Laboratory manual

Digital Signal Processing Laboratory

Faculty Laboratory Incharge : Dr. Ashwani Kumar Agagrwal

Technician Laboratory incharge : Mr. Sukhminder Singh

Electrical and Instrumentation Engineering department

Sant Longowal Institute of Engineering and Technology, Longowal

**LIST OF EXPERIMENTS**

**EXPERIMENT NO. 1.** STUDY OF VARIOUS ADDRESSING MODES OF DSP USING SIMPLE PROGRAMMING EXAMPLES.

**EXPERIMENT NO. 2.** TO PERFORM LINEAR CONVOLUTION OF TWO DISCRETE TIME SEQUENCES OF FINITE LENGTH.

**EXPERIMENT NO. 3.** TO PERFORM CIRCULAR CONVOLUTION OF TWO PERIODIC SEQUENCES x(n) and h(n).

**EXPERIMENT NO. 4.** TO PEFORM SAMPLING OF INPUT SIGNAL OBTAINED FROM A CONTINUOUS TIME SIGNAL GENERATOR AND DISPLAY IT OVER THE CRO.

**EXPERIMENT NO. 5.** TO GENERATE A SQUARE WAVE AND TRAIANGULAR WAVEFORM USING A WAVE GENERATION.

**EXPERIMENT NO. 6.** TO DESIGN A LOW PASS FILTER OF A GIVEN CUT-OFF FREQUENCY AND RAIPPLE FACTOR.

**EXPERIMENT NO. 7.** TO DESIGN A HIGH PASS FILTER OF A GIVEN CUT-OFF FREQUENCY AND RAIPPLE FACTOR.

**EXPERIMENT NO. 8.** TO PERFORM GENERATION OF CONTINUOUS TIME SIGNAL AND TO PERFORM CONVOLUTION USING MATLAB CODE.

**EXPERIMENT NO. 9**. TO STUDY THE USAGE OF CODE COMPOSER STUDIO DEVELOPMENT ENVIRONMENT, TO BUILD AND TO DEBUG EMBEDDED REAL-TIME SOFTWARE APPLICATIONS USING TMS320C5416 DSP PROCESSOR.

**EXPERIMENT NO. 10**. TO WRITE ASSEMBLY LANGUAGE PROGRAMS IN TMS320C54XX DSP PROCESSOR TO ADD, SUBTRACT, MULTIPLY AND DIVIDE TWO NUMBERS USING THE PROCESSOR INSTRUCTION AND USING VARIOUS ADDRESSING MODES TO GET THE DATA FROM MEMORY.

**EXPERIMENT NO. 1. STUDY OF VARIOUS ADDRESSING MODES OF DSP USING SIMPLE PROGRAMMING EXAMPLES**

***Aim:*** ***To Study the various addressing mode of*** [***TMS320C6745 DSP processor.***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

***Addressing Modes***

The [TMS320C55x DSP](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) supports three types of addressing modes that enable flexible access to data memory, to memory-mapped registers, to register bits, and to I/O space.

The *absolute addressing mode* allows you to reference a location by supplying all or part of an address as a constant in an instruction.

The *direct addressing mode* allows you to reference a location using an address offset.

The *indirect addressing mode* allows you to reference a location using a pointer.

Each addressing mode provides one or more types of operands. An instruction that supports an addressing-mode operand has one of the following syntax elements listed below.

*Baddr* - When an instruction contains Baddr, that instruction can access one or two bits in an accumulator (AC0–AC3), an auxiliary register (AR0–AR7), or a temporary register (T0–T3). Only the register bit test/set/clear/complement instructions support Baddr. As you write one of these instructions, replace Baddr with a compatible operand.

*Cmem* - When an instruction contains Cmem, that instruction can access a single word (16 bits) of data from data memory. As you write the instruction, replace Cmem with a compatible operand.

*Lmem* - When an instruction contains Lmem, that instruction can access a long word (32 bits) of data from data memory or from a memory-mapped registers. As you write the instruction, replace Lmem with a compatible operand.

*Smem* - When an instruction contains Smem, that instruction can access a single word (16 bits) of data from data memory, from I/O space, or from a memory-mapped register. As you write the instruction, replace Smem with a compatible operand.

*Xmem and Ymem* - When an instruction contains Xmem and Ymem, that instruction can perform two simultaneous 16-bit accesses to data memory. As you write the instruction, replace Xmem and Ymem with compatible operands.

***Absolute Addressing Modes***

*k16 absolute* - This mode uses the 7-bit register called DPH (high part of the extended data page register) and a 16-bit unsigned constant to form a 23-bit data-space address. This mode is used to access a memory location or a memory-mapped register.

*k23 absolute* - This mode enables you to specify a full address as a 23-bit unsigned constant. This mode is used to access a memory location or a memory-mapped register.

*I/O absolute* - This mode enables you to specify an I/O address as a 16-bit unsigned constant. This mode is used to access a location in I/O space.

***Direct Addressing Modes***

*DP direct* - This mode uses the main data page specified by DPH (high part of the extended data page register) in conjunction with the data page register (DP). This mode is used to access a memory location or a memory-mapped register.

*SP direct* - This mode uses the main data page specified by SPH (high part of the extended stack pointers) in conjunction with the data stack pointer (SP). This mode is used to access stack values in data memory.

*Register-bit direct* - This mode uses an offset to specify a bit address. This mode is used to access one register bit or two adjacent register bits.

