MRCET

MICROPROCESSORS AND MICROCONTROLLERS LAB

III B.Tech ECE-II Sem

ELECTRONICS & COMMUNICATION ENGINEERING

VISION

To evolve into a center of excellence in Engineering Technology through creative and innovative practices in teaching-learning, promoting academic achievement & research excellence to produce internationally accepted competitive and world class professionals.

MISSION

To provide high quality academic programmes, training activities, research facilities and opportunities supported by continuous industry institute interaction aimed at employability, entrepreneurship, leadership and research aptitude among students.

QUALITY POLICY

- **Impart up-to-date knowledge to the students in Electronics & Communication area to make them quality engineers.**
- **Make the students experience the applications on quality equipment and tools.**
- **Provide systems, resources and training opportunities to achieve continuous improvement.**
- **Maintain global standards in education, training and services.**

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: PROFESSIONALISM & CITIZENSHIP

To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.

PEO2: TECHNICAL ACCOMPLISHMENTS

To provide knowledge based services to satisfy the needs of society and the industry by providing hands on experience in various technologies in core field.

PEO3: INVENTION, INNOVATION AND CREATIVITY

To make the students to design, experiment, analyze, interpret in the core field with the help of other multi disciplinary concepts wherever applicable.

PEO4: PROFESSIONAL DEVELOPMENT

To educate the students to disseminate research findings with good soft skills and become a successful entrepreneur.

PEO5: HUMAN RESOURCE DEVELOPMENT

To graduate the students in building national capabilities in technology, education and research.

PROGRAMME SPECIFIC OBJECTIVES (PSOs)

PSO1

To develop a student community who acquire knowledge by ethical learning and fulfill the societal and industry needs in various technologies of core field.

PSO2

To nurture the students in designing, analyzing and interpreting required in research and development with exposure in multi disciplinary technologies in order to mould them as successful industry ready engineers/entrepreneurs

PSO3

To empower students with all round capabilities who will be useful in making nation strong in technology, education and research domains.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

INDEX

PART-A

PART-B

PART-A

1. INTRODUCTION TO MASM

EDITOR

An editor is a program, which allows you to create a file containing the assembly language statements for your program. As you type in your program, the editor stores the ASCII codes for the letters and numbers in successive RAM locations. When you have typed in all of your programs, you then save the file on a floppy of hard disk. This file is called source file. The next step is to process the source file with an assembler. In the MASM /TASM assembler, you should give your source file name the extension, .ASM

ASSEMBLER

An assembler program is used to translate the assembly language mnemonics for instructions to the corresponding binary codes. When you run the assembler, it reads the source file of your program from the disk, where you saved it after editing on the first pass through the source program the assembler determines the displacement of named data items, the offset of labels and pails this information in a symbol table. On the second pass through the source program, the assembler produces the binary code for each instruction and inserts the offset etc that is calculated during the first pass. The assembler generates two files on floppy or hard disk. The first file called the object file is given the extension. OBJ. The object file contains the binary codes for the instructions and information about the addresses of the instructions. The second file generated by the assembler is called assembler list file. The list file contains your assembly language statements, the binary codes for each instructions and the offset for each instruction. In MASM/TASM assembler, MASM/TASM source file name ASM is used to assemble the file. Edit source file name LST is used to view the list file, which is generated, when you assemble the file.

LINKER

A linker is a program used to join several object files into one large object file and convert to an **exe** file. The linker produces a link file, which contains the binary codes for all the combined modules. The linker however doesn't assign absolute addresses to the program, it assigns is said to be reloadable because it can be put anywhere in memory to be run. In MASM/TASM, LINK/TLIN5K source filename is used to link the file.

DEBUGGER

A debugger is a program which allows you to load your object code program into system memory, execute the program and troubleshoot are debug it the debugger allows you to look at the contents of registers and memory locations after your program runs. It allows you to change the contents of register and memory locations after your program runs. It allows you to change the contents of register and memory locations and return the program. A debugger also allows you to set a break point at any point in the program. If you inset a breakpoint the debugger will run the program up to the instruction where the breakpoint is set and stop execution. You can then examine register and memory contents to see whether the results are correct at that point. In MASM/TASM, td filename is issued to debug the file.

DEBUGGER FUNCTIONS:

1. Debugger allows looking at the contents of registers and memory locations.
2. We can extend 8-bit register to 16-bit register which the help of extended register option.
3. Debugger allows setting breakpoints at any point with the program.
4. The debugger will run the program up to the instruction where the breakpoint is set and then stop execution of program. At this point, we can examine registry and memory contents at that point.
5. With the help of dump we can view register contents.
6. We can trace the program step by step with the help of F7.
7. We can execute the program completely at a time using F8

The DOS -Debugger:

The DOS "Debug" program is an example of simple debugger that comes with MS-DOS. Hence it is available on any PC .it was initially designed to give the user the capability to trace logical errors in executable file.

Below, are summarized the basic DOS - Debugger commands

COMMAND	SYNTAX
Assemble	A [address]
Compare	C range address
Dump	D [range]
Enter	E address [list]
Fill	F range list
Go	G [=address] [addresses]
Hex	H value1 value2
Input	I port
Load	L[address] [drive][first sector][number]
Move	M range address
Name	N[pathname][argument list]
Output	O port byte
Proceed	P [=address][number]
Quit	Q
Register	R[register]
Search	S range list
Trace	T [=address][value]
Unassembled	u [range]
Write	W[address][drive][first sector][number]

MS-MASM:

Microsoft's Macro Assembler (MASM) is an integrated software package Written by Microsoft Corporation for professional software developers. It consists of an editor, an assembler, a linker and a debugger (Code View). The programmer's workbench combines these four parts into a user-friendly programming environment with built in on line help. The following are the steps used if you are to run MASM from DOS

MICROPROCESSOR LAB EXECUTION PROCEDURE

STEP1: **Opening the DOS prompt**

Click **start** menu button and click on **Run** and then type *cmd at* command prompt immediately DOS window will be appeared

STEP2: **Checking the masm installation**

To know MASAM is installed or not simply type ***masm*** *at the command prompt* upon that it replies masm version vendor (Microsoft), etc... If you get any error there is no masm in that PC

STEP3: Directory changing (create a folder with your branch and no in D drive)

Change the current directory to your won directory suppose your folder in **D** drive type the following commands to change the directory at command prompt type ***D:*** hit enter now you are in ***D*** **drive** type **cd folder name** hit the enter

Ex. **D cd ece10**

Now we are in folder cse10

STEP4: **writing the program**

At the command prompt type the **edit programname.asm**

Ex. Edit add.asm

Immediately editor window will open and there you have to write the program. Type the program in that window after completion save the Program, to save the program Go to ***file*** opt in the menu bar and select save opt now your code is ready to Assemble.

STEP5: A**ssembling, Linking and Executing the program**

Go to *file* opt click *exit* opt now DOS prompt will be displayed to assemble the program type the following commands at the DOS prompt

Masm Program Name, Program Name, Program Name, Program Name hit the enter

Ex: Masm add, add, add, add enter

OR

Ex: Masm add.asm

If there are any errors in the program assembler reports all of them at the command prompt with line no's, if there are now bugs your ready to link the program. To link the program type the following line at command prompt Link program name**,,,,,** (5 commas)

Ex: Link add,,,,,

OR

Ex: link add.obj

After linking you are ready to execute the program. To execute the program type the following command

Debug program name.exe hit the enter

Ex: Debug add.exe

Now you entered into the execution part of the program here you have to execute the program instruction by instruction (debugging) first of all press the ***r*** key(register) ***hit*** the enter key it'll displays all the registers and their initial values in HEXDECIMAL note down the values of all the register which are used in the program. To execute the next instruction press ***t*** key (TRACE) hit the enter it'll execute that instruction and displays the contents of all the register. You have to do this until you reach the last instruction of the program. After execution you have to observe the results (in memory or registers based on what you have written in the program).

STEP6: Copying list file (common for all programs):

A list file contains your code starting address and end address along with your program .For every program assembler generates a list file at your folder, programname.lst (ex. Add.lst) you should copy this to your lab observation Opening a list file

Edit programname.lst

Ex. Edit add.lst

EXPERIMENT NO.2
16 BIT ARITHMETIC OPERATIONS

AIM: Write an ALP to 8086 to perform 16-bit arithmetic operations in various Addressing Modes.

TOOLS: PC installed with MASM

ALGORITHM:

Step I : Initialize the Data segment memory.

Step II : Initialize the Extra segment memory.

Step III : Load the first number into AX register.

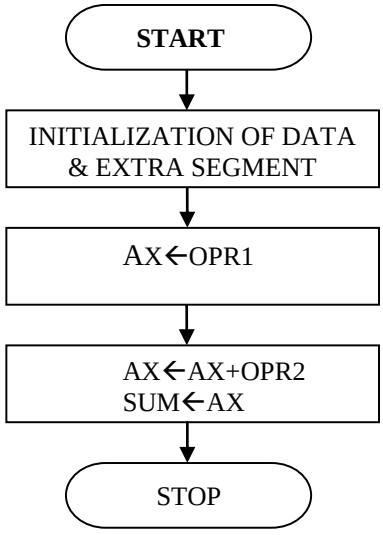
Step IV : Add two numbers.

Step V : Store the result in Extra segment.

Step VI : Terminate the program

Step VII : Stop.

FLOW CHART:



PROGRAM:

(A) 16-bit addition using different addressing modes

```
ASSUME CS: CODE, DS: DATA, ES: EXTRA
DATA SEGMENT
        OPR1 DW 5169H
        OPR2 DW 1000H
DATA ENDS
```

```
EXTRA SEGMENT
        SUM DW   ?
    EXTRA ENDS
      CODE SEGMENT
      START:
            MOV AX, DATA
            MOV DS, AX          ; REGISTER ADDRESIING MODE
            MOV AX, OPR1        ; DIRECT ADDRESSING MODE
            ADD AX, OPR2        ; DIRECT ADDRESSING MODE
            MOV SUM, AX         ; DIRECT ADDRESSING MODE
            INT 03H
      CODE ENDS
      END START
      END
```

;(B)16-bit subtraction using different addressing modes

ALGORITHM:

Step I **:** Initialize the data & extra segment memory.

Step II **:** Load the first number into AX register.

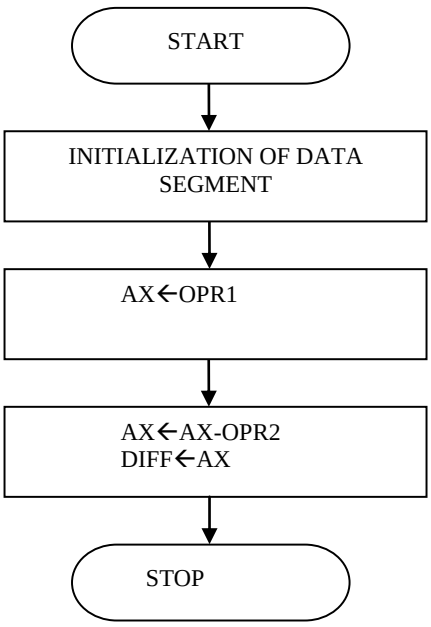
Step IV **:** Sub AX from OPR2.

Step V **:** Store result in extra segment

Step VI **:** verify the result.

Step VII **:** Stop.

FLOW CHART:



PROGRAM:

```
ASSUME CS:CODE, DS:DATA,ES:EXTRA
        DATA SEGMENT
            OPR1 DW 5169H
            OPR2 DW 1000H
        DATA  ENDS
        EXTRA SEGMENT
                DIFF  DW   ?
        EXTRA ENDS
        CODE SEGMENT
        START:
            MOV AX, DATA
            MOV DS, AX              ;REGISTER ADDRESIING MODE
            MOV AX, EXTRA
            MOV ES, AX              ;REGISTER ADDRESIING MODE
            MOV BX, OFFSET OPR1     ;DIRECT ADDRESSING MODE
            MOV AX, [BX]            ; BASE  ADDRESSING MODE/
            SUB  AX, OPR2           ;DIRECT ADDRESSING MODE
            MOV DIFF  , AX          ;DIRECT ADDRESSING MODE
            INT 03H
        CODE ENDS
        END START
        END
```

(C)16-bit Multiplication using different addressing modes

ALGORITHM:

Step I **:** Initialize the data & extra segment memory.

Step II **:** Load the first number into AX register.

Step III **:** Load the second number into BX register.

Step IV **:** Multiply AX with BX.

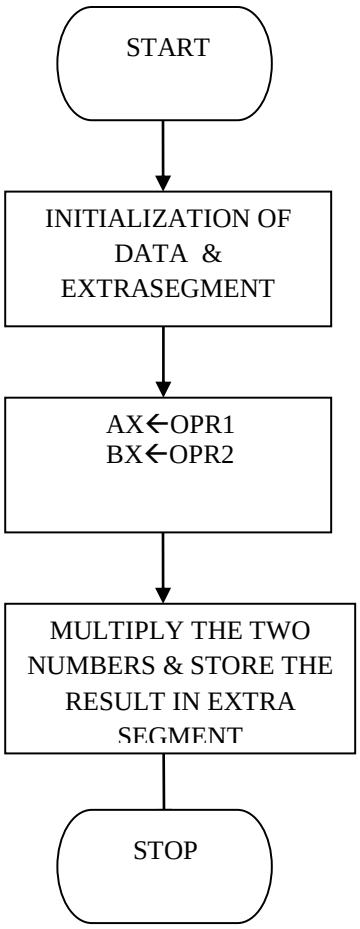
Step V **:** store lower word in accumulator into extra segment.

Step VI **:** Store Upper word in DX register into extra segment

Step VII **:** Verify the result.

Step VIII **:** Stop.

FLOW CHART:



PROGRAM:

```
ASSUME CS: CODE, DS: DATA, ES: EXTRA
        DATA SEGMENT
            OPR1 DW 5169H
            OPR2 DW 1000H
        DATA ENDS
      EXTRA SEGMENT
            RES DW 2 DUP(0)
        EXTRA ENDS
        CODE SEGMENT
        START:
            MOV AX, DATA
            MOV DS, AX          ; REGISTER ADDRESIING MODE
            MOV AX, EXTRA
            MOV ES, AX          ; REGISTER ADDRESIING MODE
            MOV SI, OFFSET OPR1
            MOV AX, [SI]        ; INDEXED  ADDRESSING MODE
```

```
        MOV BX,OPR2        ; DIRECT ADDRESSING MODE
        MUL BX             ; REGISTER ADDRESSING MODE
        MOV RES, AX        ; DIRECT ADDRESSING MODE
        MOV RES+2, DX      ; DIRECT ADDRESSING MODE
        INT 03H
CODE ENDS
END START
END
```

;(D)16-bit Division using different addressing modes

ALGORITHM:

Step I : Initialize the data & extra segment memory.

Step II : Load the first number into DX:AX register pair.

Step III : Load the second number into BX register.

Step IV : Divide DX:AX pair by BX.

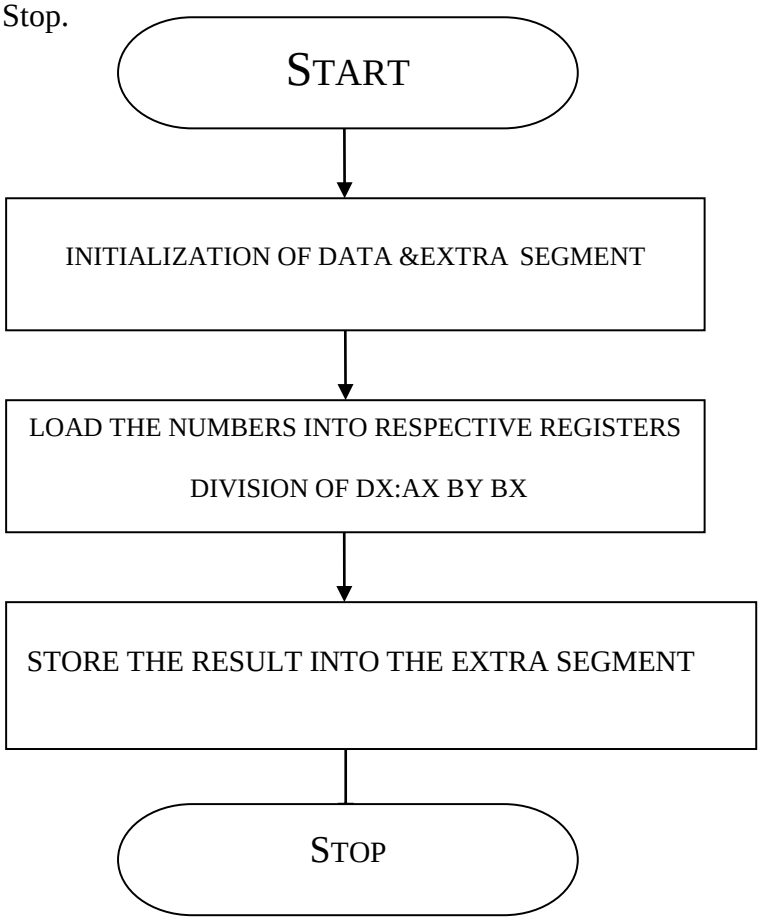
Step V : store the Quotient in AX register into extra segment.

Step VI : Store the reminder in DX register into extra segment.

Step VII : Verify the result.

Step VIII : Stop.

FLOW CHART:



PROGRAM:

```
ASSUME CS: CODE, DS:DATA, ES:EXTRA
        DATA SEGMENT
                OPR1 DD 74105169H
                OPR2 DW 7875H
        DATA   ENDS
      EXTRA SEGMENT
                DIVQ DW  ?
                DIVR DW  ?
        EXTRA ENDS
        CODE SEGMENT
        START:MOV AX, DATA
                MOV DS, AX            ; REGISTER ADDRESIING MODE
                MOV AX, EXTRA
                MOV ES, AX           ; REGISTER ADDRESIING MODE

                MOV SI, OFFSET OPR1
                MOV AX, [SI]           ; INDEXED  ADDRESSING MODE/
                MOV DX, [SI+2]         ; INDEXED  ADDRESSING MODE
                MOV BX, OPR2          ; DIRECT ADDRESSING MODE
                DIV BX                 ; REGISTER ADDRESSING MODE
                MOV   DIVQ , AX
                MOV   DIVR , DX
                INT 03H
        CODE ENDS
        END START
END
```

Result:

UNSIGNED NUMBERS

INPUT: OPR1 =

OPR2 =

OUTPUT: ALL RESULTS ARE STORED IN EXTRA SEGMENT (ES)

SUM =

DIFF=

MUL=

MUL+2=

DIVQ=

DIVR=

EXPERIMENT NO.3
SORTING AN ARRAY FOR 8086

AIM: Write and execute an ALP to 8086 processor to sort the given 16-bit numbers in Ascending and Descending order.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I: Initialize the data segment memory.

Step II: Initialize the number of elements counter

Step III : Initialize the comparisons counter..

Step IV: Load the numbers into respective registers

Step V: Compare the elements. If first element < second element goto step **VII** Else go to next step.

Step VI: Swap the numbers in the memory..

Step VII: Increment memory pointer & Decrement the comparison counter.

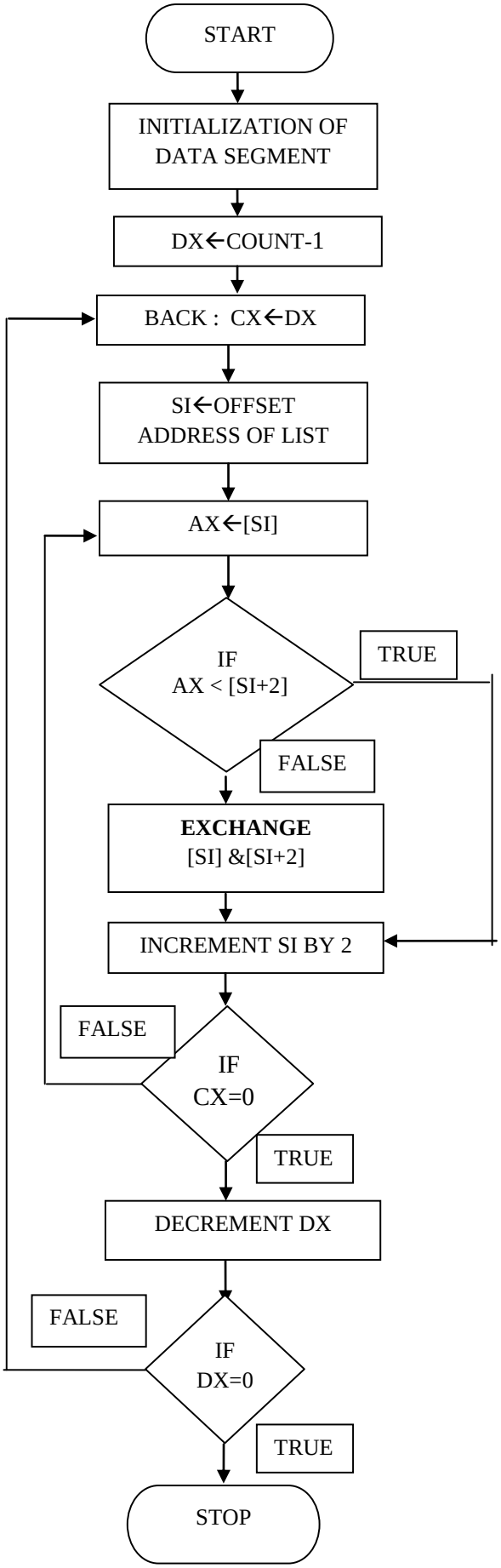
Step VIII: Is count = 0 ? if yes go to next step else go to **step IV**.

Step IX: decrement the element counter.

Step X: Is count not 0 ? go **Step III** else go to next step

Step IX: Stop & terminate the program**.**

FLOW CHART:


START
INITIALIZATION OF DATA SEGMENT
DX←COUNT-1
BACK : CX←DX
SI←OFFSET ADDRESS OF LIST
AX←[SI]
IF AX < [SI+2]
TRUE
FALSE
EXCHANGE [SI] &[SI+2]
INCREMENT SI BY 2
FALSE
IF CX=0
TRUE
DECREMENT DX
FALSE
IF DX=0
TRUE
STOP

Program:

ASCENDING ORDER

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
LIST DW 0125H,0144H,3001H,0003H,0002H
COUNT EQU 05H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
      MOV DS,AX
      MOV DX,COUNT-1
BACK: MOV CX,DX
      MOV SI, OFFSET LIST
AGAIN: MOV AX,[SI]
      CMP AX,[SI+2]
      JC GO
      XCHG AX,[SI+2]
      XCHG AX,[SI]
   GO:INC SI
      INC SI
      LOOP AGAIN
      DEC DX
      JNZ BACK
      INT 03H
CODE ENDS
END START
END
```

Result:
INPUT: (DS: 0000H) = 25H,01H,44H,01H,01H,30H,03H,00H,02H,00H
OUTPUT: (DS: 0000H) =

DESCENDING ORDER

ALGORITHM:

Step I: Initialize the data segment memory.

Step II: Initialize the number of elements counter

Step III : Initialize the comparisons counter..

Step IV: Load the numbers into respective registers

Step V: Compare the elements. If first element > second element go to **step VII**

Else go to next step.

Step VI: Swap the numbers in the memory..

Step VII: Increment memory pointer & Decrement the comparison counter.

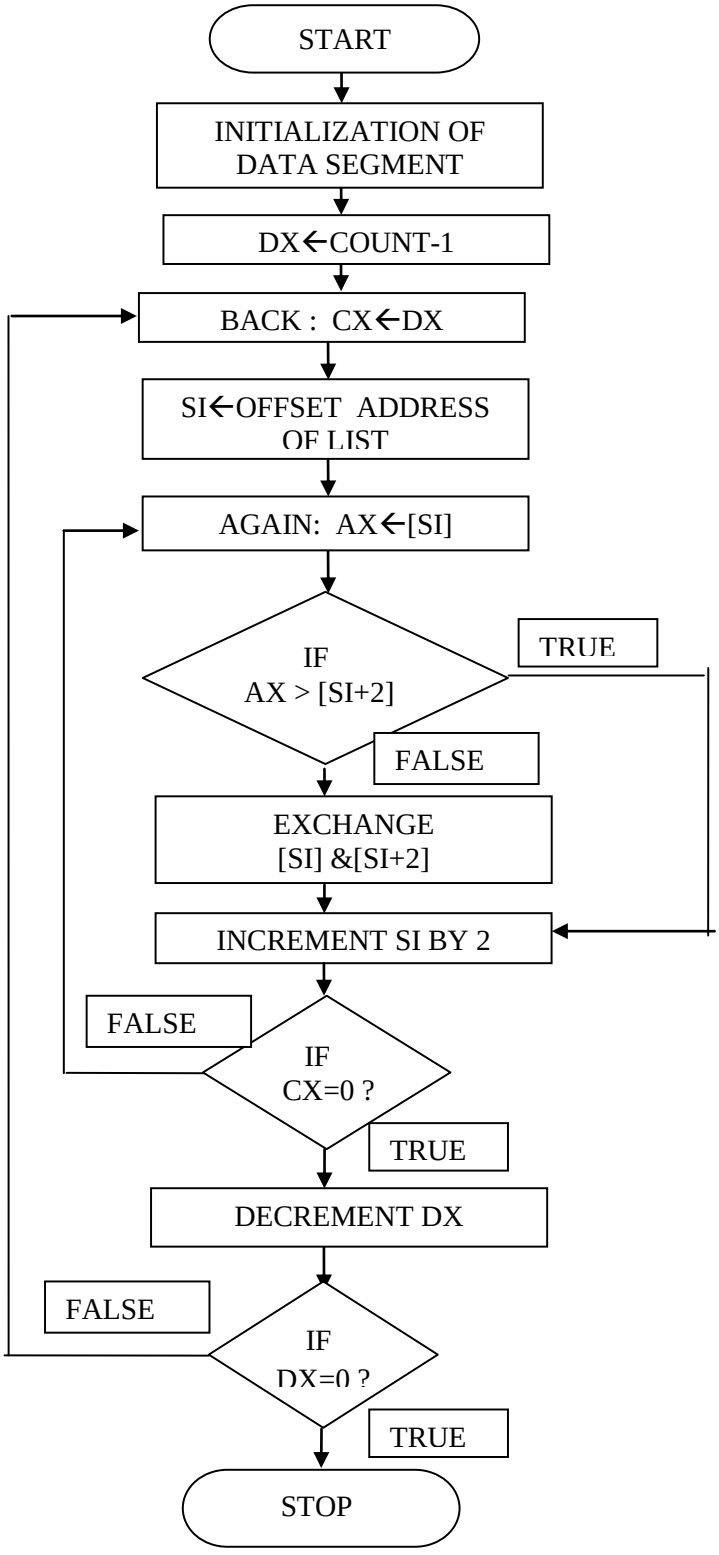
Step VIII: Is count = 0 ? if yes go to next step else go to **step IV**.

Step IX: decrement the element counter.

Step X: Is count not 0 ? go **Step III** else go to next step

Step IX: Stop & terminate the program**.**

FLOW CHART:


START
INITIALIZATION OF DATA SEGMENT
DX←COUNT-1
BACK : CX←DX
SI←OFFSET ADDRESS OF LIST
AGAIN: AX←[SI]
IF AX > [SI+2]
TRUE
FALSE
EXCHANGE [SI] &[SI+2]
INCREMENT SI BY 2
FALSE
IF CX=0 ?
TRUE
DECREMENT DX
FALSE
IF DX=0 ?
TRUE
STOP

DESCENDING ORDER

PROGRAM:

```
ASSUME CS: CODE, DS:DATA
DATA SEGMENT
LIST  DW  0125H,0144H,3001H,0003H,0002H
COUNT  EQU  05H
DATA ENDS
CODE SEGMENT
START:MOV AX,DATA
      MOV DS,AX
      MOV DX,COUNT-1
BACK:MOV CX,DX
      MOV SI,OFFSET LIST
AGAIN:MOV AX,[SI]
      CMP AX,[SI+2]
      JAE GO
      XCHG AX,[SI+2]
      XCHG AX,[SI]
   GO:INC SI
      INC SI
      LOOP AGAIN
      DEC DX
      JNZ BACK
      INT 03H
CODE ENDS
END START
END
```

Result:

INPUT: (DS: 0000H) = 25H,01H,44H,01H,01H,30H,03H,00H,02H,00H

OUTPUT: (DS: 0000H) =

EXPERIMENT NO: 4
SEARCH ING FOR CHARACTER IN A STRING

AIM: Write an ALP for searching for a number or character in a string for 8086.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I **:** Initialize the Data segment (DS) & Extra segment(ES)

Step II **:** Load the offset address of the string into SI .

Step III **:** Load the number of elements in the string into CX register

Step IV **:** Move the character to be searched into the AL register

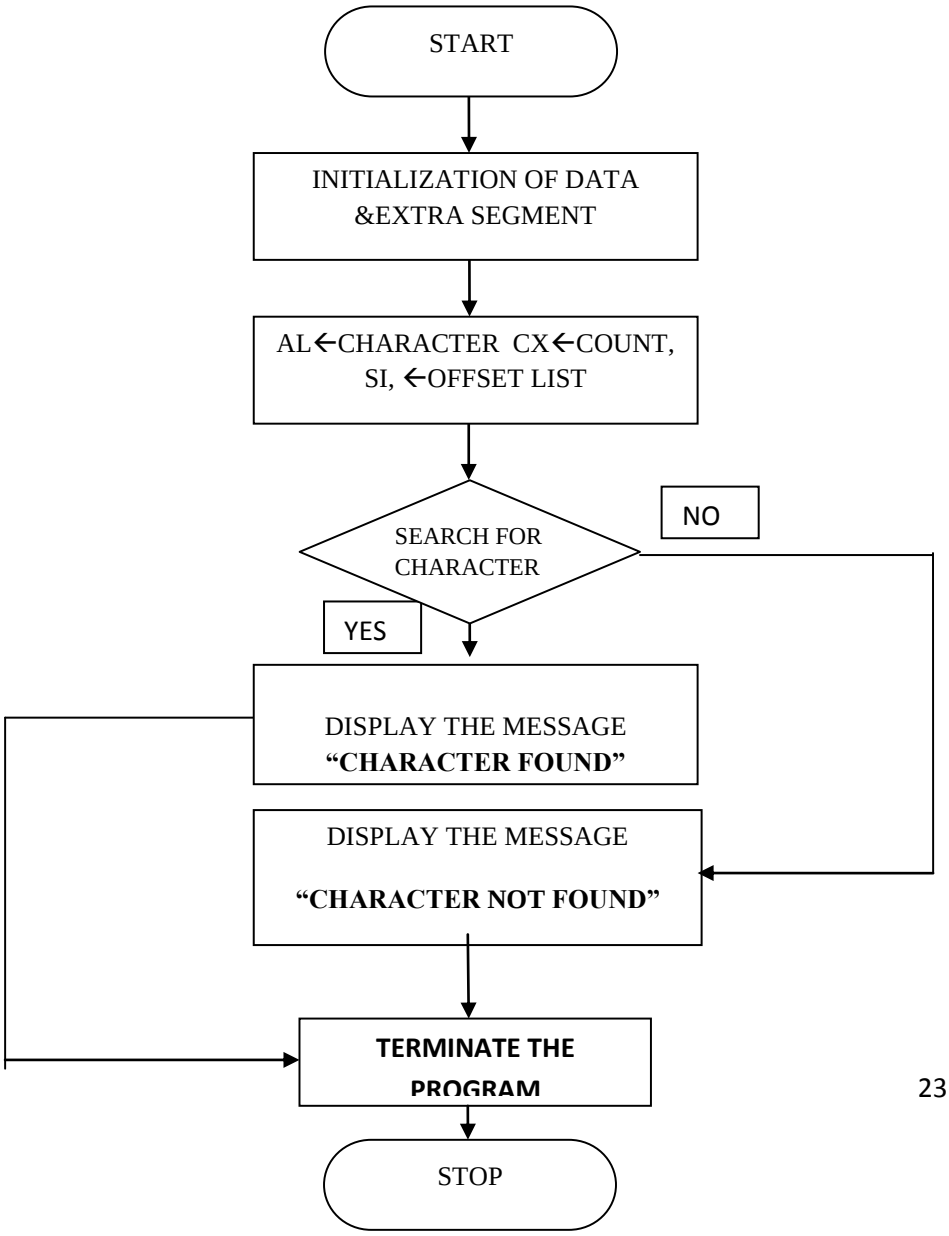
Step V **:** Scan for the character in ES. If the character is not found go to step **VII** else go to next step.

Step VI : Display the message that character found and go to step **VIII**

Step VII **:** Display the message that character not found

Step VII **:** Stop.& Terminate the program

FLOW CHART:



Program:

```
ASSUME CS: CODE, DS: DATA
        DATA SEGMENT
                STRING DB 'MRCET$'
                SLEN EQU ($-STRING)
                CHAR DB 'E'
                MSG1 DB 'THE CHARACTER IS FOUND$'
                MSG2 DB 'THE CHARACTER IS NOT FOUND$'
        DATA ENDS
      CODE SEGMENT
      START: MOV AX, DATA
                MOV DS, AX
                MOV ES, AX
                LEA SI, STRING
                MOV CX, LEN
                MOV AL, CHAR
                CLD
       REPNE SCASB
                JNZ EXIT
                LEA DX, MSG1
                MOV AH, 09H
                INT 21H
                JMP GOTOEND
        EXIT: LEA DX, MSG2
                MOV AH, 09H
                 INT 21H
   GOTOEND: MOV AH, 4CH
                INT 21H
    CODE ENDS
END START
```

RESULT: **INPUT:** **OUTPUT:**

EXPERIMENT NO.5
SRTING MANIPULATIONS FOR 8086

AIM: To write an assembly language program to move the block of data from a source BLOCK to the specified destination BLOCK.

TOOLS: PC installed with MASM 6.11

A) BLOCK TRANSFER

ALGORITHM:

Step I **:** Initialize the Data segment (DS) & Extra segment(ES)

Step II **:** Load the offset address of source and destination the string into SI and DI.

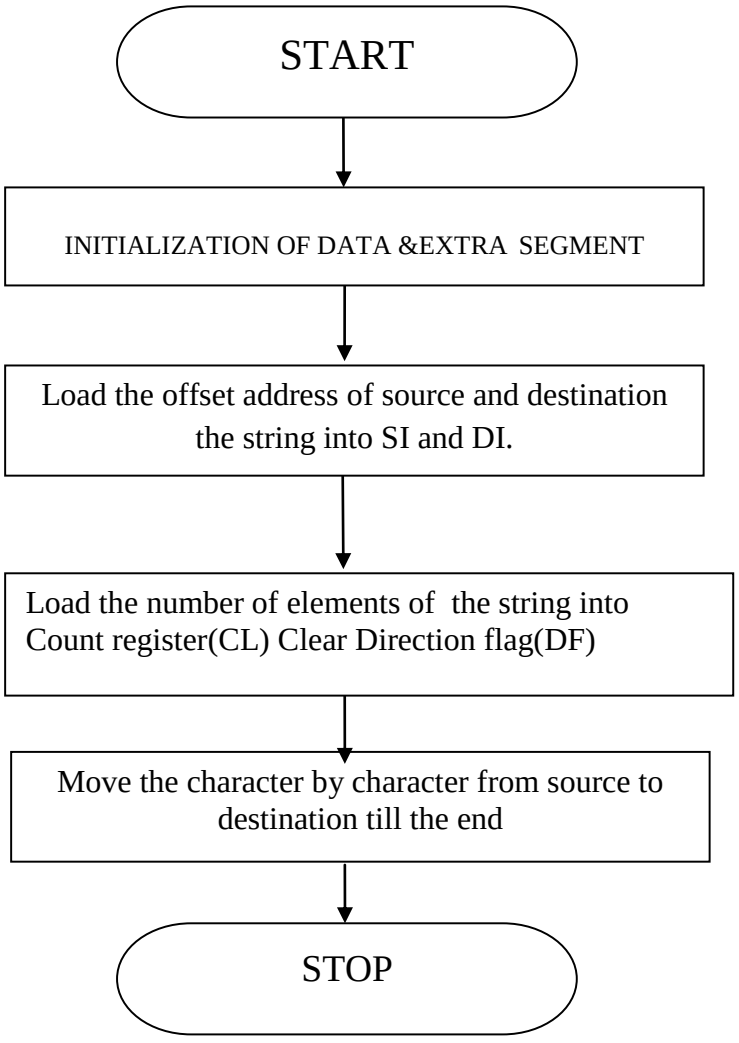
Step III **:** Load the number of elements of the string into Count register(CL)

Step IV **:** Clear Direction flag(DF) to make SI and DI into auto increment mode

Step V **:** move the character by character from source to destination till the end

Step VI **:** Stop & Terminate the program

Flow chart:



PROGRAM:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
        STRING   DB   'MICROPROCESSOR$'
        COUNT   EQU ($-STRING)
        ORG 0070H
DATA ENDS
EXTRA SEGMENT
        ORG 0010H
        STRING1 DB  ?
EXTRA ENDS
CODE SEGMENT
START:
        MOV AX,DATA
        MOV DS,AX
        MOV AX, EXTRA
        MOV ES,AX
        MOV SI,OFFSET STRING
        MOV DI,OFFSET STRING1
        MOV CL,COUNT
        CLD
   REP MOVSB
       INT 03H
CODE ENDS
END START
END
```

RESULT:

INPUT: (DS: 0000H) = MICROPROCESSOR

OUTPUT: (ES: 0010H) = MICROPROCESSOR

B) REVERSE STRING

AIM: To write an assembly language program to reverse the given string.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I **:** Initialize the Data segment (DS) & Extra segment(ES)

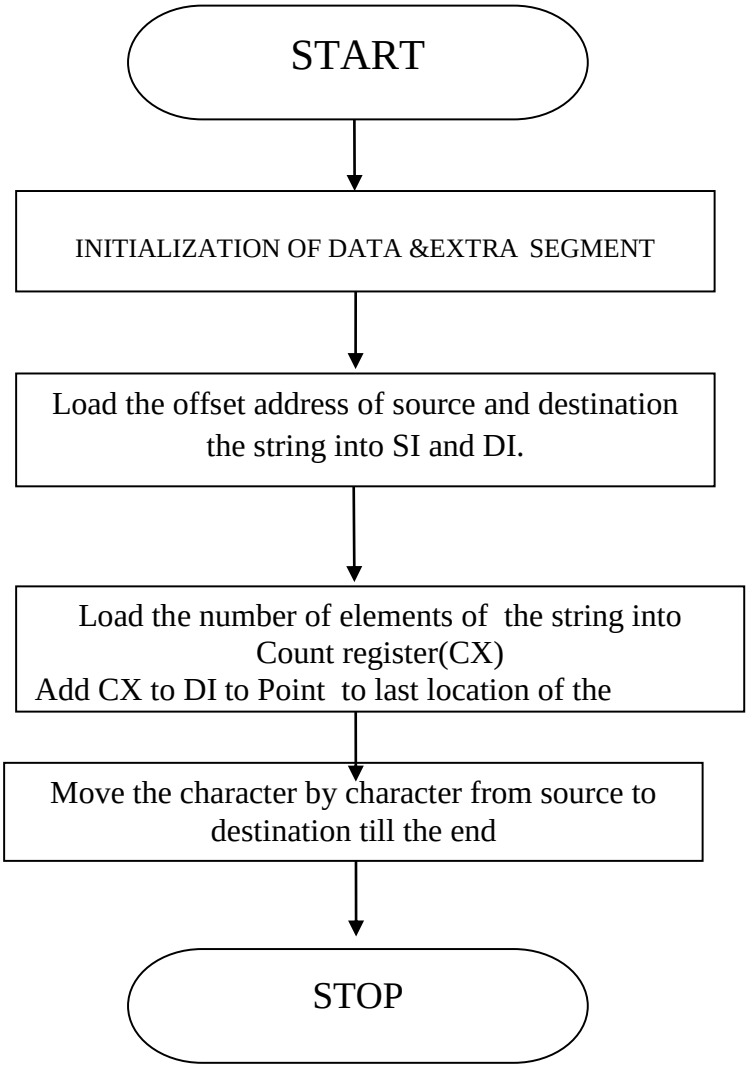
Step II **:** Load the offset address of source and destination the string into SI and DI.

