Lab 1 IPT 2022-2023

Development of applications for the Analog Devices BF 533 using the Visual DSP++ IDE

1. Introduction

Visual DSP 5.0 is the dedicated integrated development environment (IDE) used for the development of applications for Analog Devices DSPs. Applications can be developed using C/C++ or assembly. The IDE includes an editor, a compiler and debugging facilities. The Visual DSP's main window is shown in Fig.1.

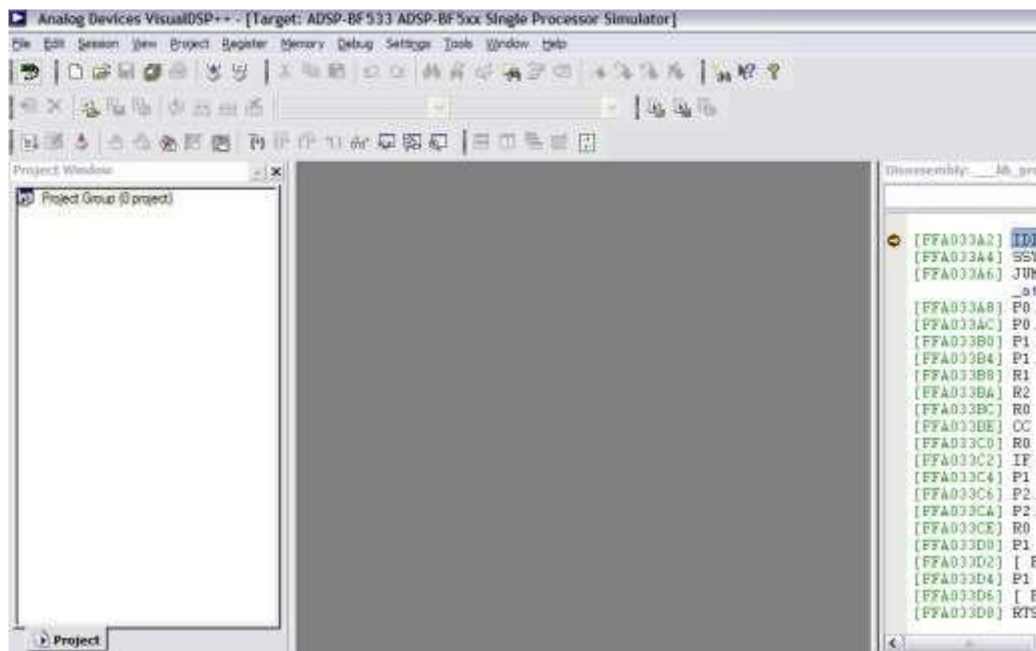

Fig.1 Visual DSP's graphical user interface

2 The structure of a simple application in Visual DSP

The stages involved in the development of a Visual DSP project illustrated using a simple application that calculates the dot product of two vectors.. Using the following notations:

$\vec{a}=(a_1,a_2,...a_N)$

$\vec{b}=(a_1,a_2,...a_N)$

the dot product in defined by:

$\vec{a} \cdot \vec{b} = a_1b_1 + a_2b_2 + ... + a_Nb_N$

The steps needed for building an application that calculates the product are listed below:

1. *Create your own folder on D:\TPI\Lab1*
2. Access the menu **File-> New-> Project**
3. *Set the parameters as in the figure below:*