*PDP direct* - This mode uses the peripheral data page register (PDP) and an offset to specify an I/O address. This mode is used to access a location in I/O space.

The DP direct and SP direct addressing modes are mutually exclusive. The mode selected depends on the CPL bit in status register ST1\_55.

* 0 DP direct addressing mode
* 1 SP direct addressing mode

The register-bit and PDP direct addressing modes are independent of the CPL bit.

***Indirect Addressing Modes***

You may use these modes for linear addressing or circular addressing.

*AR indirect* - This mode uses one of eight auxiliary registers (AR0–AR7) to point to data. The way the CPU uses the auxiliary register to generate an address depends on whether you are accessing data space (memory or memory-mapped registers), individual register bits,or I/O space.

*Dual AR indirect* - This mode uses the same address-generation process as the AR indirect addressing mode. This mode is used with instructions that access two or more data-memory locations.

*CDP indirect* - This mode uses the coefficient data pointer (CDP) to point to data. The way the CPU uses CDP to generate an address depends on whether you are accessing data space (memory or memory-mapped registers), individual register bits, or I/O space.

*Coefficient indirect* - This mode uses the same address-generation process as the CDP indirect addressing mode. This mode is available to support instructions that can access a coefficient in data memory at the same time they access two other data-memory values using the dual AR indirect addressing mode.

***Circular Addressing***

Circular addressing can be used with any of the indirect addressing modes. Each of the eight auxiliary registers (AR0–AR7) and the coefficient data pointer (CDP) can be independently configured to be linearly or circularly modified as they act as pointers to data or to register bits, see Table 3-10. This configuration is done with a bit (ARnLC) in status register ST2\_55. To choose circular modification, set the bit.

Each auxiliary register ARn has its own linear/circular configuration bit in ST2\_55.

* 0 Linear addressing
* 1 Circular addressing

The CDPLC bit in status register ST2\_55 configures the DSP to use CDP for linear addressing or circular addressing:

* 0 Linear addressing
* 1 Circular addressing

You can use the circular addressing instruction qualifier, .CR, if you want every pointer used by the instruction to be modified circularly, just add .CR to the end of the instruction mnemonic (for example, ADD.CR). The circular addressing instruction qualifier overrides the linear/circular configuration in ST2\_55.

**Result**

Thus, the various addressing mode of DSP processor [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) was studied.

**EXPERIMENT NO. 2. TO PERFORM LINEAR CONVOLUTION OF TWO DISCRETE TIME SEQUENCES OF FINITE LENGTH.**

***Aim - To perform the Linear Convolution of two given discrete sequence in*** [***TMS320C5505 KIT.***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

**Requirements**

* CCS v4
* [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)
* USB Cable
* 5V Adapter

**Theory**

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

If the input and impulse response of a system are x[n] and h[n] respectively, the convolution is given by the expression

x[n] \* h[n] = ε x[k] h[n-k]

Where k ranges between -∞ and ∞

If,x(n) is a M- point sequence

h(n) is a N – point sequence

then, y(n) is a (M+N-1) – point sequence.

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

**Procedure**

1. Open Code Composer Studio v4 .

2. In WorkSpace Launcher.

BROWSE → Select the project location and make one new folder, MAKE NEW FOLDER →Type the Workspace name, OK → OK.

3. FILE → NEW → CCS PROJECT

 Project name : Type your project name.

 Tick use default location. → NEXT

 Project type : C5500.

 Tick Debug And Release. → NEXT → NEXT.

 Output type : Executable

 Device Variant : [TMS320C55XX](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) - [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board).

 Device Endianness : big

 Code Generation Tools : TI v4.3.5

 Run time support library : automatic

 Target content : none → FINISH

4. FILE → NEW → SOURCE FILE

 Source file: Type your projectname.c( .c extension is must ).

 Type the program.

 FILE → SAVE.

5. Paste the following board library files in workspace location.

 include folder (contains header files)

 Gel folder (contains gel file)

 Library folder(contains library files)

6. Paste the Linker file in the project location.(linker file is available in cd)

Note: Those folders and linker file are availble at cd.

7. PROJECT → PROPERTIES → C/C++ BUILD → BASIC OPTION

 Target processor version(--silicon version, -mv) : 5505OK.

8. FILE → NEW → TARGET CONFIGURATION FILE

 file name: projectname. ccxml (.ccxml extension is must)

 Connection: Texas Instrument XDS100 v1 USB Emulator.

 Device: [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board). (Tick the [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board))→ SAVE → TARTGET CONFIGURATION →C55XX\_0 →BROWSE, browse the workspace location, open the gel folder and select the GEL file. → OPEN →SAVE.