Step III **:** Load the number of elements of the string into Count register(CX)

Step IV **:** Add CX to DI to Point to last location of the memory

Step V **:** move the character by character from source to destination till the end

Step VI **:** Stop & Terminate the program



PROGRAM:

```
ASSUME CS: CODE, DS: DATA ,ES:EXTRA
DATA SEGMENT
STRING1 DB 'MICROPROCESSOR$'
STRLEN EQU ($-STRING1)
DATA ENDS
 EXTRA SEGMENT
 STRING2 DB ?
 EXTRA ENDS
CODE SEGMENT
START: MOV AX, DATA
        MOV DS, AX
        MOV AX, EXTRA
        MOV ES, AX
        MOV SI, OFFSET STRING1
        MOV DI, OFFSET STRING2

        MOV CX, STRLEN-1
        ADD DI, CX
        MOV DL,'$'
        MOV ES:[DI],DL
AGAIN: DEC DI
        MOV AL,DS:[SI]
        MOV ES:[DI],AL
        INC SI
        DEC CX
        JNZ AGAIN
        INT 3H
```

RESULT:

INPUT: ' MICROPROCESSOR'

OUTPUT: 'ROSSECORPORCIM'

C) LENGTH OF THE STRING

AIM: To write an assembly language program to find the length of the given string.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I : Initialize the data segment(DS)

Step II : Initialize the counter CL with 0

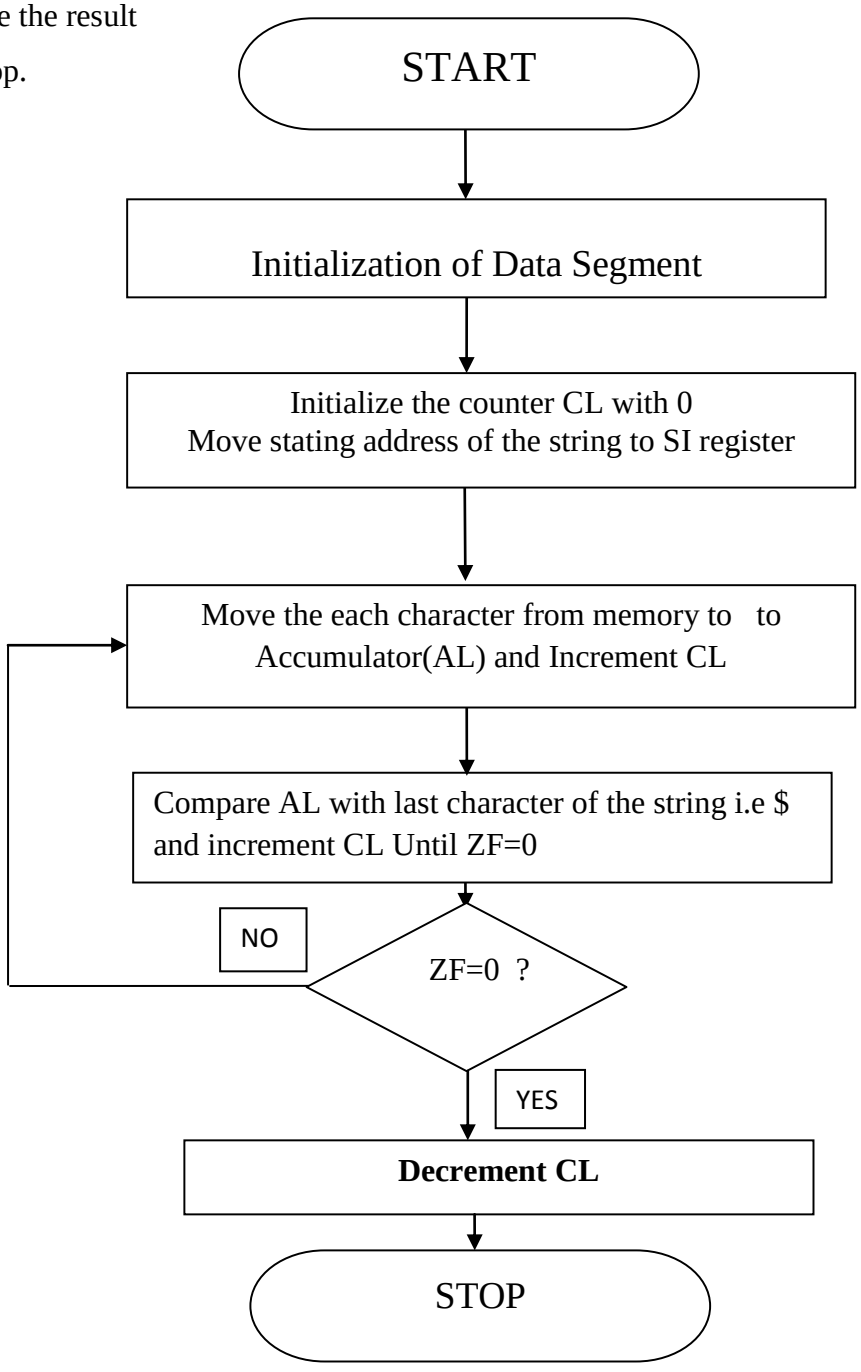
Step III : Move stating address of the string to SI register

Step IV : Move the each character from memory to to Accumulator(AL)

Step V : Compare AL with last character of the string i.e $ and increment CL Until ZF=0

Step VII : Store the result

Step VIII : Stop.



PROGRAM:

```
ASSUME CS:CODE, DS:DATA
DATA SEGMENT
        STRING1 DB 'MICROPROCESSOR AND INTERFACING LAB$'
        SLENGTH DB 0
DATA ENDS
CODE SEGMENT
START:  MOV AX, DATA
        MOV DS, AX
        SUB CL, CL
        MOV SI, OFFSET STRING1
        CLD
BACK:   LODSB
        INC CL
        CMP AL,'$'
        JNZ BACK
        DEC CL
        MOV SLENGTH, CL
        INT 03H
CODE ENDS
END START
```

RESULT: INPUT: 'MICROPROCESSOR AND INTERFACING LAB

OUTPUT:

D) STRING COMPARISON

AIM: Write an ALP to 8086 to compare the given strings.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I : Initialize the data segment (DS) & extra Segment as per requirement

Step II : Load the offset address of source and destination of the string into SI and DI.

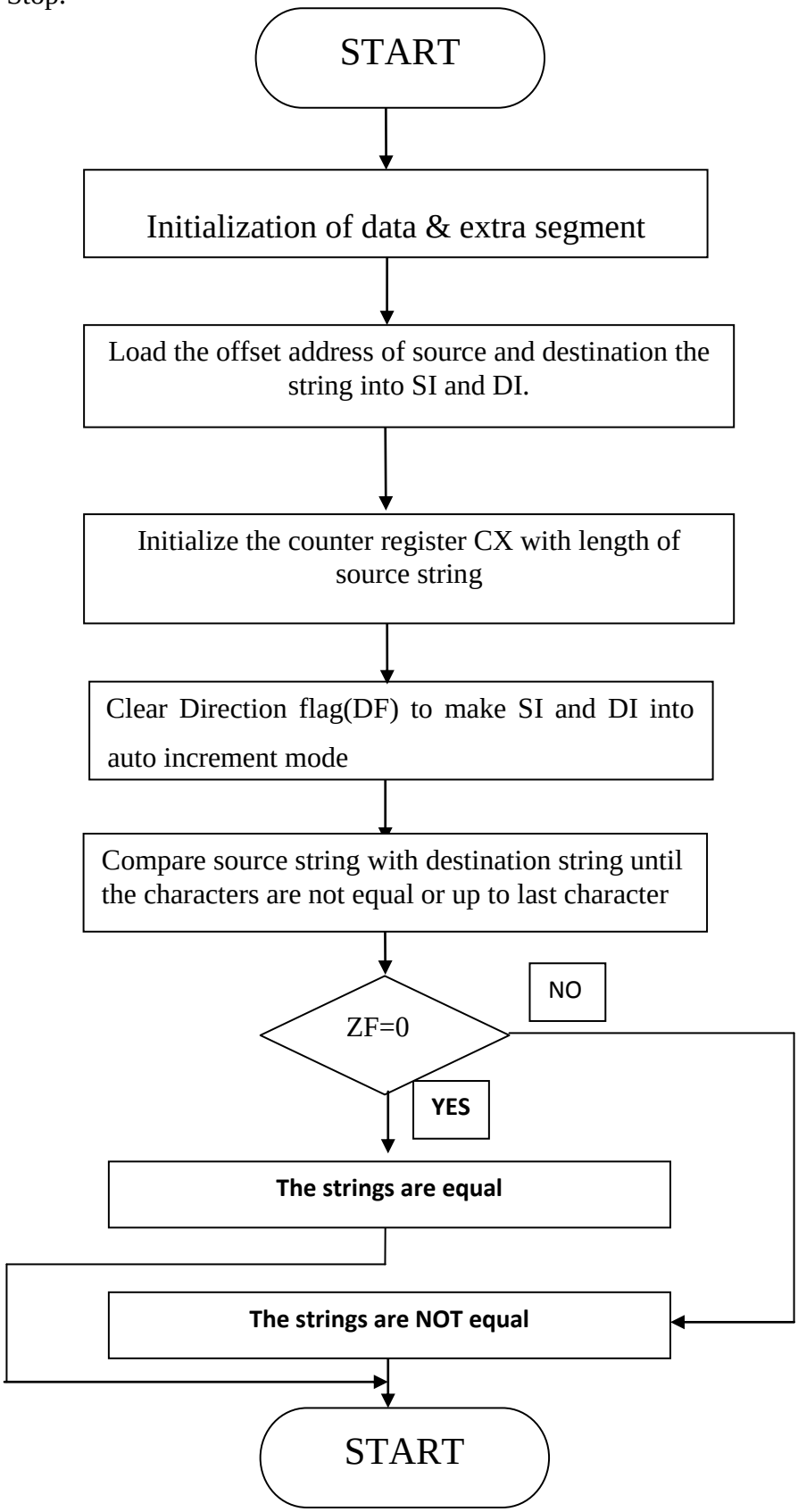
Step III : Initialize the counter register CX with length of source string

Step IV : Clear Direction flag (DF) to make SI and DI into auto increment mode

Step V : Compare source string with destination string until the characters are not equal or up to last last character

Step VII : If ZF=0 the strings are equal or otherwise the strings are not equal

Step VIII : Stop.



Program:

```
ASSUME CS: CODE, DS:DATA, ES:EXTRA
DATA SEGMENT
        STRING1 DB 'MRCET'
        STRLEN EQU ($-STRING1)
        SNOTEQUAL DB 'STRINGS ARE UNEQUAL$'
        SEQUAL DB 'STRINGS ARE EQUAL$'
DATA ENDS
EXTRA SEGMENT
        STRING2 DB 'MRCET'
EXTRA ENDS
CODE SEGMENT
      START: MOV AX,DATA
             MOV DS,AX
             MOV AX,EXTRA
             MOV ES,AX
             MOV SI,OFFSET STRING1
             MOV DI,OFFSET STRING2
             CLD
             MOV CX,STRLEN
         REPZ CMPSB
             JZ FORW
             MOV AH, 09H
             MOV DX, OFFSET SNOTEQUAL
             INT 21H
             JMP EXITP
        FORW: MOV AH,09H
             MOV DX, OFFSET SEQUAL
             INT 21H
       EXITP: MOV AH, 4CH
             INT 03H
CODE ENDS
END START
```

RESULT: INPUT:

OUTPUT:

(E) STRING INSERTION

AIM: To Write and execute an Assembly language Program (ALP) to 8086 processor to insert or delete a character/ number from the given string.

TOOLS: PC installed with MASM 6.11

ALGORITHM:

Step I : Initialize the data segment (DS) & extra segment (ES)

Step II : Load the offset address of source and destination of the string into SI and DI.

Step III : Initialize the counter register CX with length of first part of source string

Step IV : Copy the first part of STRING1 in to STRING3 of extra segment

Step V : Load the offset address of STRING2 in to SI

Step VI : Copy the STRING2 in to STRING3 of extra segment after first string of STRING1

Step VII: Load the new offset address of source of the STRING1 into SI

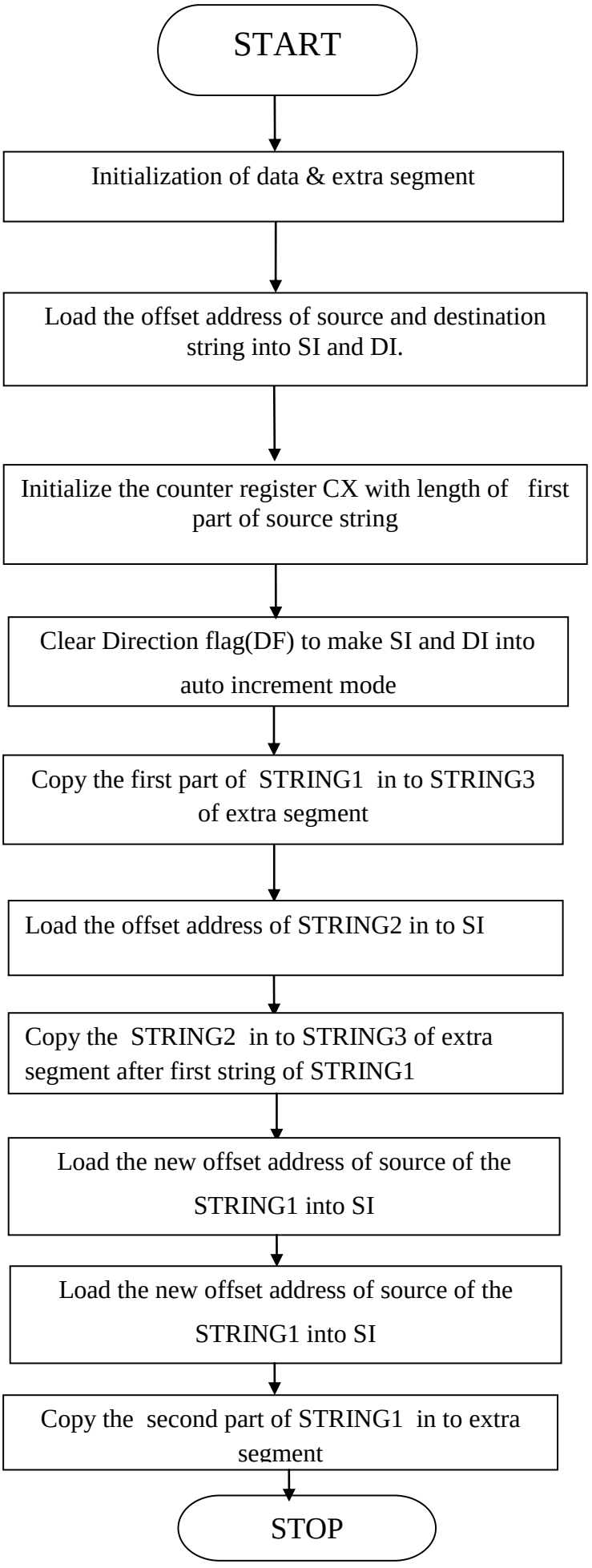
Step VIII: Copy the second part of STRING1 in to extra segment

Step IX: Stop

Program:

```
ASSUME CS: CODE, DS:DATA,ES:EXTRA
DATA SEGMENT
        STRING1 DB 'MICROPROCESSOR INTERFACING LAB$'
        STRING2 DB ‘AND ’
        STRLEN EQU ($-STRING1)
        ORG 0070H
DATA ENDS
EXTRA SEGMENT
        ORG 0010H
        STRING3 DB 38 DUP(0)
EXTRA ENDS
CODE SEGMENT
START:      MOV AX, DATA
```

FLOW CHART:


START
Initialization of data & extra segment
Load the offset address of source and destination string into SI and DI.
Initialize the counter register CX with length of first part of source string
Clear Direction flag(DF) to make SI and DI into auto increment mode
Copy the first part of STRING1 in to STRING3 of extra segment
Load the offset address of STRING2 in to SI
Copy the STRING2 in to STRING3 of extra segment after first string of STRING1
Load the new offset address of source of the STRING1 into SI
Load the new offset address of source of the STRING1 into SI
Copy the second part of STRING1 in to extra segment
STOP

```
MOV DS, AX
        MOV AX, EXTRA
        MOV ES, AX
            MOV SI, OFFSET STRING1
            MOV DI, OFFSET STRING3
            CLD
            MOV CX, 15
          REP MOVSB

        CLD
        MOV SI, OFFSET STRING2
        MOV CX,4
    REP MOVSB

        MOV SI, OFFSET STRING1
        ADD SI,15
        MOV CX, 15
    REP MOVSB
        INT 3H
CODE ENDS
END START
```

RESULT:

INPUT: **STRING1:** 'MICROPROCESSOR INTERFACING LAB'

STRING2:‘AND ’

OUTPUT: **STRING3:** ‘MICROPROCESSOR AND INTERFACING LAB'

(F) STRING DELETION

```
ASSUME CS: CODE, DS:DATA, ES:EXTRA
  DATA SEGMENT
        STRING1 DB 'MICROPROCESSOR AND INTERFACING LAB$'
        ORG 0070
  DATA ENDS
EXTRA SEGMENT
        ORG 0010H
        STRING2 DB 40 DUP (0)
EXTRA ENDS
CODE SEGMENT
START:      MOV AX, DATA
            MOV DS, AX
            MOV AX, EXTRA
            MOV ES, AX
                  MOV SI, OFFSET STRING1
                  MOV DI, OFFSET STRING2
                  CLD
                  MOV CX, 15
              REP MOVSB
            CLD
            MOV SI, OFFSET STRING1
            ADD SI, 19
            MOV CX, 15
            REP MOVSB
            INT   03H
CODE ENDS
END START
```

RESULT:

INPUT: **STRING1:** MICROPROCESSOR AND INTERFACING LAB'

OUTPUT: **STRING2:** 'MICROPROCESSOR INTERFACING LAB'

PART-B

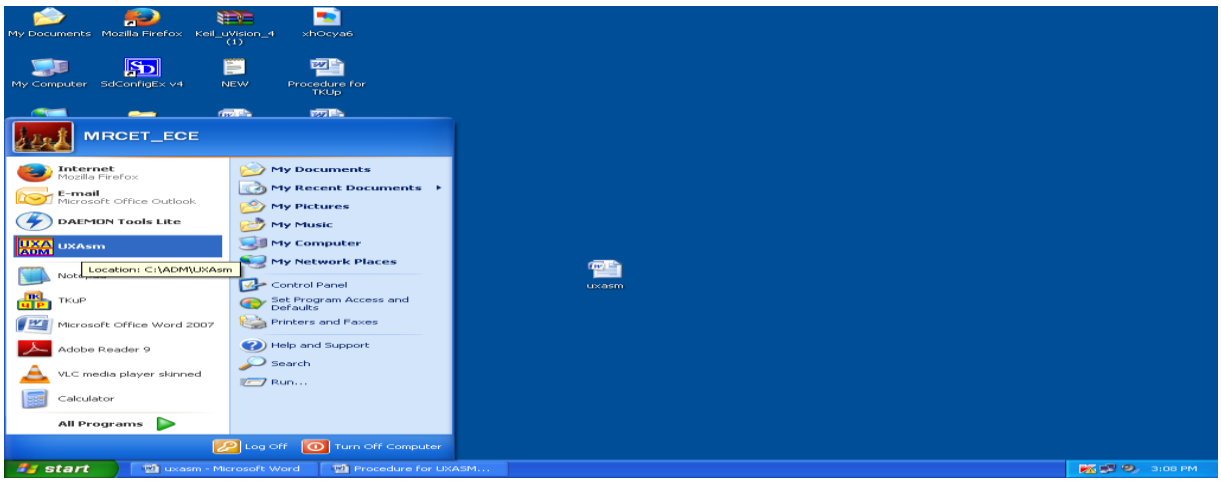
8086 Programs can also executed by using ADM TK_µP Trainer kits. The Assembly language programs(ALP) can be executed by the following steps

1. **UxAsm**
2. **TKµP**

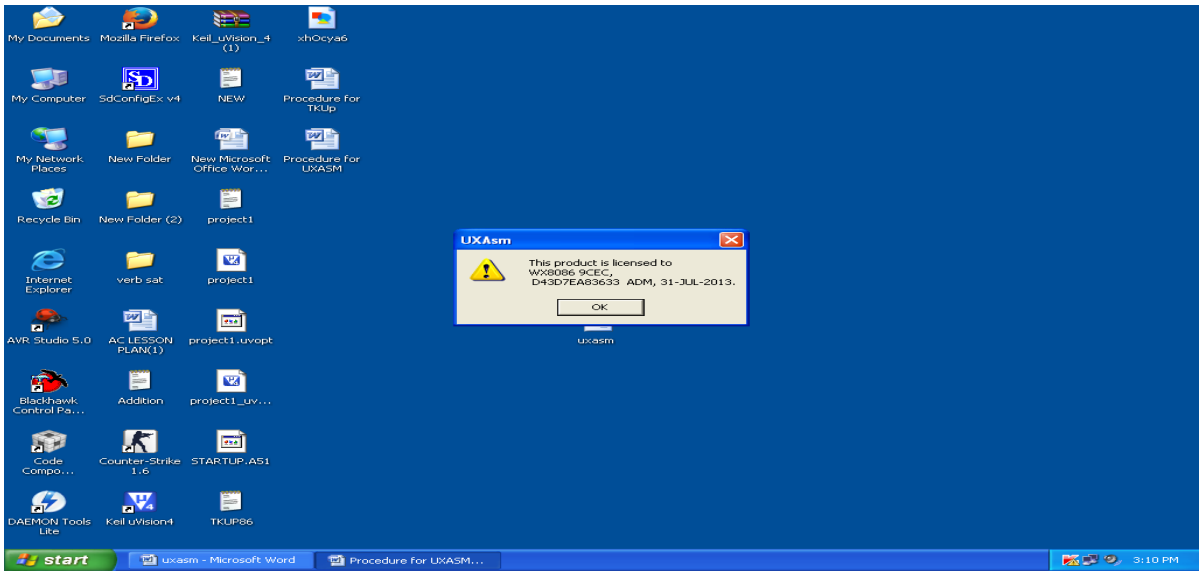
1.UxAsm : It is used to translate the Assembly language program in to Machine language (Hex File).The input file to the UxAsm is .asm and one of the output files is hex file

Procedure for UXASM:

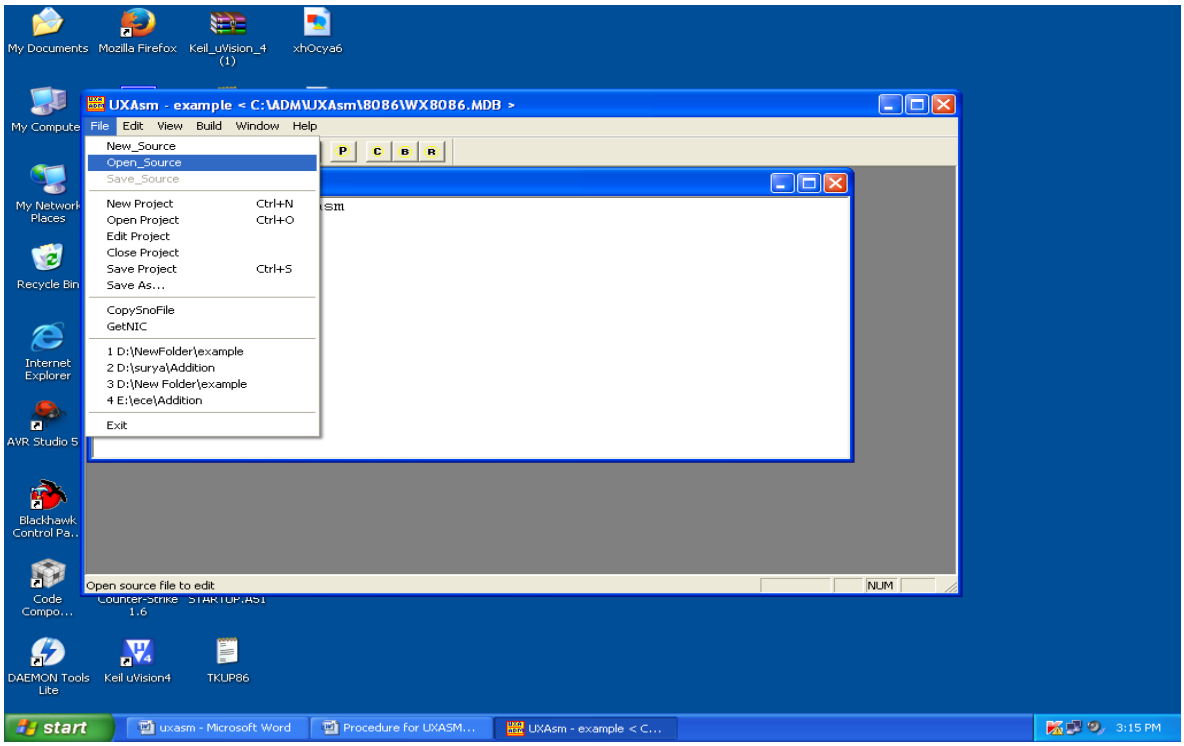
1 Go to start and select UXAsm



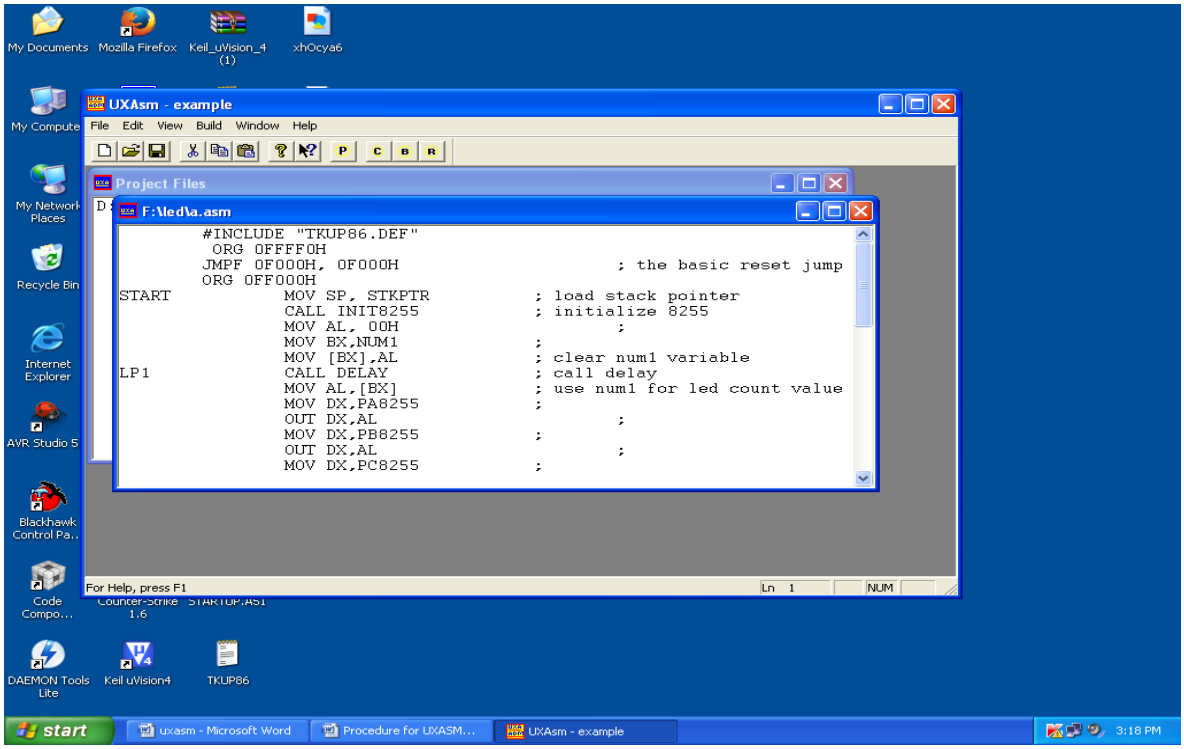
2. Verify the license by observing the following window and click "OK"



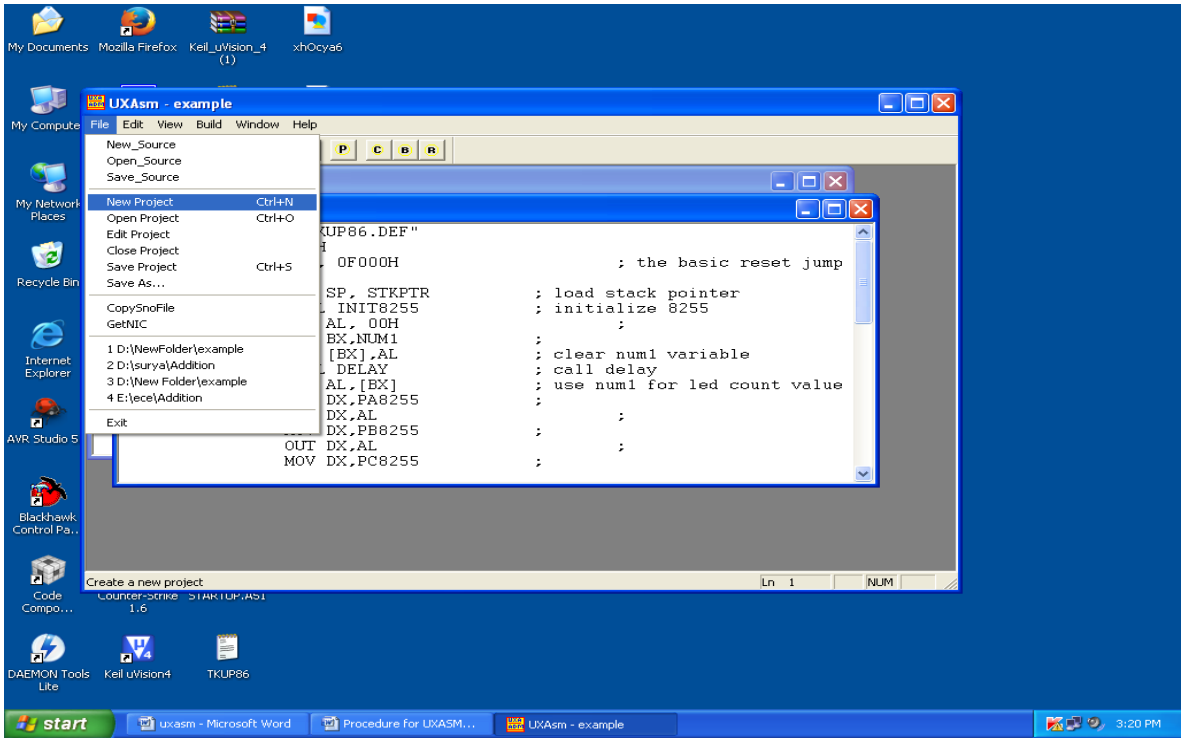
3. Go to file and select "open source" and browse the source file(.ASM file)



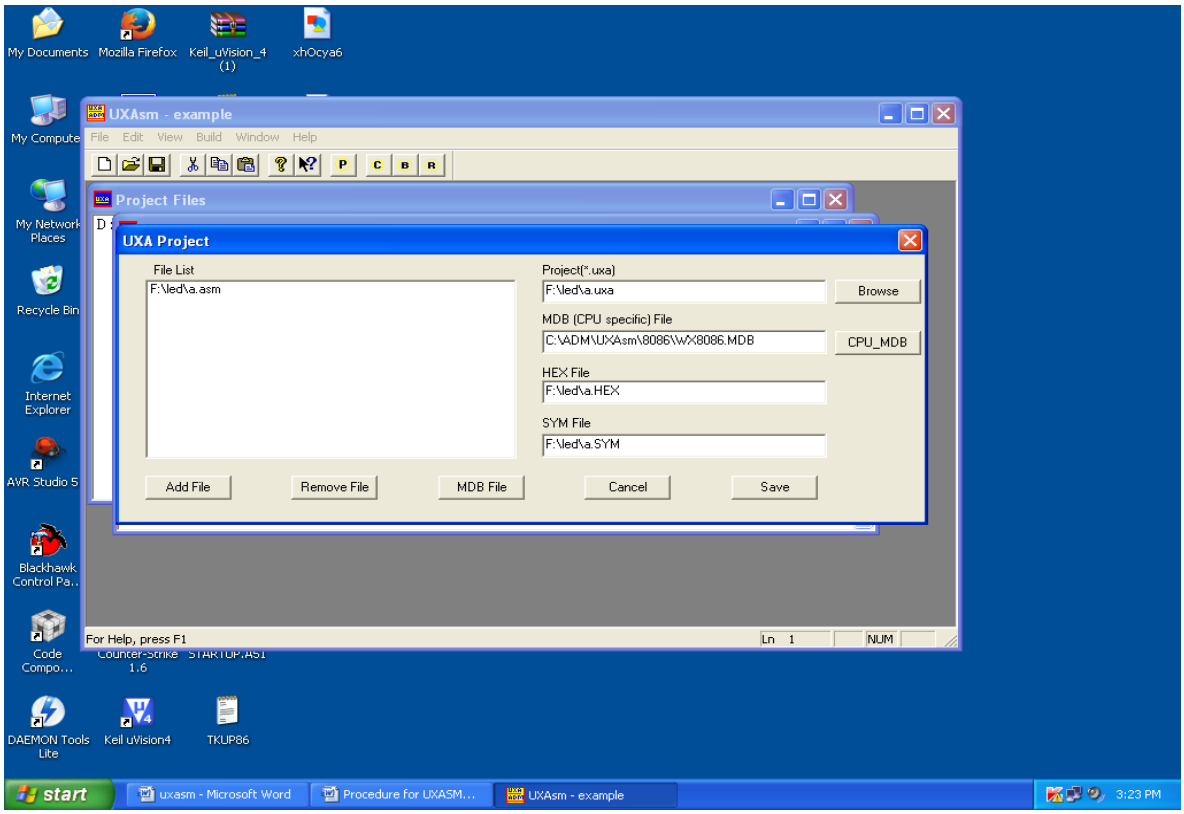
4. Observe the following window which shows the source code.



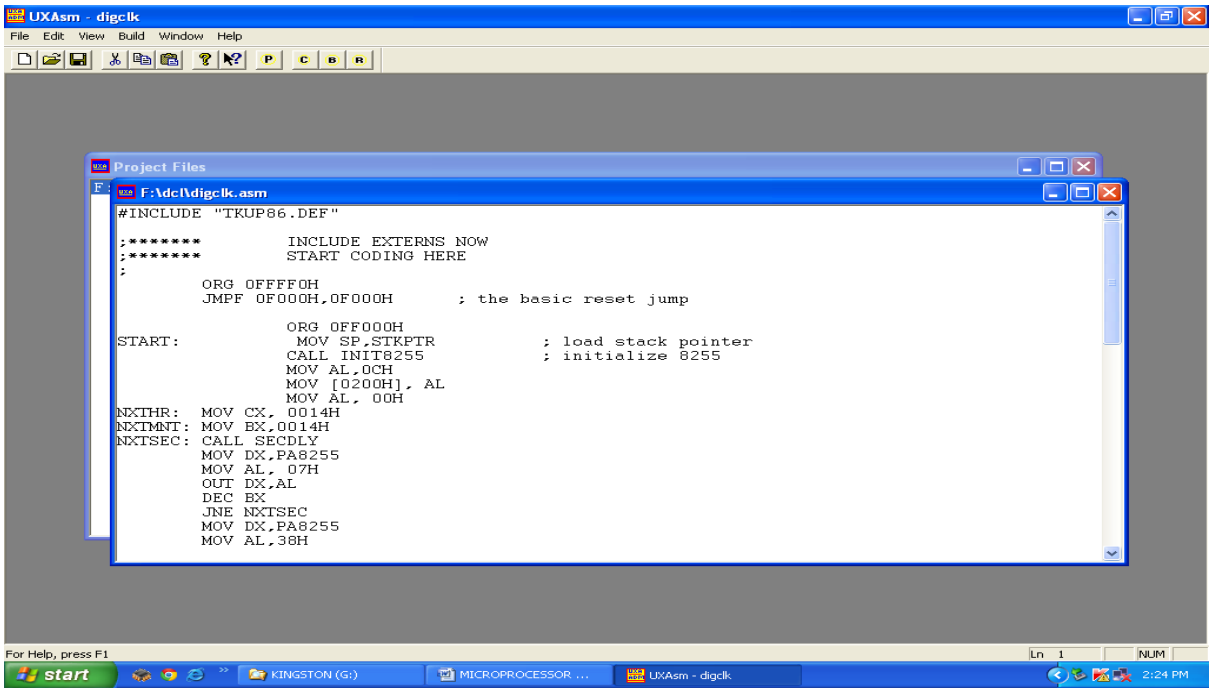
5. Again go to file select "New Project"



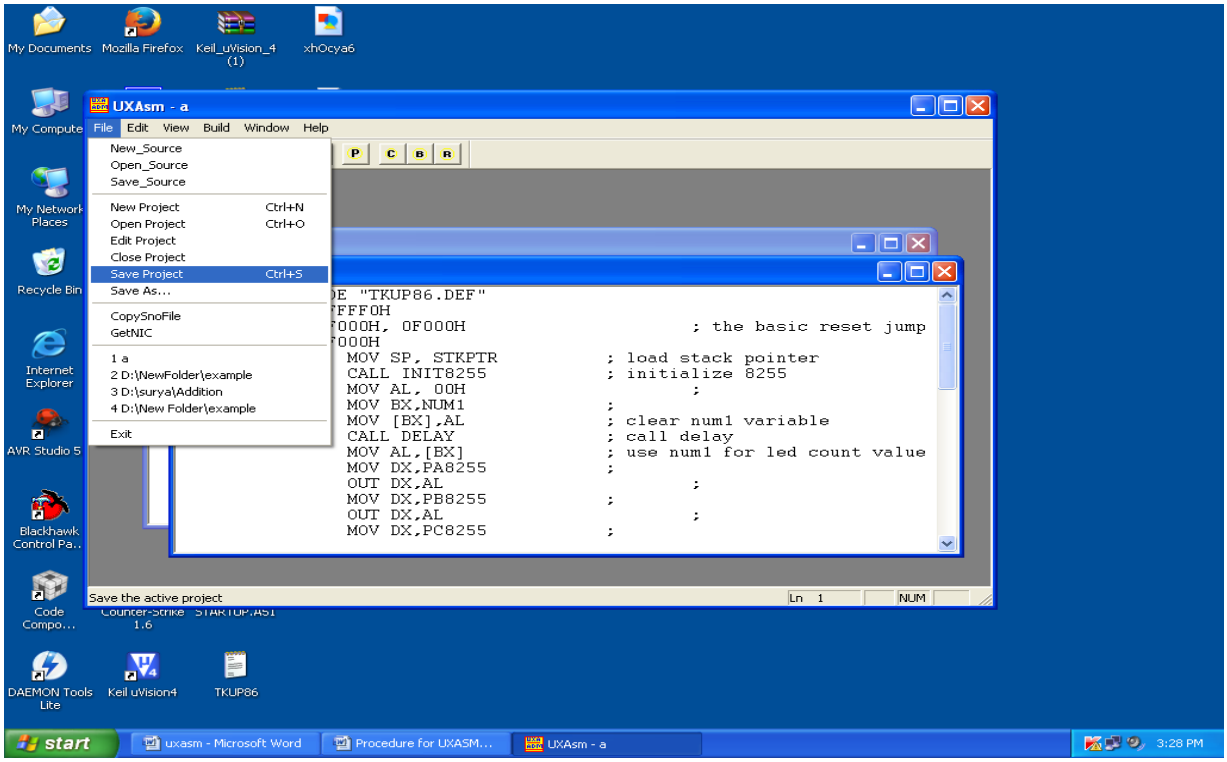
6. To add source file click on "Add File" and browse the source file and provide the source file path with .Uxa extension in the Project and press Tab and Press "Save" and click on "OK"



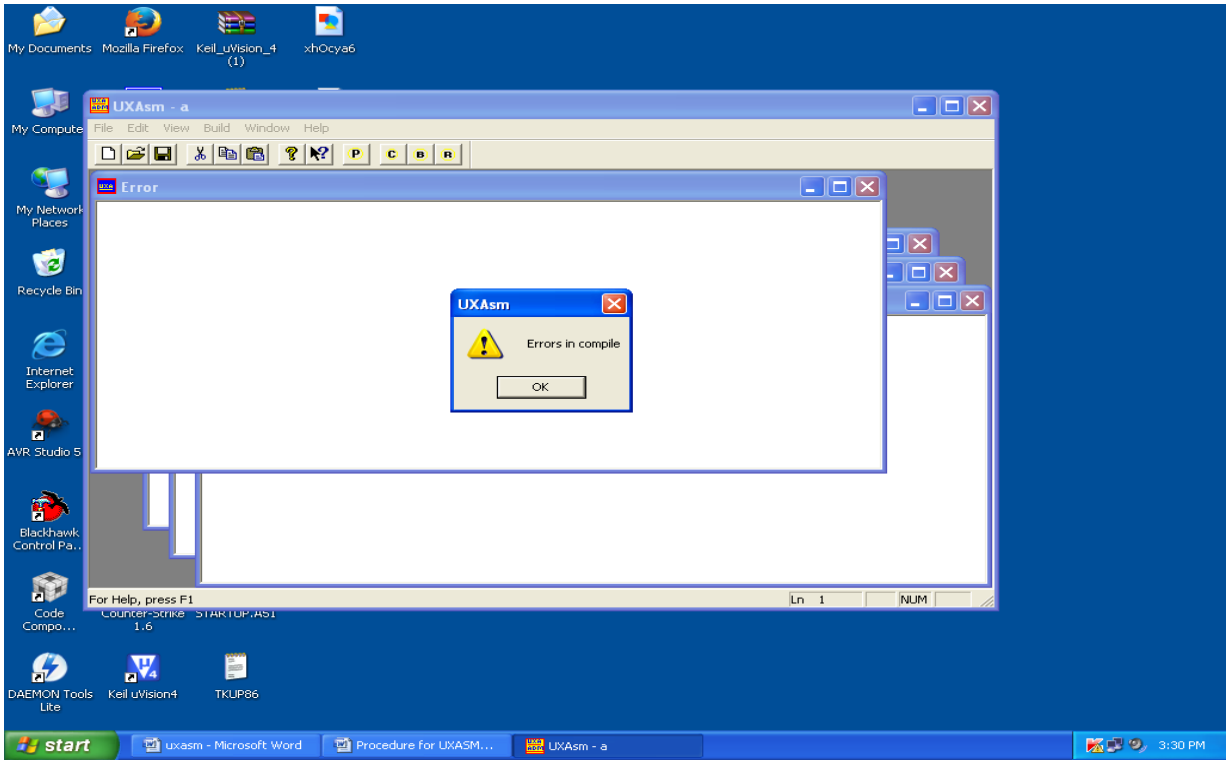
7. Observe the following window and double click on path of the File to view the program



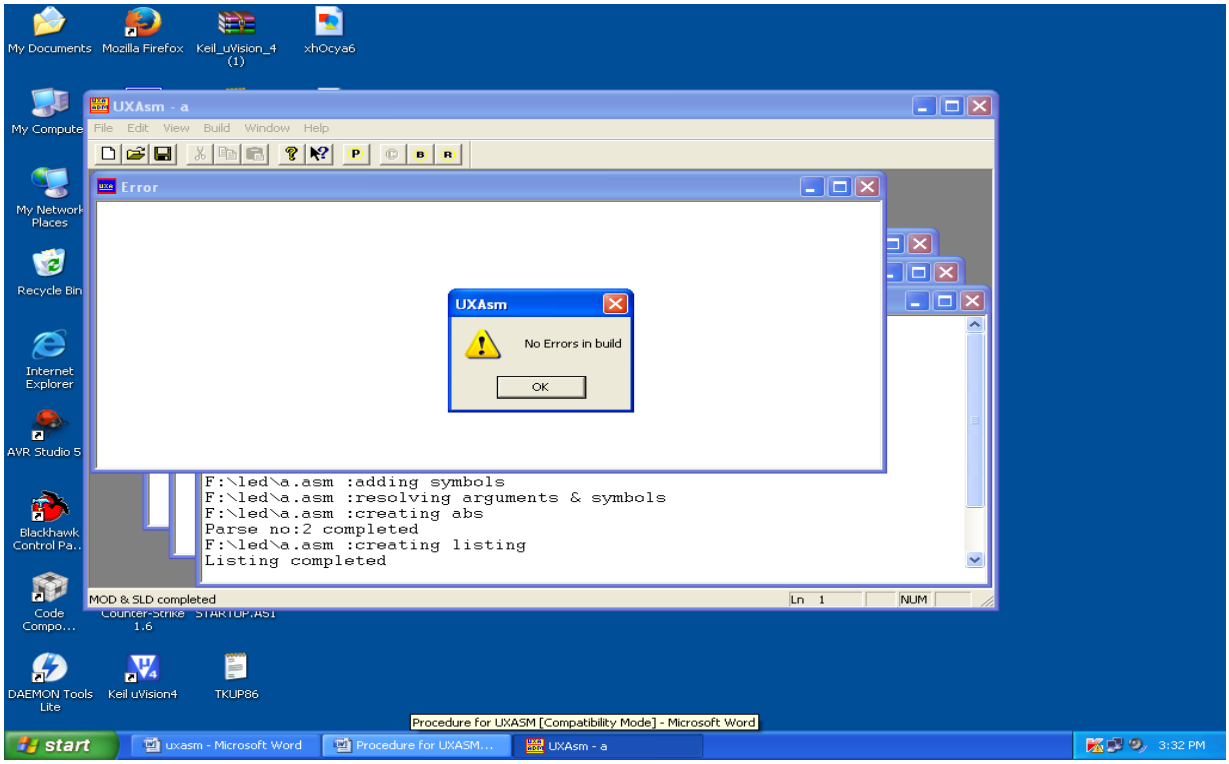
8. To save the project Go to File select "save project"



9. To compile the program , click on "C" and observe the following window



10. If any errors, Fix the errors, click on "OK" and click on "B" to Build the program



2.TKµP

Introduction:

TKµP is an ideal trainer cum development boards for Microprocessors like Z80, 8032, 8085, 8088 and 8086. All interface is provided through 10 pin polarized Box Headers. TKµP user interface software communicates with the TKµP hardware through PC parallel port LPT1 and provides fast download of hex files. The PC user interface can open multiple windows for memory Dump and List. Multiple dump windows is also useful to study memory move operations and programs.