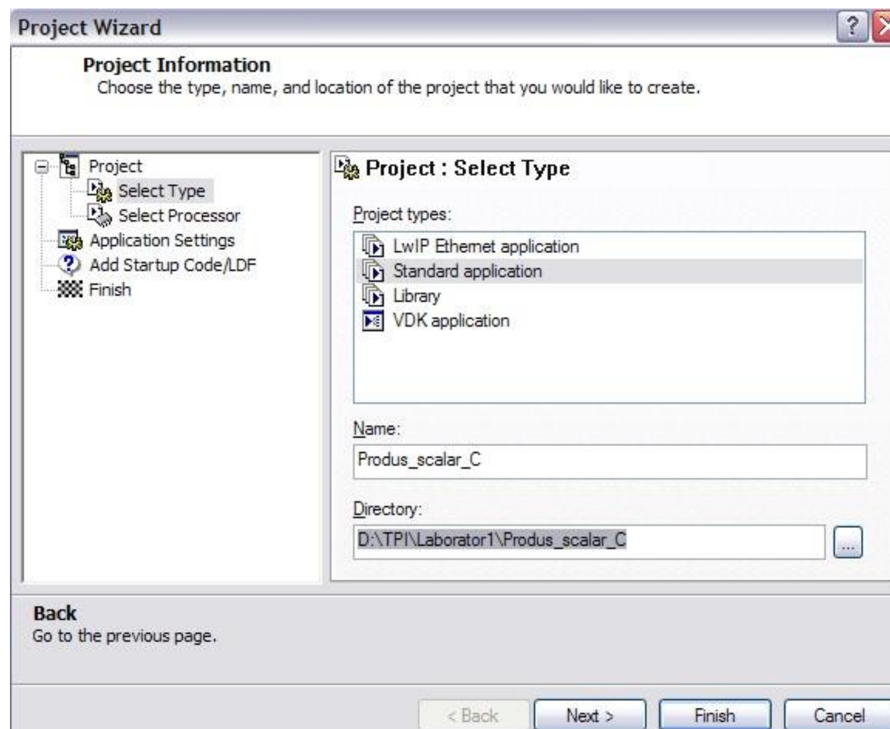

Fig.2 Parameters of an application

4. *Select the type of the processor:*

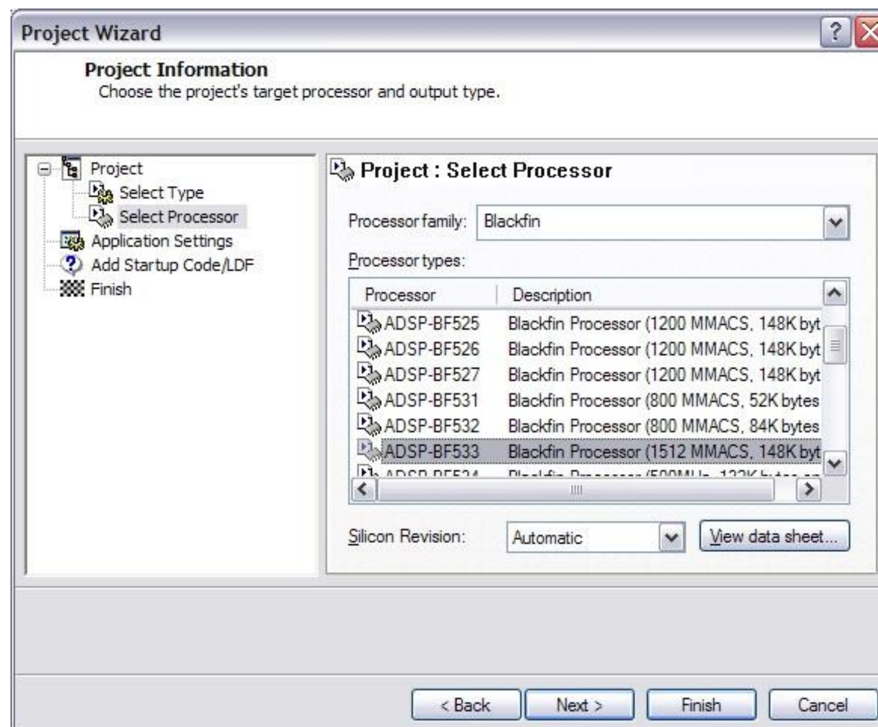

Fig.4 Selection of the processor

5. *Click on* **Next** *and, in the following window, choose the parameters as in Fig 5.* Such a choice corresponds to an empty project that will be populated with function written in the C programming language.

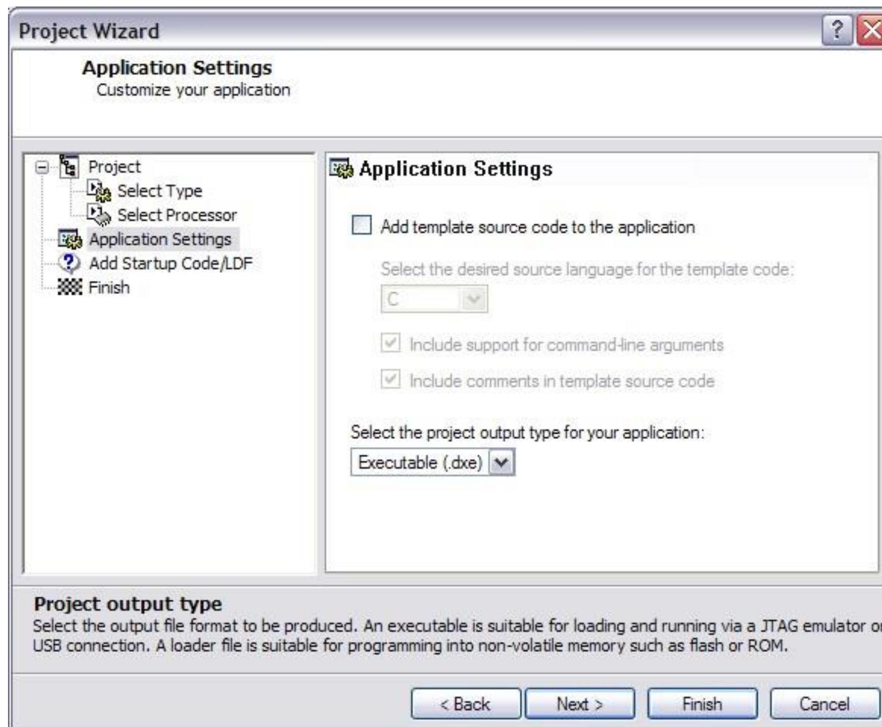

Fig.5 Type of the application and of the project

6. *Click on* **Finish,** *the window that will be displayed is shown in Fig. 6; the newly created project must be saved before continuing.*