9. In C/C++ Project window, Right click the project →REBUILD PROJECT.

10. Connections

 Connect the usb cable, PC to [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

11. TARGET → DEBUG ACTIVE PROJECT.

12. TARGET → RUN.

Enter the input

 Enter value for m 4

Enter value for n 4

Enter values for i/p x(n):1 1 1 1

Enter Values for i/p h(n):1 1 1 1

**Output**

The Value of output y[0]=1

The Value of output y[1]=2

The Value of output y[2]=3

The Value of output y[3]=4

The Value of output y[4]=3

The Value of output y[5]=2

The Value of output y[6]=1

13. TARGET ◊ HALT.

**Program**

*#include "stdio.h"*

*int x[15],h[15], y[15];*

*main()*

*{*

 *int i, j, m, n;*

 *printf("\n Enter value for m");*

 *scanf("%d", &m);*

 *printf("\n Enter value for n");*

 *scanf("%d", &n);*

 *printf("Enter values for i/p x(n):\n");*

 *for(i=0;i<m; i++)*

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

 *printf("Enter Values for i/p h(n): \n");*

 *for(i=0;i<n; i++)*

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

 *for(i=m; i<=m+n-1;i++)*

 *x[i]=0;*

 *for(i=n; i<=m+n-1;i++)*

 *h[i]=0;*

 *for(i=0;i<m+n-1;i++)*

 *{*

 *y[i]=0;*

 *for(j=0;j<=i; j++)*

 *{*

 *y[i]=y[i]+(x[j]\*h[i-j]);*

 *}*

 *}*

 *for(i=0;i<m+n-1;i++)*

 *printf("\n The Value of output y[%d]=%d",i,y[i]);*

 *printf( "\n\*\*\*ALL Tests Passed\*\*\*\n" );*

*}*

**Result**

Thus, the Linear Convolution of two given discrete sequence has performed and the result is displayed.

**EXPERIMENT NO. 3. TO PERFORM CIRCULAR CONVOLUTION OF TWO PERIODIC SEQUENCES x(n) and h(n)**

***Aim - To perform the Circular Convolution of two given discrete sequence in*** [***TMS320C5505s KIT***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

**Requirements**

* CCS v4
* [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)
* USB Cable
* 5V Adapter

**Theory**

The circular convolution, also known as cyclic convolution. A convolution operation that contains a circular shift is called circular convolution. Circular convolution of two sequences x1[n] and x2[n] is given by

x1[n]\*x2[n] = ∑ k x1[k] x2((n-k))N, 0= n =N-1

where k ranges between 0 and N-1

One of the methods to find circular convolution....

In circular convolution the length of the output sequence will be equal to length of the input sequence ie. length(y)=length(x) So first perform linear convolution using any of the methods u find easier.

If m is the length of 'x' and n is the length of the 'h' then length of 'yl' from linear conv is m+n-1.

Since length of output from circular conv is m, we will bring the last n-1 terms from 'yl' and add them to first n-1 terms.So the obtained output is circularly convoluted output.

For eg. if x= 1, 2, 3, 4 and h= 2,3,1

lin conv op ie. yl= 2,7,13,19,15,4

bring last two (n-1) terms to first two terms

so circularly convluted op is yc= 17, 11,13,19

**Procedure**

1. Open Code Composer Studio v4 .

2. In WorkSpace Launcher.

 BROWSE → Select the project location and make one new folder, MAKE NEW FOLDER →Type the Workspace name, OK →OK.

3. FILE → NEW → CCS PROJECT

 Project name: Type your project name.

 Tick use default location. →NEXT

 Project type: C5500.

 Tick Debug And Release. →NEXT → NEXT.

 Output type: Executable.

 Device Variant : [TMS320C55XX](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) - [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board).

 Device Endianness : big

 Code Generation Tools: TI v4.3.5.

 Run time support library: automatic.

 Target content: none. → FINISH

4. FILE → NEW → SOURCE FILE

 Source file: Type your projectname.c( .c extension is must ).

 Type the program.

 FILE → SAVE.

5. Paste the following board library files in workspace location.

 include folder (contains header files)

 Gel folder (contains gel file)

 Library folder(contains library files)

6. Paste the Linker file in the project location.(linker file is available in cd)

 Note: Those folders and linker file are availble at cd.

7. PROJECT → PROPERTIES → C/C++ BUILD → BASIC OPTION

 Target processor version(--silicon version, -mv) : 5505 OK.

8. FILE → NEW → TARGET CONFIGURATION FILE

 file name: projectname. ccxml (.ccxml extension is must)

 Connection: Texas Instrument XDS100 v1 USB Emulator.

 Device: [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board). (Tick the [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board))→ SAVE → TARTGET CONFIGURATION →C55XX\_0 →BROWSE, browse the workspace location, open the gel folder and select the GEL file. → OPEN →SAVE.

9. In C/C++ Project window, Right click the project →REBUILD PROJECT.

10. Connections

 Connect the usb cable, PC to [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