TKµP is made up of three sections:

1. CPU specific daughter board.

2. Base board section: It has following features

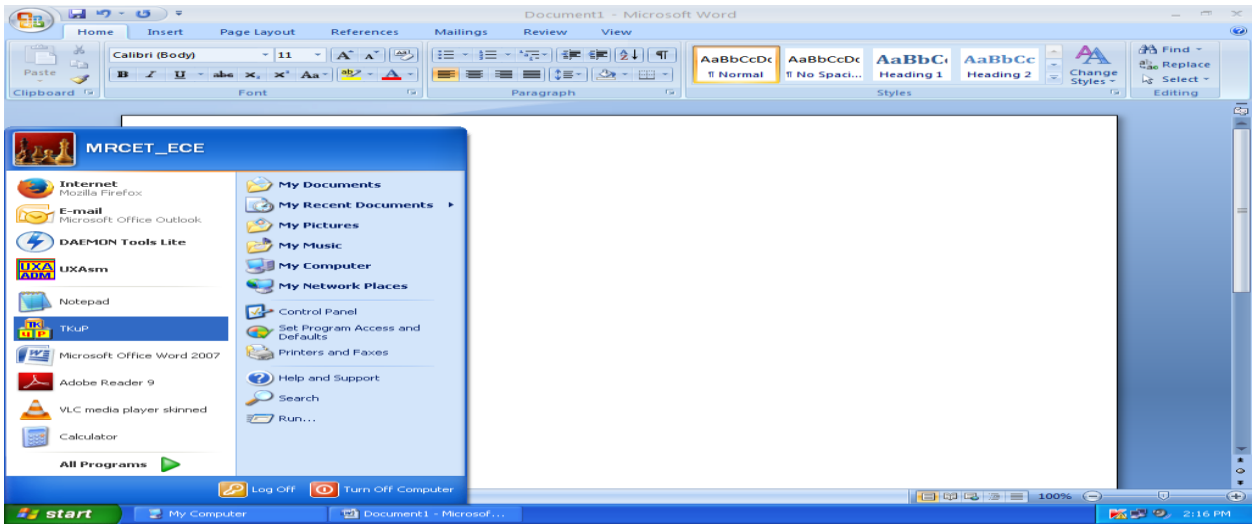
- Four sockets for memory which can accommodate maximum 4x128KB.
- 8279 key board display controller.
- 8255 IO expander.
- 8155 IO expander with timer counter.
- 8251 Asynchronous serial Transmitter and Receiver.

3. User interface section: It has following features

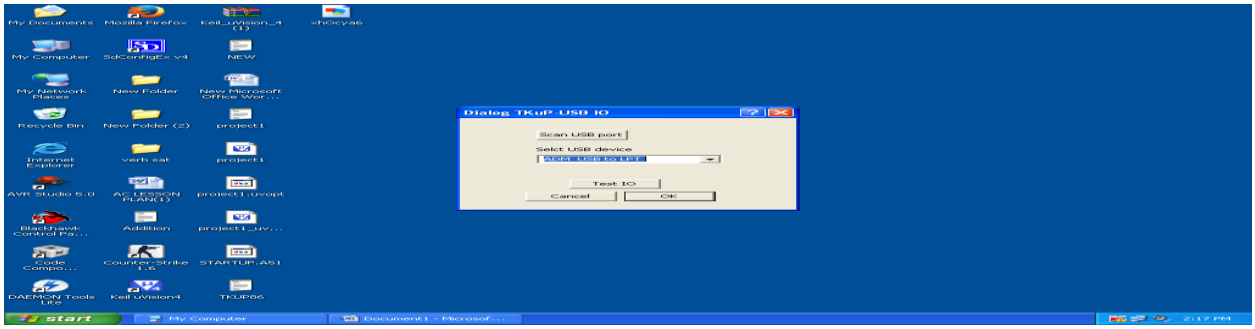
- Hex keypad.
- 8-Leds indicator.
- Four multiplexed 7-Segment displays.
- LCD 16 characters x 2 lines.
- I2C NVRAM 24C1024.
- I2C RTC PCF8583.
- I2C ADC/DAC PCF8591.
- Serial port interface through MAX232.

<u>Procedure for TKµP</u>

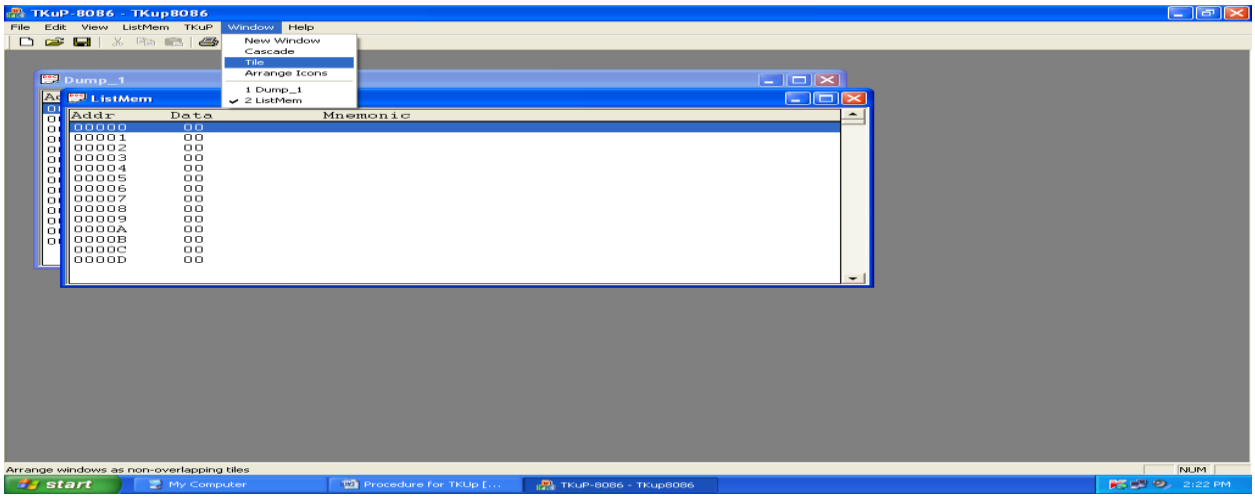
1. Go to start and select TKµP



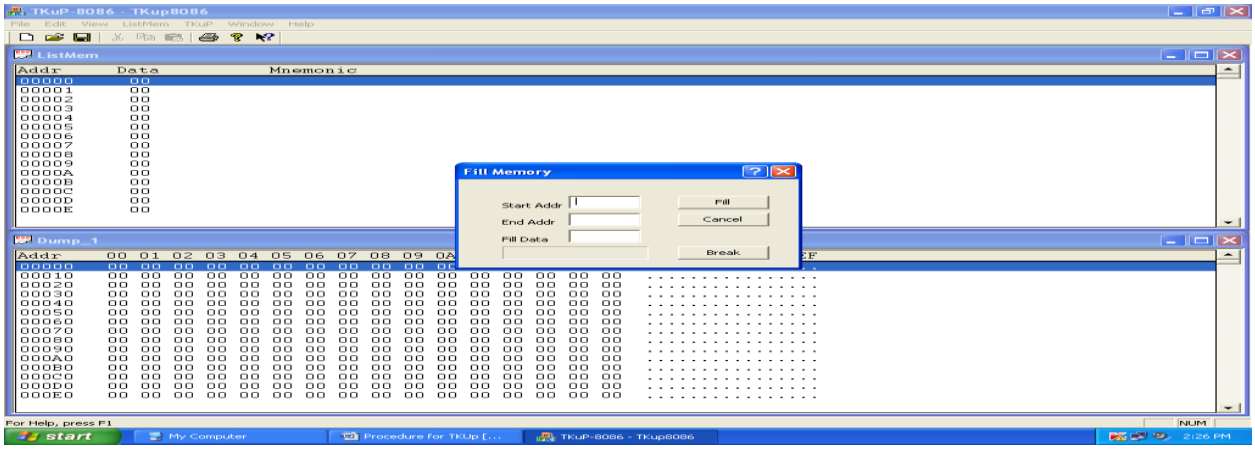
2. To Test the I/O connection click on Test I/O and click on OK



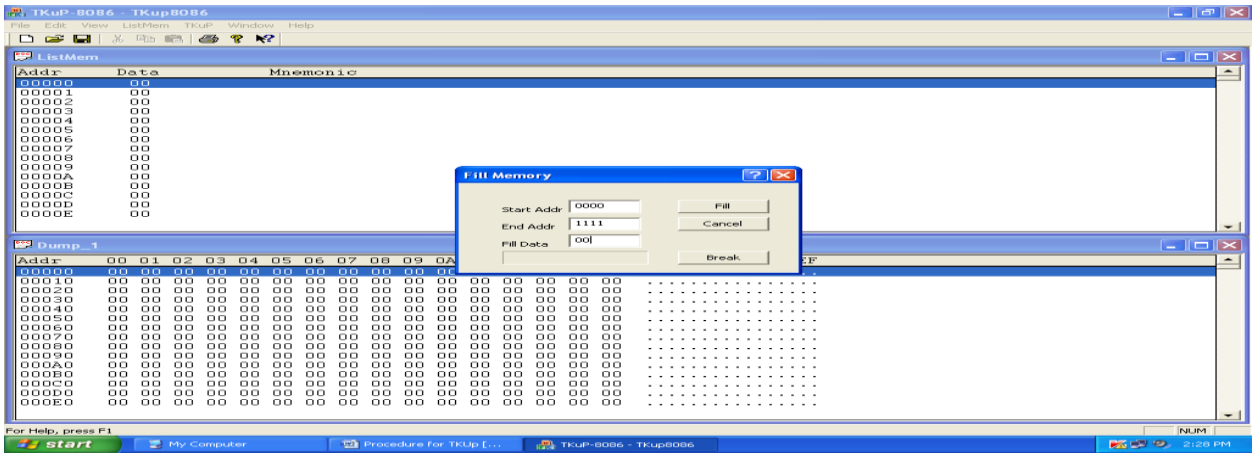
3. The following window will be displayed and go to window , select tile to avoid the overlap of windows



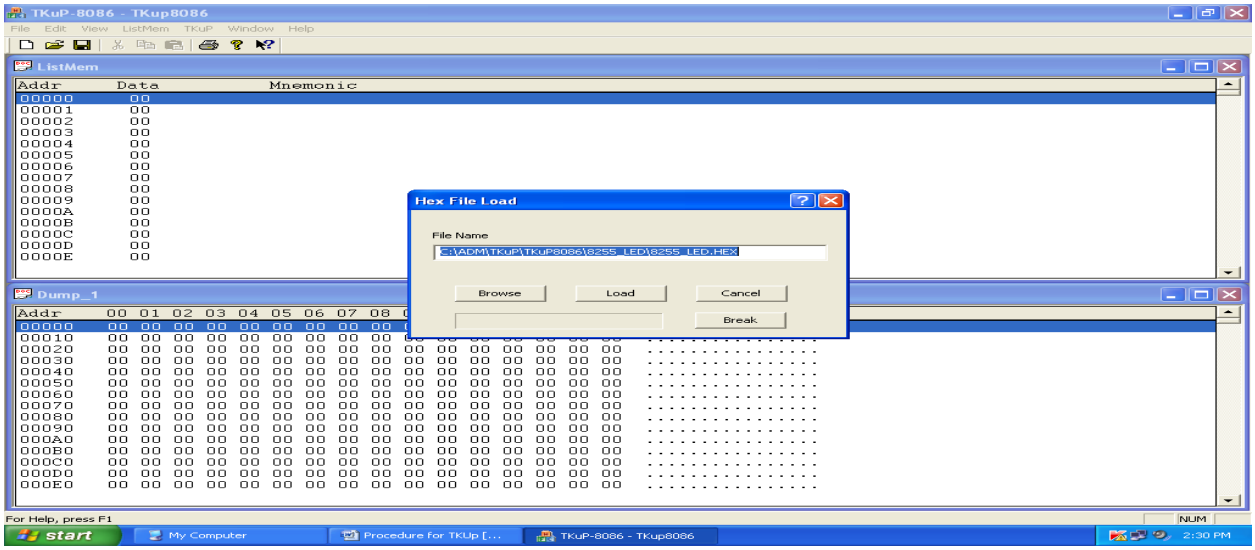
4 To clear the garbage data from dump window, go to Listmem and select fill mem



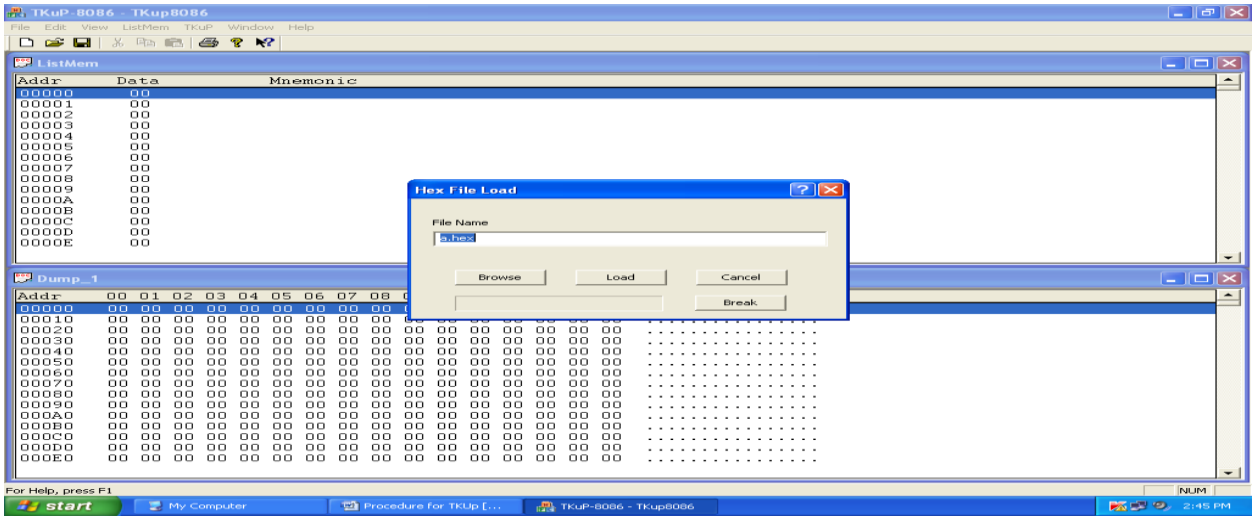
5. To clear the data from dump window ,enter start and ,end address and fill the data with 00



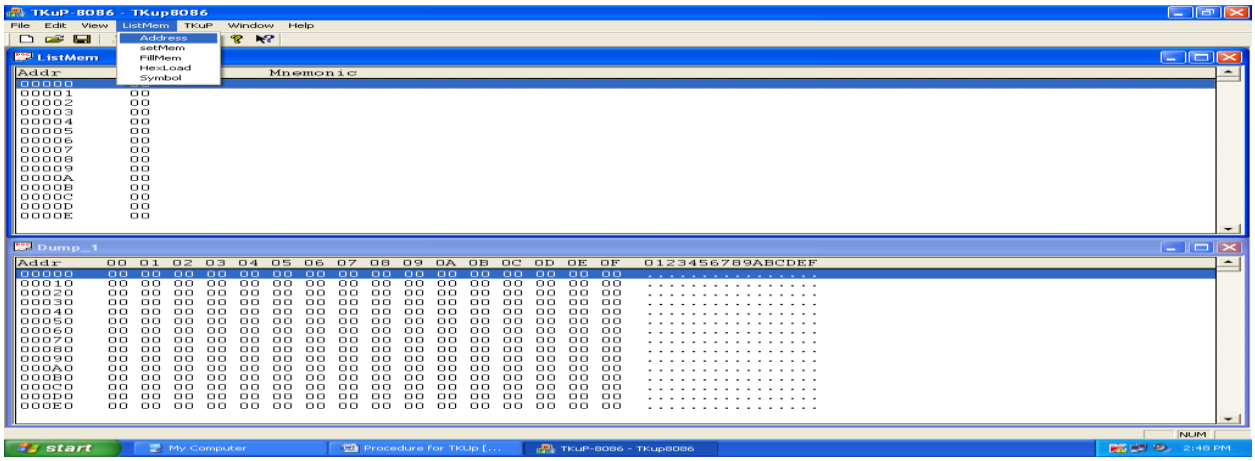
6. To load the hex file ,go to "Listmem" and select "HexLoad"



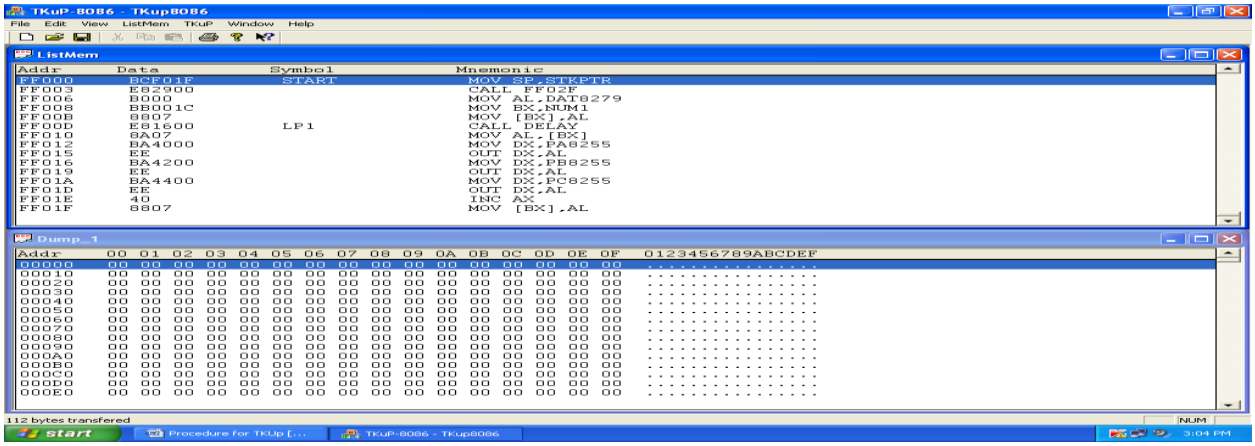
7. Browse the hex file from the source and click on "Load"



8. To view the program on the window, go to "Listmem" and select address



9. Enter the starting address of the program click "OK"



10. To verify the output, change the "SW-PP PROGRAM" switch to execution mode and verify output

EXPERIMENT NO: 6

DIGITAL CLOCK DESIGN USING 8086

AIM: Write an ALP for digital clock design using 8086

TOOLS:

i. **UXASM**
ii. **TKUP**
iii. **TKUP86 KIT**
iv. **FRC CABLE**

PROGRAM:

; **CONNECT BH4 (PORT A) TO CNLED**

```
#INCLUDE "TKUP86.DEF"

;*******     INCLUDE EXTERNS NOW
;*******     START CODING HERE
;
      ORG 0FFFF0H
        JMPF 0F000H,0F000H      ; the basic reset jump

            ORG 0FF000H
START:      MOV SP,STKPTR       ; load stack pointer
            CALL INIT8255       ; initialize 8255
            MOV AL,0CH
            MOV [0200H], AL
            MOV AL, 00H
NXTHR:      MOV CX, 003CH
NXTMNT:     MOV BX,003CH
NXTSEC:     CALL SECDLY
            MOV DX,PA8255
            MOV AL, 07H
            OUT DX,AL
```

```
                DEC BX
                JNE NXTSEC
                MOV DX,PA8255
                MOV AL,38H
                OUT DX,AL
                DEC CX
                JNE NXTMNT
                MOV DX, PA8255
                MOV AL, 0C0H
                OUT DX, AL
                MOV AX,[0200H]
                DEC AX
                MOV [0200H], AX
                JNE NXTHR

SECDLY:         PUSH AX
                PUSH BX
                PUSH CX
                PUSH DX
                MOV CX, 1234H
DLY:            NOP
                NOP
                LOOP DLY
                POP DX
                POP CX
                POP BX
                POP AX
                RET

;*******        initialize 8255
INIT8255
                MOV AL,080H
                MOV DX, CMD8255
```

```
OUT DX,AL
MOV AL,00H
MOV DX,PA8255
OUT DX,AL
MOV DX,PB8255
OUT DX,AL
MOV DX,PC8255
OUT DX,AL
RET
```

RESULT: INPUT:

OUTPUT:

EXPERIMENT NO: 7

PROGRAM FOR INTERFACING ADC&DAC TO 8086

AIM: Write an ALP for interfacing ADC to 8086

TOOLS:

i. **UXASM**
ii. **TKUP**
iii. **TKUP86 KIT**
iv. **FRC CABLE**
v. **ADC KIT**

PROGRAM:

```
; CONNECT BH4 (PORT A) TO DAC BH1A
; CONNECT BH5 (PORTB) TO DAC BH2B
; CONNECT CRO PROBES TO CND1_1 OF DAC

#INCLUDE "TKUP86.DEF"
DATA SEGMENT
        PORTA EQU 9000H
        PORTC EQU 9004H
        CNTLPRT EQU 9006H
        MEM DW 2000H
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START:  MOV AX, DATA
        MOV DS, AX
        MOV DX, CNTLPRT
        MOV AL, 98H
        OUT DX, AL
```

```
        MOV AL, 01H
        OUT DX, AL
        MOV AL, 00
        OUT DX, AL
        MOV DX, PORTC
CHK:    IN AL, DX
        AND AL, 80H
        JZ CHK
        MOV DX, PORTA
        IN AL, DX
        MOV MEM, AL
        INT 03H
CODE ENDS
END START
```

RESULT: INPUT :

OUTPUT :

INTERFACING DAC TO 8086

AIM: Write an ALP for interfacing DAC to 8086

TOOLS:

i. UXASM
ii. TKUP
iii. TKUP86 KIT
iv. FRC CABLE
v. DAC KIT

PROGRAM:

; CONNECT BH4 (PORT A) TO DAC BH1A

; CONNECT BH5 (PORTB) TO DAC BH2B

; CONNECT CRO PROBES TO CND1_1 OF DAC

```
        #INCLUDE "TKUP86.DEF"
        ORG 0FFFF0H
        JMPF 0F000H,0F000H
        ORG 0FF000H
        MOV AL,080H
        MOV DX,CMD8255
        OUT DX,AL
        MOV AL,00H
        MOV DX,PA8255
        OUT DX,AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
RPT:    MOV AL,00H
        MOV AL,0FFH
AGAIN:  MOV DX, PA8255
        OUT DX, AL
        CALL DELAY
        CALL DELAY
        CALL DELAY
```

```
        CALL DELAY
        CALL DELAY
        CALL DELAY
        INC AX
        JNE AGAIN
        JMP RPT

DELAY: MOV CX, 0FF00H
NXT2: MOV BX, 1234H
NXT:  NOP
        NOP
        NOP
        NOP
        NOP
        DEC BX
        JNE NXT
        RET
```

RESULT: INPUT :

OUTPUT :

EXPERIMENT NO: 8
PARALLEL COMMUNICATION BETWEEN TWO MICROPROCESSORS USING 8255

AIM: Write an ALP for parallel communication between two microprocessors using 8255

TOOLS:

i. **UXASM**
ii. **TKUP**
iii. **TKUP86 KIT**
iv. **FRC CABLE**

PROGRAM: FOR DATA IN KIT

```
        #INCLUDE "TKUP86.DEF"
        ORG 0FFFF0H
        JMPF 0F000H,0F000H
        ORG 0FF000H
        MOV AL,080H
        MOV DX,CMD8255
        OUT DX,AL
        MOV AL,00H
        MOV DX,PA8255
        OUT DX,AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
RPT:    MOV AL,47H
        MOV DX,PA8255
        OUT DX,AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
        JMP RPT
```

PROGRAM: FOR DATA OUT KIT

```
        #INCLUDE "TKUP86.DEF
         ORG 0FFFF0H
        JMPF 0F000H,0F000H

        ORG 0FF000H
        MOV AL,090H
        MOV DX,CMD8255
        OUT DX,AL
        MOV AL,00H
        MOV DX,PA8255
        OUT DX,AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
RPT:    MOV DX,PA8255
        IN AL,DX
        MOV [0200H],AL
        MOV DX,PB8255
        OUT DX,AL
        MOV DX,PC8255
        OUT DX,AL
`       JMP RPT
```

RESULT:

EXPERIMENT NO: 9
PROGRAM FOR INTERFACING STEPPER TO 8086

(A) ROTATE THE STEPPER MOTOR IN ANTICLOCKWISE DIRECTION

```
;       Connect 8255 Ports A   to CNLED

;****   INCLUDE DEFINATION FILES NOW
#INCLUDE "TKUP86.DEF"

;****   START CODING HERE

                ORG 0FFFF0H
                JMPF 0F000H,0F000H      ; the basic reset jump

                ORG 0FF000H
START   MOV SP,STKPTR           ; load stack pointer
                CALL INIT8255           ; initialize 8255
        LP1     MOV AL,01H      ; use num1 for led count value
                MOV DX,PA8255           ;
                OUT DX,AL       ;
                CALL DELAY              ; call delay
                MOV AL,02H      ; use num1 for led count value
                MOV DX,PA8255           ;
                OUT DX,AL
                CALL DELAY              ; call delay
                MOV AL,04H      ; use num1 for led count value
                MOV DX,PA8255           ;
                OUT DX,AL
                CALL DELAY              ; call delay
                MOV AL,08H      ; use num1 for led count value
                MOV DX,PA8255           ;
                OUT DX,AL
                CALL DELAY              ; call delay
                JMP START       ; restart again

        ;*******        Delay module

        DELAY   NOP                     ;
                MOV CX,03500H           ; load Delay count = 0x3500
                NOP             ;
        DLY1    NOP             ;
                LOOP DLY1       ;
                RET             ; end of delay

        ;*******        initialize 8255
        INIT8255
```

```
        MOV AL,080H             ; make all ports output
        MOV DX, CMD8255         ;
        OUT DX,AL       ; write to command register
        MOV AL,00H      ; clear all ports
        MOV DX,PA8255           ;
        OUT DX,AL       ;
        MOV DX,PB8255           ;
        OUT DX,AL       ;
        MOV DX,PC8255           ;
        OUT DX,AL       ;
        RET             ;
```

(B)**ROTATE THE STEPPER MOTOR IN CLOCKWISE DIRECTION**

;***** START CODING HERE

```
        ORG 0FFFF0H
        JMPF 0F000H,0F000H      ; the basic reset jump

        ORG 0FF000H
START   MOV SP,STKPTR           ; load stack pointer
        CALL INIT8255           ; initialize 8255
LP1     MOV AL,08H      ; use num1 for led count value
        MOV DX,PA8255           ;
        OUT DX,AL       ;
        CALL DELAY              ; call delay
        MOV AL,04H      ; use num1 for led count value
        MOV DX,PA8255           ;
        OUT DX,AL
        CALL DELAY              ; call delay
        MOV AL,02H      ; use num1 for led count value
        MOV DX,PA8255           ;
        OUT DX,AL
        CALL DELAY              ; call delay
        MOV AL,01H      ; use num1 for led count value
        MOV DX,PA8255           ;
        OUT DX,AL
        CALL DELAY              ; call delay
        JMP START       ; restart again

;*******      Delay module

DELAY   NOP                     ;
        MOV CX,03500H           ; load Delay count = 0x3500
        NOP             ;
DLY1    NOP             ;
        LOOP DLY1       ;
```

```
        RET             ; end of delay

;*******    initialize 8255
INIT8255
        MOV AL,080H             ; make all ports output
        MOV DX, CMD8255         ;
        OUT DX,AL       ; write to command register
        MOV AL,00H      ; clear all ports
        MOV DX,PA8255           ;
        OUT DX,AL       ;
        MOV DX,PB8255           ;
        OUT DX,AL       ;
        MOV DX,PC8255           ;
        OUT DX,AL       ;
        RET             ;
```

RESULT: INPUT:

OUTPUT:

EXPERIMENT NO: 10

ARITHMETIC, LOGICAL AND BIT MANIPULATION INSTRUCTIONS OF 8051

AIM: Write an ALP for Arithmetic, logical and bit manipulation operations in 8051

TOOLS:

i. **UXASM**
ii. **TKUP**
iii. **TKUP86 KIT**
iv. **FRC CABLE**

A) PROGRAM: FOR ARITHMETIC INSTRUCTIONS OF 8051

;Connect P1 to CNLED1

```
#INCLUDE "TKUP52.DEF"
        ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN MOV SP,#50H
        MOV R0,#20H
        MOV R1,#07H
        MOV A,R0
        ADD A,R1
        MOV P1,A
        LCALL DELAY
        MOV A,R0
        SUBB A,R1
        MOV P1,A
        LCALL DELAY
        MOV A,R0
        MOV 0F0H,R1
        MUL AB
        MOV P1,A
        LCALL DELAY
        MOV P1,0F0H
```

```
        LCALL DELAY
        MOV A,R0
        MOV 0F0H,R1
        DIV AB
        MOV P1,A
        LCALL DELAY
        MOV P1,0F0H
        LCALL DELAY
        LJMP MAIN
DELAY   NOP

        MOV R4,#020H
DLY3  MOV R3,#0FFH
DLY2  MOV R2,#0FFH
        NOP
DLY1  NOP
        NOP
        NOP
        NOP
        DJNZ R2,DLY1
        DJNZ R3,DLY2
        DJNZ R4,DLY3
        RET             ;
```

B) PROGRAM: FOR LOGICAL INSTRUCTIONS OF 8051

i) **;Connect P1 to CNLED1**

```
         #INCLUDE "TKUP52.DEF"
        ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN MOV SP,#50H
        MOV A,#35H
```

```
        ANL A,#0FH
        MOV P1,A
        ACALL DLY
        MOV A,#04H
        ORL A,#30H
        MOV P1,A
        ACALL DLY
        MOV A,#54H
        XRL A,#78H
        MOV P1,A
        ACALL DLY
        MOV A,#55H
        CPL A
        MOV P1,A
        ACALL DLY
DLY     NOP
        NOP
        MOV R4,#020H
DLY3    MOV R3,#0FFH
DLY2    MOV R2,#0FFH
        NOP
DLY1    NOP
        NOP
        NOP
        NOP
        DJNZ R2,DLY1
        DJNZ R3,DLY2
        DJNZ R4,DLY3
        RET
```

ii) ;Connect P1 to CNLED1

```
#INCLUDE "TKUP52.DEF"
                ORG 0000H
START:          LJMP MAIN
```

```
        ORG 0150H
MAIN:   MOV SP,#060H
        MOV A,#0A5H
        MOV P1,A
        LCALL SFTDL
        RR A
        MOV P1,A
        LCALL SFTDL
        SWAP A
        MOV P1,A
        LCALL SFTDL
        RL A
        MOV P1,A
        LCALL SFTDL
        SETB C
        RLC A
        MOV P1,A
        LCALL SFTDL
        RRC A
        MOV P1,A
        LCALL SFTDL
        LJMP MAIN

SFTDL   MOV R4,#50H
DL3     MOV R5,#0FFH
DL2     MOV R6,#0FFH
DL1     DJNZ R6,DL1
        DJNZ R5,DL2
        DJNZ R4,DL3
        RET
```

C) PROGRAM: FOR BIT MANIPULATION INSTRUCTIONS OF 8051
;Connect P1 to CNLED1

```
#INCLUDE "TKUP52.DEF"
        ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN MOV SP,#50H
        MOV P1,#00H
        MOV C,00H
         SETB C
        MOV P1_7,C
        LCALL SFTDL
        CLR C
        ANL C,00H
        MOV P1_7,C
        LCALL SFTDL
        CPL C
        MOV P1_3,C
        LCALL SFTDL
        ORL C,00H
        MOV P1_7,C
        LCALL SFTDL
        LJMP MAIN
SFTDL MOV R4,#50H
DL3     MOV R5,#0FFH
DL2     MOV R6,#0FFH
DL1     DJNZ R6,DL1
        DJNZ R5,DL2
        DJNZ R4,DL3
        RET
```

RESULT: INPUT :

OUTPUT:

EXPERIMENT NO: 11

TIMER/COUNTERS IN 8051

AIM: Write an ALP to verify timer/counter operation in 8051

TOOLS: i) UXASM

ii)TKUP

iii)TKUP86 KIT

iv)FRC CABLE

PROGRAM:

;Connect P1 to CNLED1

```
#INCLUDE "TKUP52.DEF"
        ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN:   MOV SP,#060H
        MOV TMOD,#01H
BACK:   MOV TL0,#075H
        MOV TH0,#0B8H
        MOV P1,#0AAH
        LCALL SFTDL
        ACALL DELAY
        MOV TL0,#00H
        MOV TH0,#00H
        MOV P1,#055H
        ACALL DELAY
        LCALL SFTDL
        SJMP BACK
        ORG 300H
DELAY:  SETB TCON4
AGAIN:  JNB TCON5,AGAIN
        CLR TCON4
        CLR TCON5
```

```
        RET

SFTDL   MOV R4,#10H
DL3     MOV R5,#0FFH
DL2     MOV R6,#0FFH
DL1     DJNZ R6,DL1
        DJNZ R5,DL2
        DJNZ R4,DL3
        RET
```

RESULT: INPUT :

OUTPUT:

EXPERIMENT NO: 12
INTERRUPT HANDLING IN 8051

AIM: Write an ALP to verify the interrupt handling in 8051

TOOLS: i) UXASM

ii)TKUP

iii)TKUP86 KIT

iv)FRC CABLE

PROGRAM:

```
#INCLUDE "TKUP52.DEF"
        ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN    MOV SP,#50H
        MOV IE,#85H
HERE    MOV P1,#7EH
        SJMP HERE
        ORG 0003H           ;INT0 ISR
        MOV  P1,#0AAH
        LCALL DELAY
        LCALL DELAY
        LCALL DELAY
        RETI
        ORG 0013H           ;INT1 ISR
        MOV  P1,#0A5H
        LCALL DELAY
        LCALL DELAY
        RETI
DELAY   NOP
        MOV R4,#020H
DLY3    MOV R3,#0FFH
DLY2    MOV R2,#0FFH
DLY1    NOP
```

```
        NOP
        DJNZ R2,DLY1
        DJNZ R3,DLY2
        DJNZ R4,DLY3
        RET
```

RESULT: INPUT :

OUTPUT :

EXPERIMENT NO: 13
UART OPERATION IN 8051

AIM: To observe the UART operation in 8051

TOOLS: i) UXASM
ii) TKUP
iii) TKUP86 KIT
iv) FRC CABLE

PROGRAM:

; CONNECT THE RS232 FROM PC TO TKUP51 KIT
;CONNECT THE Tx PIN OF 8051 TO Rx OF MAX232 AND VICE VERSA
;CONNECT PORT1 TO CNLED

```
#INCLUDE "TKUP52.DEF"
        ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN:   MOV SP,#060H
        MOV IE,#85H
        MOV TMOD,#20H
        MOV TH1,#0FAH
        MOV SCON,#50H
        SETB TCON6
RPT:    MOV SBUF,#'Y'
HERE:   JNB SCON1,HERE
        CLR SCON1
        MOV A,#'A'
        MOV P1,A
        SJMP RPT
```

RESULT: INPUT :

OUTPUT :

EXPERIMENT NO: 14
INTERFACING LCD TO 8051

AIM: Write an ALP for interfacing LCD to 8051

TOOLS: i)UXASM
ii)TKUP
iii)TKUP86 KIT
iv)FRC CABLE

PROGRAM:

```
;CONNECT BH4 TO CNLCDC
;CONNECT BH6 TO CNLCDD

#INCLUDE "TKUP52.DEF"

      ORG 0000H
START: LJMP MAIN
       ORG 0150H
       MAIN MOV SP,#060H
       LCALL INIT8255
LOOP  MOV DPTR,#CMDTBL
       LCALL INIT_LCD
       MOV DPTR,#STRTBL
LP1    MOV A,#0
       MOVC A,@A+DPTR
       CJNE A,#00,LP2
       LCALL DELAY
       LCALL DELAY
       LCALL DELAY
       LJMP  MAIN
LP2    LCALL WR_DAT
       INC DPTR
       LCALL SDELAY
```

```
        LJMP LP1

;*******     LCD init module
INIT_LCD
        MOV A,#0
        MOVC A,@A+DPTR
        CJNE A,#00,IL2
        RET
IL2     LCALL WR_CMD
        INC DPTR
        LJMP INIT_LCD

;*******     LCD Write CMD module
WR_CMD  PUSH DPH
        PUSH DPL
        MOV DPTR,#PB8255
        LCALL WRPORT
        LCALL SDELAY
        MOV A,#04
        MOV DPTR,#PA8255
        LCALL WRPORT
        LCALL SDELAY
        MOV A, #00
        MOV DPTR,#PA8255
        LCALL WRPORT
        LCALL SDELAY
        POP DPL
        POP DPH
        RET

;*******     LCD Write Data module
WR_DAT  PUSH DPH
        PUSH DPL
```

```
        MOV DPTR,#PB8255
        LCALL WRPORT
        LCALL SDELAY
        MOV A,#05H
        MOV DPTR,#PA8255
        LCALL WRPORT
        LCALL SDELAY
        MOV A,#01H
        MOV DPTR,#PA8255
        LCALL WRPORT
        LCALL SDELAY
        POP DPL
        POP DPH
        RET

;******Write Port
WRPORT      CLR P1_7
        MOVX @DPTR,A
        SETB P1_7
        RET

;******              Read Port
RDPORT      CLR P1_7
        MOVX A,@DPTR
        SETB P1_7
        RET

;*******      Delay module
SDELAY      NOP
        MOV R0,#0FFH
        MOV R1,#01H
        LJMP DLY1
        NOP
```

```
DELAY       NOP
            MOV R0,#0FFH
            MOV R1,#055H
            NOP
DLY1  DJNZ R0,DLY1
            MOV R0,#0FFH
            DJNZ R1,DLY1
            RET

;*******     initialize 8255
INIT8255
            MOV A,#080H
            MOV DPTR,#CMD8255
            LCALL WRPORT
            RET

            ORG 0500H
;*******     initialize seven segment table
CMDTBL HEX      38,0E,02,01,00
STRTBL ASCII    "HELLO ADM - TKUP"
ENDTBL HEX      00,00
```

RESULT: INPUT :

OUTPUT :

EXPERIMENT NO: 15
INTERFACING MATRIX/KEYBOARD TO 8051

AIM: Write an ALP for interfacing Matrix/keyboard to 8051

TOOLS: i) UXASM

ii) TKUP

iii) TKUP86 KIT

iv)FRC CABLE

PROGRAM:

```
;*******      8255_KBD

;*******      INCLUDE DEFINATION FILES NOW
;       1. Connect 8255 PA0-7 to CNMUX of L1C peripheral board
;       2. Connect 8255 PC0-7 to CNKEY of L1C peripheral board
;       3. Connect 8255 PB0-7 to CNSEG of L1C peripheral board
;       4. Moitor one segment showing 0000->0001->....->000F->0000 (keypress)

#INCLUDE "TKUP52.DEF"
          ORG 0000H
START:  LJMP MAIN
        ORG 0150H
MAIN MOV SP,#060H
        LCALL INIT8255
        MOV DPTR,#NUM1
        LCALL CLRMEM
        MOV DPTR,#NUM2
        LCALL CLRMEM
        MOV DPTR,#NUM3
        LCALL CLRMEM
        NOP
LOOP LCALL SCANKBD
        MOV DPTR,#NUM3
        MOVX A,@DPTR
```

```
        MOV DPTR,#SEGTBL
        MOVC A,@A+DPTR
        MOV DPTR,#PB8255
        LCALL WRPORT
        MOV A,#070H
        MOV DPTR,#PA8255
        LCALL WRPORT
        LJMP LOOP

;*******    MATRIX KBD SCAN module
;*******    Output either E0,D0,B0,70 for Row 1,2,3,4
;*******    Read PC port anded with 0x0F, expect 0F,0E,0D,0B,07

SCANKBD MOV A,#00H
SKLOOP  MOV DPTR,#NUM1
        MOVX @DPTR,A
        MOV DPTR,#KBDTBL
        MOVC A,@A+DPTR
        CJNE A,#00,SKL1
        RET
SKL1    MOV DPTR,#PC8255
        LCALL WRPORT
        MOV DPTR,#PC8255
        LCALL RDPORT
        ANL A,#0FH
        CJNE A,#0FH,SKL2
        MOV DPTR,#NUM1
        MOVX A,@DPTR
        INC A
        LJMP SKLOOP
SKL2    LJMP GETKEY

;*******    GETKEY module
```

```
GETKEY      MOV DPTR,#NUM2
            MOVX @DPTR,A
            MOV DPTR,#RETTBL
            MOVC A,@A+DPTR
            MOV R0,A
            MOV DPTR,#NUM1
            MOVX A,@DPTR
            MOV DPTR,#ROWTBL
            MOVC A,@A+DPTR
            ADD A,R0
            MOV DPTR,#NUM3
            MOVX @DPTR,A
            RET

;******          Clear memory location
CLRMEM      MOV A,#0
            MOVX @DPTR,A
            RET
WRPORT CLR P1_7
            MOVX @DPTR,A
            SETB P1_7
            RET

;******          Read Port
RDPORT      CLR P1_7
            MOVX A,@DPTR
            SETB P1_7
            RET

;*******       Delay module
SDELAY      NOP
            MOV R0,#0FFH
```

```
        MOV R1,#01H
        LJMP DLY1
        NOP
DELAY   NOP
        MOV R0,#0FFH            ; load lsb of delay=0x34FF
        MOV R1,#055H            ; load msb
        NOP
DLY1    DJNZ R0,DLY1
        MOV R0,#0FFH            ; decrement msb count
        DJNZ R1,DLY1
        RET                     ; end of delay

;*******    initialize 8255
INIT8255
        MOV A,#081H             ; make all ports output
        MOV DPTR,#CMD8255       ; write to command register
        LCALL WRPORT
        RET


        ORG 0500H
;*******    initialize seven segment table
SEGTBL  HEX  3F,06,5B,4F,66,6D,7D,07,7F,6F,77,7C,39,5E,79,71,00
KBDTBL  HEX  E0,D0,B0,70,00
RETTBL  HEX  00,00,00,00,00,00,00,03,00,00,00,02,00,01,00,00,00
ROWTBL  HEX  00,04,08,0C,00
```

RESULT: INPUT :

OUTPUT :

DIGITAL SIGNAL PROCESSING

LABORATORY MANUAL

DEPARTMENT OF ELECTRONICS AND COMMUNICATIONS ENGG.

