



TP 5 : ENVIRONNEMENT CODE COMPOSER STUDIO "CCS"
CONFIGURATION DE L'ENVIRONNEMENT CCS & UTILISATION DES DSP

Objectif : L'objectif essentiel de ce deuxième TP est toujours de se familiariser avec les outils Code Composer Studio v3.3 de Texas Instruments. Nous nous intéressons donc à l'utilisation de l'environnement CCS pour le développement d'un système DSP et l'analyse d'un programme et des résultats produits ; par simulation dans notre cas.

I- DSP et optimisation des calculs

Comme nous l'avons mentionné avant, le DSP (processeur de traitement numérique de signal) est un microprocesseur optimisé pour exécuter des applications de traitement numérique du signal (filtrage extraction de signaux,...etc.) le plus rapidement possible.

De nombreux algorithmes de traitement du signal ont besoin d'effectuer des multiplications suivies d'une addition. Les DSP accélèrent ce genre de calcul en fournissant des instructions capables de multiplier deux nombres et d'en additionner un troisième en une seule fois (fonction très utilisée des les calculs d'asservissement et de filtrage). L'instruction de ce type la plus connue est l'instruction Multiply And Accumulate (MAC).

La majorité des DSP calculent exclusivement avec des nombres entiers ; permettant une grande vitesse de traitement des données. Un additionneur entier est en effet beaucoup plus simple qu'un additionneur à virgule flottante. Cependant, certains DSP possèdent des unités de calcul en virgule flottante comme le TMS320C67x. Les instructions de manipulation de bit sont aussi très courantes. Par conséquent, pour plus d'efficacité dans certaines applications numériques, les DSP peuvent générer nativement un format de données spécial : les nombres flottants par blocs.

Les DSP sont aussi optimisés pour exécuter des boucles le plus rapidement possible. En effet, il n'est pas rare que les programmes du traitement du signal utilisent des boucles, notamment pour parcourir des tableaux. Accélérer ces boucles est donc un enjeu majeur de la conception des DSP (instructions spéciales, capables d'effectuer un test et un branchement en un seul cycle d'horloge). Ainsi, les DSP sont capables de gérer des boucles FOR en un seul cycle d'horloge,...etc. En plus, l'accès à la mémoire est aussi particulièrement optimisé sur les DSP (plusieurs accès mémoires simultanés).

II- Exercices : Les produits de convolution

Comme vous le savez, un récepteur numérique (élément final d'une liaison de télécommunication ou de radar) doit le plus souvent procéder au calcul de produits de convolution. Le calcul est plus ou moins complexe selon la méthode employée. Cette complexité peut se mesurer par deux grandeurs : le nombre d'opérations à effectuer et la quantité de mémoire nécessaire. Les opérations à effectuer sont des multiplications, des additions ou des soustractions.

A titre d'exemple, cet opérateur (produit de convolution) nous permet de calculer la valeur discrète d'un signal en sortie d'un filtre : on aura par exemple $y[n]$ la sortie filtrée du signal d'origine $x[n]$ par un filtre de réponse impulsionnelle $h[n]$. Son expression mathématique est la suivante : **$y[n] = x[n] * h[n] = \sum_k x[k] h[n - k]$** .

$X[n]$ est de M points séquence, $h[n]$ est de N points séquence, et $y[n]$ est de $(M+N-1)$ points séquence.

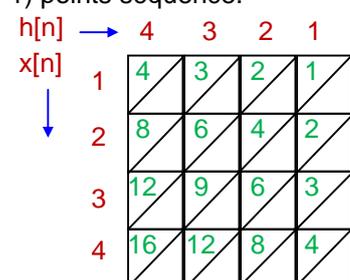
Exercice 1 : implémentation d'une **convolution linéaire**

Soit : $x[n] = \{1, 2, 3, 4\}$ et $h[n] = \{4, 3, 2, 1\}$

$M = 4$: taille de la séquence entière d'entrée des échantillons.

$N = 4$: taille de la séquence impulsionnelle.

$i = M + N - 1 = 7$: $i = 0, 1, 2, 3, 4, 5, 6, 7$ (taille de la sortie $y[i]$).



