anadigm

App Note - 208

Creating a DLL from AnadigmDesigner2 C-code

Rev: 1.0.2

Date: 10th July 2009

This application note contains a total 3 files, if you have only this pdf text document, go here
http://www.anadigm.com/sup_AppNoteLib.asp
to find the .AD2 file and example source code.

To successfully use the technique explained below you must have AnadigmDesigner2 and
Microsoft Visual C++ ver 6.0

TABLE OF CONTENTS

Purpose

This document describes how to create a Dynamic Link Library (DLL) from the AnadigmDesigner2 generated C-code files. The DLL allows another program to call both a primary configuration function and reconfiguration function from the C-code

Creating the DLL

1. Open Visual C++ v6.0 and go to Projects/New… in the menus.
2. Select “Win32 Dynamic-Link Library” and enter a project name (see figure 1).

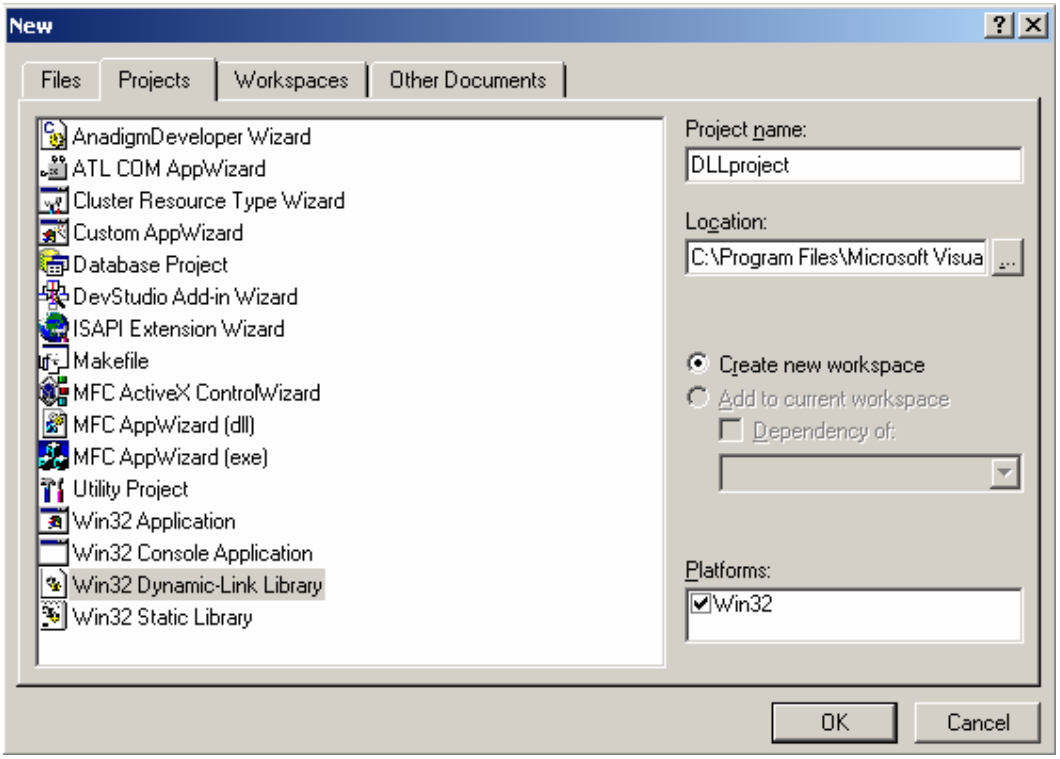

Figure 1: Creating a new DLL

3. Click on “OK” and then select “A DLL that exports some symbols” (see figure 2).

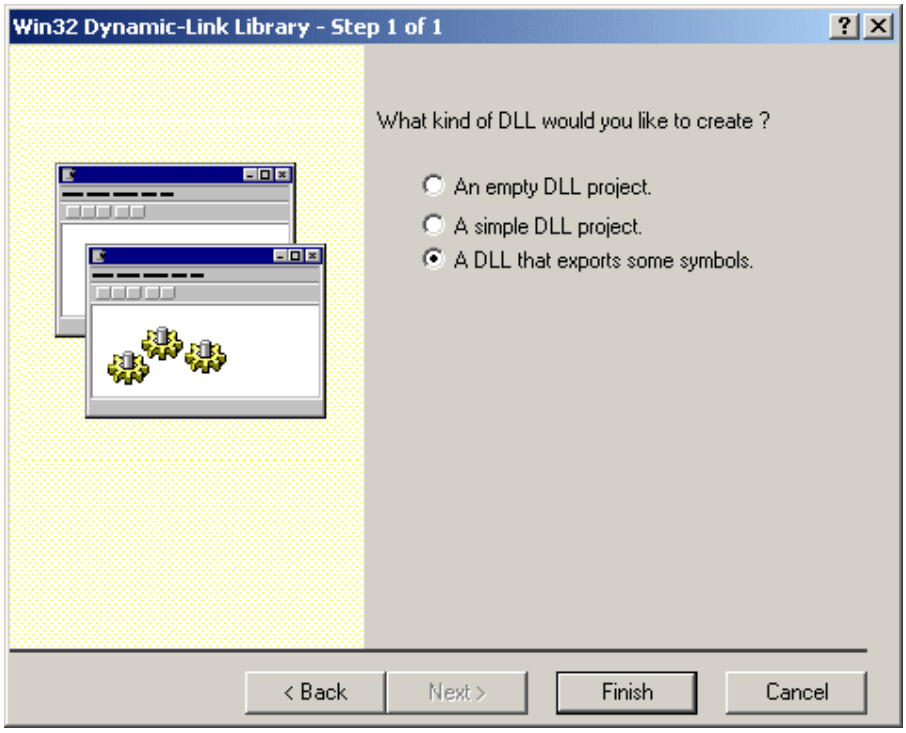

Figure 2: Select a DLL that exports symbols

4. Click on “Finish” to get the window shown in figure 3. Click on “OK”.

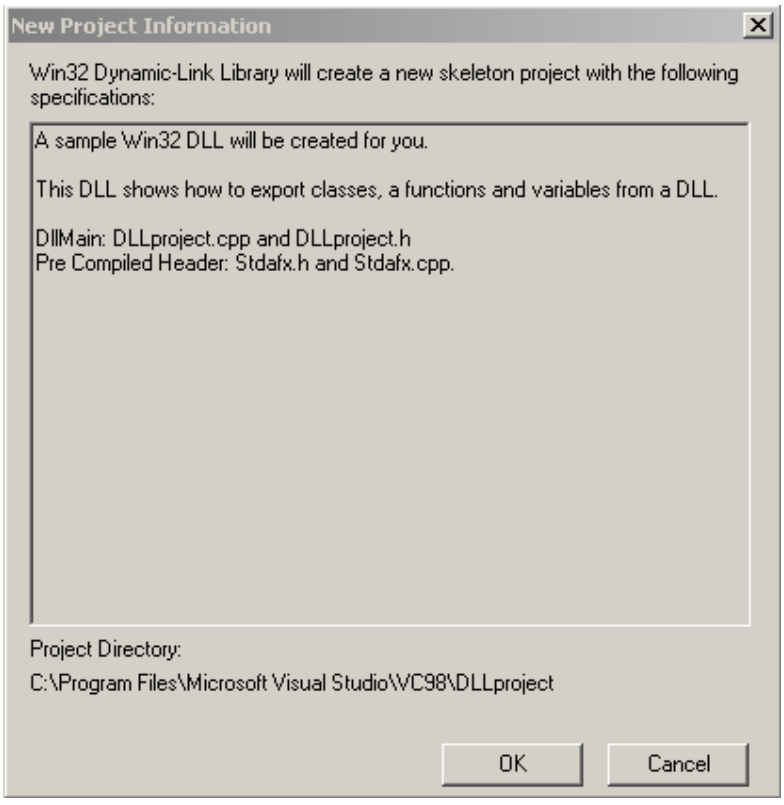

Figure 3: Confirm the project information

5. Now open AnadigmDesigner2 (called AD2 from now on) and create the required circuit. For the purpose of this document, the circuit shown in figure 4 will be used which consists of a single GainInv CAM. Save the AD2 circuit in the Visual C project area.