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS INSTITUTION: UGC, GOVT. OF INDIA)
Affiliated to JNTUH & Approved by AICTE, New Delhi
Accredited by NBA & NAAC with 'A' Grade, ISO 9001:2015 certified
Maisammaguda, Near Kompally, Secunderabad-500100

ELECTRONICS & COMMUNICATION ENGINEERING

VISION

To evolve into a center of excellence in Engineering Technology through creative and innovative practices in teaching-learning, promoting academic achievement & research excellence to produce internationally accepted competitive and world class professionals.

MISSION

To provide high quality academic programmes, training activities, research facilities and opportunities supported by continuous industry institute interaction aimed at employability, entrepreneurship, leadership and research aptitude among students.

QUALITY POLICY

- **Impart up-to-date knowledge to the students in Electronics & Communication area to make them quality engineers.**
- **Make the students experience the applications on quality equipment and tools.**
- **Provide systems, resources and training opportunities to achieve continuous improvement.**
- **Maintain global standards in education, training and services.**

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1: PROFESSIONALISM & CITIZENSHIP

To create and sustain a community of learning in which students acquire knowledge and learn to apply it professionally with due consideration for ethical, ecological and economic issues.

PEO2: TECHNICAL ACCOMPLISHMENTS

To provide knowledge based services to satisfy the needs of society and the industry by providing hands on experience in various technologies in core field.

PEO3: INVENTION, INNOVATION AND CREATIVITY

To make the students to design, experiment, analyze, interpret in the core field with the help of other multi disciplinary concepts wherever applicable.

PEO4: PROFESSIONAL DEVELOPMENT

To educate the students to disseminate research findings with good soft skills and become a successful entrepreneur.

PEO5: HUMAN RESOURCE DEVELOPMENT

To graduate the students in building national capabilities in technology, education and research.

PROGRAMME SPECIFIC OBJECTIVES (PSOs)

PSO1

To develop a student community who acquire knowledge by ethical learning and fulfill the societal and industry needs in various technologies of core field.

PSO2

To nurture the students in designing, analyzing and interpreting required in research and development with exposure in multi disciplinary technologies in order to mould them as successful industry ready engineers/entrepreneurs

PSO3

To empower students with all round capabilities who will be useful in making nation strong in technology, education and research domains.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12. **Life- long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

CONTENTS

INTRODUCTION TO MATLAB

MATLAB (MATrix LABoratory):

MATLAB is a software package for high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include the following

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow learning and applying specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include Image processing, signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The main features of MATLAB

1. Advance algorithm for high performance numerical computation ,especially in the Field matrix algebra
2. A large collection of predefined mathematical functions and the ability to define one's own functions.
3. Two-and three dimensional graphics for plotting and displaying data
4. A complete online help system
5. Powerful, matrix or vector oriented high level programming language for individual applications.
6. Toolboxes available for solving advanced problems in several application areas

EXP.NO: 1
TO FIND DFT / IDFT OF GIVEN DT SIGNAL

AIM: To find Discrete Fourier Transform and Inverse Discrete Fourier Transform of given digital signal.

Software: MATLAB

THEORY:

Basic equation to find the DFT of a sequence is given below.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$where \;\; W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \text{ [TWIDDLE FACTOR]}$$

Basic equation to find the IDFT of a sequence is given below.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1.$$

PROGRAM:

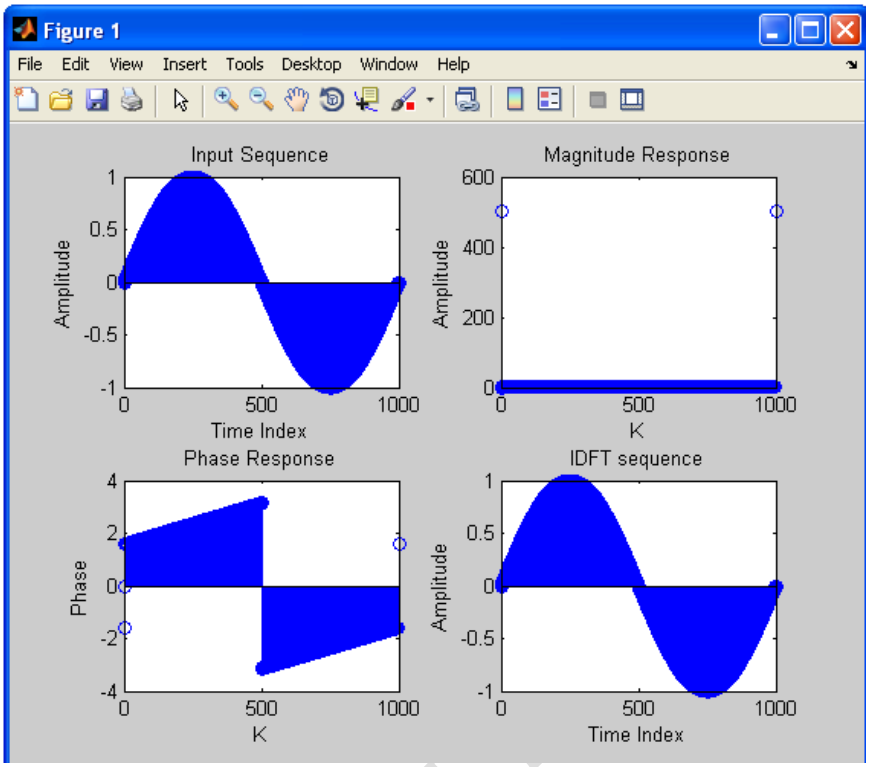
```
%% To find DFT/IDFT of a given signal/sequence
 clc;
close all;
clear all;
ts = 0:1e-3:1;
in = input('Enter 1 for signal and 2 for sequence');
if(in==1)
   xn = sin(2*pi*ts);
elseif (in==2)
   xn = randint(1,10,[1 10]);
else
   disp('Wrong value entered');
   return;
end
%xn=input('Enter the sequence x(n)'); %Get the sequence from user
```

```
ln=length(xn); %find the length of the sequence
xk=zeros(1,ln); %initilise an array of same size as that of input sequence
ixk=zeros(1,ln); %initilise an array of same size as that of input sequence
 %DFT of the sequence
 %------------------------------------------------------------
for k=0:ln-1
   for n=0:ln-1
      xk(k+1)=xk(k+1)+(xn(n+1)*exp((-i)*2*pi*k*n/ln));
   end
end
%------------------------------------------------------------
 %Plotting input sequence
 %------------------------------------------------------------
t=0:ln-1;
subplot(221);
stem(t,xn);
ylabel ('Amplitude');
xlabel ('Time Index');
title('Input Sequence');
%------------------------------------------------------------
magnitude=abs(xk); % Find the magnitudes of individual DFT points
% plot the magnitude response
 %------------------------------------------------------------
t=0:ln-1;
subplot(222);
stem(t,magnitude);
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude Response');
%------------------------------------------------------------
 phase=angle(xk); % Find the phases of individual DFT points % plot the magnitude
sequence
%------------------------------------------------------------
t=0:ln-1;
```

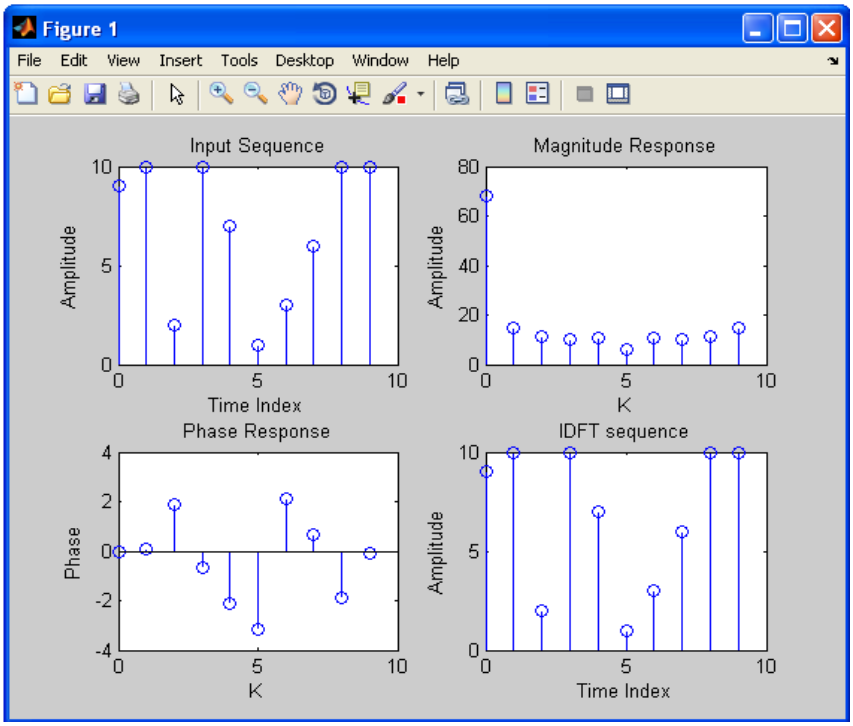
```
subplot(223);
stem(t,phase);
ylabel ('Phase');
xlabel ('K');
title ('Phase Response');
%-------------------------------------------------------
 %IDFT of the sequence
 %-------------------------------------------------------
for n=0:n-1
   for k=0:ln-1
      ixk(n+1)=ixk(n+1)+(xk(k+1)*exp(i*2*pi*k*n/ln));
   end
end
 ixk=ixk./ln;
%-------------------------------------------------------
 %code block to plot the input sequence
 %-------------------------------------------------------
t=0:ln-1;
subplot(224);
stem(t,xn);
ylabel ('Amplitude');
xlabel ('Time Index');
title ('IDFT sequence');
%-------------------------------------------------------
```

SIMULATION RESULT

- **DFT and IDFT of Sinosoidal Signal**



- **DFT and IDFT of a random sequence**



EXP.NO: 2

TO FIND FREQUENCY RESPONSE OF A GIVEN SYSTEM GIVEN IN (TRANSFER FUNCTION/ DIFFERENTIAL EQUATION FORM).

AIM: To obtain the impulse response/step response of a system described by the given difference equation

Software: MATLAB

THEORY:

- A difference equation with constant coefficients describes a LTI system. For example the difference equation **y[n] + 0.8y[n-2] + 0.6y[n-3] = x[n] + 0.7x[n-1] + 0.5x[n-2]** describes a LTI system of order 3. The coefficients 0.8, 0.7, etc are all constant i.e., they are not functions of time (n). The difference equation y[n]+0.3ny[n-1]=x[n] describes a time varying system as the coefficient 0.3n is not constant.
- The difference equation can be solved to obtain y[n], the output for a given input x[n] by rearranging as y[n] = x[n] + 0.7x[n-1]+0.5x[n-2]- 0.8y[n-2]- 0.6y[n-3] and solving.
- The output depends on the input x[n]
    - With **x[n]= δ[n],** an impulse, the computed output y[n] is the **impulse response**.
    - If **x[n]=u[n]**, a **step response** is obtained.
    - If **x[n] = cos(wn)** is a sinusoidal sequence, a **steady state response** is obtained (wherein y[n] is of the same frequency as x[n], with only an amplitude gain and phase shift-refer Fig.7.3).
    - Similarly for any arbitrary sequence of x[n], the corresponding output response y[n] is computed.
- The difference equation containing past samples of output, i.e., y[n-1], y[n-2], etc leads to a recursive system, whose impulse response is of infinite duration (IIR). For such systems the impulse response is computed for a large value of n, say n=100 (to approximate n=∞). The MATLAB function filter is used to compute the impulse response/ step response/ response to any given x[n]. Note: The filter function evaluates the convolution of an infinite sequence (IIR) and x[n], which is not possible with conv function (remember conv requires both the sequences to be finite).

- The difference equation having only y[n] and present and past samples of input (x[n], x[n-k]), represents a system whose impulse response is of finite duration (FIR). The response of FIR systems can be obtained by both the 'conv' and 'filter' functions. The filter function results in a response whose length is equal to that of the input x[n], whereas the output sequence from conv function is of a longer length (xlength + hlength-1).

ALGORITHM:

1. Input the two sequences as a and b representing the coefficients of y and x.
2. If IIR response, then input the length of the response required (say 100, which can be made constant).
3. Compute the output response using the 'filter' command.
4. Plot the input sequence & impulse response (and also step response, etc if required).

MATLAB Implementation:

MATLAB has an inbuilt function '***filter***' to solve difference equations numerically, given the input and difference equation coefficients (b,a).

y=filter(b,a,x)

where x is the input sequence, y is the output sequence which is of same length as x.

Given a difference equation $a_0y[n]+a_1y[n-1]+a_2y[n-2]=b_0x[n]+b_2x[n-2]$, the coefficients are written in a vector as $b=[b_0\ 0\ b_2]$ and $a=[a_0\ a_1\ a_2]$. Note the zero in b (x[n-1] term is missing).

Also remember a_0, the coefficient of y[n] should always be 1.

For impulse response $x[n]=\{\underset{\uparrow}{1},0,0,0,\ldots.\}$ the number of zeros = the length of the IIR response required (say 100 implemented by function zeros(1,99)).

For step response $x[n]=\{\underset{\uparrow}{1},1,1,1,1,\ldots.\}$ the number of ones = the length of the IIR response required-1 (say 100 implemented by function ones(1,100).

Similarly for any given x[n] sequence, the response y[n] can be calculated.

Given Problem

1)Difference equation y(n) - y(n-1) + 0.9y(n-2) = x(n);

Calculate impulse response h(n) and also step response at n=0,…..,100

2)Plot the steady state response y[n] to x[n]=cos(0.05πn)u(n), given y[n]-0.8y[n-1]=x[n]

MATLAB PROGRAM:

```
clc;
clear all;
close all;
%To find Impulse Response
N=input('Length of response required=');
b=[1];                          %x[n] coefficient
a=[1,-1,0.9];                   %y coefficients
x1=[1,zeros(1,N-1)];            %impulse input
n=0:1:N-1;                      %time vector for plotting
h=filter(b,a,x1);               %impulse response
disp('Impulse response');disp(h)
subplot(2,1,1);                 %plot the waveforms
stem(n,x1);
title('impulse input');
xlabel('n');
ylabel('x(n)');
subplot(2,1,2);
stem(n,h);
title('impulse response');
xlabel('n');
ylabel('h(n)');

%To find step response
N=input('Length of response required=');
x2=[ones(1,N)];                 %step input
y=filter(b,a,x2);               %step response
disp('step response');disp(y);
figure;
subplot(2,1,1);                 %plot the waveforms
stem(n,x2);
title('step input');
xlabel('n');
ylabel('u(n)');
subplot(2,1,2);
stem(n,y);
title('step response');
xlabel('n');
ylabel('y(n)');

%To find steady state response:
%y[n]-0.8y[n-1]=x[n]
N=input('Length of response required=');
b=[1];                          % x[n] coefficient
a=[1,-0.8];                     % y coefficients
n=0:1:N-1;                      % time vector
x=cos(0.05*pi*n);               % sinusoidal input
y=filter(b,a,x);                % steady  state response
disp('steady state response');disp(y);
figure;
```

```
subplot(2,1,1);                 % plot the waveforms
stem(n,x);
title('steady input');
xlabel('n');
ylabel('x(n)');
subplot(2,1,2);
stem(n,y);
title('steady state response');
xlabel('n');
ylabel('y(n)');
```

Output:

Length of response required=10

Impulse response

1.0000 1.0000 0.1000 -0.8000 -0.8900 -0.1700 0.6310 0.7840 0.2161 -0.4895

Length of response required=10

step response

1.0000 2.0000 2.1000 1.3000 0.4100 0.2400 0.8710 1.6550 1.8711 1.3816

Length of response required=35

steady state response

Columns 1 through 12

1.0000 1.7877 2.3812 2.7960 3.0458 3.1437 3.1028 2.9362 2.6580 2.2828 1.8263 1.3046

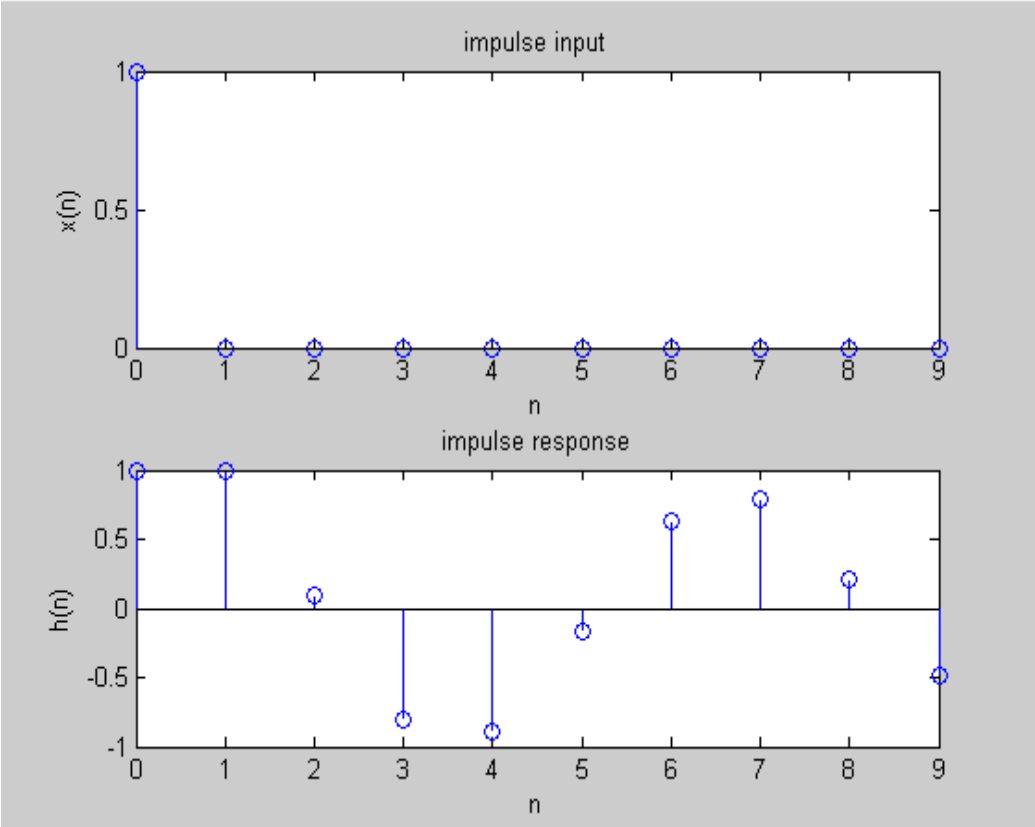
Columns 13 through 24

0.7346 0.1337 -0.4808 -1.0918 -1.6824 -2.2369 -2.7406 -3.1802 -3.5441 -3.8230 -4.0095 -4.0986

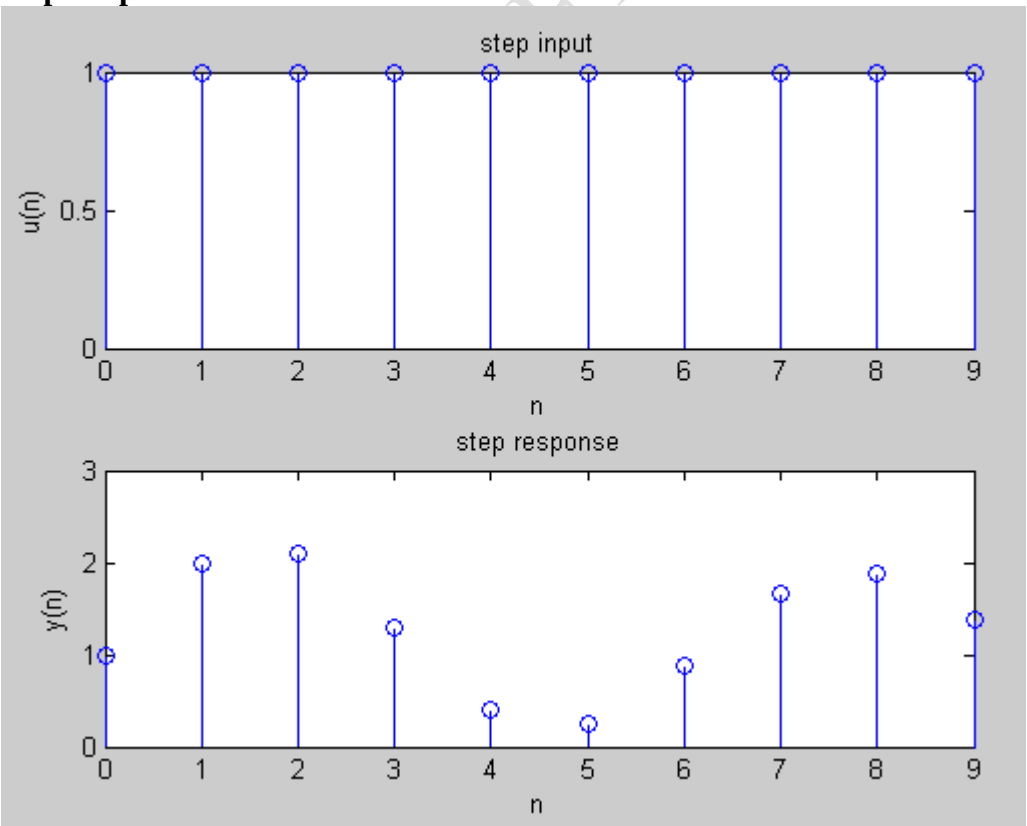
Columns 25 through 35

-4.0879 -3.9774 -3.7697 -3.4698 -3.0848 -2.6243 -2.0994 -1.5231 -0.9095 -0.2736 0.3689

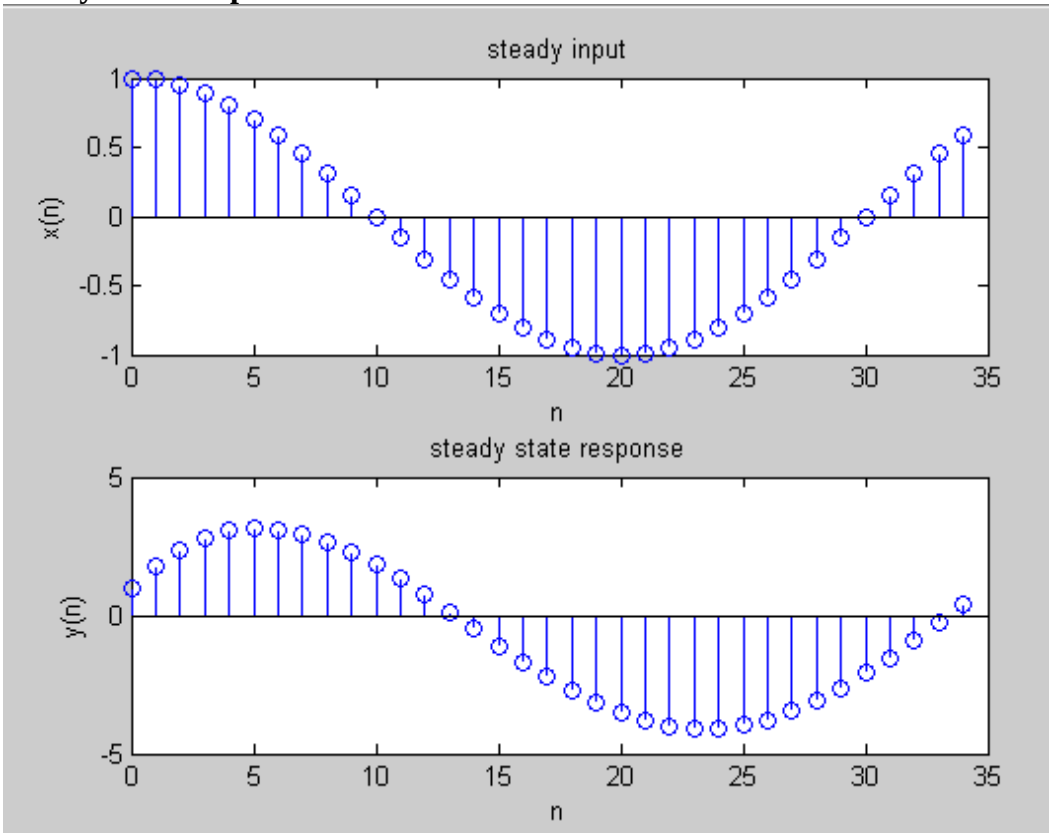
Impulse Response



Step Response



Steady State Response:



EXP.NO: 3
TO FIND THE FFT OF A GIVEN SEQUENCE

AIM: To find the FFT of a given sequence

Software: MATLAB

Theory:

DFT of a sequence

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

Where N= Length of sequence.

K= Frequency Coefficient.

n = Samples in time domain.

FFT : -Fast Fourier transformer .

There are Two methods.

1.Decimation in time (DIT FFT).

2. Decimation in Frequency (DIF FFT).

Why we need FFT ?

The no of multiplications in DFT = $N^{2.}$

The no of Additions in DFT = N(N-1).

For FFT.

The no of multiplication = N/2 $\log_2$N.

The no of additions = N $\log_2$ N.

PROGRAM:

%Finding the Fourier Transform of a given signal and plotting its

%magnitude and phase spectrum.

```
clc;
clear all;
close all;
x=input('Enter the sequence : ')
N=length(x)
xK=fft(x,N)
xn=ifft(xK)
subplot(3,1,1)
```

```
stem(abs(xK))
title('Magnitude of Fourier Spectrum')
xlabel('frequency')
ylabel('Magnitude')
subplot(3,1,2)
stem(angle(xK))
title('Phase of Fourier Spectrum')
xlabel('frequency')
ylabel('phase')
subplot(3,1,3)
stem(xn)
title('Inverse FFT')
xlabel('time')
ylabel('amplitude')
```

Output:

Enter the sequence : [1 2 4 6 8 3 -2 0 7]

x =

1 2 4 6 8 3 -2 0 7

N =

9

xK =

Columns 1 through 6

29.0000 -3.7476 - 9.3636i 5.2306 +13.6981i -7.0000 + 3.4641i -4.4829 + 2.2771i -4.4829 - 2.2771i

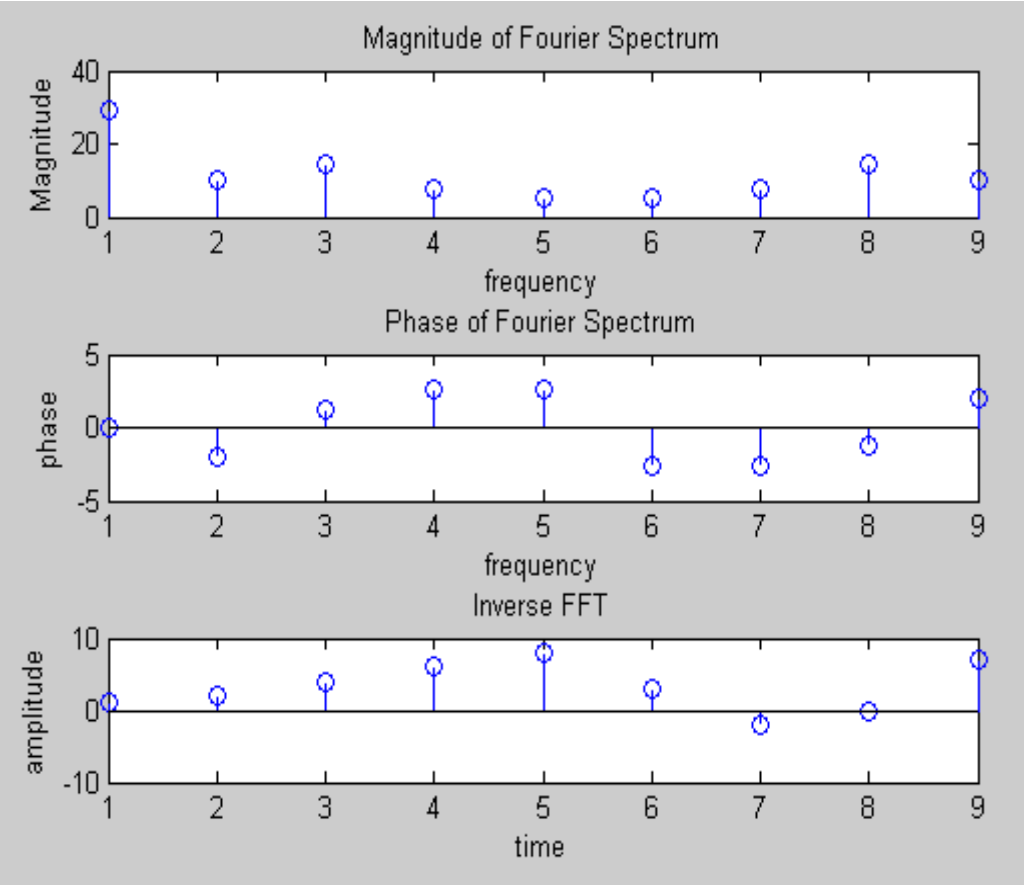
Columns 7 through 9

-7.0000 - 3.4641i 5.2306 -13.6981i -3.7476 + 9.3636i

xn =

1.0000 2.0000 4.0000 6.0000 8.0000 3.0000 -2.0000 0 7.0000

Plotting Magnitude and Phase spectrum



EXP.NO: 4
DETERMINATION OF POWER SPECTRUM OF A GIVEN SIGNAL

AIM: Determination of Power Spectrum of a given signals

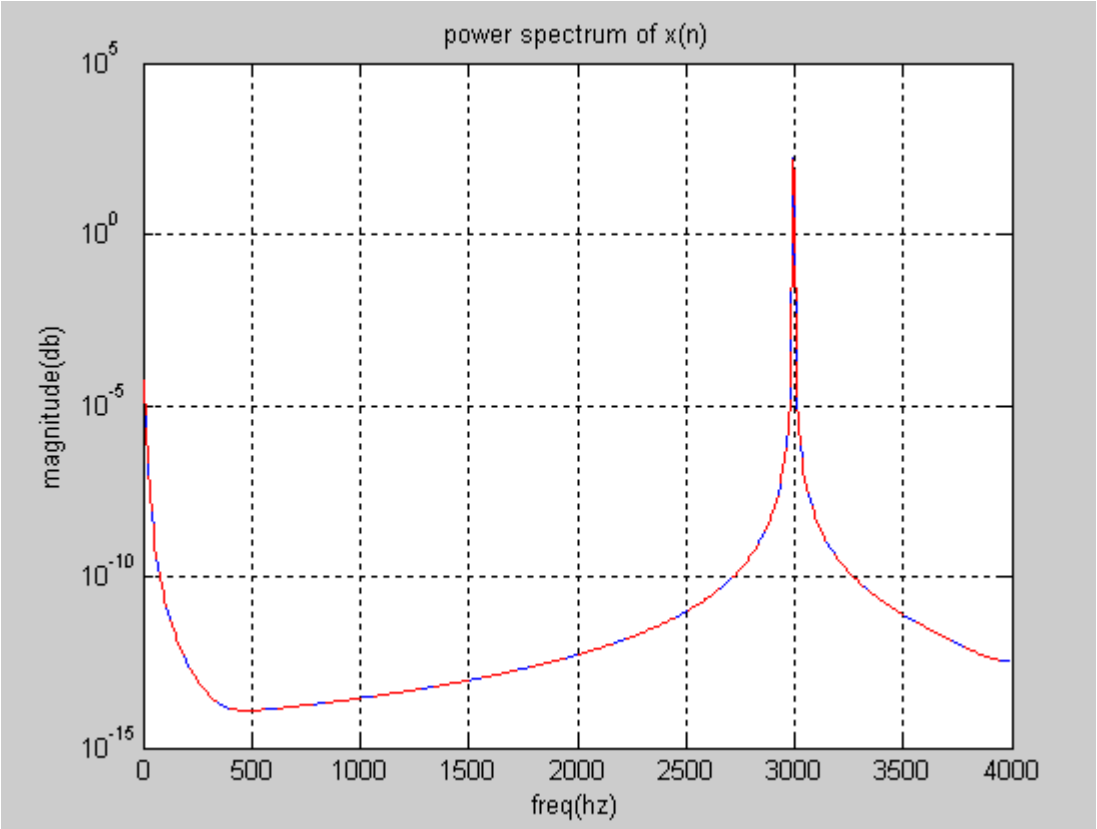
Software: MATLAB

PROGRAM:

```
clc
clear all
close all
N=1024;
fs=8000;
f=input('enter the frequency[1 to 5000]:');
n=0:N-1;
x=sin(2*pi*(f/fs)*n)
pxx=spectrum(x,N);
specplot(pxx,fs);
grid on
xlabel('freq(hz)');
ylabel('magnitude(db)');
title('power spectrum of x(n)');
```

OUTPUT:

enter the frequency[1 to 5000]:3000


power spectrum of x(n)
magnitude(db)
freq(hz)
10^5
10^0
10^-5
10^-10
10^-15
0
500
1000
1500
2000
2500
3000
3500
4000

EXP. NO: 5
IMPLEMENTATION OF LP FIR FILTER FOR A GIVEN SEQUENCE

AIM: To implement LP FIR filter for a given sequence.