11. TARGET → DEBUG ACTIVE PROJECT.

12. TARGET → RUN.

Enter the input

Enter the length of the first sequence 4

Enter the length of the second sequence 4

Enter the first sequence 1 1 1 1

Enter the second sequence 1 1 1 1

13. TARGET - HALT.

**Output**

The circular convolution is 4 4 4 4

**Program**

*#include "stdio.h"*

*int m, n, x[30], h[30], y[30], i, j, k, x2[30], a[30];*

*void main(){*

 *printf(" Enter the length of the first sequence\n");*

 *scanf("%d", &m);*

 *printf(" Enter the length of the second sequence\n");*

 *scanf("%d", &n);*

 *printf(" Enter the first sequence\n");*

 *for(i=0;i<m; i++)*

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

 *printf(" Enter the second sequence\n");*

 *for(j=0;j<n; j++)*

 *scanf("%d", &h[j]);*

 *if(m-n!=0) /\*If length of both sequences are not equal\*/*

 *{ if(m>n) /\* Pad the smaller sequence with zero\*/*

 *{ for(i=n; i<m; i++)*

 *h[i]=0;*

 *n=m;*

 *}*

 *for(i=m; i<n; i++)*

 *x[i]=0;*

 *m=n;*

 *}*

 *y[0]=0;*

 *a[0]=h[0];*

 *for(j=1;j<n; j++) /\*folding h(n) to h(-n)\*/*

 *a[j]=h[n-j];*

 */\*Circular convolution\*/*

 *for(i=0;i<n; i++)*

 *y[0]+=x[i]\*a[i];*

 *for(k=1;k<n;k++)*

 *{ y[k]=0;*

 */\*circular shift\*/*

 *for(j=1;j<n; j++)*

 *x2[j]=a[j-1];*

 *x2[0]=a[n-1];*

*for(i=0;i<n; i++) {*

 *a[i]=x2[i];*

 *y[k]+=x[i]\*x2[i];*

 *}*

 *}*

 */\*displaying the result\*/*

 *printf(" The circular convolution is\n");*

 *for(i=0;i<n; i++)*

 *printf("%d \t", y[i]);*

*}*

**Result**

Thus, the Circular Convolution of two given discrete sequence has performed and the result is displayed.

**EXPERIMENT NO. 4. TO PEFORM SAMPLING OF INPUT SIGNAL OBTAINED FROM A CONTINUOUS TIME SIGNAL GENERATOR AND DISPLAY IT OVER THE CRO.**

***Aim - To Generate a sine sample and display it using*** [***TMS320C5505 DSP KIT.***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

**Requirements**

* CCS v4
* [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)
* USB Cable
* 5V Adapter

**Theory**

The simplest method to generate Sine wave is to use Trigonometric Sin function. The Sin function will generate the samples from our specific parameter like sampling frequency, number of samples, input frequency. In this project, generating the correct sample is important. The library function "sin()" does all of the work.



Its most basic form as a function of time (t) is:

y(t) = A.sin(ωt+Φ)

Where

A, the amplitude, is the peak deviation of the function from its center position.

ω, the angular frequency, specifies how many oscillations occur in a unit time interval, in radians per second.

φ, the phase, specifies where in its cycle the oscillation begins at t = 0.

When the phase is non-zero, the entire waveform appears to be shifted in time by the amount f/? seconds. A negative value represents a delay, and a positive value represents an advance.

**Procedure**

1. Open Code Composer Studio v4 .

2. In Workspace Launcher.

 BROWSE → Select the project location and make one new folder, MAKE NEW FOLDER →Type the Workspace name, OK → OK.

3. FILE → NEW → CCS PROJECT

 Project name: Type your project name.

 tick use default location. →NEXT

 Project type: C5500.

 Tick Debug And Release. →NEXT → NEXT.

 Output type: Executable.

 Device Variant : [TMS320C55XX](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) - [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board).

 Device Endianness : big

 Code Generation Tools: TI v4.3.5.

 Run time support library: automatic.

 Target content: none. →FINISH

4.FILE → NEW → SOURCE FILE

 Source file: Type your projectname.c( .c extension is must ).

 Type the program.

 FILE → SAVE.

5. Paste the following board library files in workspace location.

 include folder (contains header files)

 Gel folder (contains gel file)

 Library folder(contains library files)

6. Paste the Linker file in the project location.(linker file is available in cd)

 Note: Those folders and linker file are availble at cd.

7. PROJECT → PROPERTIES → C/C++ BUILD → BASIC OPTION

 Target processor version(--silicon version, -mv) : 5505 OK.

8. FILE → NEW →TARGET CONFIGURATION FILE

 file name: projectname. ccxml (.ccxml extension is must)

 Connection: Texas Instrument XDS100 v1 USB Emulator.

 Device: [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board). (Tick the [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board))→ SAVE → TARTGET CONFIGURATION →C55XX\_0 →BROWSE, browse the workspace location, open the gel folder and select the GEL file. →OPEN →SAVE.

9. In C/C++ Project window, Right click the project →REBUILD PROJECT

10. Connections :

 Connect the usb cable, PC to [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

11. TARGET ? DEBUG ACTIVE PROJECT.

12. TARGET ? RUN.(wait to generate samples)

13. TARGET ? HALT.

14. TOOLS ? GRAPH ? SINGLE TIME

 Acquisition buffer size : 256

 Index increment : 1

 Start address : value.

 Display data size : 256 → Ok.

**Program**

*#include “stdio.h”*

*#define PI 3.14*

*short value[500];*

*void main()*

*{*

 *const float sampf = 1024000.0;// Sampling frequency is fixed*

 *const int inpf = 4000; // change the input frequency from 1khz to 8khz(1000 to 8000)*

 *float sampt;*

 *double teta;*

 *int i,count,nsamp,value1;*

 *sampt = 1/sampf;*

 *nsamp = sampf/inpf;*

 *printf("\n Sampling Frequency is : %f", sampf);*

 *printf("\n Sampling Time is :%f", sampt);*

 *printf("\n Input Frequency is : %d", inpf);*

 *printf("\n The number of Sample is : %d", nsamp);*

 *for(count=0;count<nsamp; count++)*

 *{*

 *teta = (2 \* PI \* inpf \* sampt \* count);*

 *printf ("\n teta = %lf", teta);*

 *value[count] = sin(teta)\*1024;*

 *printf("\t sin %lf Value is : %d", teta, value[count]);*

 *}*

 *}*

**Result**

Thus, the Sine waveform generation was generated and the sine samples is stored and displayed at graph.