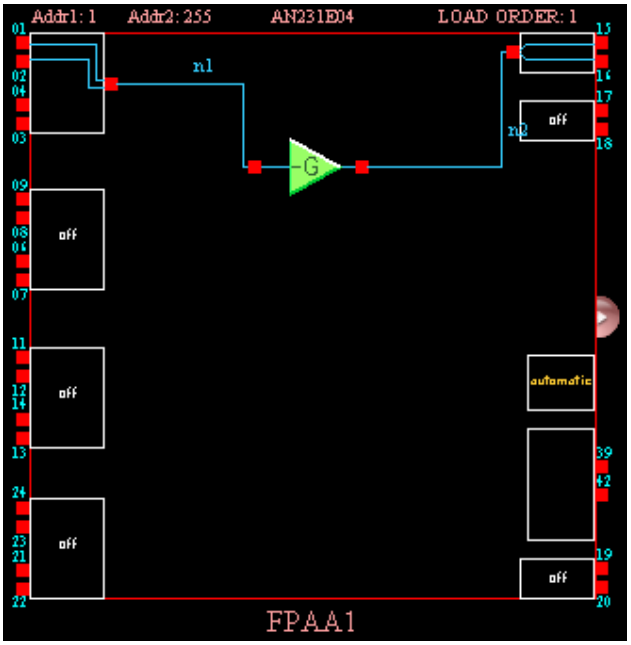

Figure 4: The AD2 circuit

6. Go to Dynamic Config./Algorithmic method… in the AD2 menus to get the window shown in figure 5.

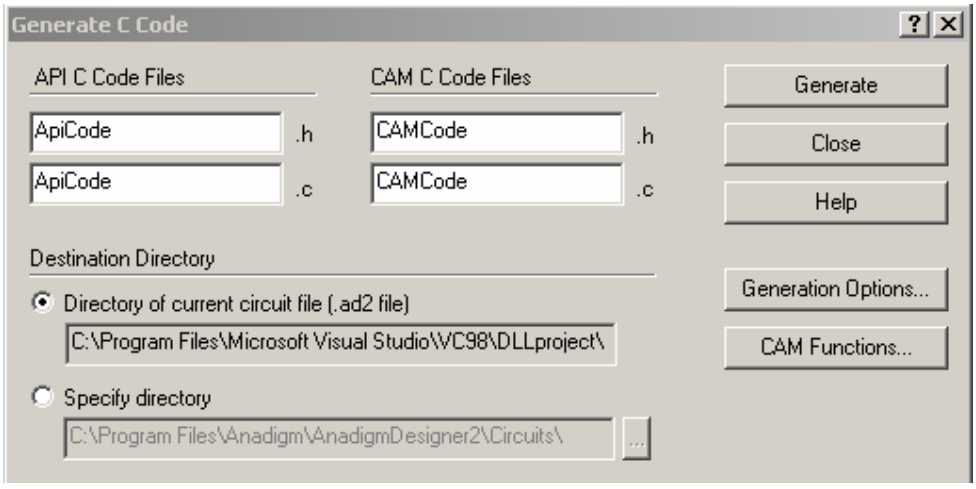

Figure 5: Generating C-code from AD2

7. Click on CAM Functions… to get the window shown in figure 6. Expand the CAM tree on the left to show all of the programmable CAMs and cells. Click on each one in turn and check the box on the right for each function that is required, and uncheck the box for functions that are not required. Note that all functions are selected by default and the DLL will still work if all functions are left selected. The only reason for deselecting functions that are not required is to reduce the amount of C-code generated by AD2.

8. Click on “OK” and then click on “Generate”. Note from figure 5 that the C files get generated into the Visual C++ working directory because we chose to save the AD2 circuit there. If the AD2 circuit is not saved in the project area, then select “Specify directory” in the window shown in figure 5 and browse to the VC++ DLLProject area.

