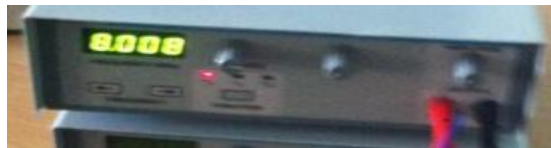
b) Function generator (input signal frequency at 1KHz)



f) C6713 DSK



g) function generator (input signal frequency at 8kHz)



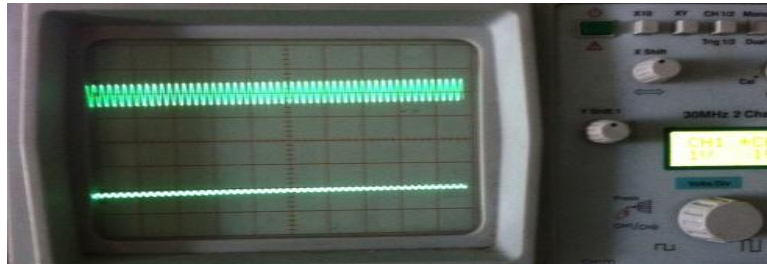
h) Waveforms of input and output from CRO



i) function generator (input signal frequency at 10kHz)



g) Waveforms of input and output from CRO (output signal attenuated)



Insert Graph Sheet (Normal)