**EXPERIMENT NO. 5. TO GENERATE A SQUARE WAVE AND TRAIANGULAR WAVEFORM USING A WAVE GENERATION**

***Aim - To Generate a Square waveform using*** [***TMS320C5505 DSP KIT.***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

**Requirements**

* CCS v4
* [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)
* USB Cable
* 5V Adapter

***Theory***

The simplest method to generate Sqaure wave is to use High Low concept for pin with delay. Square waves have an interesting mix of practice and theory. In practice, they are extremely simple. In their simplest form, they consist of an alternating sequence of amplitudes; e.g. high/low or 1's and 0's.

The same high / low logic here we implented in experiment. For particular duration the high state is out , then low state is out. Finally square wave is generated and plotted in code composer studio Graph.

**Procedure**

1. Open Code Composer Studio v4 .

2. In Workspace Launcher.

 BROWSE → Select the project location and make one new folder, MAKE NEW FOLDER →Type the Workspace name, OK → OK.

3. FILE → NEW → CCS PROJECT

 Project name: Type your project name.

 tick use default location. ?NEXT

 Project type: C5500.

 Tick Debug And Release. →NEXT → NEXT.

 Output type: Executable.

 Device Variant : [TMS320C55XX](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) - TMS320C5505.

 Device Endianness : big

 Code Generation Tools: TI v4.3.5.

 Run time support library: automatic.

 Target content: none. →FINISH

4.FILE → NEW → SOURCE FILE

 Source file: Type your projectname.c( .c extension is must ).

 Type the program.

Connect the usb cable, PC to [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

 FILE → SAVE.

5. Paste the following board library files in workspace location.

 include folder (contains header files)

 Gel folder (contains gel file)

 Library folder(contains library files)

6. Paste the Linker file in the project location.(linker file is available in cd)

 Note: Those folders and linker file are availble at cd.

7. PROJECT → PROPERTIES → C/C++ BUILD → BASIC OPTION

 Target processor version(--silicon version, -mv) : 5505 OK.

8. FILE → NEW → TARGET CONFIGURATION FILE

 file name: projectname. ccxml (.ccxml extension is must)

 Connection: Texas Instrument XDS100 v1 USB Emulator.

 Device: [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board). (Tick the [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board))→ SAVE → TARTGET CONFIGURATION →C55XX\_0 →BROWSE, browse the workspace location, open the gel folder and select the GEL file. → OPEN →SAVE.

9. In C/C++ Project window, Right click the project →REBUILD PROJECT.

10. Connections :

Connect the usb cable, PC to [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

11. TARGET → DEBUG ACTIVE PROJECT.

12. TARGET → RUN.(wait to generate samples)And it automatically halts the execution.

13. TOOLS → GRAPH → SINGLE TIME

 Acquirstion buffer size : 500

 Index increment : 1

 Start address : Square.

 Display data size : 500 → Ok.

**Program**

*#include “stdio.h”*

*#include “iostream.h”*

*unsigned short Square[500];*

*void main()*

*{*

 *int i=0,j=0,k=0;*

 *for(k=0;k<5;k++)*

 *{*

 *for(i=0;i<50;i++)*

 *{*

 *Square[j] = 0x0000FFFF;*

 *j++;*

 *}*

 *for(i=0;i<50;i++)*

 *{*

 *Square[j] = 0x0;*

 *j++;*

 *}*

 *}*

*}*

**Result**

The Square waveform was generated displayed at graph.

**EXPERIMENT NO. 6. TO DESIGN A LOW PASS FILTER OF A GIVEN CUT-OFF FREQUENCY AND RIPPLE FACTOR.**

***Aim - To implement the FIR Low pass filter using*** [***TMS320C5505 KIT***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)***.***

**Requirements**

* CCS v4
* [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)
* USB Cable
* 5V Adapter

**Theory**

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response of an Nth-order discrete-time FIR filter (i.e., with a Kronecker delta impulse input) lasts for N + 1 samples, and then settles to zero.

For a discrete time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. which defines the output sequence y[n] in terms of its input sequence x[n]:



where:

x[n] is the input signal,

y[n] is the output signal

N is the filter order;

The terms are commonly referred to as taps, based on the structure of a tapped delay line that in many implementations or block diagrams provides the delayed inputs to the multiplication operations.

**Procedure**

1. Open Code Composer Studio v4 .

2. In WorkSpace Launcher.

 BROWSE → Select the project location and make one new folder, MAKE NEW FOLDER →Type the Workspace name, OK → OK.

3. FILE → NEW → CCS PROJECT

 Project name: Type your project name.

 Tick use default location. →NEXT

 Project type: C5500.

 Tick Debug And Release. →NEXT → NEXT.

 Output type: Executable.

 Device Variant : [TMS320C55XX](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) - [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board).

 Device Endianness : big

 Code Generation Tools: TI v4.3.5.

 Run time support library: automatic.

 Target content: none. ?FINISH

4. FILE → NEW → SOURCE FILE

 Source file: Type your projectname.c( .c extension is must ).

 Type the program.

 FILE → SAVE.

5. Paste the following board library files in workspace location.

 include folder (contains header files)

 Gel folder (contains gel file)

 Library folder(contains library files)

6. Paste the Linker file in the project location.(linker file is available in cd)

 Note: Those folders and linker file are availble at cd.