Théoriquement, nous obtenons : $y[i] = \{4, 11, 20, 30, 20, 11, 4\}$

Vérifier ces résultats en utilisant CCS v3.3 (étapes mentionnées au TP précédent) et le programme ci-dessous.

```

/* programme pour implémenter une Convolution linéaire */
#include<stdio.h>
#include<math.h>
int m=4; /*taille de la séquence d'entrée des échantillons*/
int n=4; /*taille de la séquence impulsionnelle*/
int i=0,j;
int x[8]={1,2,3,4,0,0,0,0}; /* entrée de la séquence des échantillons*/
int h[8]={4,3,2,1,0,0,0,0}; /*séquence d'échantillons d'impulsion à la fin du tampon 'm' et 'n' zéros*/
int y[8];

main()
{
int i;
for(i=0;i<m+n-1;i++)
{
y[i]=0;
for(j=0;j<=i;j++)
y[i]+=x[j]*h[i-j];
}
/*Affichage des résultats*/
for(i=0;i<m+n-1;i++)
printf("%d\n",y[i]);
}

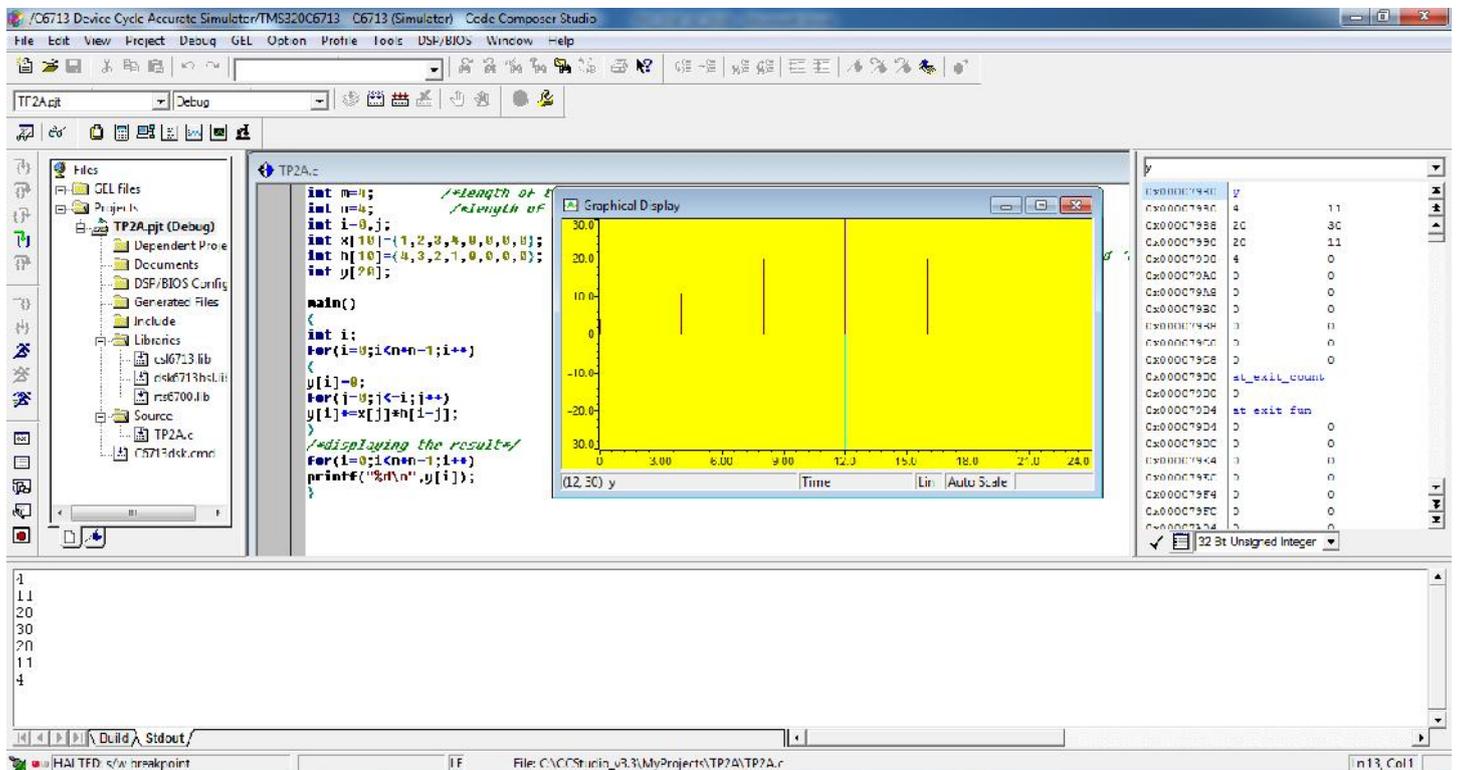
```