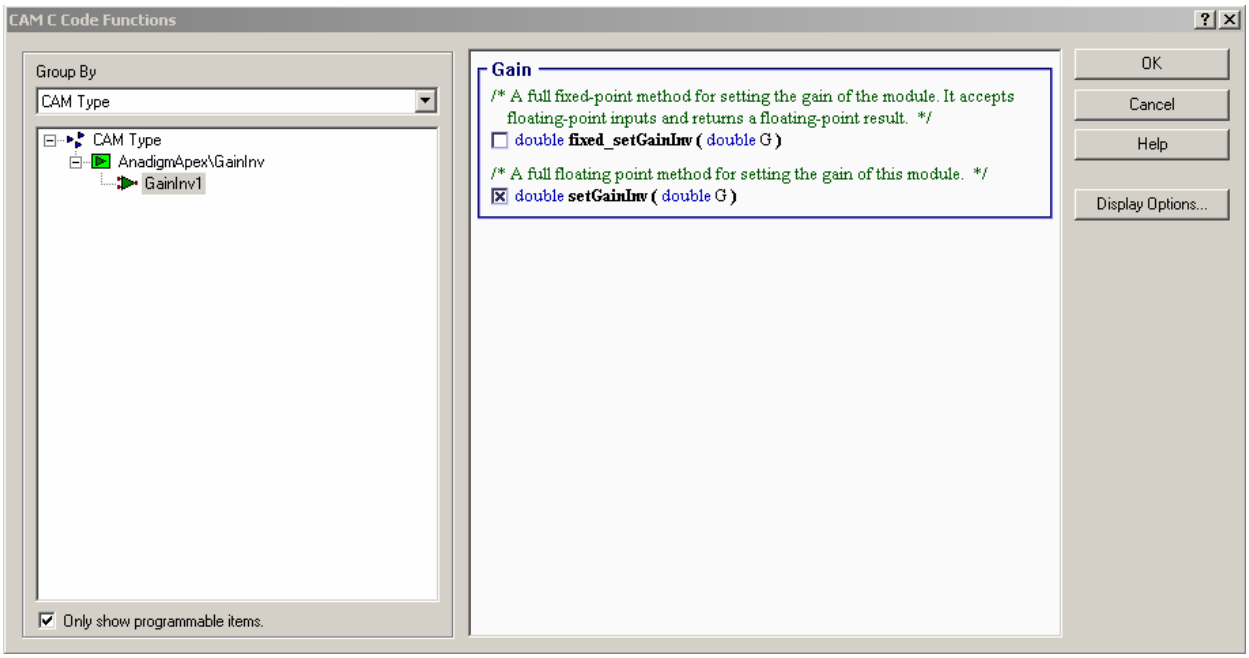

Figure 6: Selecting the CAM functions

9. Click on “Close”. 4 files should have appeared in the DLLProject area – ApiCode.c, ApiCode.h, CAMCode.c and CAMCode.h.
10. In Visual C++ go to Project/Add To Project/Files… in the menus to get the window shown in figure 7. Select the 4 files generated by AD2 and click on “OK”.

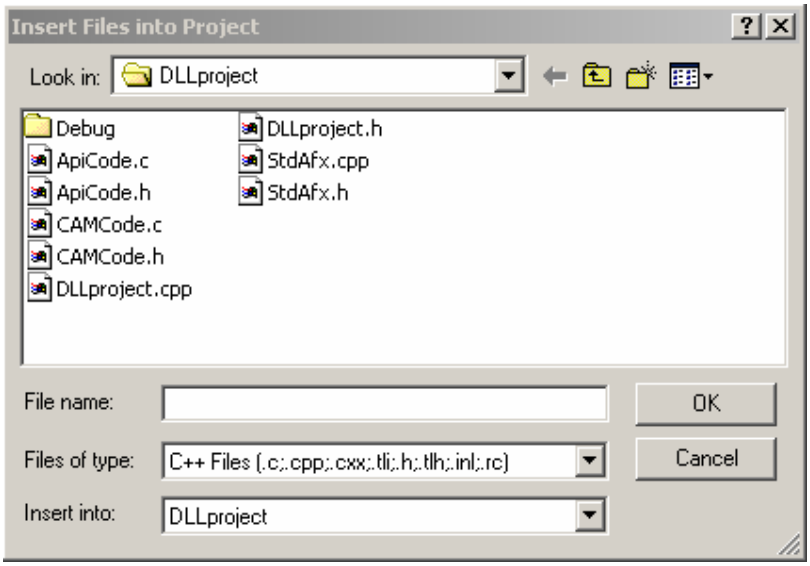

Figure 7: Adding the AD2 C-code files to the project

11. In VC++ select the FileView tab in the workspace window on the left. Expand the directory tree to show the files. They should look like figure 8.