7. PROJECT → PROPERTIES → C/C++ BUILD → BASIC OPTION

 Target processor version(--silicon version, -mv) : 5505 OK.

8. FILE → NEW → TARGET CONFIGURATION FILE

 file name: projectname. ccxml (.ccxml extension is must)

 Connection: Texas Instrument XDS100 v1 USB Emulator.

 Device: [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board). (Tick the [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board))→ SAVE → TARTGET CONFIGURATION →C55XX\_0 →BROWSE, browse the workspace location, open the gel folder and select the GEL file. → OPEN →SAVE.

9. In C/C++ Project window, Right click the project →REBUILD PROJECT.

10. Connections :

 Connect the usb cable, PC to [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

11. TARGET → DEBUG ACTIVE PROJECT.

12. TARGET → RUN.(wait few seconds)

13. TARGET ? HALT.

14. Output is displayed at Console Window.

**Program**

*// ideal low pass filter*

*#include “iostream.h”*

*#include “stdio.h”*

*#define PI 3.14*

*void main()*

*{*

 *const int sampf=10000;*

 *const int cutf=1000;*

 *double value, a, b, output;*

 *int nyqf,n,c0;*

 *int \*coeff;*

 *coeff = (int \*)0xc0001000;*

 *nyqf=sampf/2;*

 *c0=cutf/nyqf;*

*for (n=-5;n<6;n++)*

 *{*

 *if (n==0)*

 *{*

 *output = 0.5;*

 *}*

 *else*

 *a = (n \* PI)/2;*

 *b = n \* PI;*

 *value = sin(a);*

 *output = value/b;*

 *Printf ("\n The Fir Low pass filter coefficient : %f", output);*

*}*

*}*

**Result**

The FIR Low pass filter was Implemented and displayed the results in console window.

**EXPERIMENT NO. 7. TO DESIGN A HIGH PASS FILTER OF A GIVEN CUT-OFF FREQUENCY AND RIPPLE FACTOR.**

***Aim - To Implement the FIR High pass filter using*** [***TMS320C5505 KIT***](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)***.***

**Requirements**

* CCS v4
* [TMS320C5505 KIT](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)
* USB Cable
* 5V Adapter

**Theory**

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).

The impulse response of an Nth-order discrete-time FIR filter (i.e., with a Kronecker delta impulse input) lasts for N + 1 samples, and then settles to zero.

The output y of a linear time invariant system is determined by convolving its input signal x with its impulse response b.

For a discrete time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. The operation is described by the following equation, which defines the output sequence y[n] in terms of its input sequence x[n]:



where:

x[n]is the input signal,

y[n]is the output signal,

N is the filter order;

The terms are commonly referred to as taps, based on the structure of a tapped delay line that in many implementations or block diagrams provides the delayed inputs to the multiplication operations.

**Procedure**

1. Open Code Composer Studio v4 .

2. In WorkSpace Launcher.

 BROWSE → Select the project location and make one new folder, MAKE NEW FOLDER →Type the Workspace name, OK → OK.

3. FILE → NEW → CCS PROJECT

 Project name: Type your project name.

 Tick use default location. →NEXT

 Project type: C5500.

 Tick Debug And Release. →NEXT → NEXT.

 Output type: Executable.

 Device Variant : [TMS320C55XX](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) - [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board).

 Device Endianness : big

 Code Generation Tools: TI v4.3.5.

 Run time support library: automatic.

 Target content: none. →FINISH

4. FILE → NEW → SOURCE FILE

 Source file: Type your projectname.c( .c extension is must ).

 Type the program.

 FILE → SAVE.

5. Paste the following board library files in workspace location.

 include folder (contains header files)

 Gel folder (contains gel file)

 Library folder(contains library files)

6. Paste the Linker file in the project location.(linker file is available in cd)

 Note: Those folders and linker file are availble at cd.

7. PROJECT → PROPERTIES → C/C++ BUILD → BASIC OPTION

 Target processor version(--silicon version, -mv) : 5505 OK.

8. FILE → NEW → TARGET CONFIGURATION FILE

 file name: projectname. ccxml (.ccxml extension is must)

 Connection: Texas Instrument XDS100 v1 USB Emulator.

 Device: [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board). (Tick the [TMS320C5505](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board) )à SAVE à TARTGET CONFIGURATION àC55XX\_0 àBROWSE, browse the workspace location, open the gel folder and select the GEL file. à OPEN àSAVE.

9. In C/C++ Project window, Right click the project →REBUILD PROJECT.

10. Connections

 Connect the usb cable, PC to [TMS320C5505 KIT.](https://www.pantechsolutions.net/dsp-dsc-boards/tms320c5505-dsp-starter-board)

 Connect the 5v adapter.

 Power on the kit.

11. TARGET → DEBUG ACTIVE PROJECT.

12. TARGET → RUN.(wait few seconds)

13. TARGET → HALT.

14. Output is displayed at Console Window.

**Program**

*// ideal high pass filter coefficients*

*#include “stdio.h”*

*#include “iostream.h”*

*#define PI 3.14*

*void main()*

*{*

 *const int sampf=10000;*

 *const int cutf=1000;*

 *double value, a, b, output, final;*

 *int nyqf,n,c0;*

 *int \*coeff;*

 *coeff = (int \*)0xc0001000;*

 *nyqf=sampf/2;*

 *c0=cutf/nyqf;*

 *for(n=-5;n<6;n++)*

 *{*

 *if(n==0)*

 *{*

 *output = 0.5;*

 *}*

 *else*

 *a = (n \* PI)/2;*

 *b = n \* PI;*

 *value = sin(a);*

 *output = value/b;*

 *final = 1 - output;*

 *printf("\n The Fir High pass filter coefficient : %f",final);*

*}*

*}*

**Result**

Thus, the FIR High pass filter was implemented and displayed the results in console window.