Software: MATLAB

THEORY:

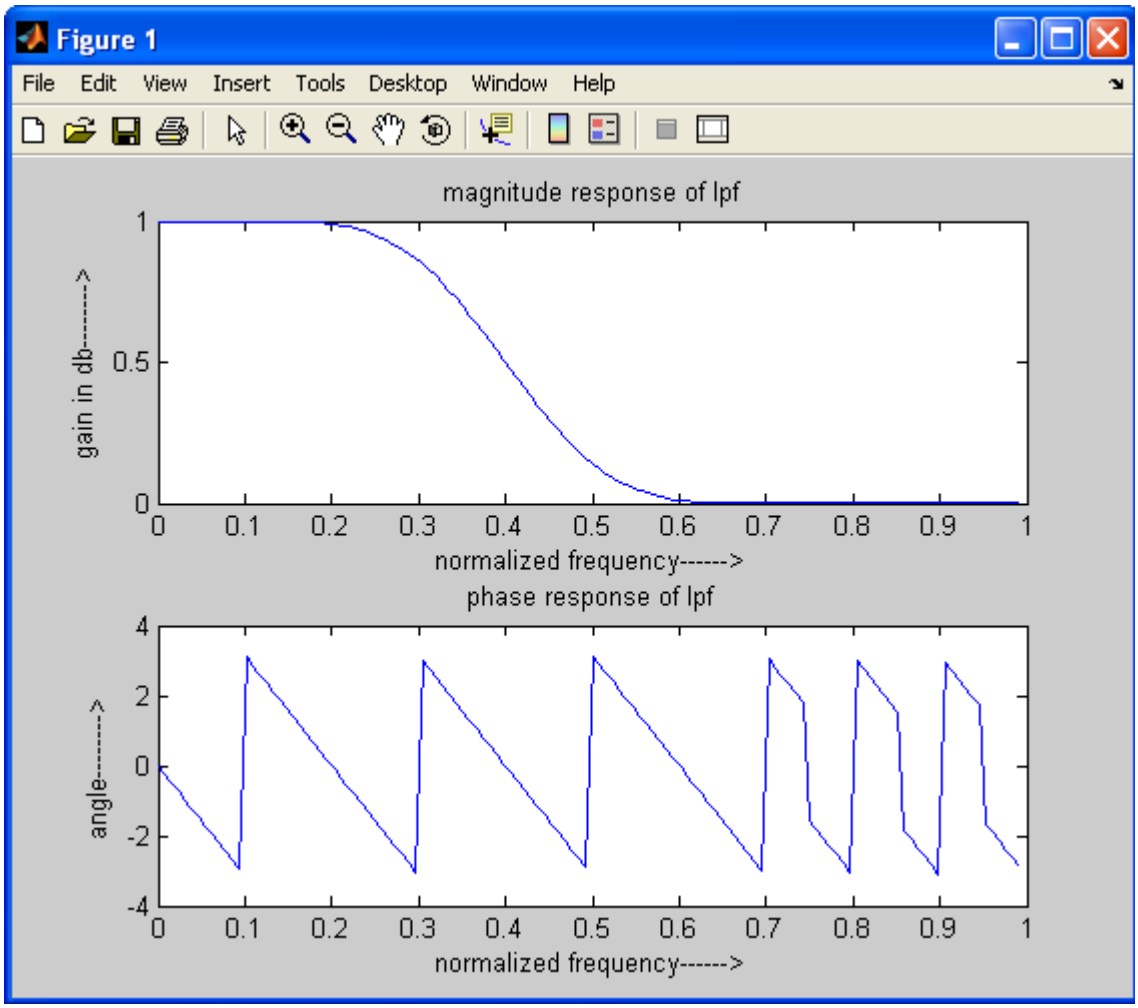
Window Name	Transition Width $\Delta\omega$ Approximate	Exact values	Min. Stop band Attenuation	Matlab Command
Rectangular	$\frac{4\Pi}{M}$	$\frac{1.8\Pi}{M}$	**21db**	**B = FIR1(N,Wn,boxcar)**
Bartlett	$\frac{8\Pi}{M}$	$\frac{6.1\Pi}{M}$	**25db**	**B = FIR1(N,Wn,bartlett)**
Hanning	$\frac{8\Pi}{M}$	$\frac{6.2\Pi}{M}$	**44db**	**B = FIR1(N,Wn,hanning)**
Hamming	$\frac{8\Pi}{M}$	$\frac{6.6\Pi}{M}$	**53db**	**B= FIR1(N,W_n,hamming)**
Blackman	$\frac{12\Pi}{M}$	$\frac{11\Pi}{M}$	**74db**	**B = FIR1(N,Wn,blackman)**

PROGRAM:

```
clc;
clear all;
close all;
n=20;
fp=200;
fq=300;
fs=1000;
fn=2*fp/fs;
window=blackman(n+1);
b=fir1(n,fn,window);
[H W]=freqz(b,1,128);
subplot(2,1,1);
plot(W/pi,abs(H));
title('magnitude response of lpf');
ylabel('gain in db-------->');
xlabel('normalized frequency------>');
```

```
subplot(2,1,2);
plot(W/pi,angle(H));
title('phase response of lpf');
ylabel('angle-------->');
xlabel('normalized frequency------>');
```

Result:



window =

-0.0000 0.0092 0.0402 0.1014 0.2008 0.3400 0.5098 0.6892 0.8492
0.9602 1.0000 0.9602 0.8492 0.6892 0.5098 0.3400 0.2008 0.1014
0.0402 0.0092 -0.0000

b = 0.0000 -0.0003 -0.0009 0.0027 0.0101
-0.0000 -0.0386 -0.0430 0.0794 0.2906
0.3999 0.2906 0.0794 -0.0430 -0.0386
-0.0000 0.0101 0.0027 -0.0009 -0.0003
0.0000

EXP. NO: 6
IMPLEMENTATION OF HP FIR FILTER FOR A GIVEN SEQUENCE

AIM: To implement HP FIR filter for a given sequence.

Software: MATLAB

THEORY:

Finite Impulse Response (FIR) Filter: The FIR filters are of non-recursive type, where by the present output sample is depending on the present input sample and previous input samples.

The transfer function of a FIR causal filter is given by,

$$H(z) = \sum_{n=0}^{N-1} h(n)z^{-n}$$

Where h(n) is the impulse response of the filter.

The Fourier transform of h(n) is

$$H(e^{jw}) = \sum_{n=0}^{N-1} h(n)e^{-jwn}$$

In the design of FIR filters most commonly used approach is using windows. The desired frequency response $H_d(e^{jw})$ of a filter is periodic in frequency and can be expanded in Fourier series. The resultant series is given by,

$$h_d(n) = (1/2\pi) \int_{-\pi}^{\pi} H(e^{jw})e^{jwn}\, dw$$

And known as Fourier coefficients having infinite length. One possible way of obtaining FIR filter is to truncate the infinite Fourier series at $n = \pm [(N-1)/2]$

Where N is the length of the desired sequence.

The Fourier coefficients of the filter are modified by multiplying the infinite impulse response with a finite weighing sequence w(n) called a window.

Where $w(n) = w(-n) \neq 0$ for $|n| \leq [(N-1)/2]$

$= 0$ for $|n| > [(N-1)/2]$

After multiplying w(n) with $h_d(n)$, we get a finite duration sequence h(n) that satisfies the desired magnitude response,

$h(n) = h_d(n)\, w(n)$ for $|n| \leq [(N-1)/2]$

$= 0$ for $|n| > [(N-1)/2]$

The frequency response $H(e^{jw})$ of the filter can be obtained by convolution of $H_d(e^{jw})$ and $W(e^{jw})$ is given by,

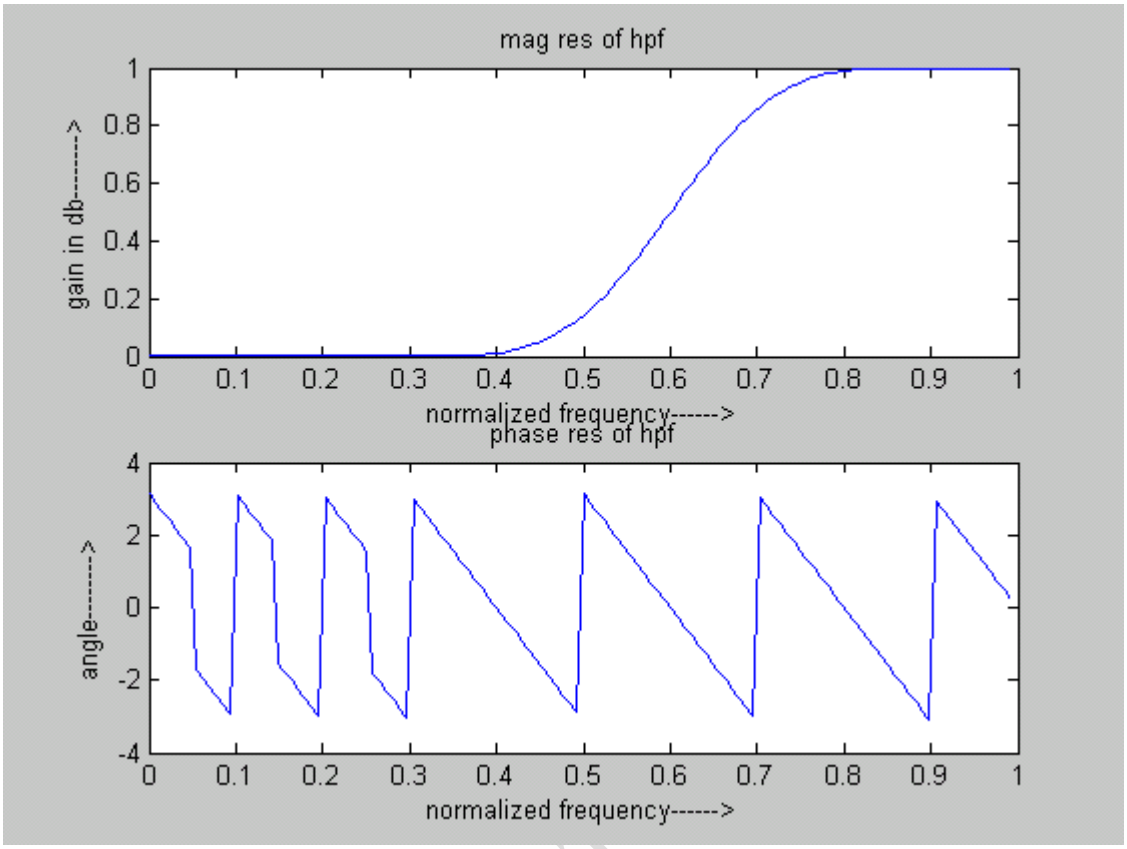
$$H(e^{jw}) = (1/2\pi) \int_{-\pi}^{\pi} H_d(e^{j\theta}) W(e^{j(w-\theta)}) d\theta$$

$$H(e^{jw}) = H_d(e^{jw}) * W(e^{jw}).$$

PROGRAM:

```
clc;
clear all;
close all;
n=20;
fp=300;
fq=200;
fs=1000;
fn=2*fp/fs;
window=blackman(n+1);
b=fir1(n,fn,'high',window);
[H W]=freqz(b,1,128);
subplot(2,1,1);
plot(W/pi,abs(H));
title('mag res of lpf');
ylabel('gain in db-------->');
xlabel('normalized frequency------>');
subplot(2,1,2);
plot(W/pi,angle(H));
title('phase res of lpf');
ylabel('angle-------->');
xlabel('normalized frequency------>');
```

RESULT:



fn= 0.6

window = -0.0000 0.0092 0.0402 0.1014 0.2008 0.3400
0.5098 0.6892 0.8492 0.9602 1.0000 0.9602 0.8492
0.6892 0.5098 0.3400 0.2008 0.1014 0.0402 0.0092
-0.0000

b =

0.0000 0.0003 -0.0009 -0.0027 0.0101
0.0000 -0.0386 0.0430 0.0794 -0.2906
0.3999 -0.2906 0.0794 0.0430 -0.0386
0.0000 0.0101 -0.0027 -0.0009 0.0003
0.0000

EXP. NO: 7
IMPLEMENTATION OF LP IIR FILTER FOR A GIVEN SEQUENCE

AIM: To implement LP IIR filter for a given sequence.

Software: MATLAB

PROGRAM:

```
clc;
clear all;
close all;
disp('enter the IIR filter design specifications');
rp=input('enter the passband ripple:');
rs=input('enter the stopband ripple:');
wp=input('enter the passband freq:');
ws=input('enter the stopband freq:');
fs=input('enter the sampling freq:');
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs,'s');
disp('Frequency response of IIR LPF is:');
[b,a]=butter(n,wn,'low','s');
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure,subplot(2,1,1);plot(om/pi,m);
title('magnitude response of IIR filter is:');
xlabel('(a) Normalized freq. -->');
ylabel('Gain in dB-->');
subplot(2,1,2);plot(om/pi,an);
title('phase response of IIR filter is:');
xlabel('(b) Normalized freq. -->');
ylabel('Phase in radians-->');
```

OUTPUT:

```
enter the IIR filter design specifications
```

enter the passband ripple:15

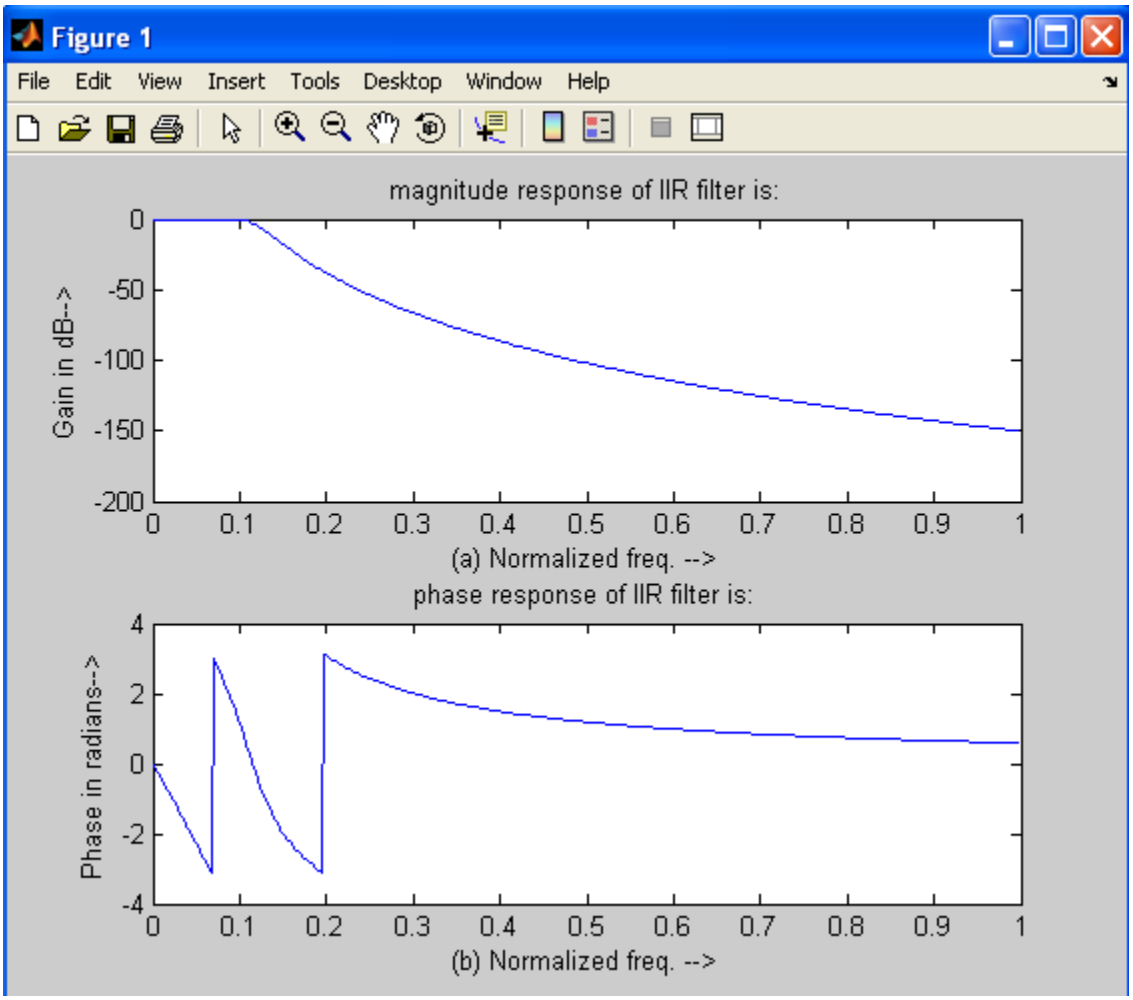
enter the stopband ripple:60

enter the passband freq:1500

enter the stopband freq:3000

enter the sampling freq:7000

Frequency response of IIR LPF is:



EXP. NO: 8
IMPLEMENTATION OF HP IIR FILTER FOR A GIVEN SEQUENCE

AIM: To implement HP IIR filter for a given sequence.

Software: MATLAB

PROGRAM:

```
clc;
clear all;
close all;
disp('enter the IIR filter design specifications');
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband freq');
ws=input('enter the stopband freq');
fs=input('enter the sampling freq');
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs,'s');
disp('Frequency response of IIR HPF is:');
[b,a]=butter(n,wn,'high','s');
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure,subplot(2,1,1);plot(om/pi,m);
title('magnitude response of IIR filter is:');
xlabel('(a) Normalized freq. -->');
ylabel('Gain in dB-->');
subplot(2,1,2);plot(om/pi,an);
title('phase response of IIR filter is:');
xlabel('(b) Normalized freq. -->');
ylabel('Phase in radians-->');
```

OUTPUT:

```
enter the IIR filter design specifications
```

enter the passband ripple15

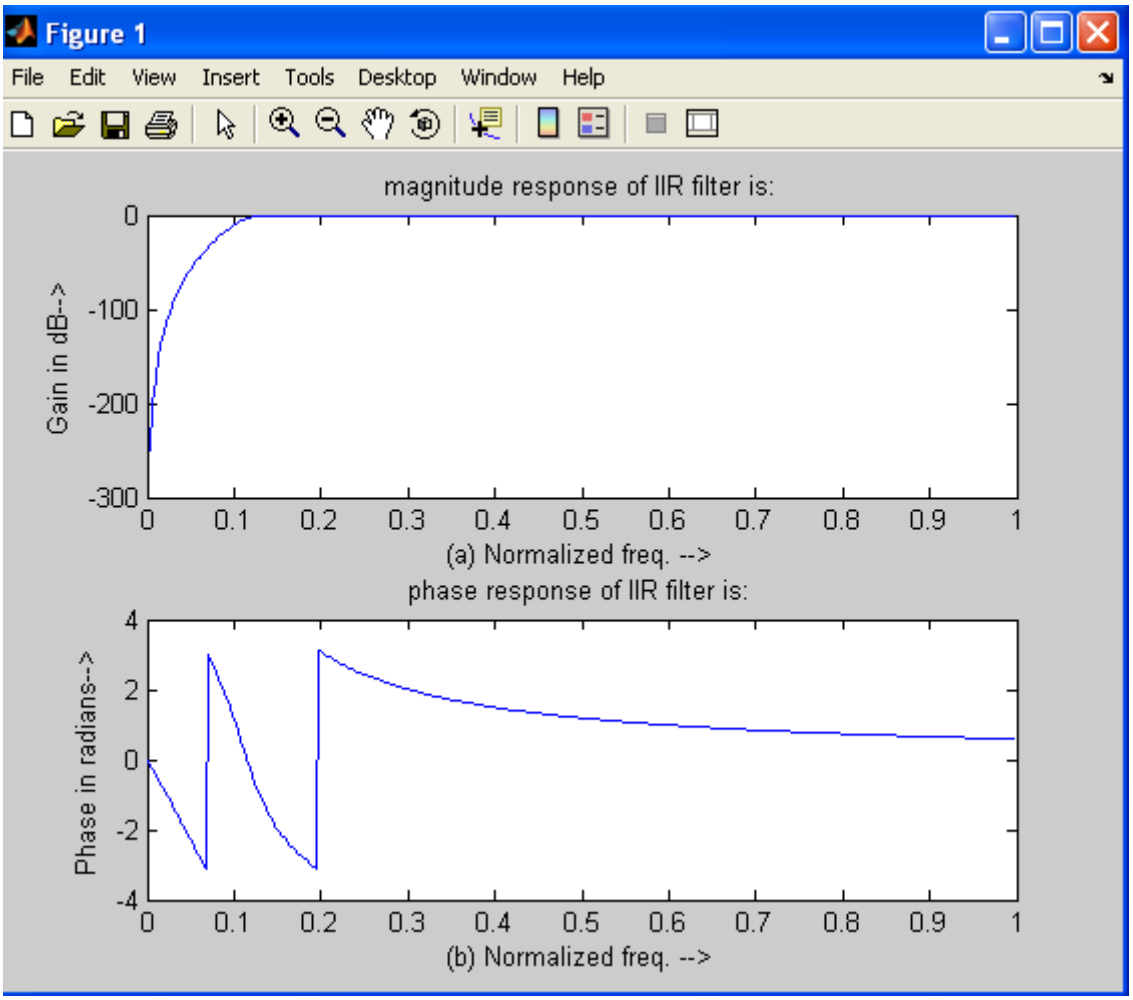
enter the stopband ripple60

enter the passband freq1500

enter the stopband freq3000

enter the sampling freq7000

Frequency response of IIR HPF is:



EXP. NO: 9

GENERATION OF SINUSOIDAL SIGNAL THROUGH FILTERING

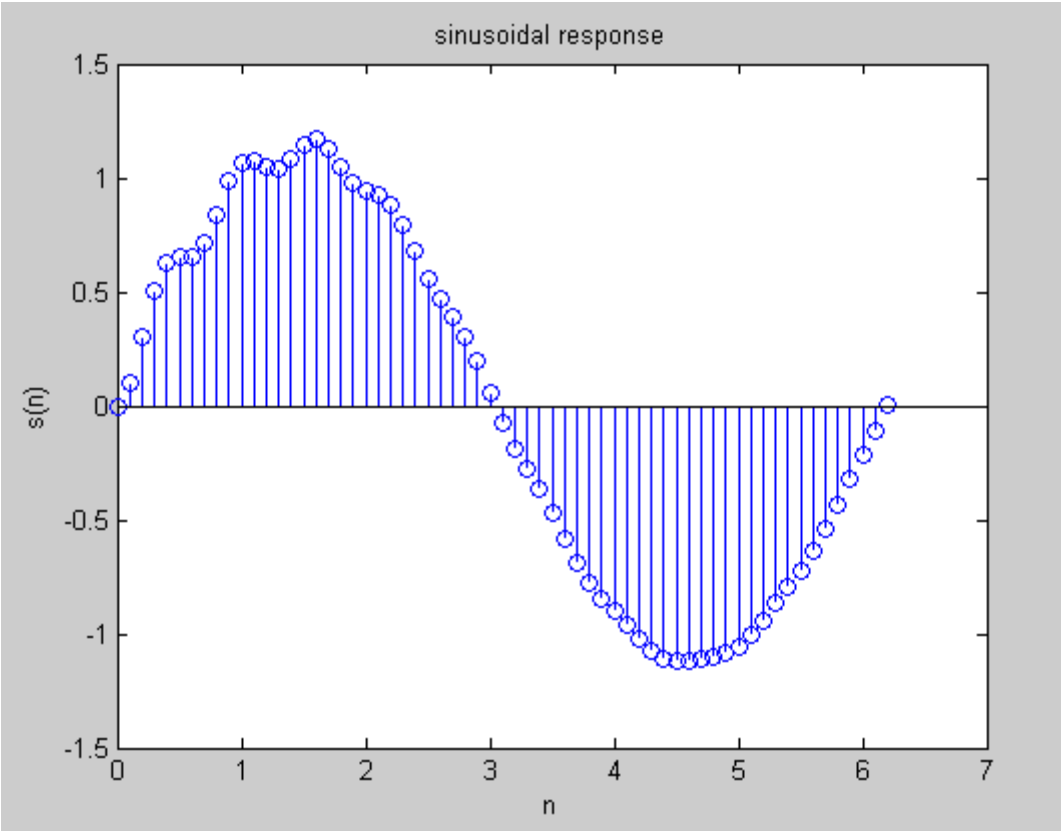
AIM: To generate a sinusoidal signal through filtering.

Software: MATLAB

PROGRAM:

```
close all;
clear all;
clc;
b=[1];
a=[1,-1,0.9];
n=[-20:120];
t=0:0.1:2*pi;
x=sin(t);
s=filter(b,a,x);
stem(t,s);
title('sinusoidal response');
xlabel('n');
ylabel('s(n)');
```

OUTPUT:



EXP. NO: 10
GENERATION OF DTMF SIGNALS

AIM: To generate DTMF Signals using MATLAB Software.

Software: MATLAB

THEORY:

The DTMF stands for "*D*ual *T*one *M*ulti *F*requency", and is a method of representing digits with tone frequencies, in order to transmit them over an analog communications network, for example a telephone line. In telephone networks, DTMF signals are used to encode dial trains and other information.

Dual-tone Multi-Frequency (DTMF) signaling is the basis for voice communications control and is widely used worldwide in modern telephony to dial numbers and configure switchboards. It is also used in systems such as in voice mail, electronic mail and telephone banking.

A DTMF signal consists of the sum of two sinusoids - or tones - with frequencies taken from two mutually exclusive groups. These frequencies were chosen to prevent any harmonics from being incorrectly detected by the receiver as some other DTMF frequency. Each pair of tones contains one frequency of the low group (697 Hz, 770 Hz, 852 Hz, 941 Hz) and one frequency of the high group (1209 Hz, 1336 Hz, 1477Hz) and represents a unique symbol.

Frequency	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

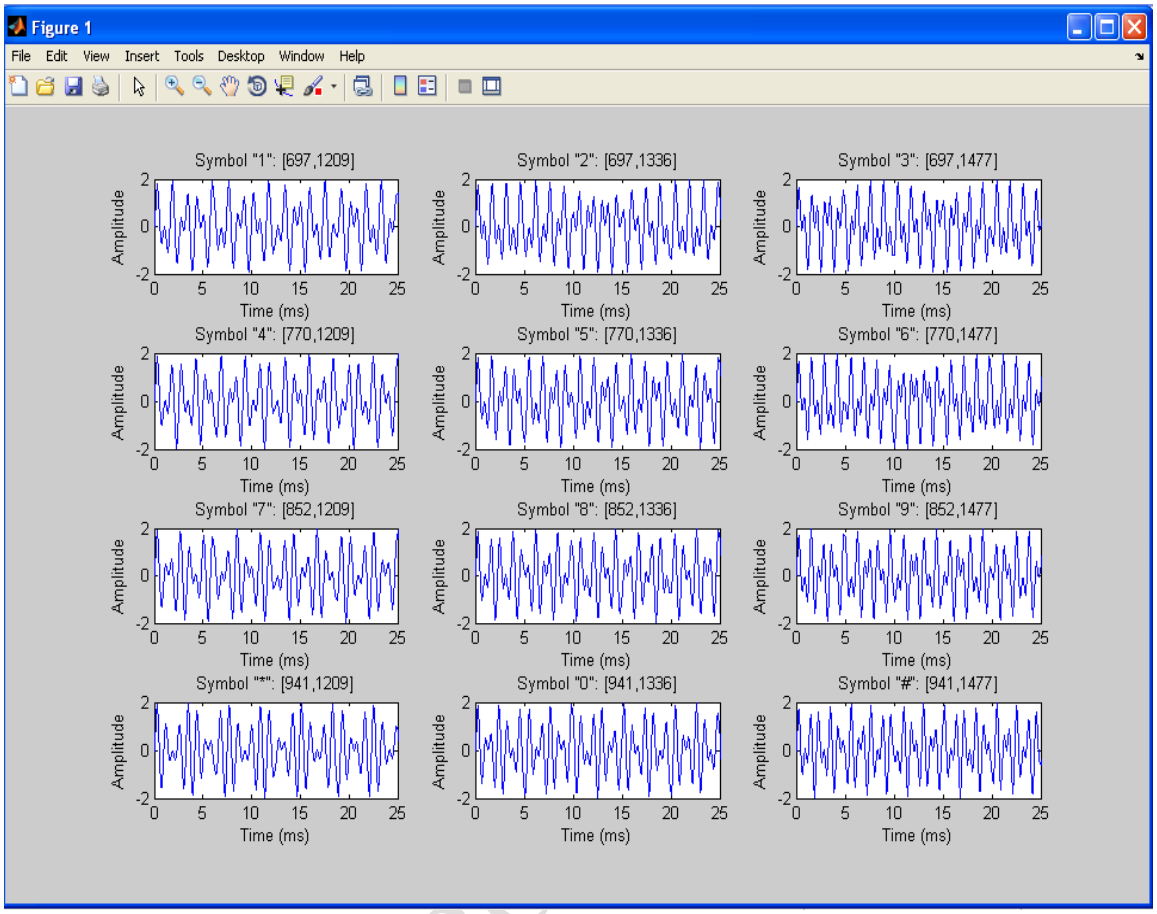
Program:

```
clc
close all
clear all
 %generate the twelve frequency pairs
symbol = {'1','2','3','4','5','6','7','8','9','*','0','#'};
lfg = [697 770 852 941]; % Low frequency group
hfg = [1209 1336 1477];  % High frequency group
f  = [];
```

```
for c=1:4,
   for r=1:3,
      f = [ f [lfg(c);hfg(r)] ];
   end
end
f';
 %generate and visualize the DTMF tones
 Fs  = 8000;       % Sampling frequency 8 kHz
N = 800;          % Tones of 100 ms
t   = (0:N-1)/Fs; % 800 samples at Fs
tones = zeros(N,size(f,2));
for toneChoice=1:12,
   % Generate tone
   tones(:,toneChoice) = sum(sin(f(:,toneChoice)*2*pi*t)';
   % Plot tone
   subplot(4,3,toneChoice),plot(t*1e3,tones(:,toneChoice));
   title(['Symbol "', symbol{toneChoice},'":
[',num2str(f(1,toneChoice)),',',num2str(f(2,toneChoice)),']'])
   set(gca, 'Xlim', [0 25]);
   ylabel('Amplitude');
    xlabel('Time (ms)');
end
```

Generation of DTMF Signals:

Output



EXP. NO: 11
IMPLEMENTATION OF DECIMATION PROCESS

AIM: program to verify the decimation of given sequence.

SOFTWARE: MATLAB 7.0 and above

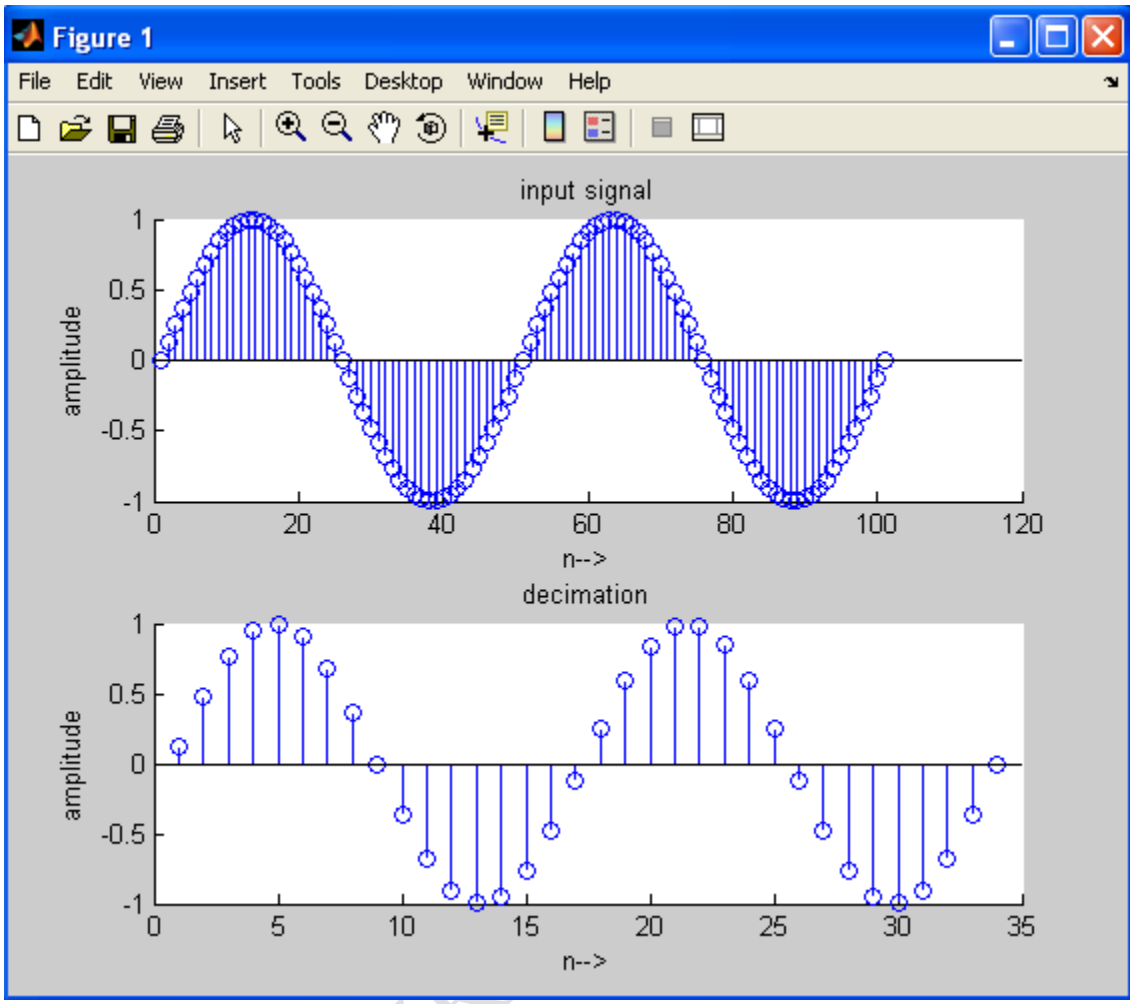
Signal Processing Toolbox

PROGRAM:

```
clc;
clear all;
close all;
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%input signal
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
fm=20;
fs=1000;
t=0.1;
dt=1/fs;
n=0:dt:t;
m=sin(2*pi*fm*n);
subplot(2,1,1);
stem(m);
xlabel('n-->');
ylabel('amplitude');
title('input signal');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%decimation
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
r=input(' enter the length of decimation factor r= ');
d=decimate(m,r);
subplot (2,1,2);
stem(d);
xlabel('n-->');
ylabel('amplitude');
title('decimation');
```

OUTPUT:

enter the length of decimation factor b= 3



EXP. NO: 12
IMPLEMENTATION OF INTERPOLATION PROCESS

AIM: program to verify the decimation of given sequence.

SOFTWARE: MATLAB 7.0 and above

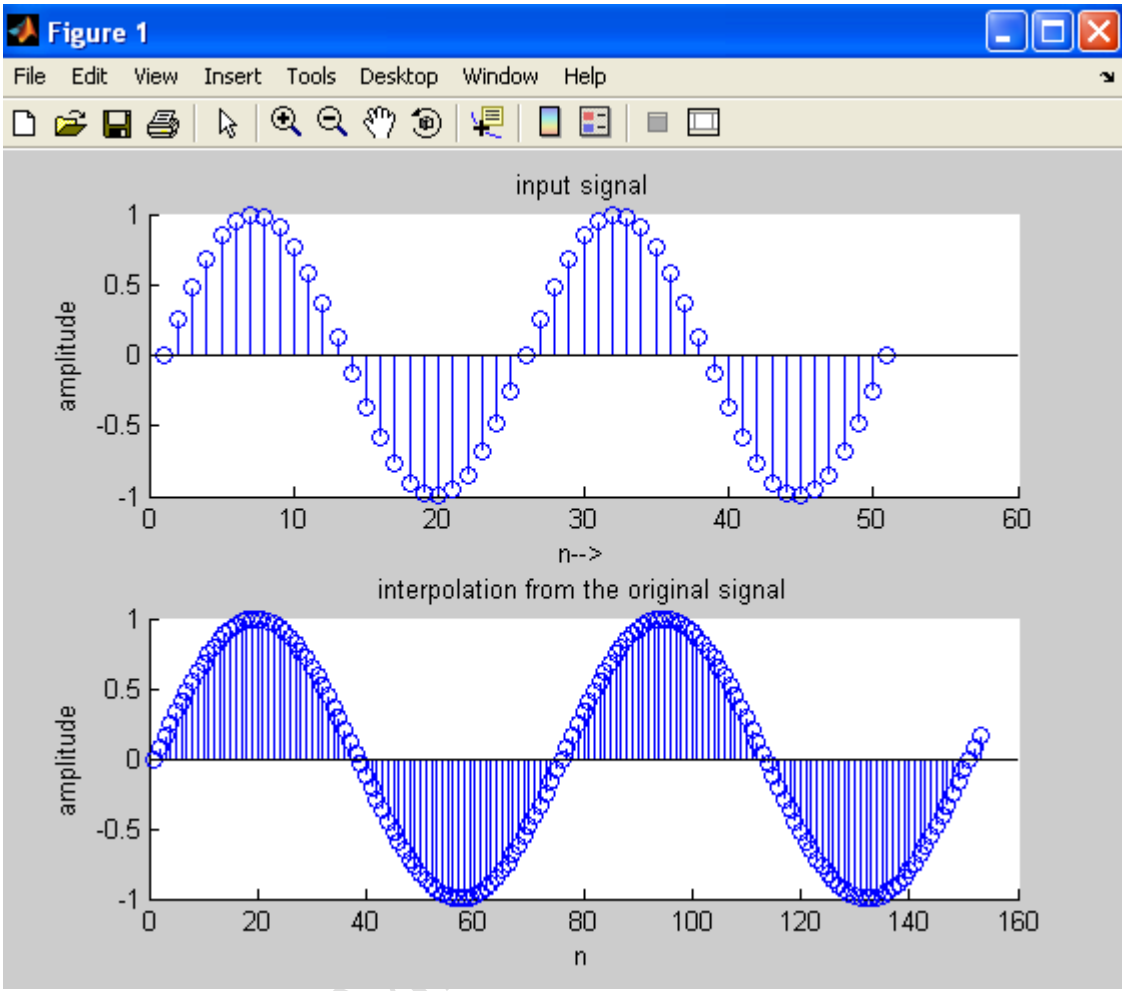
Signal Processing Toolbox

PROGRAM:

```
clc;
clear all;
close all;
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%input signal
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
fm=20;
fs=500;
t=0.1;
dt=1/fs;
n=0:dt:t;
m=sin(2*pi*fm*n);
subplot(2,1,1);
stem(m);
xlabel('n-->');
ylabel('amplitude');
title('input signal');
%from the origina signal interpolation
b=input('enter the length of interpolation factor b= ');
i=interp(m,b);
subplot(2,1,2);
stem(i);
xlabel('n');
ylabel('amplitude');
title('interpolation from the original signal');
```

OUTPUT:

enter the length of interpolation factor b= 3



EXP. NO: 13
IMPLEMENTATION OF I/D SAMPLING RATE CONVERTERS

AIM: program to implement sampling rate conversion.

SOFTWARE: MATLAB 7.0 and above

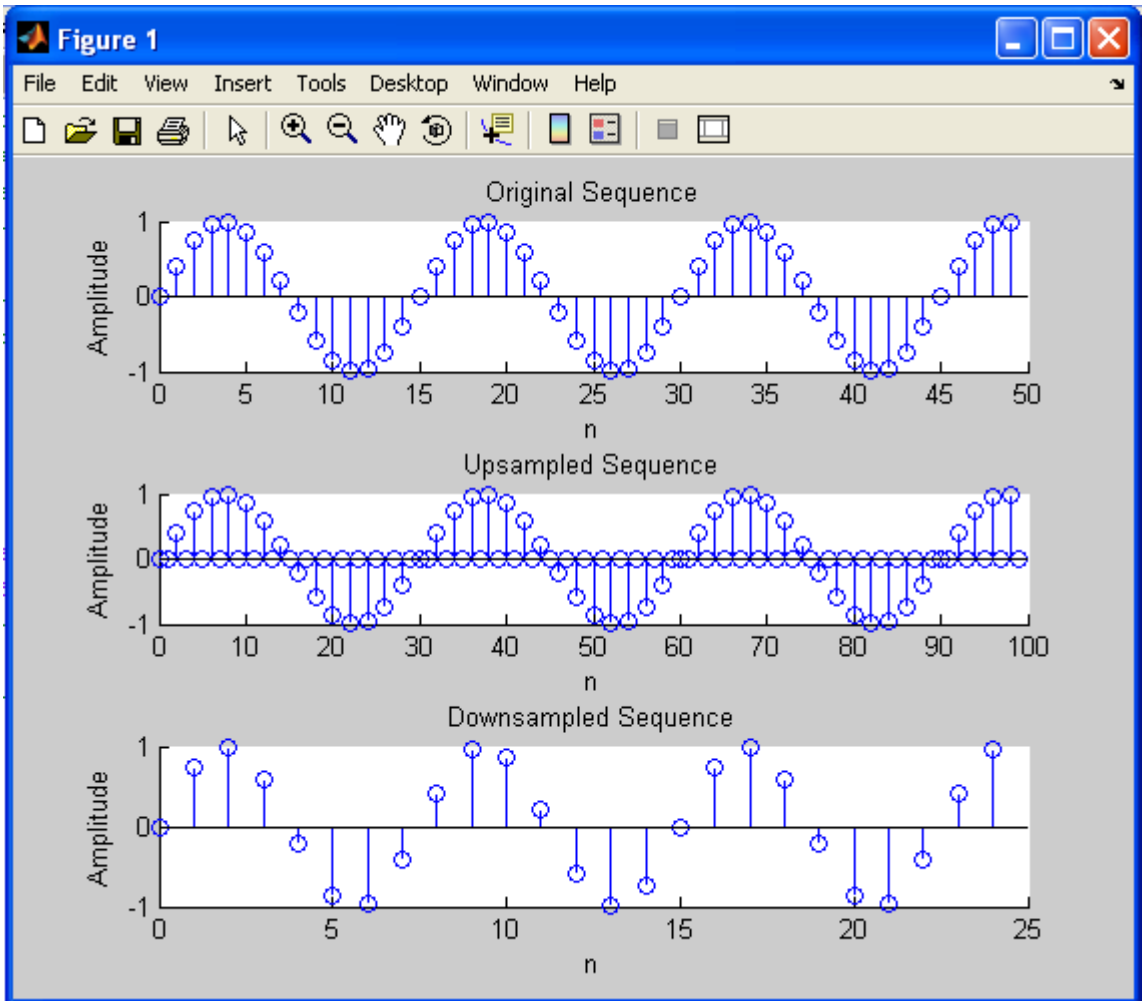
Signal Processing Toolbox

PROGRAM:

```
clc;% clear comand window
clear all; % clear work space
close all; %close all figure windows
%input signal
N=50 ;% no of samples
n=0:1:N-1;
x=sin(2*pi*n/15);% input signal
subplot(3,1,1)
stem(n,x);
xlabel('n');
ylabel('Amplitude');
title('Original Sequence');
%up sampling
L=2;% upsampling factor
x1=[zeros(1,L*N)];
n1=1:1:L*N;
j =1:L:L*N;
x1(j)=x;
subplot(3,1,2)
stem(n1-1,x1);
xlabel('n');
ylabel('Amplitude');
title('Upsampled Sequence');
% down sampling
M=2;
x2=x(1:M:N);
n2=1:1:N/M;
subplot(3,1,3)
```

```
stem(n2-1,x2);
xlabel('n');
ylabel('Amplitude');
title('Downsampled Sequence');
```

Output waveforms for Sampling Rate Conversion:



PART B - LIST OF EXPERIMENTS USING DSP PROCESSOR ARCHITECTURE AND INSTRUCTION SET OF DSPCHIP-TMS320C5515

Introduction to the TMS320C55x:

The TMS320C55x digital signal processor (DSP) represents the latest generation of 'C5000 DSPs from Texas Instruments. The 'C55x is built on the proven legacy of the 'C54x and is source code compatible with the 'C54x, protecting the customer's software investment. Following the trends set by the 'C54x, the 'C55x is optimized for power efficiency, low system cost, and best-in-class performance for tight power budgets. With core power dissipation as low as 0.05 mW/MIPS at 0.9V, and performance up to 800 MIPS (400 MHz), the TMS320C55x offers a cost-effective solution to the toughest challenges in personal and portable processing applications as well as digital communications infrastructure with restrictive power budgets. Compared to a 120-MHz 'C54x, a 300-MHz 'C55x will deliver approximately 5X higher performance and dissipate one-sixth the core power dissipation of the 'C54x. The 'C55x core's ultra-low power dissipation of 0.05mW/MIPS is achieved through intense attention to low-power design and advanced power management techniques. The 'C55x designers have implemented an unparalleled level of power-down configurability and granularity coupled with unprecedented power management that occurs automatically and is transparent to the user.