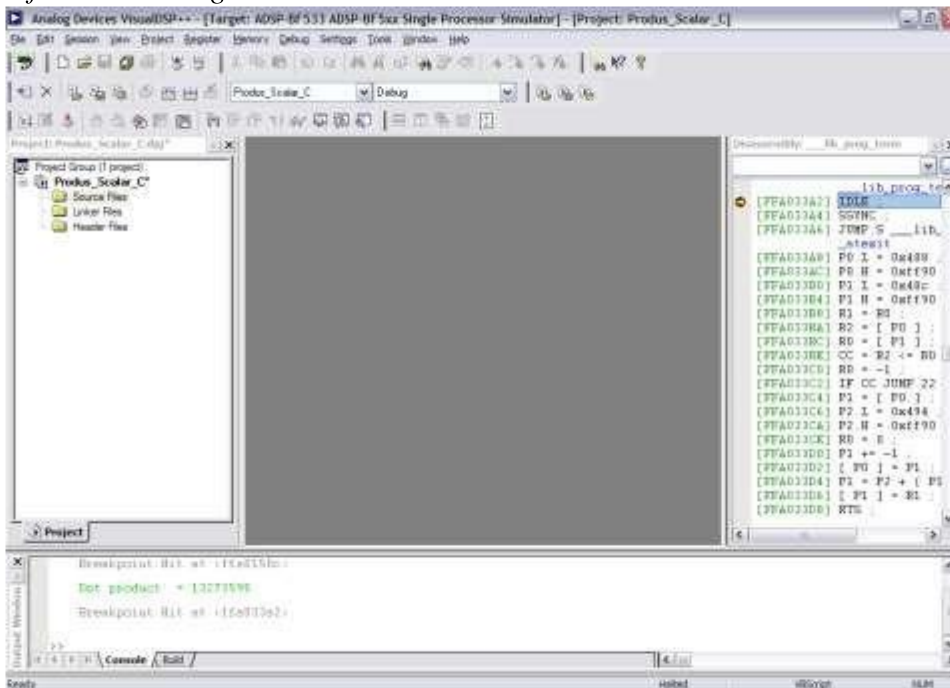

Fig.6 The Produs_Scalar_C Visual DSP project

7. *Access the menu* **File-> New-> File***; in the newly created file insert the code included in appendix 1 and save the file under the name* ***produs_scalar_c.c***. The code includes a function

having the prototype *int a_prod_b(int *, int *);* the function can be used to calculate the dot product.

8. *Insert the newly created file into the project:*

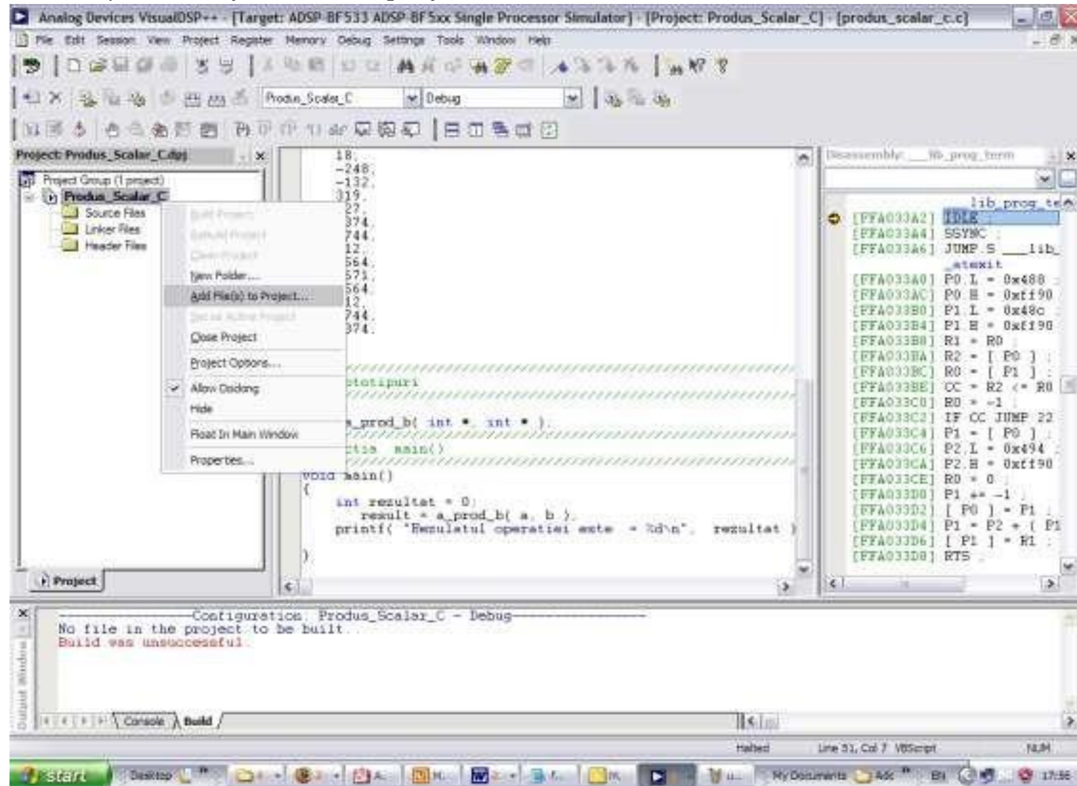

Fig.7 File insertion into the project

9. *Go to* **Project->Build project** ; the compiler will indicate a compiling error due to the fact that the function a_prod_b does not have a body yet:

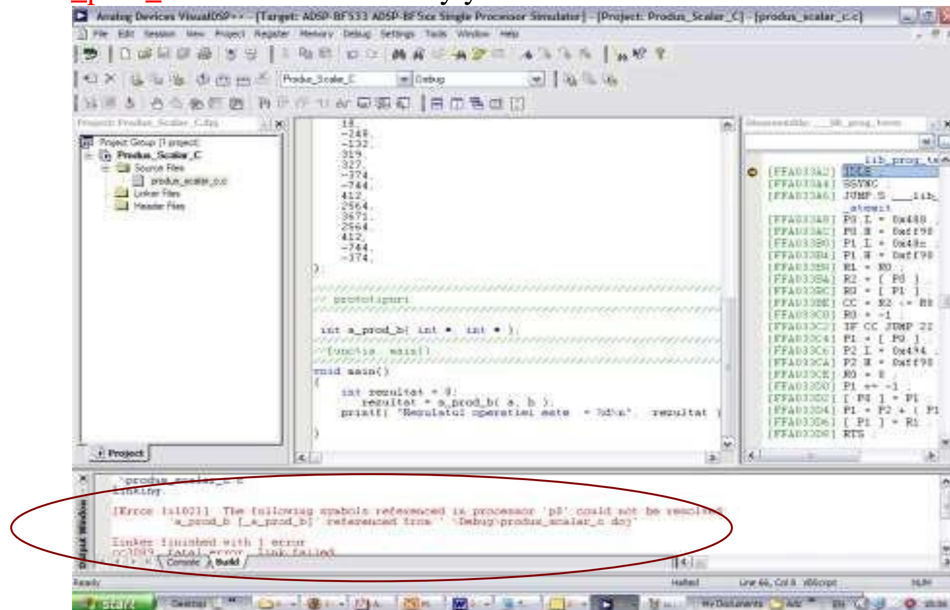

Fig.8 Compiling errors in Visual DSP

10. *Repeat 7 in creating a new file produs_scalar_implementare.c that will include the code included in appendix 2*. The structure of the project modifies accordingly (fig.9). Run the program and note the obtained result.

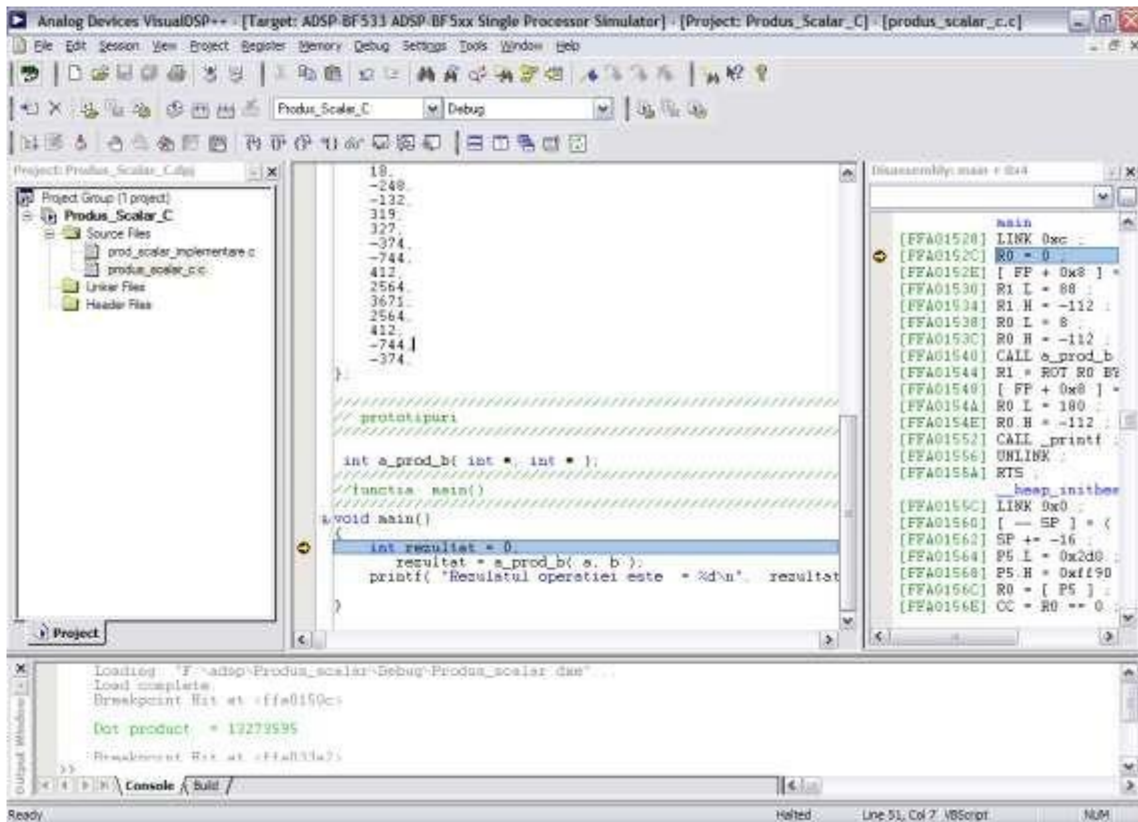

Fig.9 ***The structure of the final project***

Visual DSP includes facilities for computing the execution speed of the code for an eventual optimization. These facilities can be accessed through **Tools-> Linear Profiling.**

11. *Access the menu indicated above and select* ***New Profile*;** this allows a user to inspect the execution time associated to some function in percent (**Execution percent**) or in clock periods (**Sample Count**).
12. Filtering can be performed in only to measure the performance of user predefined functions; in order to do this go to the menu **Properties** and define the filter as shown below.