Address	Value	Comment
0x00007850	y	
0x0000785C	4	11
0x00007864	20	30
0x0000786C	20	11
0x00007874	4	0
0x0000787C	x*1	
0x0000787C	0	0
0x00007884	y*2	
0x00007884	0	0
0x0000788C	h*1	
0x0000788C	0	0
0x00007894	_look	
0x00007894	0	0
0x0000789C	_unlock	
0x0000789C	0	0
0x000078A4	memory_size	
0x000078A4	1024	
0x000078AC	sys_tree	
0x000078AC	33443	
0x000078B4	init	
0x000078B4	0	

Property	Value
Display Type	Single Time
Graph Title	Graphical Display
Start Address	y
Acquisition Buffer Size	CC
Index Increment	1
Display Data Size	3C
DSP Data Type	Fixed signed integer
Q-value	C
Composing Rate (Hz)	1
Plot Data From	Left to right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	C
Axes Display	On
Time Display Unit	s
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Bar
Grid Style	Zero Line
Cursor Mode	Data Cursor

Property	Value
Display Type	Single Time
Graph Title	Graphical Display
Start Address	y
Acquisition Buffer Size	25
Index Increment	1
Display Data Size	25
DSP Data Type	8-bit signed integer
Q-value	C
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	r
Axes Display	On
Time Display Unit	s
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Bar

The screenshot shows the CCS v3.3 interface. The main window displays the source code for TP2A.c, which implements a linear convolution. The console window shows the output of the program, which matches the expected results: 4, 11, 20, 30, 20, 11, 4. The variable watch window shows the memory addresses and values for the arrays x, h, and y.



Exercice 2 : implémentation d'une **convolution linéaire avec manipulation des données via adressage mémoire.**