The 'C55x core delivers twice the cycle efficiency of the 'C54x through a dual-MAC (multiply-accumulate) architecture with parallel instructions, additional accumulators, ALUs, and data registers. An advanced instruction set, a superset to that of the 'C54x, combined with expanded busing structure complements the new hardware execution units. The 'C55x continues the standard set by the 'C54x in code density leadership for lower system cost. The 'C55x instructions are variable byte lengths ranging in size from 8 bits to 48 bits. With this scalable instruction word length, the 'C55x can reduce control code size per function by up to 40% more than 'C54x. Reduced control code size means reduced memory requirements and lower system cost.

Key Features of the 'C55x

The 'C55x incorporates a rich set of features that provide processing efficiency, low-power dissipation, and ease of use. Some of these features are listed in Table

Feature(s)	Benefit(s)
A 32 x 16-bit Instruction buffer queue	Buffers variable length instructions and implements efficient block repeat operations
Two 17-bit x17-bit MAC units	Execute dual MAC operations in a single cycle
One 40-bit ALU	Performs high precision arithmetic and logical operations
One 40-bit Barrel Shifter	Can shift a 40-bit result up to 31 bits to the left, or 32 bits to the right
One 16-bit ALU	Performs simpler arithmetic in parallel to main ALU
Four 40-bit accumulators	Hold results of computations and reduce the required memory traffic
Twelve independent buses: – Three data read buses – Two data write buses – Five data address buses – One program read bus – One program address bus	Provide the instructions to be processed as well as the operands for the various computational units in parallel —to take advantage of the 'C55x parallelism.
User-configurable *IDLE* Domains	Improve flexibility of low-activity power management

Overview of the C5515 eZdsp USB Stick

The C5515 eZdsp USB Stick is an evaluation tool for the Texas Instruments TMS320C5515 Digital Signal Processor (DSP). This USB bus powered tool allows the user to evaluate the following items:

• The TMS320C5515 processor along with its peripherals

• The TLV320AIC3204 codec

• The Code Composer Studio IDETM software development tools

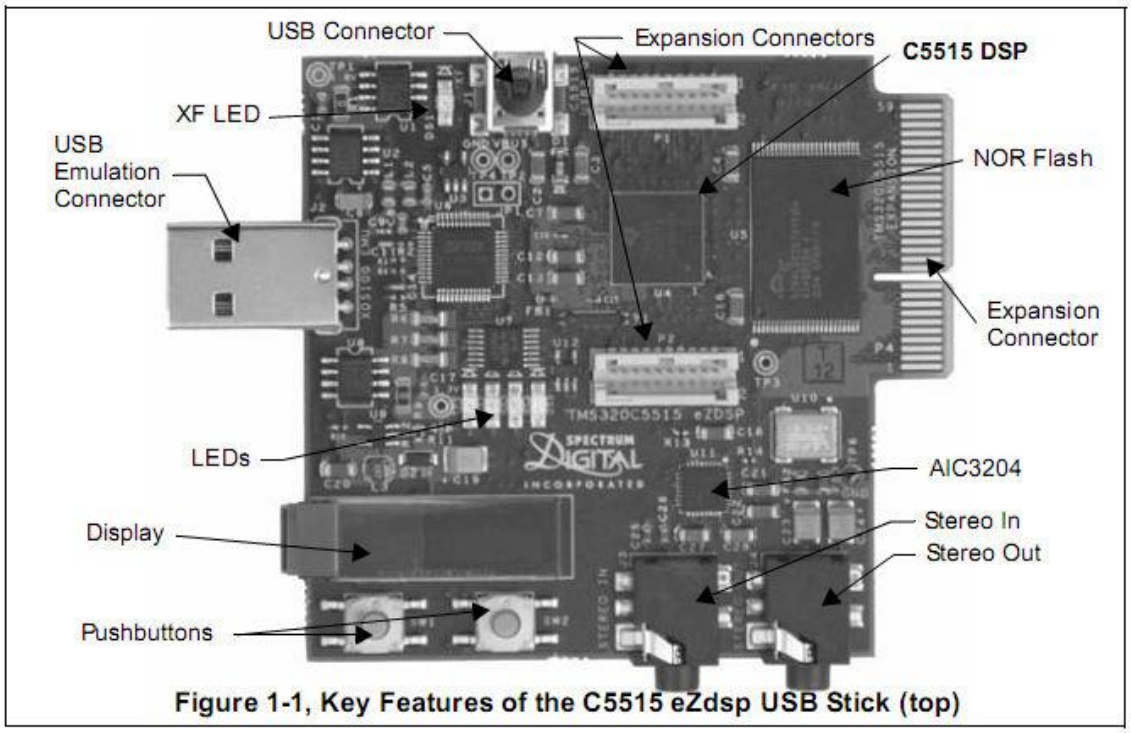


Figure 1-1, Key Features of the C5515 eZdsp USB Stick (top)

Key Features of the C5515 eZdsp USB Stick

The C5515 eZdsp USB Stick has the following features:

• Texas Instrument's TMS320C5515 Digital Signal Processor

• Texas Instruments TLV320AIC3204 Stereo Codec (stereo in, stereo out)

• Micro SD connector

• USB 2.0 interface to C5515 processor

• 32 Mb NOR flash

• I2C OLED display

• 5 user controlled LEDs

• 2 user readable push button switches

• Embedded USB XDS100 JTAG emulator

• Bluetooth board interface

• Expansion edge connector

• Power provided by USB interface

• Compatible with Texas Instruments Code Composer Studio v4

• USB extension cable

C5515 eZdsp USB Stick Block Diagram

The block diagram of the C5515 eZdsp USB Stick is shown below.

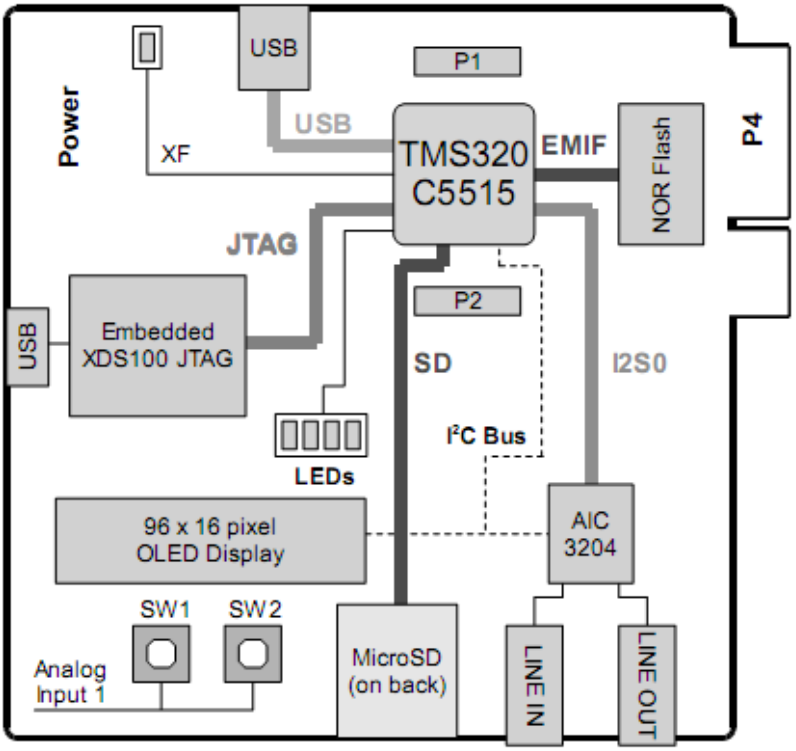


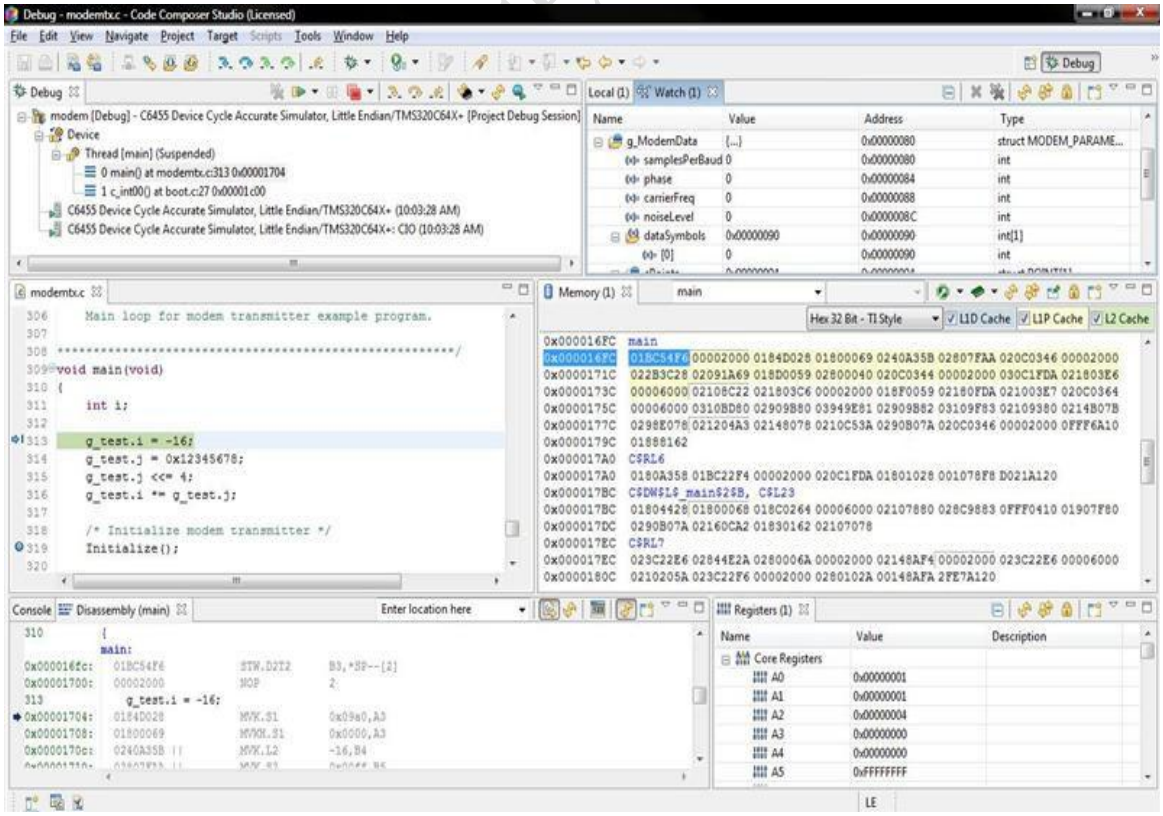
Figure 1-3, C5515 eZdsp USB Stick Block Diagram

CODE COMPOSER STUDIO

INTRODUCTION TO CODE COMPOSER STUDIO

Code Composer Studio™ (CCS or CCStudio) is the integrated development environment for TI's DSPs, microcontrollers and application processors. CCStudio includes a suite of tools used to develop and debug embedded applications. It includes compilers for each of TI's device families, source code editor, project build environment, debugger, profiler, simulators and many other features. CCStudio provides a single user interface taking users through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before and add functionality to their application thanks to sophisticated productivity tools.

CCStudio version 4 (CCSv4) is based on the Eclipse open source software framework. CCSv4 is based on Eclipse because it offers an excellent software framework for development environments a standard framework many embedded software vendors. CCSv4 combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature rich development environment for embedded developers.



Features

Debugger

CCStudio's integrated debugger has several capabilities and advanced breakpoints to simplify development. Conditional or hardware breakpoints are based on full C expressions, local variables or registers. The advanced memory window allows you to inspect each level of memory so that you can debug complex cache coherency issues. CCStudio supports the development of complex systems with multiple processors or cores. Global breakpoints and synchronous operations provide control over multiple processors and cores.

Profiling

CCStudio's interactive profiler makes it easy to quickly measure code performance and ensure the efficient use of the target's resources during debug and development sessions. The profiler allows developers to easily profile all C/C++ functions in their application for instruction cycles or other events such as cache misses/hits, pipeline stalls and branches.

Profile ranges can be used to concentrate efforts on high-usage areas of code during optimization, helping developers produce finely-tuned code. Profiling is available for ranges of assembly, C++ or C code in any combination. To increase productivity, all profiling facilities are available throughout the development cycle.

Scripting

Some tasks such as testing need to run for hours or days without user interaction. To accomplish such a task, the IDE should be able to automate common tasks. CCStudio has a complete scripting environment allowing for the automation of repetitive tasks such as testing and performance benchmarking. A separate scripting console allows you to type commands or to execute scripts within the IDE.

Image Analysis and Visualization

CCStudio has many image analysis and graphic visualization. It includes the ability to graphically view variables and data on displays which can be automatically refreshed. CCStudio can also look at images and video data in the native format (YUV, RGB) both in the host PC or loaded in the target board.

Compiler

TI has developed C/C++ compilers specifically tuned to maximize the processor's usage and performance. TI compilers use a wide range of classical, application-oriented,

and sophisticated device-specific optimizations that are tuned to all the supported architectures.

Some of these optimizations include:

• Common sub-expression elimination
• Software pipelining
• Strength Reduction
• Auto increment addressing
• Cost-based register allocation
• Instruction predication
• Hardware looping
• Function In-lining
• Vectorization

TI compilers also perform program level optimizations that evaluate code performance at the application level. With the program level view, the compiler is able to generate code similar to an assembly program developer who has the full system view. This application level view is leveraged by the compiler to make trade-offs that significantly increase the processor performance.

The TI ARM and Microcontroller C/C++ compilers are specifically tuned for code size and control code efficiency. They offer industry leading performance and compatibility.

Simulation

Simulators provide a way for users to begin development prior to having access to a development board. Simulators also have the benefit of providing enhanced visibility into application performance and behavior. Several simulator variants are available allowing users to trade off cycle accuracy, speed and peripheral simulation, with some simulators being ideally suited to algorithm benchmarking and others for more detailed system simulation. Hardware Debugging (Emulation)

TI devices include advanced hardware debugging capabilities. These capabilities include:

• IEEE 1149.1 (JTAG) and Boundary Scan
• Non-intrusive access to registers and memory
• Real-time mode which provides for the debugging of code that interacts with interrupts that must not be disabled. Real-time mode allows you to suspend background code at break events while continuing to execute time-critical interrupt service routines.

• Multi-core operations such as synchronous run, step, and halt. This includes crosscore triggering, which provides the ability to have one core trigger other cores to halt. Advanced Event Triggering (AET) which is available on selected devices, allows a user to halt the CPU or trigger other events based on complex events or sequences such as invalid data or program memory accesses. It can non-intrusively measure performance and count system events (for example, cache events).

CCStudio provides Processor Trace on selected devices to help customers find previously "invisible" complex real-time bugs. Trace can detect the really hard to find bugs – race conditions between events, intermittent real-time glitches, crashes from stack overflows, runaway code and false interrupts without stopping the processor. Trace is a completely nonintrusive debug method that relies on a debug unit inside the processor so it does not interfere or change the application's real-time behavior. Trace can fine tune code performance and cache optimization of complex switch intensive multi-channel applications. Processor Trace supports the export of program, data, timing and selected processor and system events/interrupts. Processor Trace can be exported either to an XDS560 Trace external JTAG emulator, or on selected devices, to an on chip buffer Embedded Trace Buffer (ETB).

Real time operating system support

CCSv4 comes with two versions of TI's real time operating system:

• DSP/BIOS 5.4x is a real-time operating system that provides pre-emptive multitasking services for DSP devices. Its services include ISR dispatching, software interrupts, semaphores, messages, device I/O, memory management, and power management. In addition, DSP/BIOS 5.x also includes debug instrumentation and tooling, including low-overhead print and statistics gathering.

• BIOS 6.x is an advanced, extensible real-time operating system that supports ARM926, ARM Cortex M3, C674x, C64x+, C672x, and 28x-based devices. It offers numerous kernel and debugging enhancements not available in DSP/BIOS 5.x, including faster, more flexible memory management, events, and priority-inheritance mutexes.

Note: BIOS 6.x includes a DSP/BIOS 5.x compatibility layer to support easy migration of application source code.

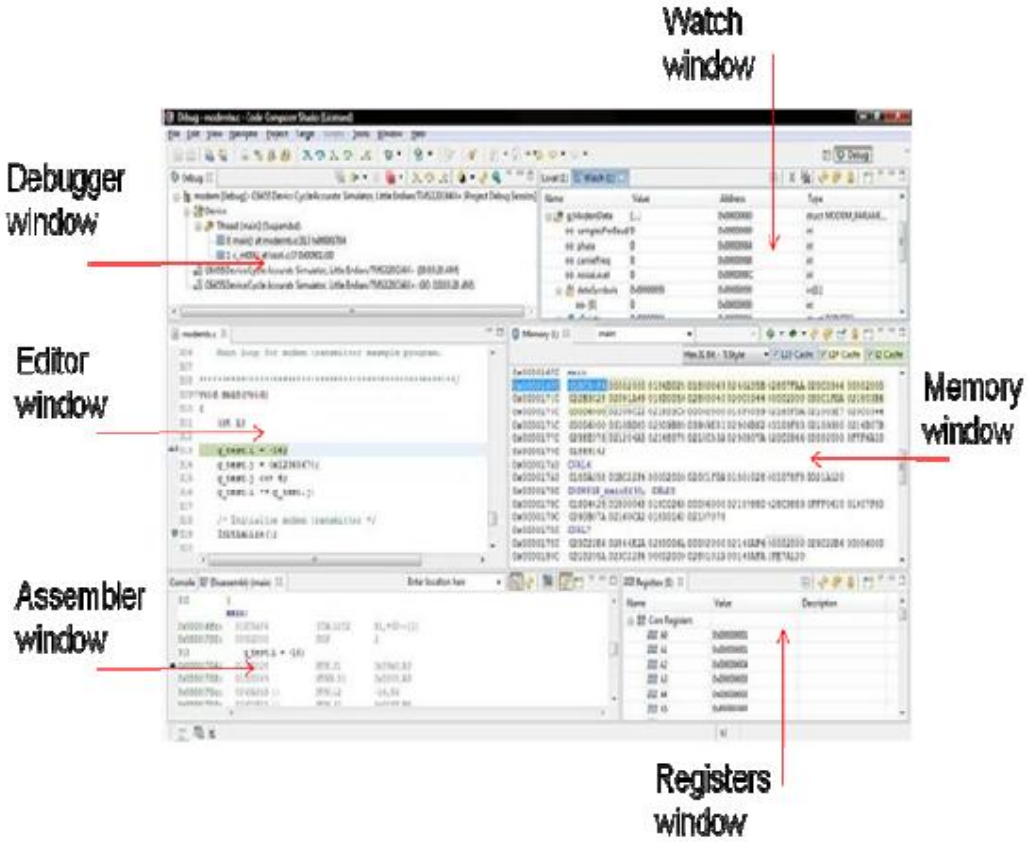
Step 1:

Open the code composer studio (CCSV4) and give a name to workspace and store it in the default path itself.

Note: don't assign other than default path unless you are familiar with eclipse frame work based CCSV4

Step 2:

Project windows overview



EXP.NO: 14
COMPUTATION OF N- POINT DFT OF A GIVEN SEQUENCE

Aim: To compute the N (=4/8/16) point DFT of the given sequence

EQUIPMENTS:

- Host (PC) with windows (95/98/Me/XP/NT/2000).
- TMS320C5515 DSP Starter Kit (DSK).

Theory:

The N point DFT of discrete time signal x[n] is given by the equation

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} \quad ; \quad k = 0,1,2,\ldots N-1$$

Where N is chosen such that $N \geq L$, where L=length of x[n]. To implement using C program we use the expression $e^{\frac{-j2\pi kn}{N}} = \cos\left(\frac{2\pi kn}{N}\right) - j\sin\left(\frac{2\pi kn}{N}\right)$ and allot memory space for real and imaginary parts of the DFT X(k)

Program

```
//dft.c N-point DFT of sequence read from lookup table
#include <stdio.h>
#include <math.h>
#define PI 3.14159265358979
#define N 64
#define TESTFREQ 10000.0
#define SAMPLING_FREQ 64000.0
typedef struct
{
float real;
float imag;
} COMPLEX;
float x1[N],y1[N];
COMPLEX samples[N];
void dft(COMPLEX *x)
{
COMPLEX result[N];
int k,n,i;
for (k=0 ; k<N ; k++)
```

```
{
result[k].real=0.0;
result[k].imag = 0.0;
for (n=0 ; n<N ; n++)
{
result[k].real += x[n].real*cos(2*PI*k*n/N) + x[n].imag*sin(2*PI*k*n/N);
result[k].imag += x[n].imag*cos(2*PI*k*n/N) - x[n].real*sin(2*PI*k*n/N);
}
}
for (k=0 ; k<N ; k++)
{
x[k] = result[k];
}
printf("output");
for (i = 0 ; i < N ; i++) //compute magnitude
{
x1[i] = (int)sqrt(result[i].real*result[i].real + result[i].imag*result[i].imag);
printf("\n%d = %f",i,x1[i]);
}
}
void main() //main function
{
int n;
for(n=0 ; n<N ; n++)
{
y1[n] = samples[n].real = cos(2*PI*TESTFREQ*n/SAMPLING_FREQ);
samples[n].imag = 0.0;
printf("\n%d = %f",n,samples[n].real);
}
printf("real input data stored in array samples[]\n");
printf("\n"); // place breakpoint here
dft(samples); //call DFT function
printf("done!\n");
 }
```

Code Flow:

• Step 1 - Select no. of points for DFT(Eg: 64)

• Step 2 – Generate a sine wave of frequency ‘f ‘ (eg: 10 Hz with a sampling rate = No. of Points of DFT(eg. 64)) using math library function.

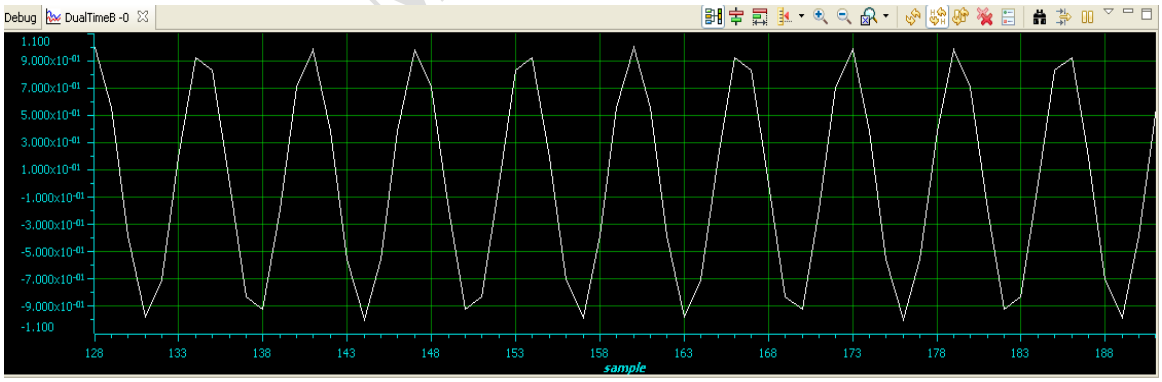
• Step 3 - Take sampled data and apply DFT algorithm.

Execution Procedure:

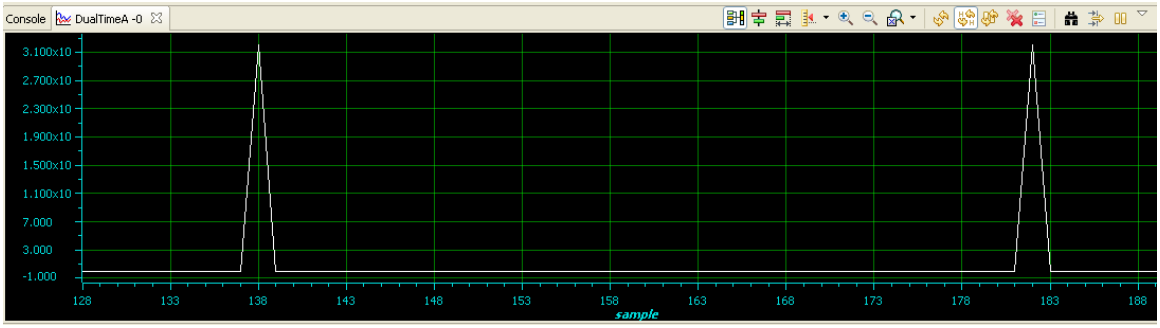
- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select DFT Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure from Step 46 to 51)
- Give Right Click on Your Dft.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- From Tools select Graph(Dual Time) , give properties and select OK.

Result:

Input Signal:



Output Signal :



Graph Properties:

Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	64
Dsp Data Type	32 bit floating point
Index Increment	1
Interleaved Data Sources	☐ false
Q_Value	0
Sampling Rate HZ	1
Start Address A	x1
Start Address B	y1
Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	64
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph A	☐ false
Use Dc Value For Graph B	☐ false

EXP.NO: 15
IMPLEMENTATION OF FFT OF GIVEN SEQUENCE

AIM: To compute the FFT of the given sequence

EQUIPMENTS:

1. Host (PC) with windows (95/98/Me/XP/NT/2000).
2. TMS320C5515 DSP Starter Kit (DSK).

FFT Algorithm

The FFT has a fairly easy algorithm to implement, and it is shown step by step in the list below. This version of the FFT is the Decimation in Time Method

1. Pad input sequence, of N samples, with Zero's until the number of samples is the nearest power of two.

e.g. 500 samples are padded to 512 (2^9)

2. Bit reverse the input sequence.

e.g. 3 = 011 goes to 110 = 6

3. Compute (N / 2) two sample DFT's from the shuffled inputs. See "Shuffled Inputs"
4. Compute (N / 4) four sample DFT's from the two sample DFT's. See "Shuffled Inputs"
5. Compute (N / 2) eight sample DFT's from the four sample DFT's. See "Shuffled Inputs"
6. Until the all the samples combine into one N-sample DFT

PROGRAM:

Main.c

```
#include "usbstk5515.h"
#include <math.h>
#include <stdio.h>
#define PTS 64 //no of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n); //FFT prototype
float iobuffer[PTS]; //as input and output buffer
float x1[PTS]; //intermediate buffer
short i; //general purpose index variable
short buffercount = 0; //number of new samples in iobuffer
short flag = 0; //set to 1 by ISR when iobuffer full
COMPLEX w[PTS]; //twiddle constants stored in w
```

```
COMPLEX samples[PTS]; //primary working buffer
void main(void)
{
for(i=0;i<PTS;i++)
{
iobuffer[i]=0;
x1[i]=0;
}
printf("\n input");
for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/(PTS*2.0)); //Re component of twiddle constants
w[i].imag =-sin(2*PI*i/(PTS*2.0)); //Im component of twiddle constants
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
iobuffer[i] = sin(2*PI*10*i/64.0);/*10- > freq, 64 -> sampling freq*/
printf("\n%d = %f",i,iobuffer[i]);
samples[i].real=0.0;
samples[i].imag=0.0;
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
printf("\n output");
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(samples[i].real*samples[i].real +
samples[i].imag*samples[i].imag);
printf("\n%d = %f",i,x1[i]);
```

```
}
} //end of main
fft.c
#define PTS 64 //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS]; //twiddle constants stored in w
void FFT(COMPLEX *Y, int N) //input sample array, # of points
{
COMPLEX temp1,temp2; //temporary storage variables
int i,j,k; //loop counter variables
int upper_leg, lower_leg; //index of upper/lower butterfly leg
int leg_diff; //difference between upper/lower leg
int num_stages = 0; //number of FFT stages (iterations)
int index, step; //index/step through twiddle constant
i = 1; //log(base2) of N points= # of stages
do
{
num_stages +=1;
i = i*2;
}while (i!=N);
leg_diff = N/2; //difference between upper&lower legs
step = (PTS*2)/N; //step between values in twiddle.h
for (i = 0;i < num_stages; i++) //for N-point FFT
{
index = 0;
for (j = 0; j < leg_diff; j++)
{
for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
{
lower_leg = upper_leg+leg_diff;
temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
```

```
(Y[lower_leg]).real = temp2.real*(w[index]).real
-temp2.imag*(w[index]).imag;
(Y[lower_leg]).imag = temp2.real*(w[index]).imag
+temp2.imag*(w[index]).real;
(Y[upper_leg]).real = temp1.real;
(Y[upper_leg]).imag = temp1.imag;
}
index += step;
}
leg_diff = leg_diff/2;
step *= 2;
}
j = 0;
for (i = 1; i < (N-1); i++) //bit reversal for resequencing data
{
k = N/2;
while (k <= j)
{
j = j - k;
k = k/2;
}
j = j + k;
if (i<j)
{
temp1.real = (Y[j]).real;
temp1.imag = (Y[j]).imag;
(Y[j]).real = (Y[i]).real;
(Y[j]).imag = (Y[i]).imag;
(Y[i]).real = temp1.real;
(Y[i]).imag = temp1.imag;
}
}
return;
}
```

Code Flow:

• Step 1 - Select no. of points for FFT(Eg: 64)

• Step 2 – Generate a sine wave of frequency ‘f ‘ (eg: 10 Hz with a sampling rate = No. of Points of FFT(eg. 64)) using math library function.

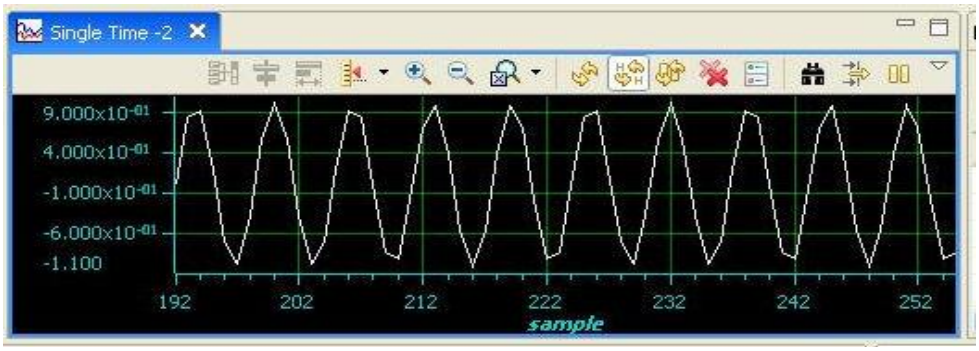
• Step 3 - Take sampled data and apply FFT algorithm.

Execution Procedure:

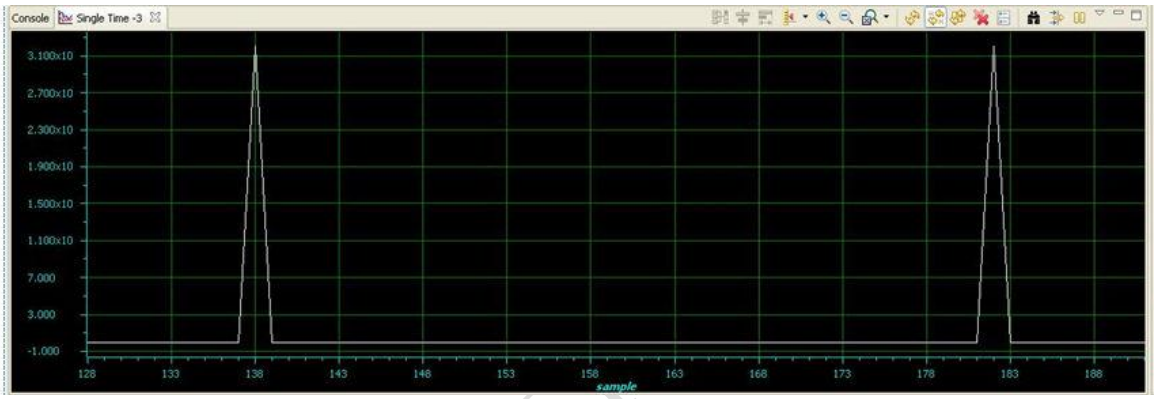
- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select FFT Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure from Step 46 to 51)
- Give Right Click on Your fft.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- From Tools select Graph(Dual Time) , give properties and select OK.

Result :

Input Signal:



Output Signal:



EXP.NO: 16
POWER SPECTRUM

Aim: To verify the power spectrum using DSP processor.

Equipments required:

1. Host (PC) with windows (95/98/Me/XP/NT/2000).
2. TMS320C5515 DSP Starter Kit (DSK).

Program:

Main.c

```
#include "usbstk5515.h"
#include <math.h>
#include <stdio.h>
#define PTS 64 //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n); //FFT prototype
void apply_fft(void);
float iobuffer[PTS]; //as input and output buffer
float x1[PTS]; //intermediate buffer
float x[PTS];
short i; //general purpose index variable
short buffercount = 0; //number of new samples in iobuffer
short flag = 0; //set to 1 by ISR when iobuffer full
COMPLEX w[PTS]; //twiddle constants stored in w
COMPLEX samples[PTS]; //primary working buffer
void main(void)
{
float sum=0.0 ;
int n,k,i;
for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/(PTS*2.0)); /*Re component of twiddle constants*/
w[i].imag =-sin(2*PI*i/(PTS*2.0)); /*Im component of twiddle constants*/
}
/****************Input Signal X(n) *************************/
```

```
for(i=0;i<PTS;i++)
{
x[i] = sin(2*PI*5*i/PTS);
// Signal x(Fs)=sin(2*pi*f*i/Fs);
samples[i].real=0.0;
samples[i].imag=0.0;
}
/*********************Auto Correlation of X(n)=R(t) ***********/
for(n=0;n<PTS;n++)
{
sum=0;
for(k=0;k<PTS-n;k++)
{
sum=sum+(x[k]*x[n+k]); // Auto Correlation R(t)
}
iobuffer[n] = sum;
}
/********************** FFT of R(t) ************************/
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
/********************* PSD ********************/
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(samples[i].real*samples[i].real + samples[i].imag*samples[i].imag);
}
}
```

FFT.c

Refer previous FFT experiment

Code Flow:

• Step 1 - Select no. of points for FFT(E.g.: 64)

• Step 2 – Generate a sine wave of frequency ‘f ‘(e.g.: 10 Hz with a sampling rate = No. of Points of FFT (e.g. 64)) using math library function.

• Step 3 - Compute the Auto Correlation of Sine wave

• Step4 - Take output of auto correlation, apply FFT algorithm.

Execution Procedure:

_ Open CCstudio setup

_ Go to File Menu, select Import option.

_ In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.

_ In Select root Directory Browse for project file where it is located.

_ Select PSD Project folder and Finish it.

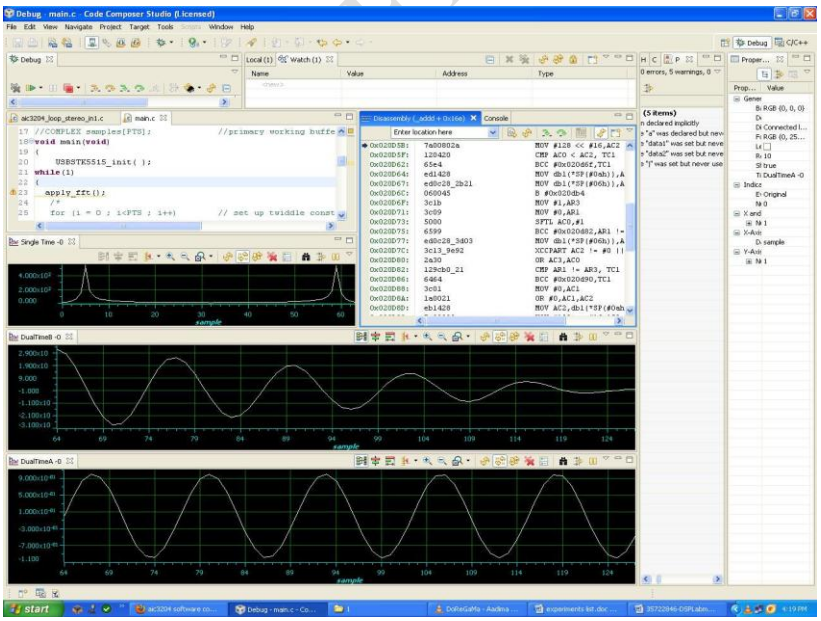
_ Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure from Step 46 to 51)

_ Give Right Click on Your psd.out file under Binaries and select Load program Option.

_ Now Go to Target select Run.

_ From Tools select Graph(Dual Time and single Time) , give properties and select OK.

Output:



EXP.NO: 17

IMPLEMENTATION OF LP FIR FILTER FOR GIVEN SEQUENCE & IMPLEMENTATION OF HP FIR FILTER FOR GIVEN SEQUENCE

Aim: The aim of this laboratory exercise is to design and implement a Digital FIR Filter & observe its frequency response. In this experiment we design a simple FIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components, which are outside the required band.

EQUIPMENTS:

- Host (PC) with windows(95/98/Me/XP/NT/2000).
- TMS320C5515 DSP Starter Kit (DSK).

Finite Impulse Response (FIR) Filter: The FIR filters are of non-recursive type, where by the present output sample is depending on the present input sample and previous input samples.

The transfer function of a FIR causal filter is given by,

$$H(z) = \sum_{n=0}^{N-1} h(n) Z^{-n}$$

Where h(n) is the impulse response of the filter.

The Fourier transform of h(n) is

$$H(e^{jw}) = \sum_{n=0}^{N-1} h(n) e^{-jwn}$$

In the design of FIR filters most commonly used approach is using windows.

The desired frequency response Hd(ejw) of a filter is periodic in frequency and can be expanded in Fourier series. The resultant series is given by,

$$h_d(n) = (1/2\pi) \int_{-\pi}^{\pi} H(e^{jw}) e^{jwn}\, dw$$

And known as Fourier coefficients having infinite length. One possible way of obtaining FIR filter is to truncate the infinite Fourier series at n = ± [(N-1)/2]

Where N is the length of the desired sequence.

The Fourier coefficients of the filter are modified by multiplying the infinite impulse response with a finite weighing sequence w(n) called a window.

Where w(n) = w(-n) ≠ 0 for |n| ≤ [(N-1)/2]

= 0 for |n| > [(N-1)/2]

After multiplying w(n) with hd(n), we get a finite duration sequence h(n) that satisfies the desired magnitude response,

h(n) = hd(n) w(n) for |n| ≤ [(N-1)/2]

= 0 for |n| > [(N-1)/2]

The frequency response H(ejw) of the filter can be obtained by convolution of Hd(ejw) and

W(ejw) is given by,

$$H(e^{jw}) = (1/2\pi)\int_{-\pi}^{\pi} H_d(e^{j\theta})e^{jw-\theta}\,d\theta$$

H(ejw) = Hd(ejw) * W(ejw).

DESIGNING AN FIR FILTER :

Following are the steps to design linear phase FIR filters Using Windowing Method.

I. Clearly specify the filter specifications.

Eg: Order= 30; Sampling Rate= 8000 samples/sec; Cut off Freq. = 400 Hz.