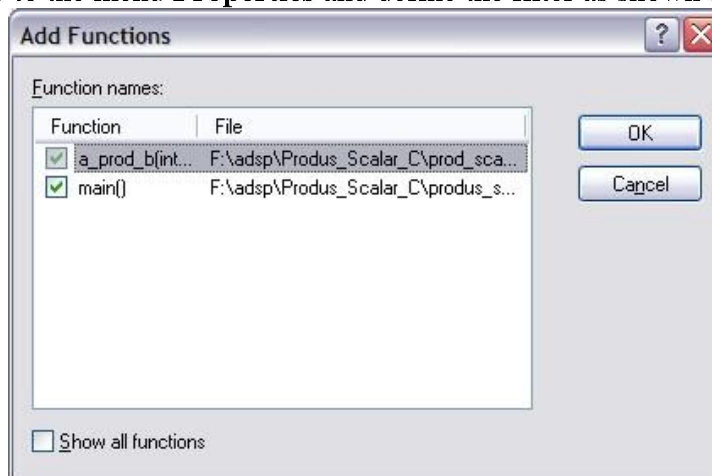

Fig.10 Filters for the linear profiling tool

13. *Recompile the project and run it again.* The profiling data should be show as illustrated în fig.11.

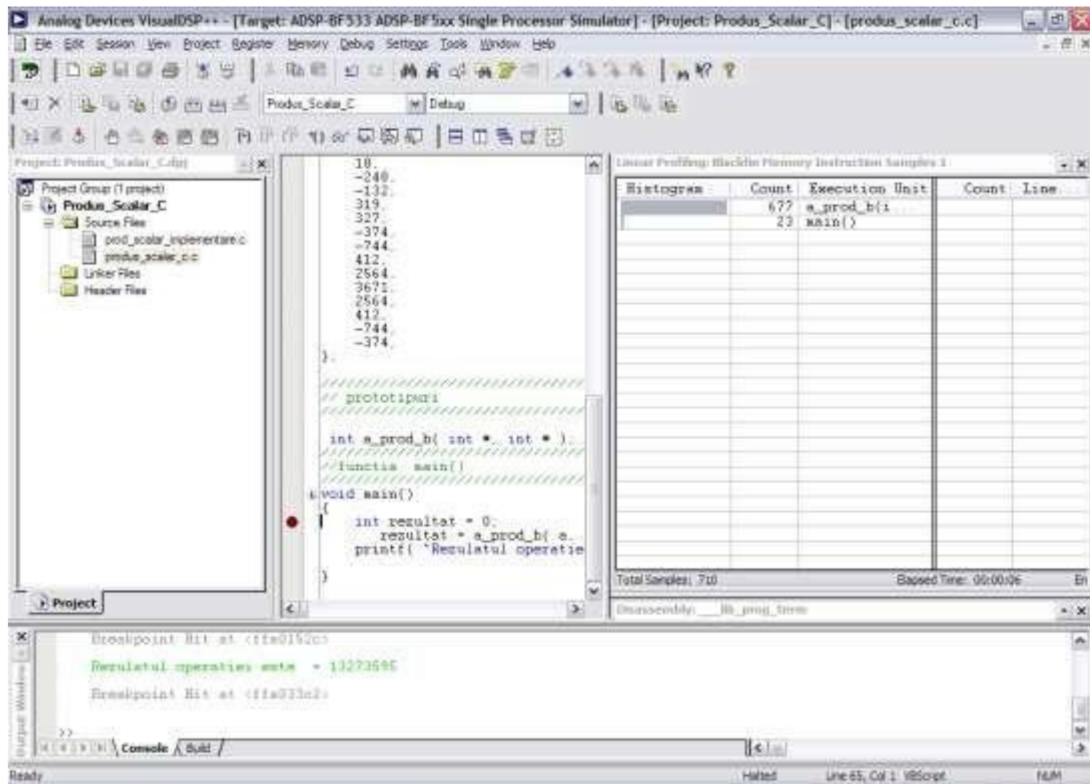

Fig.11 The linear profiling tool included in Visual DSP 5.0

3 Combined assembly and C language programming

Even if Visual DSP includes a powerful compiler, full advantage of the knowledge of the processor's architecture can be achieved by using combined assembly language and C based instructions for decreasing the execution time. Assembly instruction can be directly inserted in the C code or they can be included in separate files having the implicit extension **.asm.** Such files must be included in the project prior to the compilation process.

As an example an external function will be defined; its prototype will be included in the C file and its body, shown in appendix 3, will be included in a separate file. The following steps must e followed:

1. *Similarly to 7* and *10 create a new file named prod_asm.asm*; in the newly created file insert the code included in appendix 3
2. *Insert the file into the project.*
3. Modify the file that holds the main function by inserting*:*

```
/////////////////////////////////////////////////////////////////////////////
// prototipuri
/////////////////////////////////////////////////////////////////////////////

int a_prod_b( int *, int * );
extern int a_prod_b_asm( int *, int * );
```

4.*Modify the body of the main function as follows:*

```
void main()
{
            int tip_implementare=1;   int rezultat
= 0;       if (tip_implementare==0)
 rezultat = a_prod_b( a, b );     else
                  rezultat = a_prod_b_asm( a, b );
            printf( "Rezultatul operatiei este  = %d\n",  rezultat );

}
```

5. Recompile the project and run it again. Note the results obtained in the profiling window. What do you observe?

4 Visual DSP facilities for working with images

The Visual DSP IDE includes a includes an utility called **Image Viewer** that allows an image to be loaded either from the internal DSP memory (or on the memory installed on the development board) or from an external device (typically an image stored on the HDD of a PC). Image Viewer can be launched through the menu **View-> Debug Windows-> Image Viewer** and it has the graphical user interface shown in fig.12.

