

Linear Convolution using DSP Kit

Aim: To perform linear convolution of two sequences using DSP Kit.

Theory

Linear convolution is one of the fundamental operations used extensively in signal and system in electrical engineering. It has applications in areas like audio processing, signal filtering, imaging, communication systems and more. In simple terms, linear convolution is the process of combining two signals or functions to produce a third signal or function. Formally, the linear convolution of two functions $f(t)$ and $g(t)$ is defined as: The formula for linear convolution of two discrete signals $x[n]$ and $h[n]$ is given by:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n - k]$$

In the context of linear convolution in DSP, this operation is applied to digital signals. DSP systems utilize algorithms to perform convolution efficiently, often leveraging Fast Convolution methods to handle large datasets and real-time processing.

Procedure

1. Set Up New CCS Project

Open Code Composer Studio.

Go to File → New → CCS Project.

Target Selection: Choose C674X Floating point DSP, TMS320C6748.

Connection: Select Texas Instruments XDS 100v2 USB Debug Probe.

Name the project and click Finish.

2. Write and Configure the Program

Write the C code for generating and storing a sine wave, configuring it to access data at specified memory locations.

Assign the input X_n and filter H_n values to specified addresses:

X_n : Start at 0x80010000, populate subsequent values at offsets like 0x80010004 for each additional input.

H_n : Start at 0x80011000 with similar offsets for additional values.

Lengths of X_n and H_n should be defined at 0x80012000 and 0x80012004, respectively.

3. Configure Output Location in Code

In the code, configure the output to store convolution results at specific memory addresses starting from 0x80013000, with each result at an offset of 0x04.

4. Save the Program

Go to File → Save As and save the code with a filename like main.c.

Remove any default main.c program that might exist in the project.

5. Build and Debug the Program

Select Debug to build and load the program on the DSP.

Once the build is complete, select Run to execute.

6. Execute and Verify Output

In the Debug perspective, click Resume to run the code.

Use the Memory Browser in Code Composer Studio to verify the output at the memory location 0x80013000:

Check 0x80013000 for the first convolution result, 0x80013004 for the second, and so on.

Cross-check the values with the expected convolution results for accuracy.

Program

```
//#include<fastmath67x.h>

#include<math.h>

void main()
{
    int *Xn,*Hn,*Output;
    int *XnLength,*HnLength;
    int i,k,n,l,m;
    Xn=(int *)0x80010000; //input x(n)
    Hn=(int *)0x80011000; //input h(n)
    XnLength=(int *)0x80012000; //x(n) length
    HnLength=(int *)0x80012004; //h(n) length
    Output=(int *)0x80013000; // output address
    l=*XnLength; // copy x(n) from memory address to variable l
    m=*HnLength; // copy h(n) from memory address to variable m
```

```

for(i=0;i<(l+m-1);i++) // memory clear
{
Output[i]=0; // o/p array
Xn[l+i]=0; // i/p array
Hn[m+i]=0; // i/p array
}
for(n=0;n<(l+m-1);n++)
{
for(k=0;k<=n;k++)
{
Output[n] =Output[n] + (Xn[k]*Hn[n-k]); // convolution operation.
}
}
}

```

Result

Performed Linear Convolution using DSP Kit.

Observation

Xn

0x80010000 - 1

0x80010004 - 2

0x80010008 - 3

Hn

0x80011000 - 1

0x80011004 - 2

XnLength

0x80012000 - 3

HnLength

0x80012004 - 2

Output

0x80013000 - 1

0x80013004 - 4

0x80013008 - 7

0x8001300C - 6