II. Compute the cut-off frequency Wc

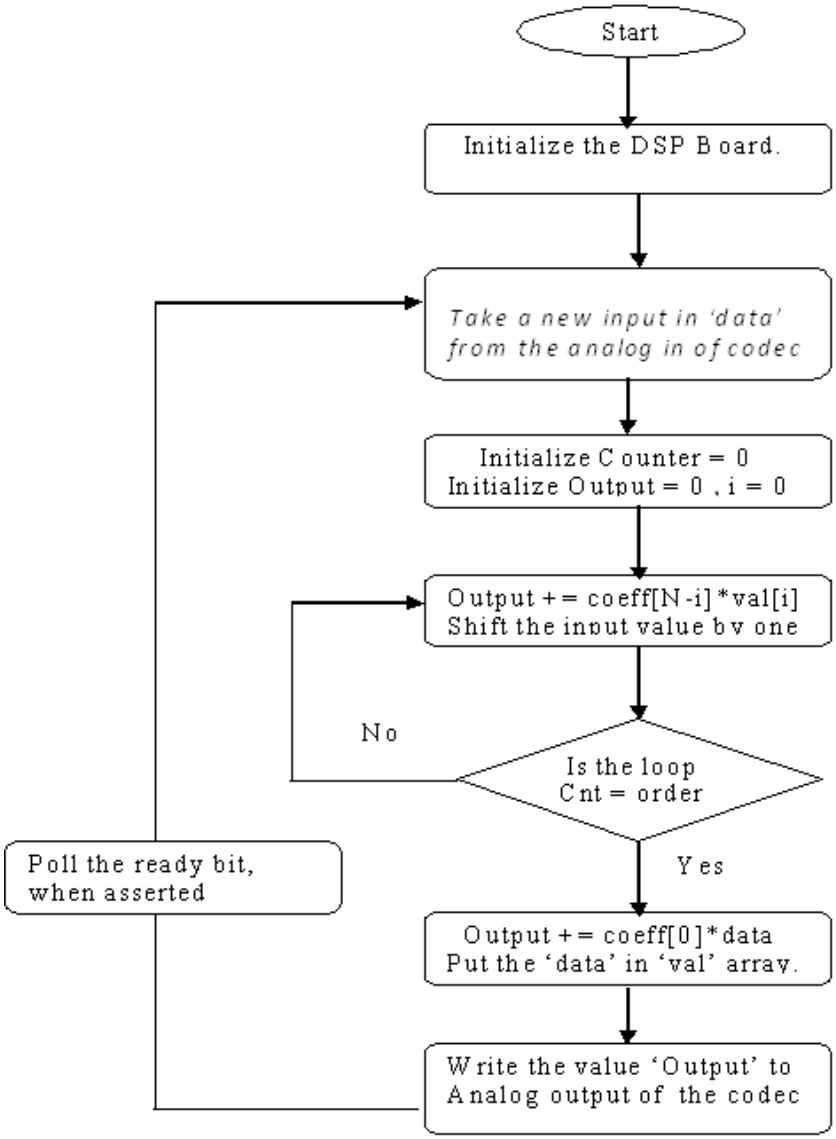
Eg: Wc = 2*pie* fc / Fs= 2*pie* 400/8000= 0.1*pie

III. Compute the desired Impulse Response h d (n) using particular Window.

Eg: b_rect1=fir1(order, Wc , 'high',boxcar(31));

IV. Convolve input sequence with truncated Impulse Response x (n)*h (n).

FLOW CHART TO IMPLEMENT FIR FILTER:



Coefficients for FIR Low Pass Kaiser filter:

Cutoff freq: 8khz, sampling freq: 24khz

#define N 82 //length of filter

short h[N]= { 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 2, -2, 0, 2, -2, 0, 2, -2, 0, 3, -3, 0, 4, -4, 0, 5, -6, 0, 10, -14, 0, 70, 70, 0, -14, 10, 0, -6, 5, 0, -4, 4, 0, -3, 3, 0, -2, 2, 0, -2, 2, 0, -2, 2, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0 };

Coefficients for FIR Low Pass rectangular filter:

Cutoff freq: 8khz, sampling freq: 24khz

#define N 82 //length of filter

short h[N]= { 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 2, -2, 0, 2, -2, 0, 2, -2, 0, 3, -3, 0, 4, -4, 0, 5, -6, 0, 10, -14, 0, 70, 70, 0, -14, 10, 0, -6, 5, 0, -4, 4, 0, -3, 3, 0, -2, 2, 0, -2, 2, 0, -2, 2, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0 };

Coefficients for FIR Low Pass triangular filter:

Cutoff freq: 8khz, sampling freq: 24khz

#define N 82 //length of filter

short h[N]= { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -2, 0, 2, -2, 0, 3, -3, 0, 5, -6, 0, 9, -13, 0, 70, 70, 0, -13, 9, 0, -6, 5, 0, -3, 3, 0, -2, 2, 0, -2, 1, 0, -1, 1, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

Coefficients for FIR high Pass Kaiser filter:

Cutoff freq: 4khz, sampling freq: 12khz

#define N 82 //length of filter

short h[N]= { 1, 0, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 2, -1, -1, 2, -1, -1, 2, -1, -1, 2, -1, -1, 3, -2, -2, 4, -2, -2, 5, -3, -4, 9, -6, -8, 27, -41, 41, -27, 8, 6, -9, 4, 3, -5, 2, 2, -4, 2, 2, -3, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 0, -1 };

Coefficients for FIR high Pass rectangular filter:

Cutoff freq: 4khz, sampling freq: 12khz

#define N 82 //length of filter

short h[N]= { 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 2, -1, -1, 2, -1, -1, 2, -1, -1, 2, -1, -1, 3, -2, -2, 4, -2, -2, 5, -3, -4, 9, -6, -8, 27, -41, 41, -27, 8, 6, -9, 4, 3, -5, 2, 2, -4, 2, 2, -3, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 1, -1 };

Coefficients for FIR high Pass triangular filter:

Cutoff freq: 4khz, sampling freq: 12khz

#define N 82 //length of filter

short h[N]= { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, -1, 1, -1, -1, 1, -1, -1, 2, -1, -1, 3, -2, -2, 5, -3, -3, 8, -5, -8, 27, -41, 41, -27, 8, 5, -8, 3, 3, -5, 2, 2, -3, 1, 1, -2, 1, 1, -1, 1, 1, -1, 1, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

C-Program:

Main.c

```
#include "stdio.h"
#include "usbstk5515.h"
void main( void )
{
/* Initialize BSL */
```

```
USBSTK5515_init();
printf( "playing audio ::::: \n" );
while(1)
{
aic3204_test( );
}
}
```

Aic3204_test.c:

```
#define AIC3204_I2C_ADDR 0x18
#include "usbstk5515.h"
#include "usbstk5515_gpio.h"
#include "usbstk5515_i2c.h"
#include "stdio.h"
extern Int16 aic3204_tone_headphone( );
extern Int16 aic3204_loop_stereo_in1( );
Int16 AIC3204_rget( Uint16 regnum, Uint16* regval )
{
Int16 retcode = 0;
Uint8 cmd[2];
cmd[0] = regnum & 0x007F; // 7-bit Register Address
cmd[1] = 0;
retcode |= USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
retcode |= USBSTK5515_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );
*regval = cmd[0];
USBSTK5515_wait( 10 );
return retcode;
}
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
{
Uint8 cmd[2];
cmd[0] = regnum & 0x007F; // 7-bit Register Address
cmd[1] = regval; // 8-bit Register Data
return USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}
```

```
Int16 aic3204_test( )
{
SYS_EXBUSSEL = 0x6100; // Enable I2C bus
USBSTK5515_I2C_init( ); // Initialize I2C
USBSTK5515_wait( 100 ); // Wait
if ( aic3204_loop_stereo_in1( ) )
return 1;
return 0;
}
```

aic3204_loop_stereo_in1.c:

```
#include "stdio.h"
#include "usbstk5515.h"
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20
#define N 82
Int16 data1, data2, data3, data4;
int sample,n,k,l;
Int16 dly0[N];
Int32 lyn,ryn;
/*//hpkaiser12-4
Int16 h[N]= {1, 0, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 2, -1, -1, 2, -1, -1, 2, -1, -1,
2, -1, -1, 3, -2, -2, 4, -2, -2, 5, -3, -4, 9, -6, -8, 27, -41, 41, -27, 8, 6, -9, 4, 3, -5, 2, 2, -4, 2,
2, -3, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 0, -1 };*/
/*//hprect12-4
Int16 h[N]= { 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 2, -1, -1, 2, -1, -1, 2, -1, -1,
2, -1, -1, 3, -2, -2, 4, -2, -2, 5, -3, -4, 9, -6, -8, 27, -41, 41, -27, 8, 6, -9, 4, 3, -5, 2, 2, -4, 2,
2, -3, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -2, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 1, -1, 1, 1, -1 };*/
/*//hptrig12-4
Int16 h[N]= { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, -1, 1, -1, -1, 1, -1, -1, 2, -
1, -1, 3, -2, -2, 5, -3, -3, 8, -5, -8, 27, -41, 41, -27, 8, 5, -8, 3, 3, -5, 2, 2, -3, 1, 1, -2, 1, 1, -
1, 1, 1,-1, 1, 0, -1, 0, 0, -1, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0};*/
/*//lpkaiser24-8
```

```
Int16 h[N]= {0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1,0, 1, -1, 0, 1, -1, 0, 2, -2, 0, 2, -2,0, 2, -2, 0,
3, -3, 0, 4, -4, 0, 5, -6, 0, 10, -14, 0, 70, 70, 0, -14, 10, 0, -6, 5, 0, -4, 4, 0, -3, 3, 0, -2, 2, 0,
-2, 2,0, -2, 2, 0, -1, 1, 0, -1, 1, 0, -1, 1,0, -1, 1, 0, -1, 1, 0, -1, 1, 0 };*/
/*//lprect24-8
short h[N]= {0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1,0, 1, -1, 0, 1, -1, 0, 2, -2, 0, 2, -2,0, 2, -2, 0,
3, -3, 0, 4, -4, 0, 5, -6,0, 10, -14, 0, 70, 70, 0, -14, 10, 0, -6, 5, 0, -4, 4, 0, -3, 3, 0, -2, 2, 0, -
2, 2,0, -2, 2, 0, -1, 1, 0, -1, 1, 0, -1, 1,0, -1, 1, 0, -1, 1, 0, -1, 1, 0 };*/
/*//lptrig24-8
Int16 h[N]= { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -2, 0, 2,
-2, 0, 3, -3, 0, 5, -6,0, 9, -13, 0, 70, 70, 0, -13, 9, 0, -6, 5,0, -3, 3, 0, -2, 2, 0, -2, 1, 0, -1, 1,
0, -1, 1, 0, -1, 1, 0, 0, 0, 0, 0, 0,0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};*/
//lpkaiser48-8
Int16 h[N]= {-1, 0, 1, 1, 1, -1, -1, -1, 1, 1, 1, -1,-1, -1, 1, 2, 1, -1, -2, -1, 1, 2, 1, -1, -2, -1,
1, 3, 2, -2, -4, -2, 2, 5, 3, -4,-9, -6, 8, 27, 41, 41, 27, 8, -6, -9, -4, 3,5, 2, -2, -4, -2, 2, 3, 1, -
1, -2, -1, 1,2, 1, -1, -2, -1, 1, 2, 1, -1, -1, -1, 1,1, 1, -1, -1, -1, 1, 1, 1, 0, -1};
Int16 aic3204_loop_stereo_in1( )
{
Int16 j, i = 0;
/* Configure AIC3204 */
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 1, 1 ); // Reset codec
AIC3204_rset( 0, 1 ); // Point to page 1
AIC3204_rset( 1, 8 ); // Disable crude AVDD generation from DVDD
AIC3204_rset( 2, 1 ); // Enable Analog Blocks, use LDO power
AIC3204_rset( 0, 0 ); // Select page 0
/* PLL and Clocks config and Power Up */
AIC3204_rset( 27, 0x0d ); // BCLK and WCLK is set as o/p to AIC3204(Master)
AIC3204_rset( 28, 0x00 ); // Data ofset = 0
AIC3204_rset( 4, 3 ); // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN <-PLL CLK
AIC3204_rset( 6, 8 ); // PLL setting: J=8
AIC3204_rset( 7, 15 ); // PLL setting: HI_BYTE(D)
AIC3204_rset( 8, 0xdc ); // PLL setting: LO_BYTE(D)
AIC3204_rset( 30, 0x88 ); // For 32 bit clocks per frame in Master mode ONLY
// BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz = 32*fs
```

```
AIC3204_rset( 5, 0x91 ); // PLL setting: Power up PLL, P=1 and R=1
AIC3204_rset( 13, 0 ); // Hi_Byte(DOSR) for DOSR = 128 decimal or 0x0080 DAC oversamppling
AIC3204_rset( 14, 0x80 ); // Lo_Byte(DOSR) for DOSR = 128 decimal or 0x0080
AIC3204_rset( 20, 0x80 ); // AOSR for AOSR = 128 decimal or 0x0080 for decimation filters 1 to 6
AIC3204_rset( 11, 0x88 ); // Power up NDAC and set NDAC value to 8
AIC3204_rset( 12, 0x82 ); // Power up MDAC and set MDAC value to 2
AIC3204_rset( 18, 0x88 ); // Power up NADC and set NADC value to 8
AIC3204_rset( 19, 0x82 ); // Power up MADC and set MADC value to 2
/* DAC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 12, 0x08 ); // LDAC AFIR routed to HPL
AIC3204_rset( 13, 0x08 ); // RDAC AFIR routed to HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 64, 0x02 ); // Left vol=right vol
AIC3204_rset( 65, 0x00 ); // Left DAC gain to 0dB VOL; Right tracks Left
AIC3204_rset( 63, 0xd4 ); // Power up left,right data paths and set channel
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 16, 0x06 ); // Unmute HPL , 6dB gain
AIC3204_rset( 17, 0x06 ); // Unmute HPR , 6dB gain
AIC3204_rset( 9, 0x30 ); // Power up HPL,HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
USBSTK5515_wait( 500 ); // Wait
/* ADC ROUTING and Power Up */
AIC3204_rset( 0, 1 ); // Select page 1
AIC3204_rset( 0x34, 0x30 ); // STEREO 1 Jack
// IN2_L to LADC_P through 40 kohm
AIC3204_rset( 0x37, 0x30 ); // IN2_R to RADC_P through 40 kohmm
AIC3204_rset( 0x36, 3 ); // CM_1 (common mode) to LADC_M through 40 kohm
AIC3204_rset( 0x39, 0xc0 ); // CM_1 (common mode) to RADC_M through 40 kohm
AIC3204_rset( 0x3b, 0 ); // MIC_PGA_L unmute
AIC3204_rset( 0x3c, 0 ); // MIC_PGA_R unmute
AIC3204_rset( 0, 0 ); // Select page 0
```

```
AIC3204_rset( 0x51, 0xc0 ); // Powerup Left and Right ADC
AIC3204_rset( 0x52, 0 ); // Unmute Left and Right ADC
AIC3204_rset( 0, 0 );
USBSTK5515_wait( 200 ); // Wait
/* I2S settings */
I2S0_SRGR = 0x0;
I2S0_CR = 0x8010; // 16-bit word, slave, enable I2C
I2S0_ICMR = 0x3f; // Enable interrupts
/* Play Tone */
for(l=0;l<N;l++)
dly0[l]=0;
for ( i = 0 ; i < 5 ; i++ )
{
for ( j = 0 ; j < 1000 ; j++ )
{
for ( sample = 0 ; sample < N ; sample++ )
{
/* Read Digital audio input */
data3 = I2S0_W0_MSW_R; // 16 bit left channel received audio data
data1 = I2S0_W0_LSW_R;
data4 = I2S0_W1_MSW_R; // 16 bit right channel received audio data
data2 = I2S0_W1_LSW_R;
while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag
lyn=0;
for(k=0;k<N;k++)
{
dly0[(N-1)-k]=dly0[(N-1)-k-1];
dly0[0]=data3;
}
for(k=0;k<N;k++)
lyn+=((h[k])*dly0[k]); //fir low pass filter
data1=lyn>>1;
data2=data1;
/* Write Digital audio input */
```

```
I2S0_W0_MSW_W = data1; // 16 bit left channel transmit audio data
I2S0_W0_LSW_W = 0;
I2S0_W1_MSW_W = data2; // 16 bit right channel transmit audio data
I2S0_W1_LSW_W = 0;
while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag
}
}
}
/* Disble I2S */
I2S0_CR = 0x00;
return 0;
}
```

Execution Procedure:

- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select FIR Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure from Step 46 to 51)
- Give Right Click on Your fir.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- Observe the audio (Give Input from Stereo in and listen Filtered Output from Stereo Out).

EXP.NO: 18

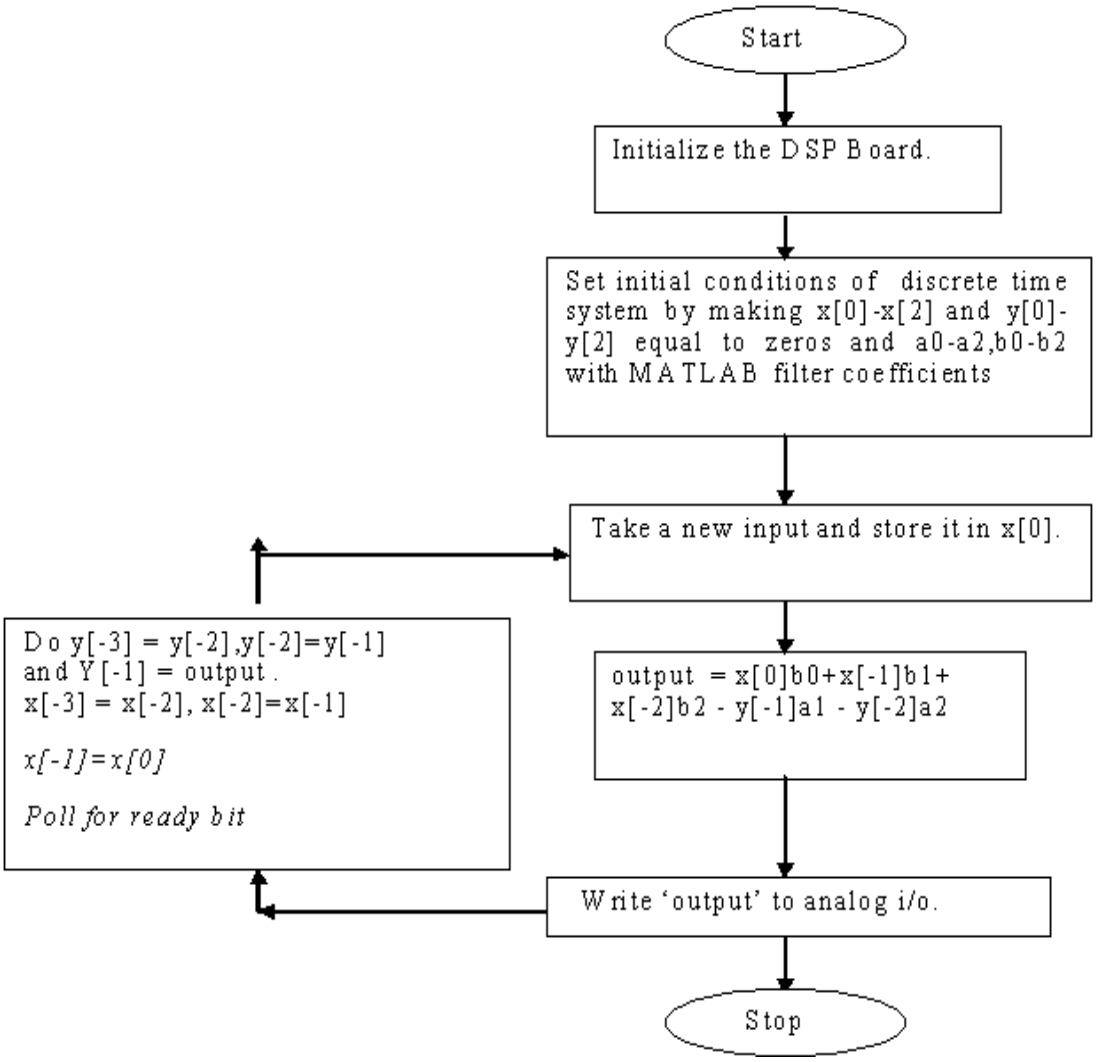
IMPLEMENTATION OF LP IIR FILTER FOR GIVEN SEQUENCE & IMPLEMENTATION OF HP IIR FILTER FOR GIVEN SEQUENCE

Aim: The aim of this laboratory exercise is to design and implement a Digital IIR Filter & observe its frequency response. In this experiment we design a simple IIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components which are outside the required band

EQUIPMENTS NEEDED:

- Host (PC) with windows(95/98/Me/XP/NT/2000).
- TMS320C5515 DSP Starter Kit (DSK).
- Audio Jack Cable

FLOWCHART FOR IIR IMPLEMENTATION:



IIR high pass filter coefficients:

#define N 83 //length of filter

Short h[N]={1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1, 3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1, 1,3,1,3,1,3,1,3,1,2,1};

IIR low pass filter coefficients:

#define N 83 //length of filter

Shorth[N]={1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1, 3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,2,1 };

c-Program:

main.c

```
#include "stdio.h"
#include "usbstk5515.h"
void main( void )
{
/* Initialize BSL */
USBSTK5515_init();
printf( "playing audio ::::: \n" );
while(1)
{
aic3204_test( );
}
}
```

Aic3204_test.c:

```
#define AIC3204_I2C_ADDR 0x18
#include "usbstk5515.h"
#include "usbstk5515_gpio.h"
#include "usbstk5515_i2c.h"
#include "stdio.h"
extern Int16 aic3204_tone_headphone( );
extern Int16 aic3204_loop_stereo_in1( );
Int16 AIC3204_rget( Uint16 regnum, Uint16* regval )
{
Int16 retcode = 0;
Uint8 cmd[2];
cmd[0] = regnum & 0x007F; // 7-bit Register Address
cmd[1] = 0;
```

```
retcode |= USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
retcode |= USBSTK5515_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );
*regval = cmd[0];
USBSTK5515_wait( 10 );
return retcode;
}
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
{
Uint8 cmd[2];
cmd[0] = regnum & 0x007F; // 7-bit Register Address
cmd[1] = regval; // 8-bit Register Data
return USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}
Int16 aic3204_test( )
{
SYS_EXBUSSEL = 0x6100; // Enable I2C bus
USBSTK5515_I2C_init( ); // Initialize I2C
USBSTK5515_wait( 100 ); // Wait
if ( aic3204_loop_stereo_in1( ) )
return 1;
return 0;
}
```

aic3204_loop_stereo_in1.c:

```
#include "stdio.h"
#include "usbstk5515.h"
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20
#define N 83
Int16 data1, data2, data3, data4;
int sample, n, k, l;
Int16 dly0[N];
Int16 dly1[N];
Int32 lyn,lyn0,lyn1;
```

```
Int16 h[N]={1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,
3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,3,1,2,1
};
Int16 aic3204_loop_stereo_in1 ( )
{
/* Pre-generated sine wave data, 16-bit signed samples */
Int16 j, i = 0;
/* Configure AIC3204 */
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 1, 1 ); // Reset codec
AIC3204_rset( 0, 1 ); // Point to page 1
AIC3204_rset( 1, 8 ); // Disable crude AVDD generation from DVDD
AIC3204_rset( 2, 1 ); // Enable Analog Blocks, use LDO power
AIC3204_rset( 0, 0 ); // Select page 0
/* PLL and Clocks config and Power Up */
AIC3204_rset( 27, 0x0d ); // BCLK and WCLK is set as o/p to AIC3204(Master)
AIC3204_rset( 28, 0x00 ); // Data ofset = 0
AIC3204_rset( 4, 3 ); // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN <-PLL
CLK
AIC3204_rset( 6, 8 ); // PLL setting: J=8
AIC3204_rset( 7, 15 ); // PLL setting: HI_BYTE(D)
AIC3204_rset( 8, 0xdc ); // PLL setting: LO_BYTE(D)
AIC3204_rset( 30, 0x88 ); // For 32 bit clocks per frame in Master mode ONLY
// BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz = 32*fs
AIC3204_rset( 5, 0x91 ); // PLL setting: Power up PLL, P=1 and R=1
AIC3204_rset( 13, 0 ); // Hi_Byte(DOSR) for DOSR = 128 decimal or 0x0080 DAC
oversamppling
AIC3204_rset( 14, 0x80 ); // Lo_Byte(DOSR) for DOSR = 128 decimal or 0x0080
AIC3204_rset( 20, 0x80 ); // AOSR for AOSR = 128 decimal or 0x0080 for decimation
filters 1 to 6
AIC3204_rset( 11, 0x88 ); // Power up NDAC and set NDAC value to 8
AIC3204_rset( 12, 0x82 ); // Power up MDAC and set MDAC value to 2
AIC3204_rset( 18, 0x88 ); // Power up NADC and set NADC value to 8
AIC3204_rset( 19, 0x82 ); // Power up MADC and set MADC value to 2
```

```
/* DAC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 12, 0x08 ); // LDAC AFIR routed to HPL
AIC3204_rset( 13, 0x08 ); // RDAC AFIR routed to HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 64, 0x02 ); // Left vol=right vol
AIC3204_rset( 65, 0x00 ); // Left DAC gain to 0dB VOL; Right tracks Left
AIC3204_rset( 63, 0xd4 ); // Power up left,right data paths and set channel
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 16, 0x06 ); // Unmute HPL , 6dB gain
AIC3204_rset( 17, 0x06 ); // Unmute HPR , 6dB gain
AIC3204_rset( 9, 0x30 ); // Power up HPL,HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
USBSTK5515_wait( 500 ); // Wait
/* ADC ROUTING and Power Up */
AIC3204_rset( 0, 1 ); // Select page 1
AIC3204_rset( 0x34, 0x30 ); // STEREO 1 Jack
// IN2_L to LADC_P through 40 kohm
AIC3204_rset( 0x37, 0x30 ); // IN2_R to RADC_P through 40 kohmm
AIC3204_rset( 0x36, 3 ); // CM_1 (common mode) to LADC_M through 40 kohm
AIC3204_rset( 0x39, 0xc0 ); // CM_1 (common mode) to RADC_M through 40 kohm
AIC3204_rset( 0x3b, 0 ); // MIC_PGA_L unmute
AIC3204_rset( 0x3c, 0 ); // MIC_PGA_R unmute
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 0x51, 0xc0 ); // Powerup Left and Right ADC
AIC3204_rset( 0x52, 0 ); // Unmute Left and Right ADC
AIC3204_rset( 0, 0 );
USBSTK5515_wait( 200 ); // Wait
/* I2S settings */
I2S0_SRGR = 0x0;
I2S0_CR = 0x8010; // 16-bit word, slave, enable I2C
I2S0_ICMR = 0x3f; // Enable interrupts
/* Play Tone */
for(l=0;l<N;l++)
```

```
{
dly0[l]=0;
dly1[l]=0;
}
for ( i = 0 ; i < 5 ; i++ )
{
for ( j = 0 ; j < 1000 ; j++ )
{
for ( sample = 0 ; sample < N ; sample++ )
{
/* Read Digital audio input */
data3 = I2S0_W0_MSW_R; // 16 bit left channel received audio data
data1 = I2S0_W0_LSW_R;
data4 = I2S0_W1_MSW_R; // 16 bit right channel received audio data
data2 = I2S0_W1_LSW_R;
while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag
dly0[sample]=data3;
lyn0=0;
lyn1=0;
for(k=0;k<=sample;k++)
lyn0+=((h[k])*dly0[sample-k]); //fir low pass filter
for(k=1;k<=sample;k++)
lyn1+=((h[k])*dly1[sample-k]); //fir low pass filter
lyn=lyn0-lyn1;
dly1[sample]=lyn;
data1=lyn<<1;
data2=lyn<<1;
/* Write Digital audio input */
I2S0_W0_MSW_W = data1; // 16 bit left channel transmit audio data
I2S0_W0_LSW_W = 0;
I2S0_W1_MSW_W = data2; // 16 bit right channel transmit audio data
I2S0_W1_LSW_W = 0;
while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag
}
```

```
}
}
/* Disble I2S */
I2S0_CR = 0x00;
return 0;
}
```

Execution Procedure:

- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select FIR Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure from Step 46 to 51)
- Give Right Click on Your fir.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- Observe the audio (Give Input from Stereo in and listen Filtered Output from Stereo Out).

EXP.NO: 19
GENERATION OF SINUSOIDAL SIGNAL THROUGH FILTERING

Program:

Main.c

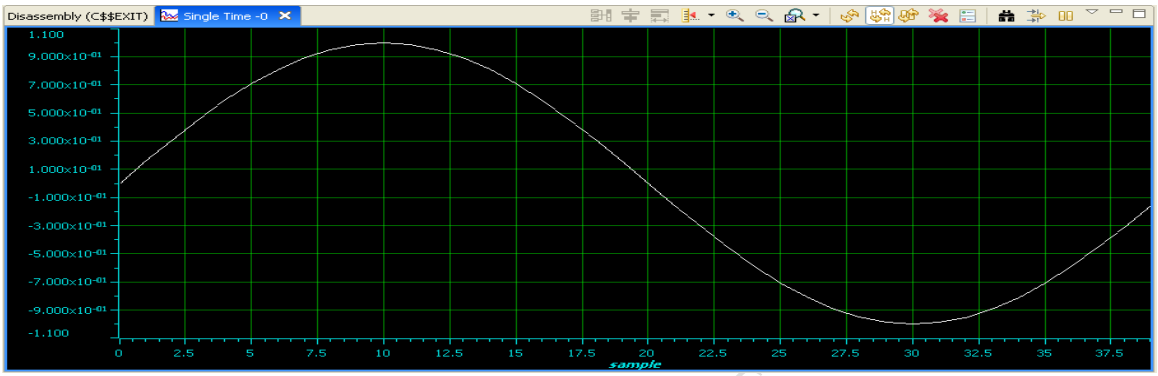
```
#include <usbstk5515.h>
#include<stdio.h>
#include <math.h>
#define PTS 40 //no of points for FFT
#define PI 3.14159265358979
float output;
float y[40];
void main()
{
int n;
for (n=0; n<PTS; n++)
{
/*y[n]=A•y[n–1]+B•y[n–2] A=1.9754 and B=–1. Examining the behavior
of this IIR filter by its transfer function as below:
//y[n] = 1.9754•y[n–1] – y[n–2] */
if(n<2)
y[n]=sin(2*PI*n/PTS);
else
y[n]=(1.9754*y[n-1])-(y[n-2]);
printf("%f,",y[n]);
}
}
```

Execution Procedure:

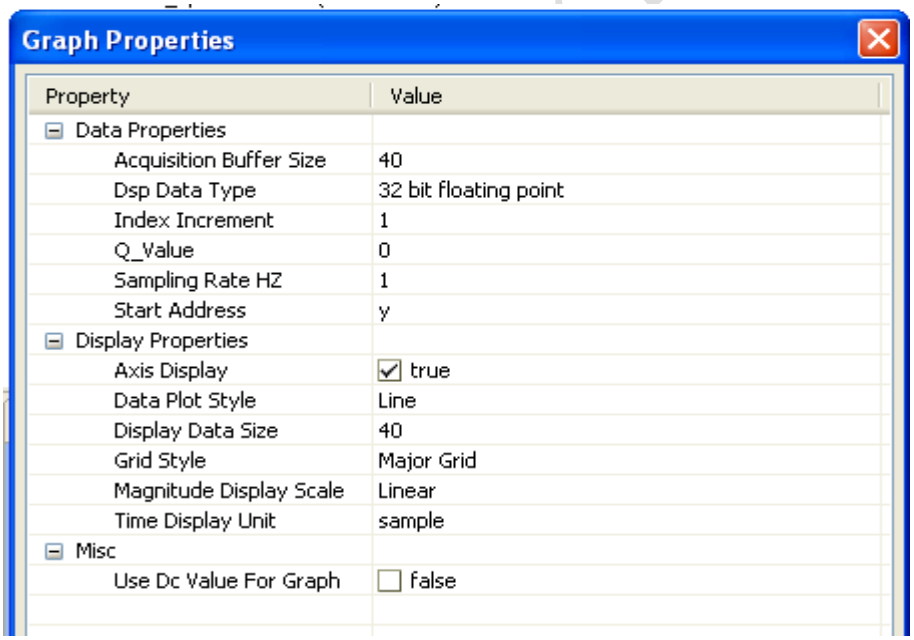
- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then
- next.
- In Select root Directory Browse for project file where it is located.
- Select sinusoidal through Filtering Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual

- procedure from Step 46 to 51)
- Give Right Click on Your sin.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- In Tools menu select Graph (single Time) set the properties and watch.

Result:



Graph Proeprties:



EXP.NO: 20
GENERATION OF DTMF SIGNALS

Program:

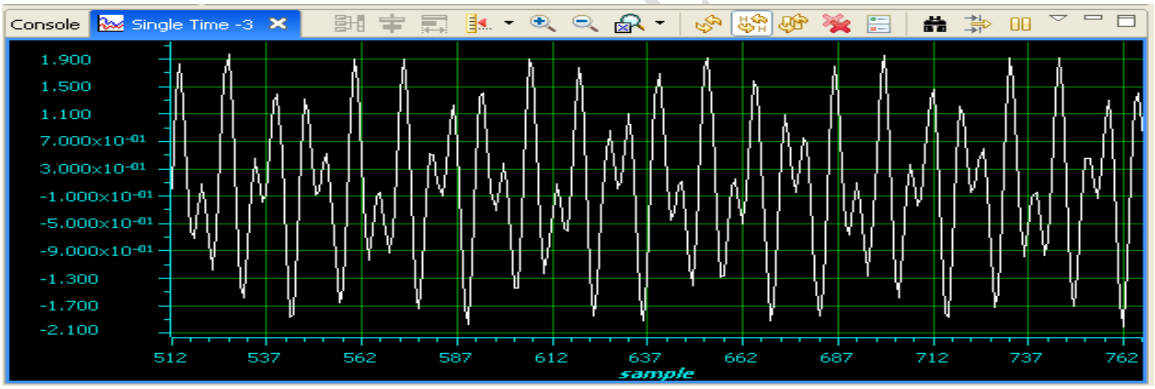
```
#include <usbstk5515.h>
#include<stdio.h>
#include <math.h>
#define PTS 256
#define FS 8000 //sampling frequency of 8khz
#define pi 3.14159265358979
int lfg[ ]={697,770,852,941}; // Low frequency group
int hfg[ ]={1209,1336,1477}; // High frequency group
float num[12][PTS];
float t[PTS];
void main()
{
int n,tone_select=0,r=0,c=0;
printf("DTMF Signal Generation:");
for(n=0;n<PTS;n++) {
t[n] = 2*pi*n/FS; }
tone_select=0;
for(r=0;r<4;r++) //loop for selection lfg index
{
for(c=0;c<3;c++) //loop for selection hfg index
{
for(n=0;n<PTS;n++)
{
num[tone_select][n]=sin(lfg[r]*t[n])+sin(hfg[c]*t[n]);
//printf("%d %f \n",n,t[n],num[tone_select][n]);
}
tone_select++;
}
}
printf("Done");
}
```

Execution Procedure:

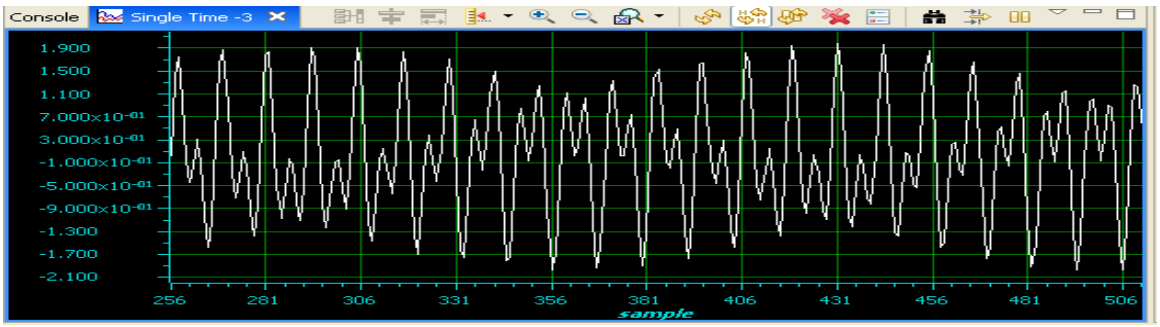
- Open CCstudio setup
- Go to File Menu, select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select DTMF Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it. (Follow the above given manual procedure from Step 46 to 51).
- Give Right Click on Your dtmf.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- In Tools menu select Graph (single Time) set the properties and watch.

Result:

DTMF 1:



DTMF 2:



EXP.NO: 21
IMPLEMENTATION OF DECIMATION PROCESS

Program:

```
#include <usbstk5515.h>
#include<stdio.h>
#include <math.h>
#define PTS 256 //no of points for FFT
#define PI 3.14159265358979
float y1[PTS],y2[PTS],y3[PTS];
main()
{
int n,m;
printf("Enter Vaue for Decimation Factor\n");
scanf("%d",&m);
printf("Original signal samples:\n");
for(n=0;n<=PTS;n++) //original signal
{
y1[n]=sin(2*PI*10*(n)/(PTS));
printf("%d, %f\n",n,y1[n]);
}
printf("Decimated signal samples:\n");
for(n=0;n<=PTS;n++) //intrpol
{
y2[n]=sin(2*PI*10*n/(m*PTS));
printf("%d, %f\n",n,y2[n]);
}
} //end of main
```

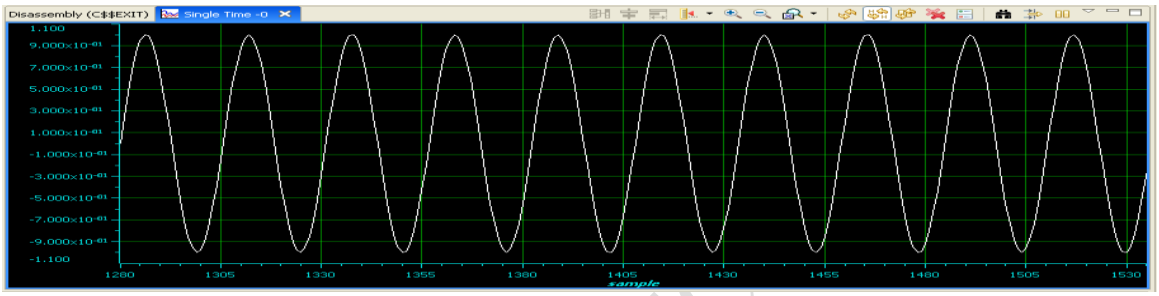
Execution Procedure:

- Open CCstudio setup
- Go to File Menu, select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select Decimation Project folder and Finish it.

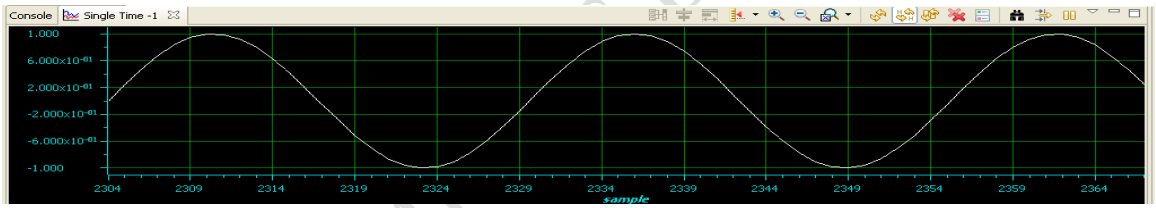
- Now Connect the DSP Kit to PC and Launch it. (Follow the above given manual procedure from Step 46 to 51).
- Give Right Click on Your decimation.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- In Tools menu select Graph (Dual Time) set the properties and watch input signal and decimated signal.

Result:

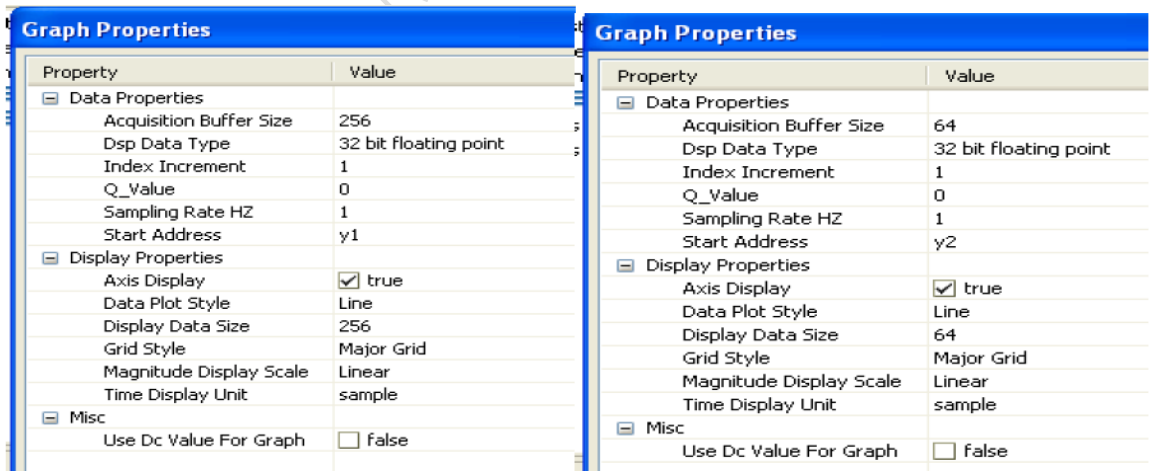
Input signal:



Output Signal:



Graph Properties:



EXP.NO: 22

IMPLEMENTATION OF INTERPOLATION PROCESS

Program:

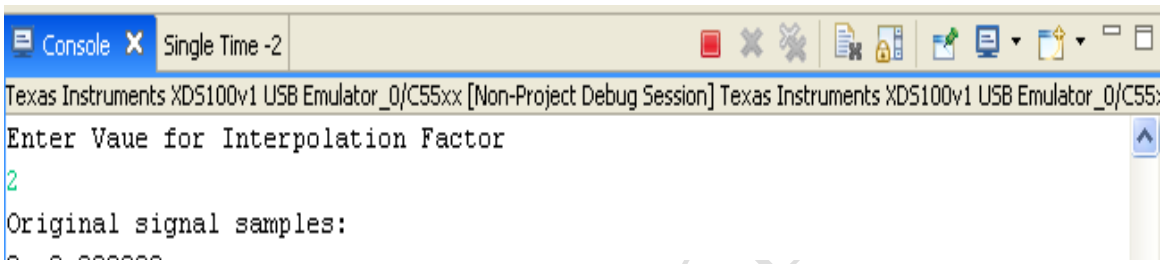
```
#include <usbstk5515.h>
#include<stdio.h>
#include <math.h>
#define PTS 256 //no of points for FFT
#define PI 3.14159265358979
float y1[PTS],y2[PTS],y3[PTS];
main()
{
int n,m;
printf("Enter Vaue for Interpolation Factor\n");
scanf("%d",&m);
printf("Original signal samples:\n");
for(n=0;n<=PTS;n++) //original signal
{
y1[n]=sin(2*PI*10*(n)/(PTS));
printf("%d, %f\n",n,y1[n]);
}
printf("Interpolated signal samples:\n");
for(n=0;n<=PTS;n++) //intrpol
{
y2[n]=sin(2*PI*10*(m*n)/(PTS));
printf("%d, %f\n",n,y2[n]);
}
}
```


Execution Procedure:

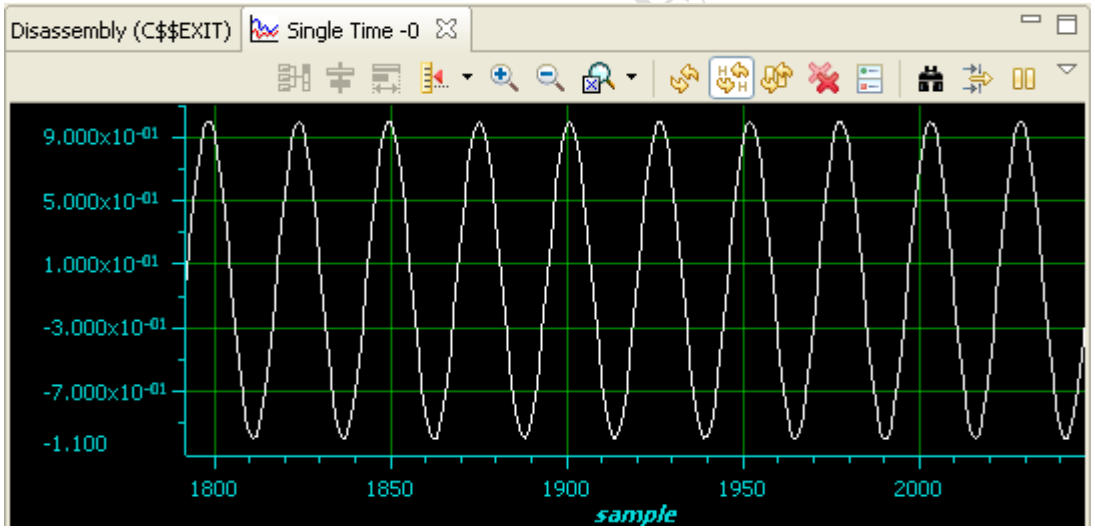
- Open CCstudio setup
- Go to File Menu, select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.