**EXPERIMENT NO. 8. TO PERFORM GENERATION OF CONTINUOUS TIME SIGNAL AND TO PERFORM CONVOLUTION USING MATLAB CODE.**

***Aim - To Generate a continuous sinusoidal time signals and to perform convolution Using MATLAB.***

**Requirements**

* MATLAB 2007 later

**Procedure**

1. OPEN MATLAB

2. File → New → Script.

 Type the program in untitled window

3. File → Save → type filename.m in matlab workspace path

4. Debug → Run. Wave will displayed at Figure dialog box.

***Program (i)***

*% here we generating the sine wave using sin function,*

*% The input frequency is 1 Khz*

*% Assuming The Sampling frequency is 5 Mhz*

*clear all;*

*Finput = 1000;*

*Fsampling = 5000000;*

*Tsampling = 1 / Fsampling;*

*Nsample = Fsampling/ Finput;*

*N = 0:5\*Nsample-1;*

*x=sin(2 \* pi \* Finput \* Tsampling \* N);*

*plot(x);*

*title('Sine Wave Generation');*

*xlabel('Time -- >');*

*ylabel('Amplitude-- >');*

*grid on;*

5. OPEN MATLAB

6. File → New → Script.

 Type the program in untitled window.

 File → Save → type filename.m in matlab workspace path.

7. Debug → Run. Wave will displayed at Figure dialog box.

 Note: Change the input through program in Xn and Hn.

***Program (ii)***

*clear all;*

*Xn = [1,1,1,1];*

*Hn = [1,1,1,1];*

*x=length(Xn);*

*h = length(Hn);*

*N = x + h - 1;*

*Yn = conv(Xn, Hn);*

*subplot(2,2,1);*

*stem(Xn);*

*title('Xn');*

*xlabel('Length of First Input Sequence');*

*ylabel('Input Value');*

*subplot(2,2,2);*

*stem(Hn);*

*title('Hn');*

*xlabel('Length of Second Input Sequence');*

*ylabel('Input Value');*

*subplot(2,2,3);*

*stem(Yn);*

*title('Yn');*

*xlabel('Length of Output Sequence');*

*ylabel('Output Value');*

**Result**

Thus the Signal was generated using MATLAB.

Thus the Linear convolution was performed using MATLAB. **EXPERIMENT NO. 9. USAGE OF CODE COMPOSER STUDIO DEVELOPMENT ENVIRONMENT, TO**

 **BUILD AND TO DEBUG EMBEDDED REAL-TIME SOFTWARE.**

***AIM - TO STUDY THE USAGE OF CODE COMPOSER STUDIO DEVELOPMENT ENVIRONMENT, TO BUILD AND TO DEBUG EMBEDDED REAL-TIME SOFTWARE APPLICATIONS USING TMS320C5416 DSP PROCESSOR.***

**APPARATUS REQUIRED:**

|  |  |  |
| --- | --- | --- |
| S. NO. | ITEM | QUANTITY |
| 1 | TMS320VC5416 DSK Development Board | 1 |
| 2. | PC with Code Composer Studio IDE | 1 |
| 3 | USB Cable | 1 |
| 4 | +5V Universal Power Supply | 1 |
| 5 | AC Power cord | 1 |

**THEORY**

Texas Instruments’ Code Composer Studio development tools are bundled with the 5416DSK providing the user with an industrial-strength integrated development environment for C and assembly programming. Code Composer Studio communicates with the DSP using an on-board JTAG emulator through a USB interface.

ASSEMBLY LANGUAGE PROGRAMMING IN C54X

The Assembly language programming is done using the following assembler directives

*.text section*

 It consists of the assembly program, which has to be translated in object code by the assembler, and it is loaded into the program memory for execution.

*.data section.*

 It consists of the constants and variables which are intialized and are loaded into the data memory area. The origin of the data memory is given in the .cmd file.

*.bss section.*

 It is used to reserve a block of memory which is uninitalized.

*.mmregs*

 It permits to use the memory mapped registers to be referred using the names such as AR0, SP etc.

*.include “XX”*

 It informs the assembler to insert a list of instructions in the file “XX” while assembling.

.end

 To end the assembly language program.

*.equ*

 To equate a symbol with a constant value.