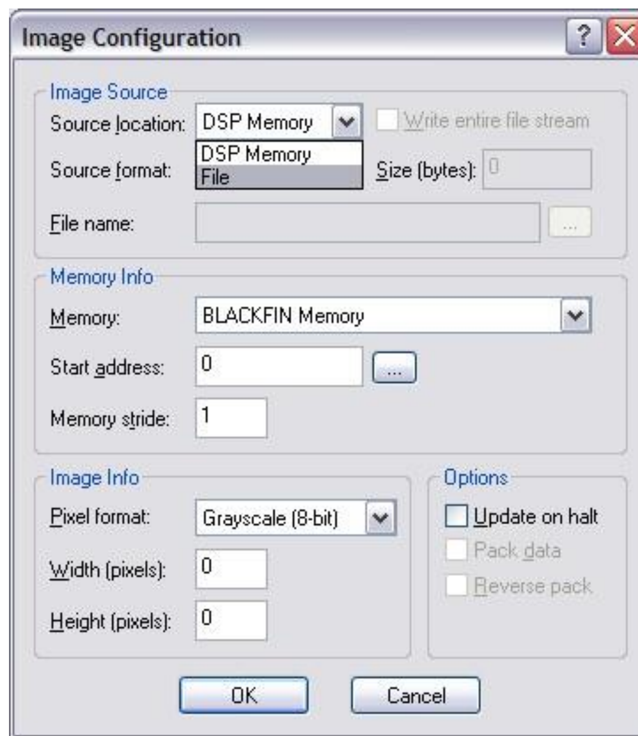

Fig.12 Image Viewer GUI

The following example shows how Visual DSP can be used for accessing and processing images.

1. *Copy from the course web site the project Image Viewer and place it on your own directory D:\ Lab 1* (Fig.13)

2. The application uses 2 variables : an input variable named *imgInput* and an output variable *imgOutput*, both being defined as unidimensional tables with 16384 positions each. The size of the tables is sufficient for holding images having 128x128 pixels.
3. For initializing the two variables and for allowing the images to be displayed two Image Viewer windows must be employed. Their settings are shown in Fig. 14.