- Select Interpolation Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it. (Follow the above given manual procedure from Step 46 to 51).
- Give Right Click on Your interpolation.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- In Tools menu select Graph (Dual Time) set the properties and watch input signal and decimated signal.

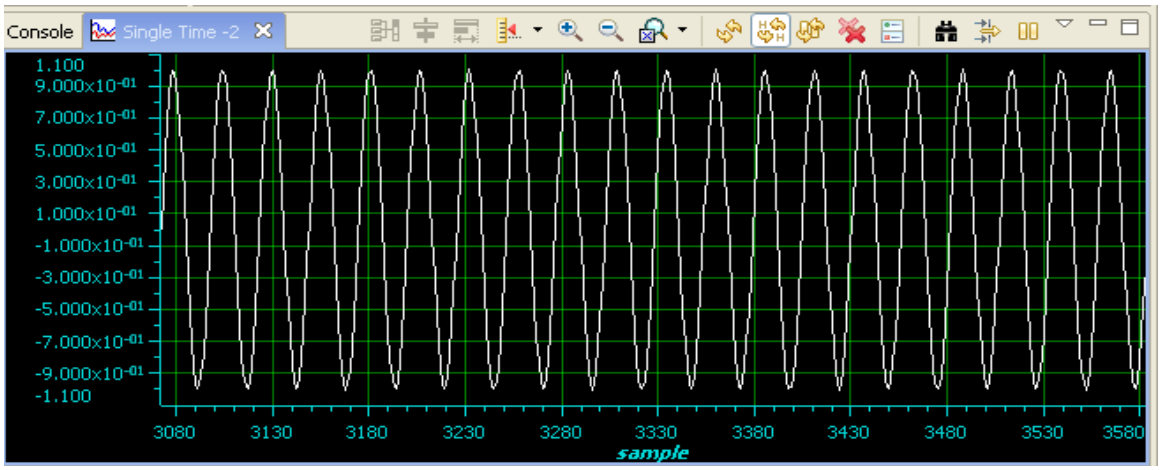
Result :



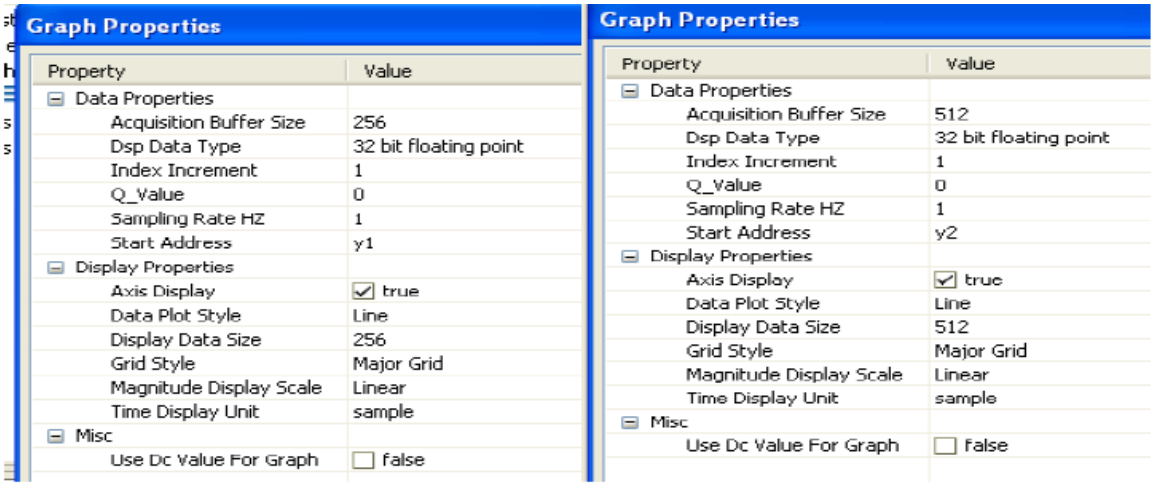
Input Signal:



Output Signal:



Graph Properties:



Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	256
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	y1
Display Properties	
Axis Display	true
Data Plot Style	Line
Display Data Size	256
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	false

Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	512
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	y2
Display Properties	
Axis Display	true
Data Plot Style	Line
Display Data Size	512
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	false

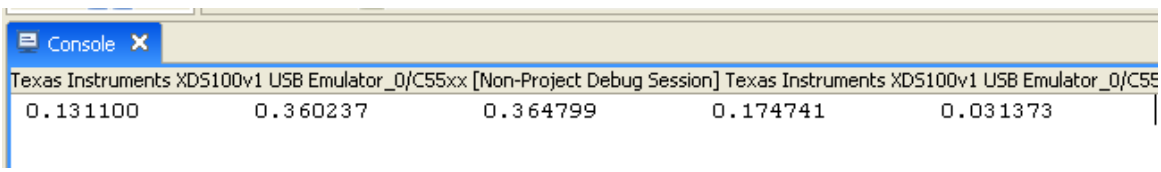
EXP.NO: 23
IMPULSE RESPONSE OF FIRST ORDER AND SECOND ORDER SYSTEMS

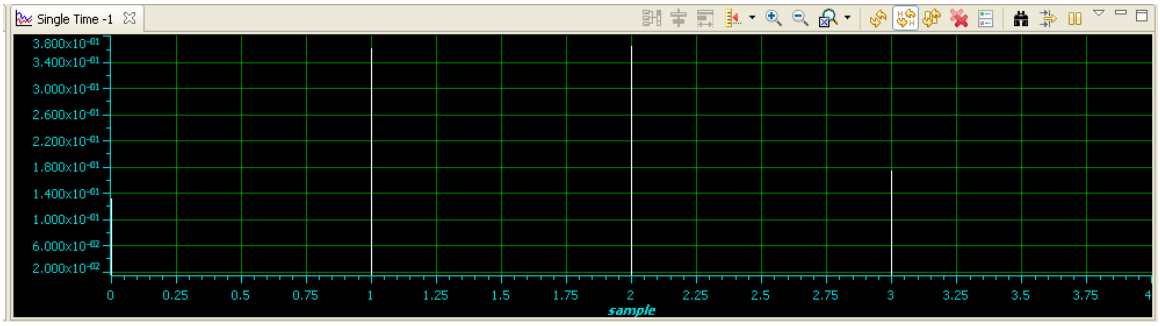
Program:

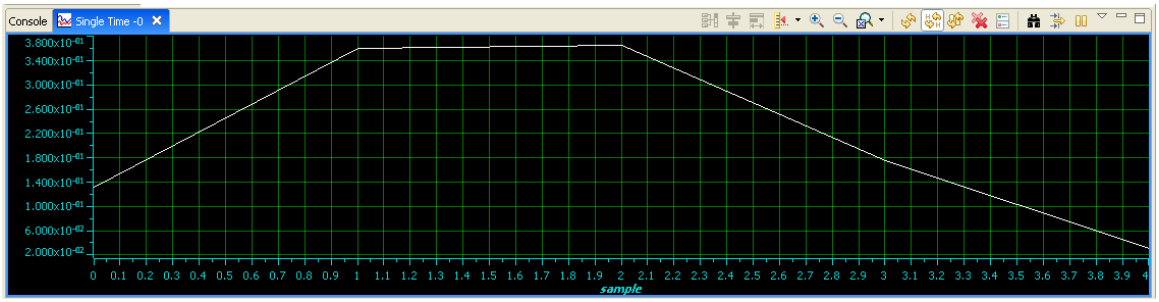
```
#include<stdio.h>
#define Order 2
#define Len 5
float h[Len] = {0.0,0.0,0.0,0.0,0.0},sum;
void main()
{
int j, k;
float a[Order+1] = {0.1311, 0.2622, 0.1311};
float b[Order+1] = {1, -0.7478, 0.2722};
for(j=0; j<Len; j++)
{
sum = 0.0;
for(k=1; k<=Order; k++)
{
if ((j-k) >= 0)
sum = sum+(b[k]*h[j-k]);
}
if (j <= Order)
h[j] = a[j]-sum;
else
h[j] = -sum;
printf (" %f ",h[j]);
}
}
```


Output:

0.131100 0.360237 0.364799 0.174741 0.031373







Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	5
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	h
Display Properties	
Axis Display	true
Data Plot Style	Bar
Display Data Size	5
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	false

Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	5
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	h
Display Properties	
Axis Display	true
Data Plot Style	Line
Display Data Size	5
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	false

FOR FIRST ORDER DIFFERENCE EQUATION.

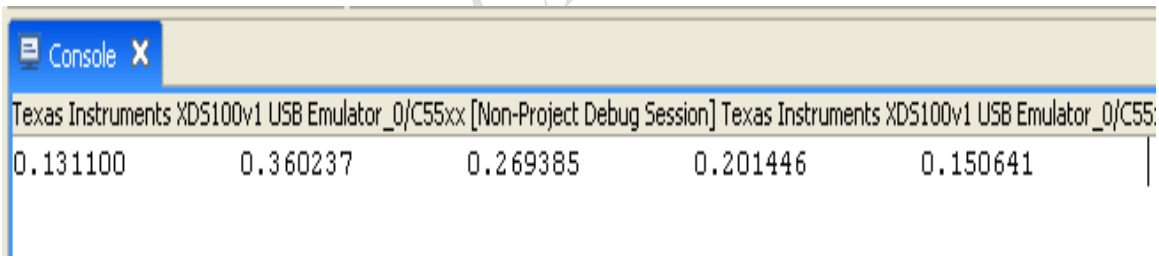
Program:

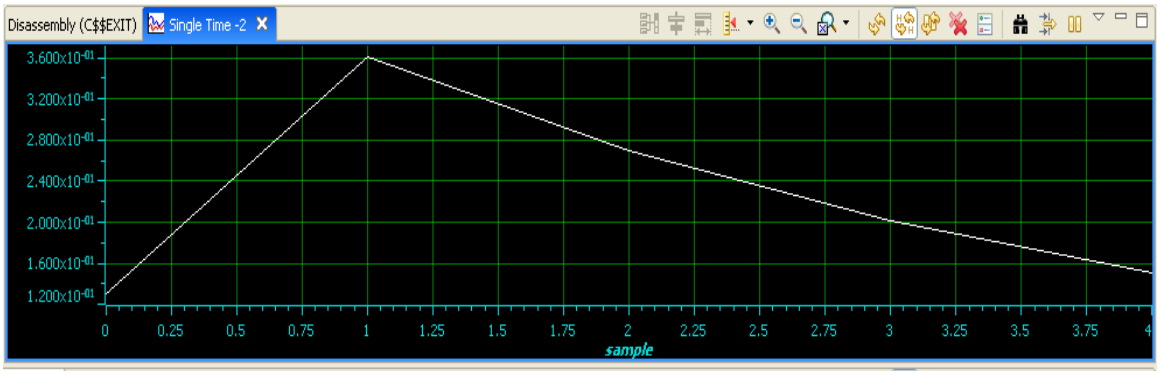
```
#include<stdio.h>
#define Order 1
#define Len 5
float h[Len] = {0.0,0.0,0.0,0.0,0.0},sum;
void main()
{
int j, k;
float a[Order+1] = {0.1311, 0.2622};
```

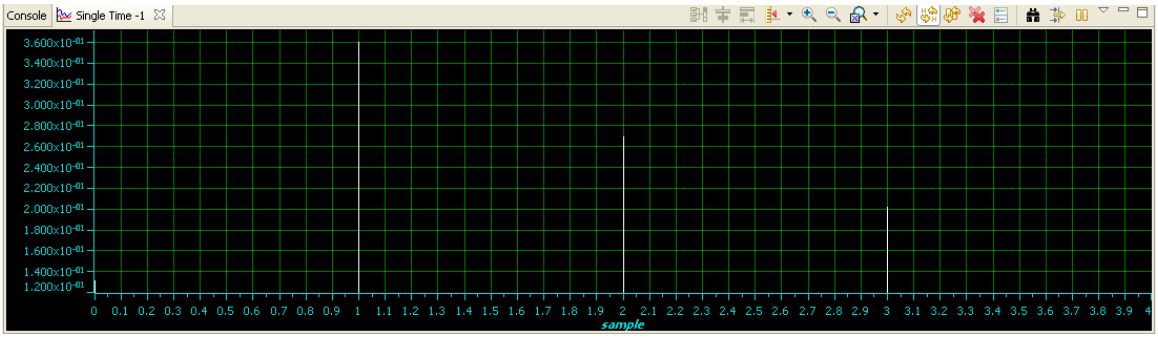
```
float b[Order+1] = {1, -0.7478};
for(j=0; j<Len; j++)
{
sum = 0.0;
for(k=1; k<=Order; k++)
{
if((j-k)>=0)
sum = sum+(b[k]*h[j-k]);
}
if(j<=Order)
h[j] = a[j]-sum;
else
h[j] = -sum;
printf("%f ", j, h[j]);
}
}
```

Output:

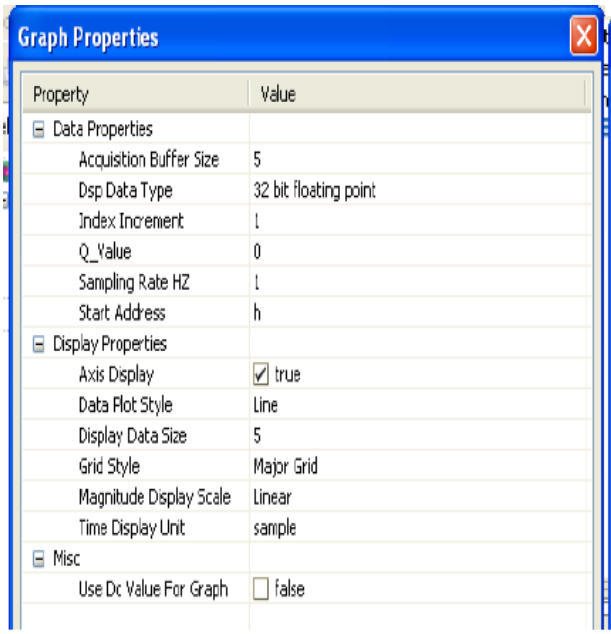
0.131100 0.360237 0.269385 0.201446 0.150641

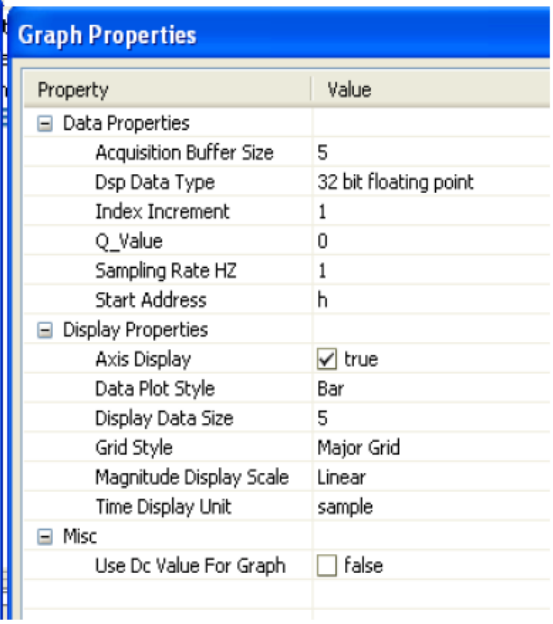






Graph Properties:





EXP. NO 24
AUDIO APPLICATIONS

Aim: Audio applications such as to plot a time and frequency display of microphone plus a cosine using DSP. Read a wav file and match with their respective spectrograms.

Program:

```
#include <math.h>
#include <stdio.h>
#include "usbstk5515.h"
#define PTS 64
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n);
void cosine_plot(void);
Int16 aic3204_test( );
extern Int16 dly0[PTS];
COMPLEX data[PTS],samples[PTS];
COMPLEX w[PTS];
#define PI 3.14159265358979
float x1[PTS],y1[PTS],iobuffer[PTS];
int i;
void main( void )
{
/* Initialize BSL */
USBSTK5515_init();
for (i = 0;i<PTS;i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/(PTS*2.0)); //Re component of twiddle constants
w[i].imag =-sin(2*PI*i/(PTS*2.0)); //Im component of twiddle constants
}
printf( "Cosine Plot ::::: \n" );
cosine_plot();
printf( "playing audio ::::: \n" );
while(1)
{
```

```
aic3204_test( );
//for(i=0;i<PTS;i++)
// printf( "%d \n",dly0[i]);
for(i=0;i<PTS;i++)
{
data[i].real = dly0[i];
data[i].imag = 0.0;
}
FFT(data,PTS);
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(data[i].real*data[i].real + data[i].imag*data[i].imag);
printf("\n%d = %f",i,x1[i]);
}
}
}
void cosine_plot()
{
int i;
for (i = 0 ; i < PTS ; i++) //swap buffers
{
iobuffer[i] = cos(2*PI*2*i/64.0);/*10- > freq, 64 -> sampling freq*/
printf("\n%d = %f",i,iobuffer[i]);
samples[i].real=0.0;
samples[i].imag=0.0;
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
printf("\n output");
```

```
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
y1[i] = sqrt(samples[i].real*samples[i].real + samples[i].imag*samples[i].imag);
printf("\n%d = %f",i,x1[i]);
}
}
```

aic3204_loop_stereo_in1.C

```
#include "stdio.h"
#include "usbstk5515.h"
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20
#define N 64
Int16 data1, data2, data3, data4;
int sample,n,k,l;
Int16 dly0[N];
Int32 lyn,ryn;
Int16 aic3204_loop_stereo_in1( )
{
// Int16 j, i = 0;
/* Configure AIC3204 */
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 1, 1 ); // Reset codec
AIC3204_rset( 0, 1 ); // Point to page 1
AIC3204_rset( 1, 8 ); // Disable crude AVDD generation from DVDD
AIC3204_rset( 2, 1 ); // Enable Analog Blocks, use LDO power
AIC3204_rset( 0, 0 ); // Select page 0
/* PLL and Clocks config and Power Up */
AIC3204_rset( 27, 0x0d ); // BCLK and WCLK is set as o/p to AIC3204(Master)
AIC3204_rset( 28, 0x00 ); // Data ofset = 0
AIC3204_rset( 4, 3 ); // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN <-PLL CLK
AIC3204_rset( 6, 8 ); // PLL setting: J=8
AIC3204_rset( 7, 15 ); // PLL setting: HI_BYTE(D)
AIC3204_rset( 8, 0xdc ); // PLL setting: LO_BYTE(D)
```

```
AIC3204_rset( 30, 0x88 ); // For 32 bit clocks per frame in Master mode ONLY
// BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz = 32*fs
AIC3204_rset( 5, 0x91 ); // PLL setting: Power up PLL, P=1 and R=1
AIC3204_rset( 13, 0 );// Hi_Byte(DOSR) for DOSR = 128 decimal or 0x0080 DAC
oversamppling
AIC3204_rset( 14, 0x80 ); // Lo_Byte(DOSR) for DOSR = 128 decimal or 0x0080
AIC3204_rset( 20, 0x80 ); // AOSR for AOSR = 128 decimal or 0x0080 for decimation
filters 1 to 6
AIC3204_rset( 11, 0x88 ); // Power up NDAC and set NDAC value to 8
AIC3204_rset( 12, 0x82 ); // Power up MDAC and set MDAC value to 2
AIC3204_rset( 18, 0x88 ); // Power up NADC and set NADC value to 8
AIC3204_rset( 19, 0x82 ); // Power up MADC and set MADC value to 2
/* DAC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 12, 0x08 ); // LDAC AFIR routed to HPL
AIC3204_rset( 13, 0x08 ); // RDAC AFIR routed to HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 64, 0x02 ); // Left vol=right vol
AIC3204_rset( 65, 0x00 ); // Left DAC gain to 0dB VOL; Right tracks Left
AIC3204_rset( 63, 0xd4 ); // Power up left,right data paths and set channel
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 16, 0x06 ); // Unmute HPL , 6dB gain
AIC3204_rset( 17, 0x06 ); // Unmute HPR , 6dB gain
AIC3204_rset( 9, 0x30 ); // Power up HPL,HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
USBSTK5515_wait( 500 ); // Wait
/* ADC ROUTING and Power Up */
AIC3204_rset( 0, 1 ); // Select page 1
AIC3204_rset( 0x34, 0x30 ); // STEREO 1 Jack
// IN2_L to LADC_P through 40 kohm
AIC3204_rset( 0x37, 0x30 ); // IN2_R to RADC_P through 40 kohmm
AIC3204_rset( 0x36, 3 ); // CM_1 (common mode) to LADC_M through 40 kohm
AIC3204_rset( 0x39, 0xc0 ); // CM_1 (common mode) to RADC_M through 40 kohm
AIC3204_rset( 0x3b, 0 ); // MIC_PGA_L unmute
```

```
AIC3204_rset( 0x3c, 0 ); // MIC_PGA_R unmute
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 0x51, 0xc0 ); // Powerup Left and Right ADC
AIC3204_rset( 0x52, 0 ); // Unmute Left and Right ADC
AIC3204_rset( 0, 0 );
USBSTK5515_wait( 200 ); // Wait
/* I2S settings */
I2S0_SRGR = 0x0;
I2S0_CR = 0x8010; // 16-bit word, slave, enable I2C
I2S0_ICMR = 0x3f; // Enable interrupts
/* Play Tone */
for(l=0;l<N;l++)
dly0[l]=0;
for ( sample = 0; sample < N ; sample++ )
{
USBSTK5515_waitusec(25);
/* Read Digital audio input */
data3 = I2S0_W0_MSW_R; // 16 bit left channel received audio data
data1 = I2S0_W0_LSW_R;
data4 = I2S0_W1_MSW_R; // 16 bit right channel received audio data
data2 = I2S0_W1_LSW_R;
while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag
// printf("data3.. %d \n",data3);
USBSTK5515_wait(5000);
dly0[sample] = data3;
/* Write Digital audio input */
I2S0_W0_MSW_W = data3; // 16 bit left channel transmit audio data
I2S0_W0_LSW_W = 0;
I2S0_W1_MSW_W = data3; // 16 bit right channel transmit audio data
I2S0_W1_LSW_W = 0;
while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag
}
/* Disble I2S */
I2S0_CR = 0x00;
```

```
return 0;
}
aic3204_test.c
#define AIC3204_I2C_ADDR 0x18
#include "usbstk5515.h"
#include "usbstk5515_gpio.h"
#include "usbstk5515_i2c.h"
#include "stdio.h"
extern Int16 aic3204_tone_headphone( );
extern Int16 aic3204_loop_stereo_in1( );
Int16 AIC3204_rget( Uint16 regnum, Uint16* regval )
{
Int16 retcode = 0;
Uint8 cmd[2];
cmd[0] = regnum & 0x007F; // 7-bit Register Address
cmd[1] = 0;
retcode |= USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 1 );
retcode |= USBSTK5515_I2C_read( AIC3204_I2C_ADDR, cmd, 1 );
*regval = cmd[0];
USBSTK5515_wait( 10 );
return retcode;
}
Int16 AIC3204_rset( Uint16 regnum, Uint16 regval )
{
Uint8 cmd[2];
cmd[0] = regnum & 0x007F; // 7-bit Register Address
cmd[1] = regval; // 8-bit Register Data
return USBSTK5515_I2C_write( AIC3204_I2C_ADDR, cmd, 2 );
}
Int16 aic3204_test( )
{
SYS_EXBUSSEL = 0x6100; // Enable I2C bus
USBSTK5515_I2C_init( ); // Initialize I2C
USBSTK5515_wait( 100 ); // Wait
```

```
if ( aic3204_loop_stereo_in1( ) )
return 1;
return 0;
}
```

FFT.C

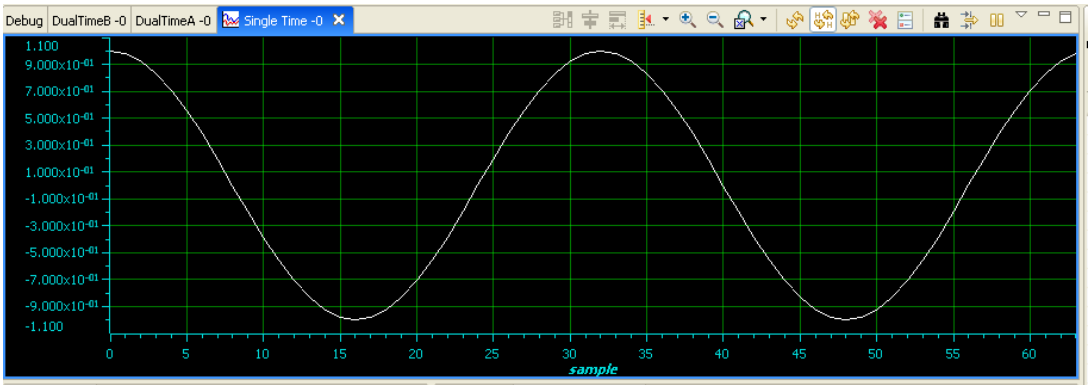
Refer previous FFT experiment

Execution Procedure:

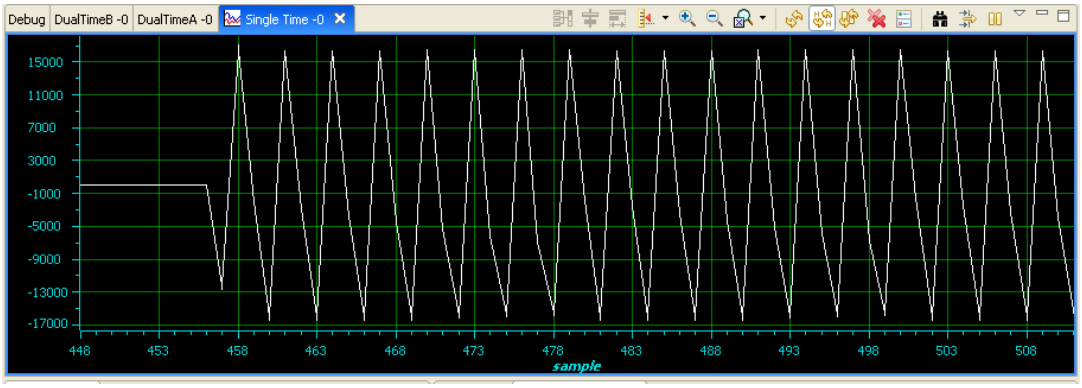
- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select Audio Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure
- from Step 46 to 51)
- Give Right Click on Your Audio.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- Observe the audio (Give Input from Stereo in and listen Filtered Output
- from Stereo Out).
- In Tools menu select Graph (Dual Time) set the properties and watch.

Result:

Cosine time plot



Audio time plot



Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	64
Dsp Data Type	16 bit signed integer
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	dly0
Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	64
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	☐ false

Audio time plot **Frequency plot**

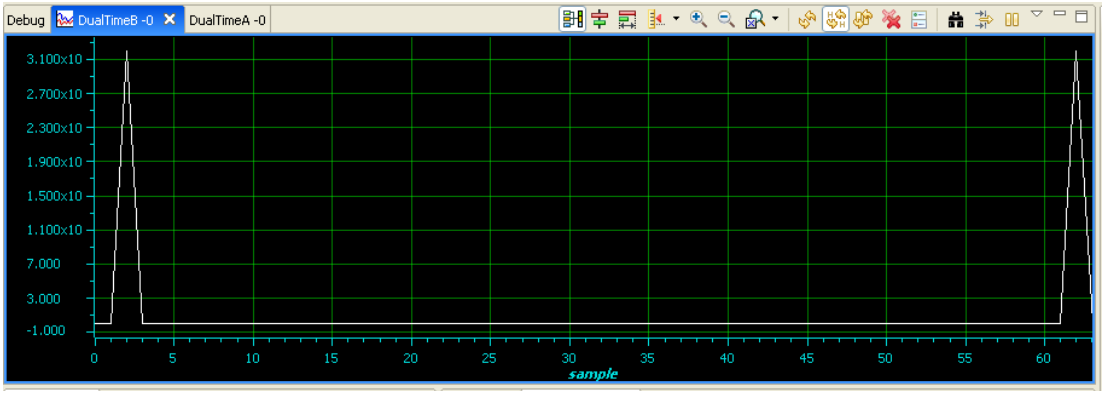
Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	64
Dsp Data Type	32 bit floating point
Index Increment	1
Interleaved Data Sources	☐ false
Q_Value	0
Sampling Rate HZ	1
Start Address A	x1
Start Address B	y1
Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	64
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Use Dc Value For Graph A	☐ false
Use Dc Value For Graph B	☐ false

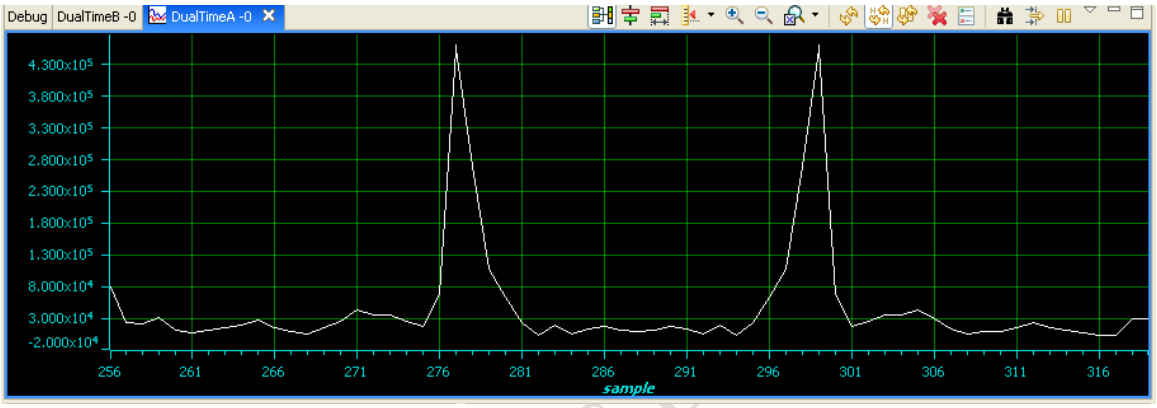
Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	64
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	1
Start Address	iobuffer
Display Properties	
Axis Display	☑ true
Data Plot Style	Line
Display Data Size	64
Grid Style	Major Grid
Magnitude Display Scale	Linear
Time Display Unit	sample
Misc	
Use Dc Value For Graph	☐ false

Cosine time plot



Cosine Frequency plot



EXP. NO.25
NOISE REMOVAL

Noise removal programs:

(i) Add noise above 3kHz and then remove (using adaptive filters)

(ii) Interference suppression using 400 Hz tone

Program code:

Main.c

```
#include<usbstk5515.h>
#include<stdio.h>
Int16 aic3204_test( );
void main()
{
USBSTK5515_init();
while(1)
{
aic3204_test();
}
}
```

aic3204_loop_stereo_in1.c

```
#include "stdio.h"
#include "usbstk5515.h"
#include<math.h>
extern Int16 AIC3204_rset( Uint16 regnum, Uint16 regval);
#define Rcv 0x08
#define Xmit 0x20
#define beta 1.5E-8 /*rate of convergence */
#define N1 30 /*# of coefficients */
#define NS 128 /*# of output sample points*/
#define Fs 8000 /*sampling frequency */
#define pi 3.1415926
//#define DESIRED 1000*sin(2*pi*T*1000/Fs) /*desired signal */
#define ADDNOISE 1000*sin(2*pi*T*3000/Fs) /*additive noise */
#define REFNOISE 1000*cos(2*pi*T*3000/Fs) /*reference noise*/
Int16 data1, data2, data3, data4;
```

```
int sample,n,k,l;
Int16 dly0[NS];
Int32 lyn,ryn;
int I,T;
float W[N1+1];
float Delay[N1+1];
float Y, E, DPLUSN,IO_INPUT[NS],IO_OUTPUT[NS],a[NS],b[NS];
Int16 aic3204_loop_stereo_in1( )
    {
// Int16 j, i = 0;
/* Configure AIC3204 */
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 1, 1 ); // Reset codec
AIC3204_rset( 0, 1 ); // Point to page 1
AIC3204_rset( 1, 8 ); // Disable crude AVDD generation from DVDD
AIC3204_rset( 2, 1 ); // Enable Analog Blocks, use LDO power
AIC3204_rset( 0, 0 ); // Select page 0
/* PLL and Clocks config and Power Up */
AIC3204_rset( 27, 0x0d ); // BCLK and WCLK is set as o/p to AIC3204(Master)
AIC3204_rset( 28, 0x00 ); // Data ofset = 0
AIC3204_rset( 4, 3 ); // PLL setting: PLLCLK <- MCLK, CODEC_CLKIN <-PLL CLK
AIC3204_rset( 6, 8 ); // PLL setting: J=8
AIC3204_rset( 7, 15 ); // PLL setting: HI_BYTE(D)
AIC3204_rset( 8, 0xdc ); // PLL setting: LO_BYTE(D)
AIC3204_rset( 30, 0x88 ); // For 32 bit clocks per frame in Master mode ONLY
// BCLK=DAC_CLK/N =(12288000/8) = 1.536MHz = 32*fs
AIC3204_rset( 5, 0x91 ); // PLL setting: Power up PLL, P=1 and R=1
AIC3204_rset( 13, 0 );// Hi_Byte(DOSR) for DOSR = 128 decimal or 0x0080 DAC
oversamppling
AIC3204_rset( 14, 0x80 ); // Lo_Byte(DOSR) for DOSR = 128 decimal or 0x0080
AIC3204_rset( 20, 0x80 ); // AOSR for AOSR = 128 decimal or 0x0080 for decimation
filters 1 to 6
AIC3204_rset( 11, 0x88 ); // Power up NDAC and set NDAC value to 8
    AIC3204_rset( 12, 0x82 ); // Power up MDAC and set MDAC value to 2
```

```
AIC3204_rset( 18, 0x88 ); // Power up NADC and set NADC value to 8
AIC3204_rset( 19, 0x82 ); // Power up MADC and set MADC value to 2
/* DAC ROUTING and Power Up */
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 12, 0x08 ); // LDAC AFIR routed to HPL
AIC3204_rset( 13, 0x08 ); // RDAC AFIR routed to HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
AIC3204_rset( 64, 0x02 ); // Left vol=right vol
AIC3204_rset( 65, 0x00 ); // Left DAC gain to 0dB VOL; Right tracks Left
AIC3204_rset( 63, 0xd4 ); // Power up left,right data paths and set channel
AIC3204_rset( 0, 0x01 ); // Select page 1
AIC3204_rset( 16, 0x06 ); // Unmute HPL , 6dB gain
AIC3204_rset( 17, 0x06 ); // Unmute HPR , 6dB gain
AIC3204_rset( 9, 0x30 ); // Power up HPL,HPR
AIC3204_rset( 0, 0x00 ); // Select page 0
USBSTK5515_wait( 500 ); // Wait
/* ADC ROUTING and Power Up */
AIC3204_rset( 0, 1 ); // Select page 1
AIC3204_rset( 0x34, 0x30 ); // STEREO 1 Jack
// IN2_L to LADC_P through 40 kohm
AIC3204_rset( 0x37, 0x30 ); // IN2_R to RADC_P through 40 kohmm
AIC3204_rset( 0x36, 3 ); // CM_1 (common mode) to LADC_M through 40 kohm
AIC3204_rset( 0x39, 0xc0 ); // CM_1 (common mode) to RADC_M through 40 kohm
AIC3204_rset( 0x3b, 0 ); // MIC_PGA_L unmute
    AIC3204_rset( 0x3c, 0 ); // MIC_PGA_R unmute
AIC3204_rset( 0, 0 ); // Select page 0
AIC3204_rset( 0x51, 0xc0 ); // Powerup Left and Right ADC
AIC3204_rset( 0x52, 0 ); // Unmute Left and Right ADC
AIC3204_rset( 0, 0 );
USBSTK5515_wait( 200 ); // Wait
/* I2S settings */
I2S0_SRGR = 0x0;
I2S0_CR = 0x8010; // 16-bit word, slave, enable I2C
I2S0_ICMR = 0x3f; // Enable interrupts
```

```
/* Play Tone */
for(l=0;l<NS;l++)
dly0[l]=0;
for(T=0;T<N1;T++)
{
W[T] = 0.0;
Delay[T] = 0.0;
}
for ( T = 0; T < NS ; T++ )
{ USBSTK5515_waitusec(25);
/* Read Digital audio input */
data3 = I2S0_W0_MSW_R; // 16 bit left channel received audio data
data1 = I2S0_W0_LSW_R;
data4 = I2S0_W1_MSW_R; // 16 bit right channel received audio data
data2 = I2S0_W1_LSW_R;
    while((Rcv & I2S0_IR) == 0); // Wait for interrupt pending flag
// printf("data3.. %d \n",data3);
USBSTK5515_wait(5000);
dly0[T] = data3;
Delay[0] = REFNOISE; /*adaptive filter's input*/
b[T]=DPLUSN = dly0[T]+ADDNOISE; /*desired + noise, d+n */
Y = 0;
for(I=0;I<N1;I++)
Y += (W[I] * Delay[I]); /*adaptive filter output */
E = DPLUSN-Y; /*error signal */
for(I=N1;I>0;I--)
{
W[I] = W[I]+(beta*E*Delay[I]); /*update weights */
if (I != 0)
Delay[I] = Delay[I-1]; /*update samples */
}
/* Write Digital audio input */
I2S0_W0_MSW_W = E; // 16 bit left channel transmit audio data
I2S0_W0_LSW_W = 0;
```

```
I2S0_W1_MSW_W = E; // 16 bit right channel transmit audio data
I2S0_W1_LSW_W = 0;
while((Xmit & I2S0_IR) == 0); // Wait for interrupt pending flag
}
/* Disble I2S */
    I2S0_CR = 0x00;
return 0;
}
/*
void apply_noise()
{
for(T=0;T<NS;T++) //# of output samples
{
a[T] = dly0[T];
IO_OUTPUT[T] = E; //overall output E
IO_INPUT[T]= DPLUSN; // store d + n
printf("%d %f %f\n",T,IO_INPUT[T],IO_OUTPUT[T]);
}
}*/
```

And one more .c file same as aic3204_test.c in the above experiment.

Execution Procedure:

- Open CCstudio setup
- Go to File Menu , select Import option.
- In the Import Window under CCs choose Existing CCS/CCE Eclipse project then next.
- In Select root Directory Browse for project file where it is located.
- Select FIR Project folder and Finish it.
- Now Connect the DSP Kit to PC and Launch it.(Follow the above given manual procedure
- from Step 46 to 51)
- Give Right Click on Your fir.out file under Binaries and select Load program Option.
- Now Go to Target select Run.
- Observe the audio (Give Input from Stereo in and listen Filtered Output
- from Stereo Out).