Refaire l'exemple précédent en utilisant la méthode de manipulation des données via mémoire (programme ci-dessous).

```

#include<stdio.h>
#include<math.h>

#define xn 4
#define hn 4
void main()
{
    int *x,*h,*y,i,n,k;

    x = (int *)0x80010000;
    h = (int *)0x80020000;
    y = (int *)0x80020500;

    for(i=0;i<(xn+hn-1);i++)
    {
        y[i]=0;
        x[xn+i]=0;
        h[hn+i]=0;
    }

    for(n=0;n<(xn+hn-1);n++)
    {
        for(k=0;k<=n;k++)
        {
            y[n] = (y[n]) + ((x[k])*(h[n-k]));
        }
    }
}

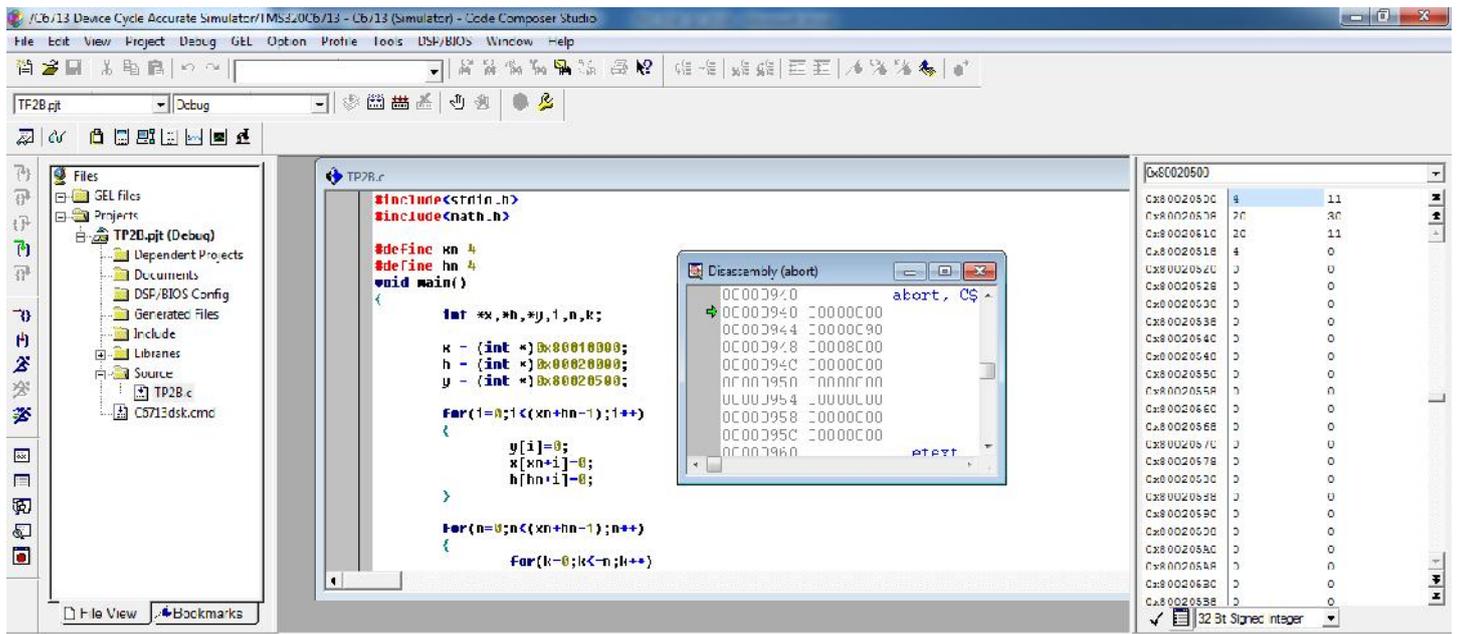
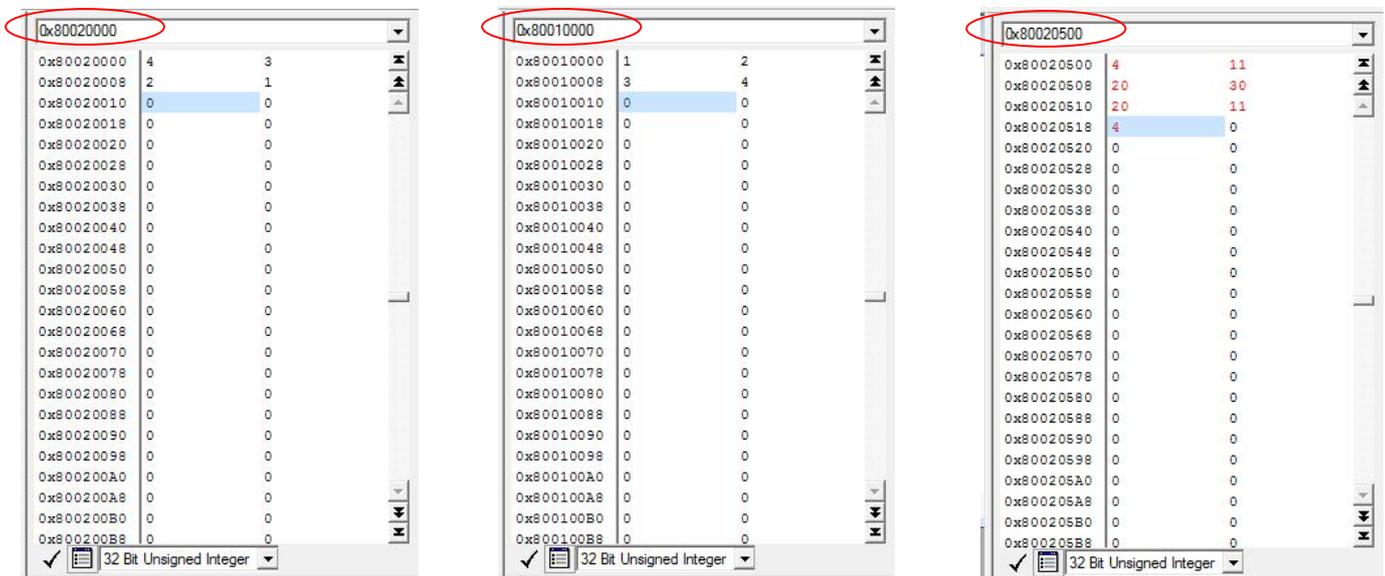
```

Entrer les valeurs de x et h puis Reload program... et visualiser les résultats (via les adresses mémoires).