*.word x,y, ….z*

 It reserves 16 bit locations and initialises them with values x,y,….z

*sum.asm*

*.include "5416\_IV.asm"*

*.word 0003h,0004h*

 *.data*

 *.text*

*begin STM #1000h,AR1 {memory location of P}*

 *STM #1001h,AR2 {memory location of Q}*

 *STM #1500h,AR1{Result is stored in the memory}*

 *LD \*AR1,A {This accumulator loads that accumulator A*

*with the data in the memory 1000h}*

 *LD \*AR2,B {{This accumulator loads that accumulator A*

*with the data in the memory 1001h}*

 *ADD A,0,B {accumulator A is added eith accumulator B*

 *and output will be in accumulator B}*

 *STL B,\*AR3 (output will be stored in the memory location*

 *1500h}*

*.end*

**PROCEDURE FOR REAL TIME PROGRAMS:**

1. Connect CRO to the Socket Provided for LINE OUT.
2. Connect a Signal Generator to the LINE IN Socket.
3. Switch on the Signal Generator with a sine wave of frequency 500 Hz. and Vp-p=1.5v
4. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.
5. Create a new project with name codec.pjt.( c:\ccstudio\_v3.1\myprojects\codec)
6. From the File Menu 🡺 new 🡺 DSP/BIOS Configuration 🡺select “dsk6713.cdb”
7. Add the library file “dsk6713bsl.lib” to the current project

Path 🡪 “C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib”

1. Build, Load and Run the program.
2. You can notice the input signal of 500 Hz. appearing on the CRO verifying the codec configuration.
3. You can also pass an audio input and hear the output signal through the speakers.
4. You can also vary the sampling frequency using the DSK6713\_AIC23\_setFreq Function in the “codec.c” file and repeat the above steps.

**RESULT**: Thus the usage of Code Composer studio was studied and the sample programs were run and the results were seen.

**EXPERIMENT 10. ASSEMBLY LANGUAGE PROGRAMS IN TMS320C54XX DSP PROCESSOR TO ADD, SUBTRACT, MULTIPLY AND DIVIDE TWO NUMBERS.**

***AIM - TO WRITE ASSEMBLY LANGUAGE PROGRAMS IN TMS320C54XX DSP PROCESSOR TO ADD, SUBTRACT, MULTIPLY AND DIVIDE TWO NUMBERS USING THE PROCESSOR INSTRUCTION AND USING VARIOUS ADDRESSING MODES TO GET THE DATA FROM MEMORY.***

**APPARATUS REQUIRED**

|  |  |  |
| --- | --- | --- |
| *S. NO.* | *ITEM* | *QUANTITY* |
| 1 | TMS320VC5416 DSK Development Board | 1 |
| 2. | PC with Code Composer Studio IDE | 1 |
| 3 | USB Cable | 1 |
| 4 | +5V Universal Power Supply | 1 |
| 5 | AC Power cord | 1 |

(i) ADDITION using INDIRECT ADDRESSING

 In Indirect addressing any location in the 64 K word data space can be accessed via a 16-bit address. Indirect addressing uses the auxiliary registers (AR’s) to point out a location in data or program memory.

**ALGORITHM**

*Step 1: Store any auxiliary register say AR1 with any address location for example 1000h*

*Step 2: Store another auxiliary register say AR2 with another address location e.g. 1001h.*

*Step 3: Store another auxiliary register say AR3 with address location for example 1500h*

*Step 4: Load first data from the first address location to accumulator A using indirect addressing.*

*Step 5: Load second data from second location to accumulator B using indirect addressing.*

*Step 6:Add the two data in the accumulators A and B.*

*Step 7: Store output in accumulator to the memory location addressed by the auxillary register.*

*Step 8: End the program.*

PROGRAM FOR ADDITION USING INDIRECT ADDRESSING:

|  |  |  |
| --- | --- | --- |
| *LABEL* | *MNEMONICS* | *COMMENTS* |
| Begin: | .include "5416\_IV.asm".word 0003h, 0004h.data .textSTM #1000h,AR1STM #1001h,AR2STM #1500h,AR3 LD \*AR1, A LD \*AR2, B ADDC A, 0, B STL B, \*AR3. end | Begin the programMemory location of PMemory location of QMemory location of the result.The accumulator A is loaded with the data in the memory location pointed by AR1.The accumulator B is loaded with the data in the memory location pointed byAR2.Accumulator A is added with accumulator B and the output will be in accumulator BThe result in accumulator B is stored to the memory location pointed by AR3.End the program |

(ii) MULTIPLICATION USING DIRECT ADDRESSING:

In direct addressing the instruction carries the 7-bit data memory address offset in the instruction itself. The offset address is added with the DP (Data pointer) or SP (Stack Pointer) to get the 16-bit data memory address.

**ALGORITHM:**

*Step 1: Reset Compiler mode bit to zero to use the DP register to generate data memory address.*

*Step 2: Initialize the DP register with page number.*

*Step 3: Load the value from the location pointed by DP and offset in the Accumulator A.*

*Step 4: Load the second value from the location pointed by DP and offset in the Accumulator B.*

*Step 5: Multiply both the values in A & B.*

*Step 6: Store the value in the accumulator to the memory location pointed by the offset.*

*Step 7: End the program.*

PROGRAM FOR MULTIPLICATION USING DIRECT ADDRESSING:

|  |  |  |
| --- | --- | --- |
| *LABEL* | *MNEMONICS* | *COMMENTS* |
| Begin: | .include "5416\_IV.asm".word 0003h, 0004h.data .textRSBX CPLLD #20h, DPLD 01h,ALD 02h, 2, BMPY A, 0, BSTL B, 15h. end | Begin the programCompiler mode bit CPL is reset to zero.Loads the DP register with the value 20h that corresponds to the page address 1000hLoads the value from address location 1001h to the accumulator ALoads the second value from address location 1002h to the accumulator B left shifted by 2 bits.Multiplies accumulator A & B.The result in accumulator B is stored to the memory location pointed by AR3.End the program. |

**RESULT**:

 The addition and multiplication operation is performed using TMS320C5416