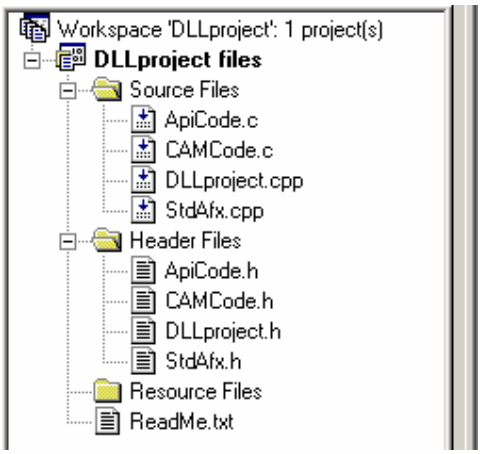

Figure 8: Workspace window

12. Double click on DLLProject.h to show the code in the edit window on the right (figure 9). Delete the following code:

```
// This class is exported from the DLLProject.dll
class DLLPROJECT_API CDLLProject {
public:
      CDLLProject(void);
      // TODO: add your methods here.
};
extern DLLPROJECT_API int nDLLProject;
DLLPROJECT_API int fnDLLProject(void);
```

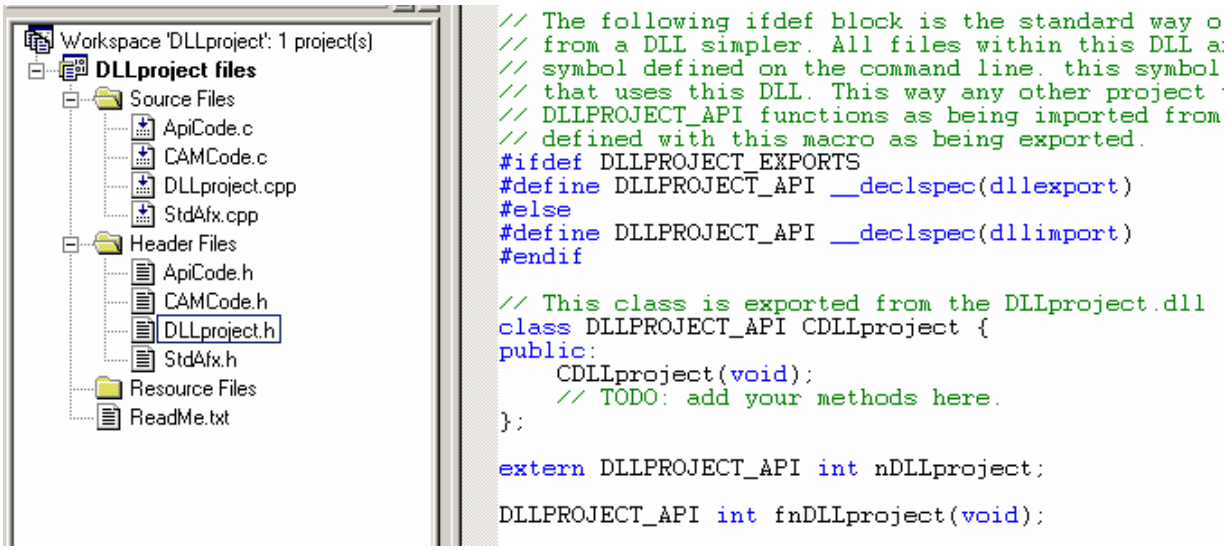

Figure 9: Project header file

13. Replace the 6 lines of REM at the top of the header file with the following code:

```
#ifndef DLLPROJECT__H
#define DLLPROJECT__H
```

14. Add the following code to the end of the header file:

```
#ifdef __cplusplus
extern "C" {
#endif

      DLLPROJECT_API int _stdcall MyFunction(IN const
short FunctionParameter);

#ifdef __cplusplus
}
#endif

#endif
```

15. The header file should now look like figure 10. In this code there is one example function called MyFunction which takes a single parameter and returns an integer. The user can add more functions here. Note that when a function takes a parameter from the Visual Basic calling program that is an integer, it must be declared as a short integer in VC++.

```
#ifndef DLLPROJECT__H
#define DLLPROJECT__H

#ifdef DLLPROJECT_EXPORTS
#define DLLPROJECT_API __declspec(dllexport)
#else
#define DLLPROJECT_API __declspec(dllimport)
#endif

#ifdef __cplusplus
extern "C" {
#endif

    DLLPROJECT_API int _stdcall MyFunction(IN const short FunctionParameter);

#ifdef __cplusplus
}
#endif

#endif
```

Figure 10: User's header file

16. Double click on DLLProject.cpp in the workspace window. Add the following includes at the top:

```
#include "apicode.h"
#include "camcode.h"
```