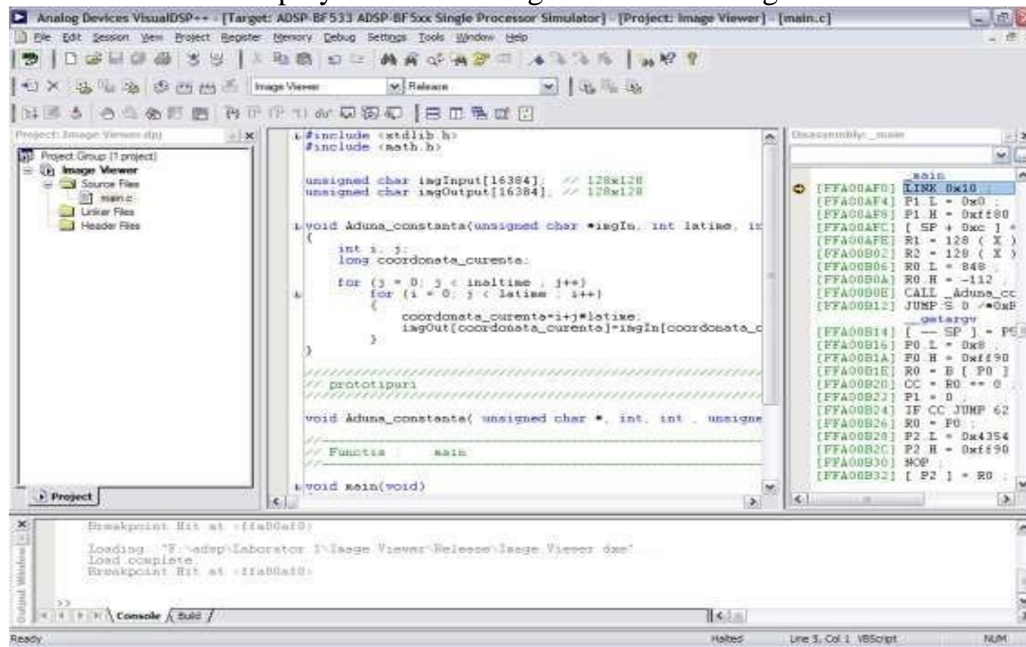

Fig.13 The Image Viewer application

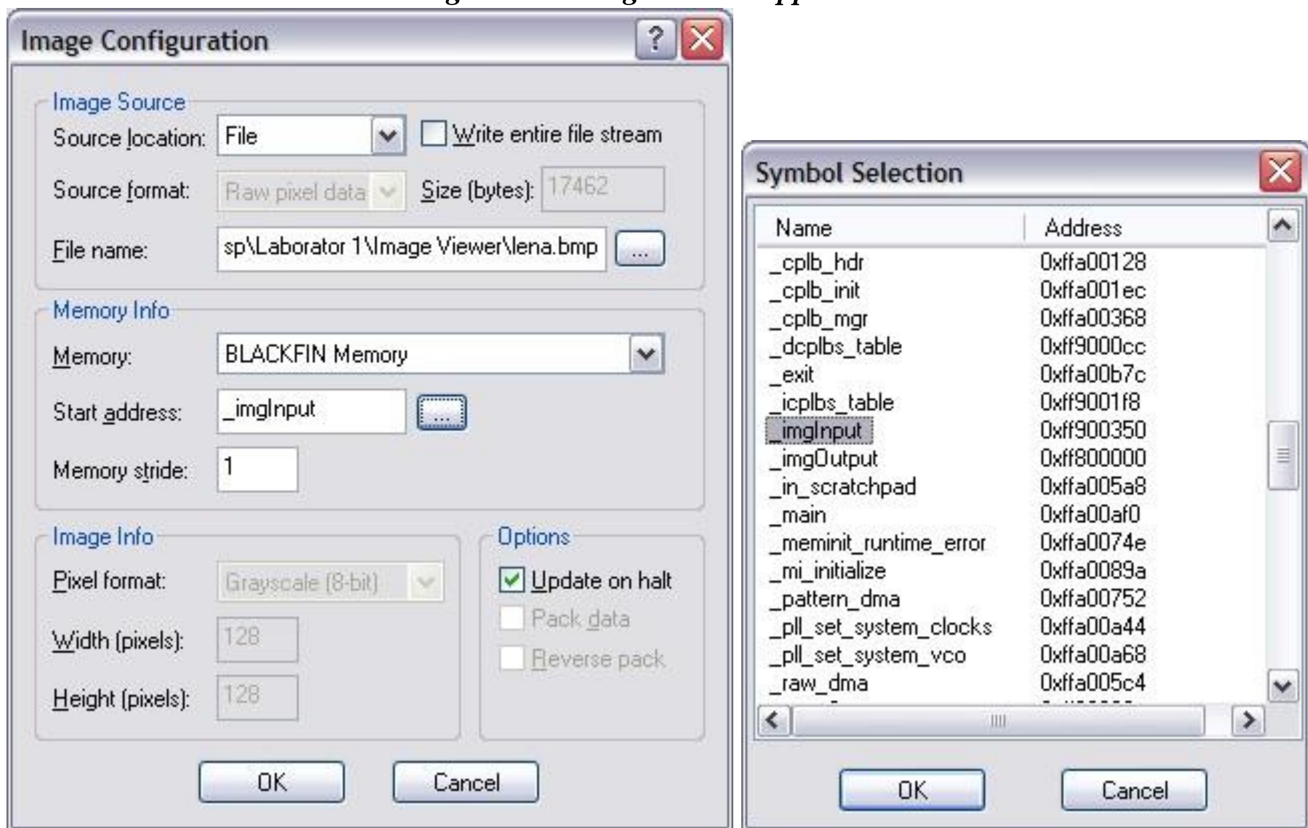

Fig.14 Settings for the imgInput variable

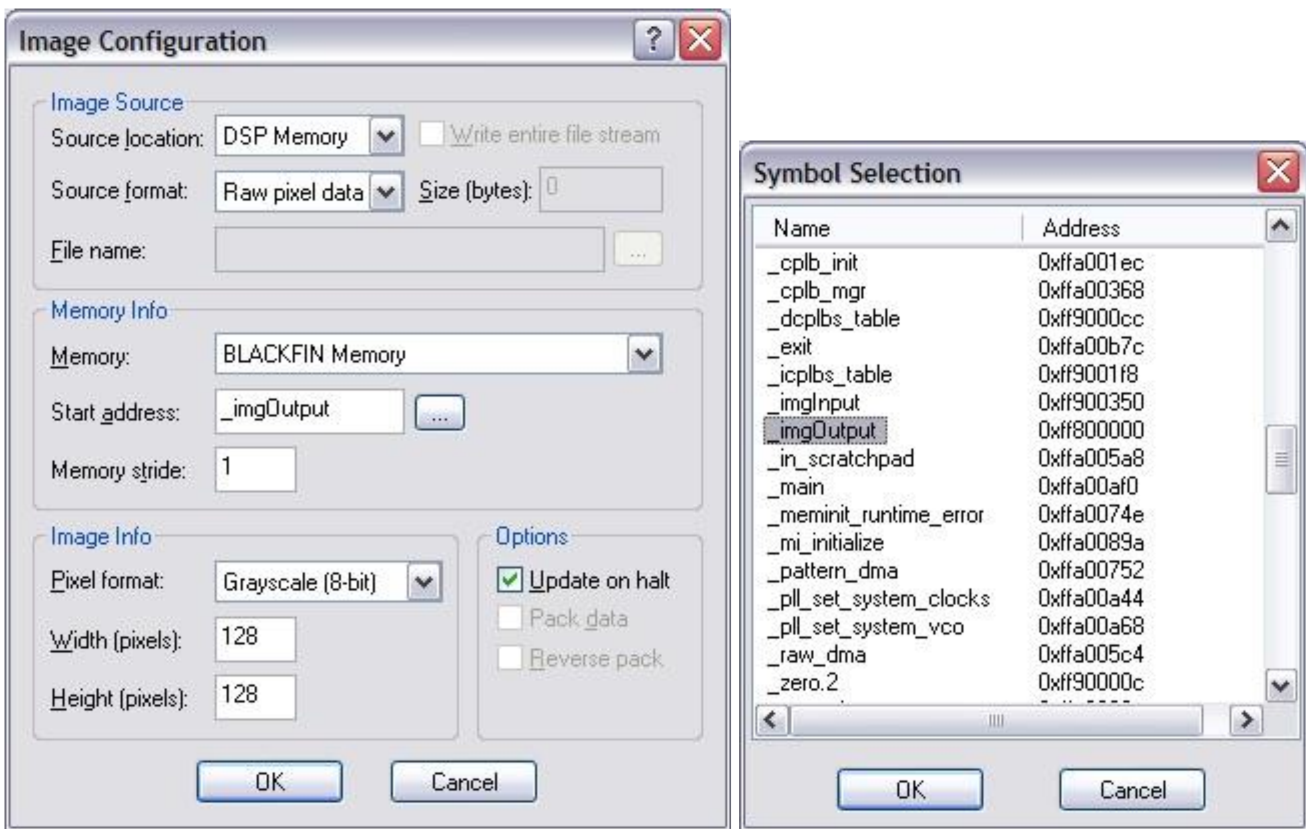

Fig.15 Settings for the imgOutput variable

4. Compile the project (**F7 can be used as a shortcut**), and reinitialize the Image Viewer windows:

5. Run the program (**F5** can be used as a shortcut**)** and analyze the result after 1 min by stopping the application (**Shift+F5**)

5 Exercises

1. Using the first application as an example, propose an implementation for an application written in C that will compute the sum of two vectors.
2. Modify the image viewer application for allowing histogram equalization to take place. The body of the histogram equalization function is included in appendix 4. After inserting the code identify the role of each instruction and insert commentaries into the code. Run the program and note the result on the output image.

Appendix 1. The produs_scalar_c.c file

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
////////////////////////////////////////////////////////////////////////////////
// variabile globale
////////////////////////////////////////////////////////////////////////////////

int a[] = {
        66,
        140,
        48,
        4,
        -121,
        -178,
        -146,
        14,
        231,
        383,
        328,
        -15,
        -607,
        -1286,
        -1827,
        6160,
        -1827,
        -1286,
        -607,
        -15
};

int b[] = {
        86,
        115,
        -77,
        -103,
        24,
        187,
        18,
        -248,
        -132,
        319,
        327,
        -374,
        -744,
        412,
        2564,
        3671,
        2564,
        412,
        -744,
        -374,
};
```

```
///////////////////////////////////////////////////////////////////////////////
// prototipuri
///////////////////////////////////////////////////////////////////////////////
int a_prod_b( int *, int * );
/////////////////////////////////////////////////////////////////////////////// //functia  main()
///////////////////////////////////////////////////////////////////////////////