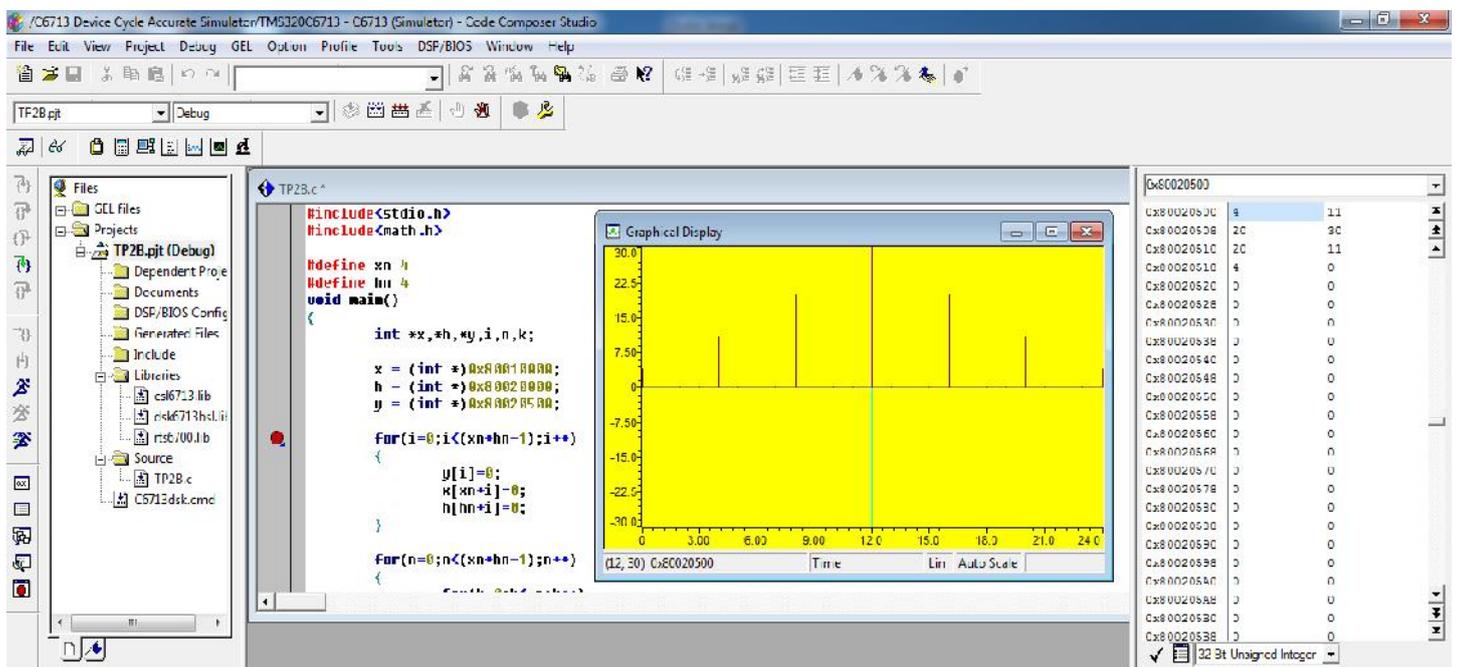
$x[n] = \{1, 2, 3, 4\}$

$h[n] = \{4, 3, 2, 1\}$

Théoriquement, nous obtenons les mêmes résultats trouvés précédemment : $y[n] = \{4, 11, 20, 30, 20, 11, 4\}$



Remarque : Concernant le message restant (1 Warning), il est possible de surmonter ce message : **Project Build Options Linker Stack Size (-stack) : 0x400 ok**