17. Delete the following code:

```
// This is an example of an exported variable
DLLPROJECT_API int nDLLProject=0;

// This is an example of an exported function.
DLLPROJECT_API int fnDLLProject(void)
{
      return 42;
}

// This is the constructor of a class that has been
exported.
// see DLLProject.h for the class definition
CDLLProject::CDLLProject()
```

```
{
        return;
}
```

18. Replace the deleted code with the code below:

```
DLLPROJECT_API int _stdcall MyFunction(IN const short
FunctionParameter)
{
        int variable;

        variable = 2 * FunctionParameter;
        return variable;
}
```

19. Figure 11 shows all the code in DLLProject.cpp. There is only one function defined here and it is trivial since all it does is return double the input parameter. The user can enter his own functions as required (see section 26).

```
// DLLproject.cpp : Defines the entry point for the DLL application.
//
#include "stdafx.h"
#include "DLLproject.h"
#include "apicode.h"
#include "camcode.h"

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            break;
    }
    return TRUE;
}


DLLPROJECT_API int _stdcall MyFunction(IN const short FunctionParameter)
{
    int variable;

    variable = 2 * FunctionParameter;
    return variable;
}
```

Figure 11: User's source code

20. Double click on StdAfx.h in the workspace window and place the following lines below the "#include <windows.h>" line:

```
#include "apicode.h"
#include "camcode.h"
```

21. Go to Build/Set Active Configuration… in the menus and select "DLLProject – Win32 Release" (figure 12) and click on OK.

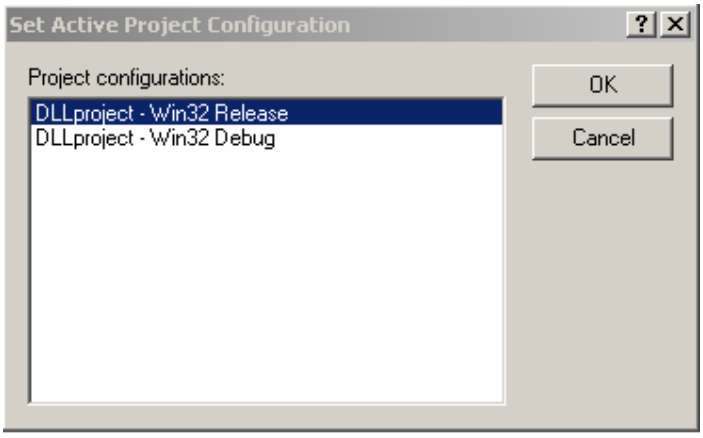

Figure 12: Set active project configuration

22. Go to Project/Settings… in the menus. Select the C/C++ tab and from the pull-down menu next to “Category”, select “Precompiled Headers”. Select the option button labelled “Not using precompiled headers” (figure 13).