void main()
{
	int rezultat = 0;
	rezultat = a_prod_b( a, b );
	printf( "Rezultatul operatiei este  = %d\n",  rezultat );

}
```

Appendix 2. The produs_scalar_implementare.c file

```
const int N = 20;
//////////////////////////////////////////////////////////////////////////////
// int a_prod_b( int*, int* )
//////////////////////////////////////////////////////////////////////////////

int a_prod_b( int *a, int *b )
{
   int i;
   int rezultat = 0.0;

   for( i=0; i<N; i++ )
   {
                        rezultat += ( a[i] * b[i] );
            }

   return( rezultat );
}
```

Appendix 3. The prod_asm.asm file

```
.section program;
// cod ce va fi scris în memoria program

.global _a_prod_b_asm;
//functia a_prod_b_asm este vizibila in intreg proiectul. Caracterul "_" este o directiva ce indica likeditorului ca este //o
functie scrisa de utilizator

//inceput corp functie  _a_prod_b_asm:

// Registrii interni
    P0 = R0;
    I0 = R1;
    P1 = 19;
    R0 = 0;
    NOP;
    R1 = [P0++];
    R2 = [I0++];
//Bucla
    LSETUP (begin_loop, end_loop) LC0 = P1;

    begin_loop:     R1 *= R2;
                            R2 = [I0++];
    end_loop:       R0= R0 + R1 (NS) || R1 = [P0++] || NOP;

    R1 *= R2;
    R0 = R0 + R1;
    RTS;

_a_prod_b_asm.end:
//sfarsit corp functie
```

\

Appendix 4. Body of the histogram equalization function

```
void Egalizare_Histo (unsigned char * intrare, unsigned char *iesire, int latime, int inaltime)
{
        int i, j,k;
        int marime=latime*inaltime;
        float suma;          unsigned
char * p1, *p2;

        p1=&intrare[0];

        for (i=0; i<16384;i++)
                                histo_ini[*p1++]++;



                for (i=0; i<256;i++)
                                histo_ini[i]=histo_ini[i]/marime;

        for (i=0; i<256;i++)
        {
                suma=0.0;
                for (k=0;k<i;k++)
                {
                                suma=suma+histo_ini[k];
                }
                histo_modif[i]=suma;
        }
        p1=&intrare[0];
        p2=&iesire[0];   for
(i=0; i<16384;i++)
                                *p2++=histo_modif[*p1++]*255;
}
```