Exercice 3 : implémentation de la **convolution circulaire**

En bref, la convolution circulaire de deux séquences $x_1[n]$ et $x_2[n]$ est donnée par l'expression mathématique suivante :

$$X_1[n] * x_2[n] = x.$$

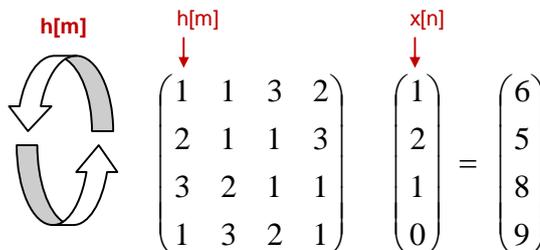
A titre d'exemple, prenons l'exemple suivant :

$$m = 4$$

$$n = 3$$

$$h[m] = \{1, 2, 3, 1\}$$

$$x[n] = \{1, 2, 1\}$$


$$\begin{matrix} h[m] & & h[m] & & x[n] \\ \downarrow & & \downarrow & & \downarrow \\ \begin{pmatrix} 1 & 1 & 3 & 2 \\ 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \end{pmatrix} & & \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix} & = & \begin{pmatrix} 6 \\ 5 \\ 8 \\ 9 \end{pmatrix} \end{matrix}$$

Théoriquement, nous obtenons les mêmes résultats trouvés précédemment : $y[m] = \{6, 5, 8, 9\}$

/ programme pour implémenter la convolution circulaire */*

```
#include<stdio.h>
```

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

```
void main()
```

```
{
printf(" Enter la taille de la première séquence\n");
scanf("%d",&m);
printf(" Enter la taille de la deuxième séquence\n");
scanf("%d",&n);
printf(" Enter la première séquence\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf(" Enter la deuxième séquence\n");
for(j=0;j<n;j++)
scanf("%d",&h[j]);
if(m-n!=0)
```

```
/*Si les tailles des séquences ne sont pas égales*/
```

```
{
if(m>n) /* La faible séquence par zéro*/
```

```
{
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++)
```

```
a[j]=h[n-j]; /*Convolution circulaire*/
```

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

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

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

```
{
y[k]=0; /*Décalage circulaire */
```

```
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];
```

```
}
```

```
}
```

```
/*Affichage des résultats*/
```

```
printf(" La convolution circulaire est : \n");
```

```
for(i=0;i<n;i++)printf("%d \t",y[i]);
```

```
}
```

Address	Value	Comment
0x0000C790	y	
0x0000C790	6	5
0x0000C798	8	9
0x0000C7A0	0	0
0x0000C7A8	0	0
0x0000C7B0	0	0
0x0000C7B8	0	0
0x0000C7C0	0	0
0x0000C7C8	0	0
0x0000C7D0	0	0
0x0000C7D8	0	0
0x0000C7E0	0	0
0x0000C7E8	0	0
0x0000C7F0	0	0
0x0000C7F8	0	0
0x0000C800	0	0
0x0000C808	x2	
0x0000C808	0	1
0x0000C810	2	1
0x0000C818	0	0

32 Bit Signed Integer

Graph Property Dialog

Graph Title	Graphical Display
Start Address	y
Acquisition Buffer Size	15
Index Increment	1
Display Data Size	15
DSP Data Type	8-bit signed integer
Q-value	0
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Bar
Grid Style	Zero Line

OK Cancel Help

C6713 Device Cycle Accurate Simulator/TMS320C6713 C6713 (Simulator) Code Composer Studio

TP2C.pjt Debug

```

/* programme pour implémenter la convolution circulaire */
#include <stdio.h>
int m,n,x[30],h[30],v[30],i,j,k,x2[30],a[30];
void main()
{
    printf(" Enter la taille de la première séquence\n");
    scanf("%d",&n);
    printf(" Enter la taille de la deuxième séquence\n");
    scanf("%d",&n);
    printf(" Enter la première séquence\n");
    for(i=0;i<n;i++)
        scanf("%d",&x[i]);
    printf(" Enter la deuxième séquence\n");
    for(j=0;j<n;j++)
        scanf("%d",&h[j]);
    if(n!=0)
        /*si les tailles des séquences ne sont pas égal
    <
    if(n>n)
        /* la faible séquence par zéros/
    <
    for(i=n;i<n;i++)
        h[i]=0;
  
```

Enter la deuxième séquence

```

1
2
1
La convolution circulaire est :
6 5 8 9
  
```

File: C:\CCStudio_v3.3\MyProjects\TP2C\TP2C.c

Quick Watch

Recalculate

Name	Value	Ty...	Ra
y	0x0000C790	int...	he
[0]	6	int	de
[1]	5	int	de
[2]	8	int	de
[3]	9	int	de
[4]	0	int	de
[5]	0	int	de

Add To Watch

Close

Réalisé par : Dr. S. ABADLI.