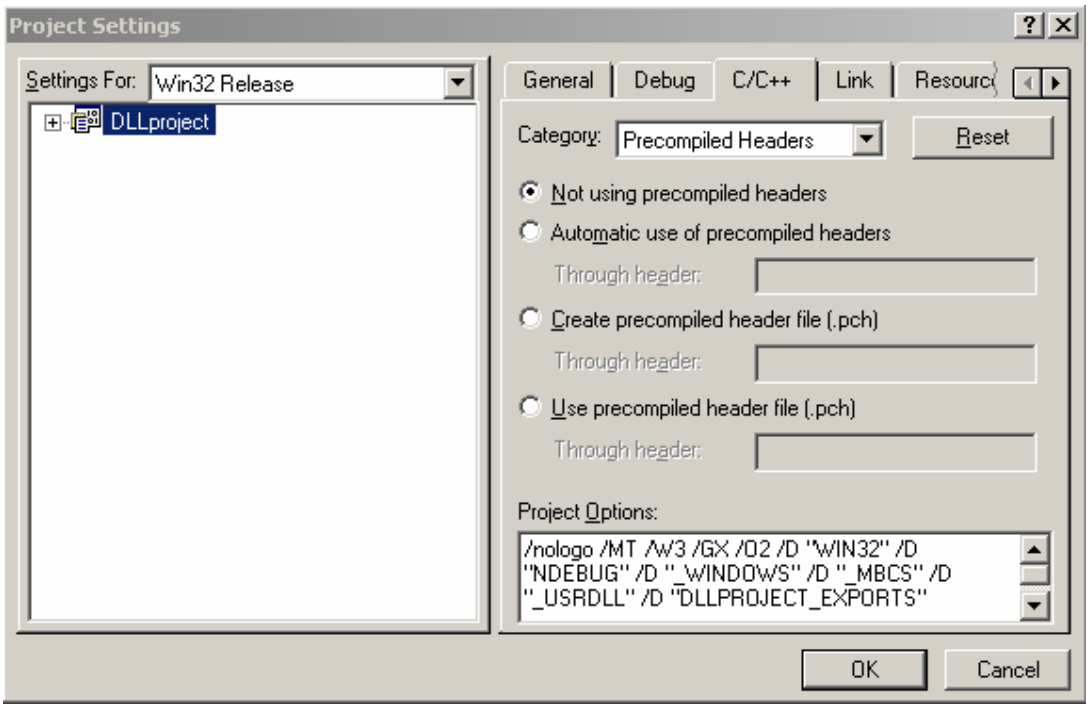

Figure 13: Unselecting precompiled headers

23. Select the Link tab, select “General” from the pull-down menu labelled “Category”. Check the box labelled “Generate mapfile” (figure 14). Then click OK.

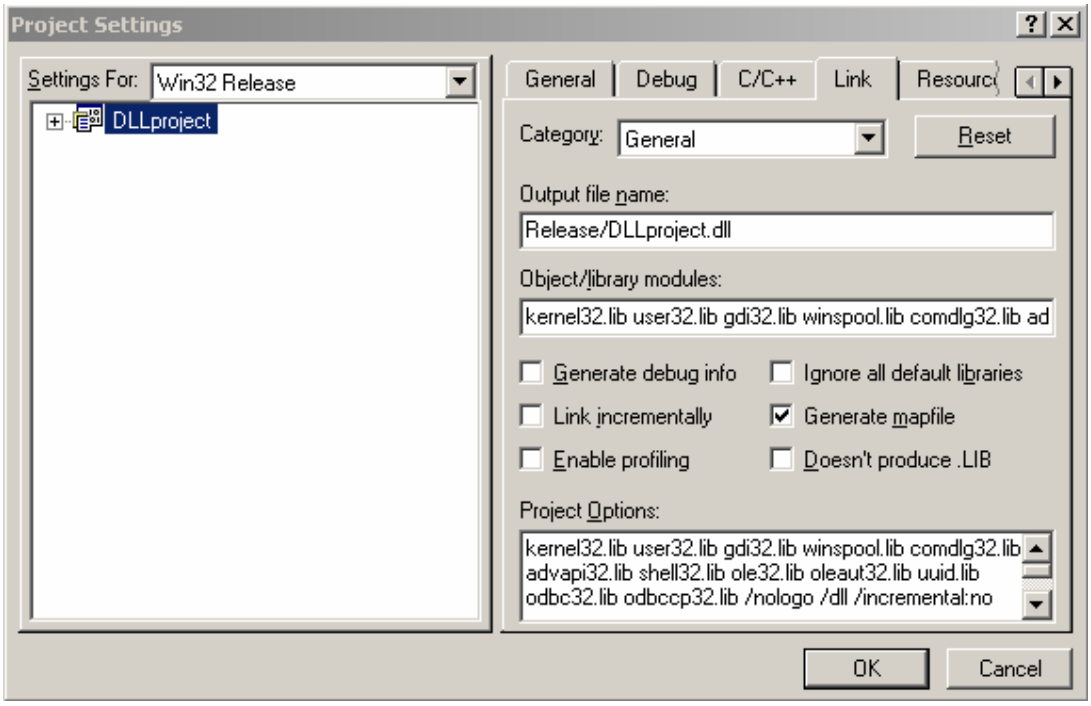

Figure 14: Selecting option to generate mapfile

24. Go to Build/Rebuild All in the menus to compile the DLL. The output from the compile should appear in the output window at the bottom (figure 15). There should be no errors. Don't worry if there are warnings from the CAMCode.c file.

```
--------------------Configuration: DLLproject - Win32 Release--------------------
Compiling...
StdAfx.cpp
Compiling...
DLLproject.cpp
Generating Code...
Compiling...
CAMCode.c
ApiCode.c
Generating Code...
Linking...
   Creating library Release/DLLproject.lib and object Release/DLLproject.exp

DLLproject.dll - 0 error(s), 0 warning(s)
```

Figure 15: Compiler output

25. The DLL can be found in the Release directory within the VC++ working directory. Also in the Release directory there should be a file called "DLLProject.MAP". Open this file with any text editor and look for the following line:

```
0001:00000010      _MyFunction@4      10001010 f   DLLProject.obj
```

This line identifies the function name that must be used by the calling function, in this example it is "`_MyFunction@4`".

26. So far only one trivial function has been written into the DLL. In order to make the DLL useful in interfacing to an FPAA, it is necessary to add functions that can build and provide access to both primary configuration and reconfiguration data. Below are some functions that do this, along with a description of what each does. The full listings for these functions can be found in the attached source code (DLLproject.cpp). The source code also contains detailed REM statements that describe what each line of the code does.

```
//this function carries out a primary configuration of the FPAA
DLLPROJECT_API void _stdcall PriConfig()

//this function returns reconfigures the FPAA circuit with a new gain
DLLPROJECT_API void _stdcall Reconfig(IN const double G1)
```

The MAP function names for these functions can be found in the .MAP file from the lines below:

```
0001:00000020      _PriConfig@0               10001020 f   DLLproject.obj
0001:00000070      _Reconfig@8                10001070 f   DLLproject.obj
```

1 Modifying the DLL

1. If the AD2 circuit needs to be changed, make the required modifications to the AD2 circuit and regenerate the C-code files as in steps 6 to 8 above. Note that if new CAM functions are required then these must be selected (figure 6).
2. It is not necessary to add these C-code files to the project again. If VC++ is already open then it will give a warning (figure 16) that the files have been modified and will ask if the files should be reloaded. Click on “Yes”.

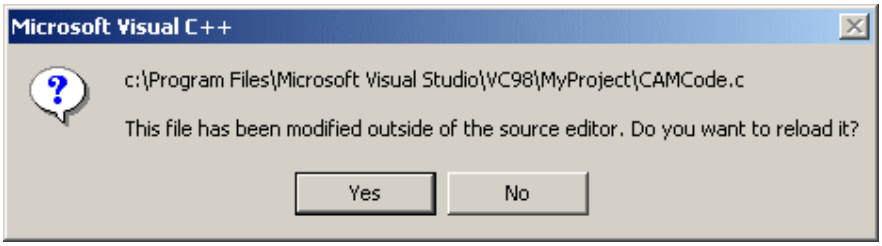

Figure 16: Reloading new AD2 C-code files

3. Recompile the project as in step 24. The new DLL file will be found in the Release directory within the VC++ working directory.
4. If any of the functions that are called from the DLL have been modified (or new ones added), the .MAP file in the Release directory must be opened with a text editor in order to find the new function names that must be used by the calling software. Note that simply changing the number or type of parameters in a function will change its name. For example, if the MyFunction() function that was used in the previous section had another parameter added as below:

```
DLLPROJECT_API int _stdcall MyFunction(IN const short FunctionParameter,
IN const short Parameter2)
```

then the function name in the .MAP file would be:

```
_MyFunction@8
```

or if the function was changed to:

```
DLLPROJECT_API int _stdcall MyFunction(IN const short FunctionParameter,
IN const double Parameter2)
```

then the function name in the .MAP file would be:

```
_MyFunction@12
```

2 Calling DLL Functions from Visual Basic

To be able to call the DLL functions from Visual Basic (VB), it is necessary to add a module to the VB project. To do this, go to the “Project” menu and select “Add Module”. Select the “New” tab and click on “Open”. You will be presented with a blank page. Enter the following code onto this page:

```
DefInt A-Z
Declare Function MyFunc Lib "DLLproject.DLL" Alias "_MyFunction@4" (ByVal G1 As Integer)
As Integer
Declare Function Pri_Config Lib "DLLproject.DLL" Alias "_PriConfig@0" ()
Declare Function Re_Config Lib " DLLproject.DLL" Alias "_Reconfig@8" (ByVal G1 As Double)
```

The 3 functions can now be called from anywhere in the VB project using the following function calls:

variable2 = **MyFunc**(variable1)

This function takes a variable and returns a variable (this is the trivial multiply by 2 function).

Pri_Config

This function takes no variables and returns no variables. It performs a primary configuration of the FPAA.

Re_Config(gain)

This function takes a variable but does not return one. It reconfigures the FPAA with a new value of gain.

go here
http://www.anadigm.com/sup_AppNoteLib.asp
to find the .AD2 file and example source code.

For more product information, please visit Anadigm® at: www.anadigm.com

Or email support@anadigm.com