ANALOG DEVICES

Technical Notes on using Analog Devices' DSP components and development tools
Contact our technical support by phone: (800) ANALOG-D or e-mail: dsp.support@analog.com
Or visit our on-line resources http://www.analog.com/dsp and http://www.analog.com/dsp/EZAnswers

Using the Expert Linker for Multiprocessor LDFs

Contributed by Maikel Kokaly-Bannourah *July 17, 2003*

Introduction

This EE-Note explains the use of the *Expert Linker* (EL) for creating Linker Description Files (LDFs) for Multiprocessor (MP) systems.

Although, this concept applies to VisualDSP++™ for all SHARC® Processor families (ADSP-21x6x and ADSP-TSxxx), the examples shown throughout this document are for the ADSP-TS101S TigerSHARC® Processor.

The example code used for this note is based on EE-167 *"Introduction to TigerSHARC Multiprocessor Systems Using VisualDSP++™"* and it was written using VisualDSP++ 3.0 Service Pack 1 for TigerSHARC (please note that *"expertlinker_fixes.zip"* must be installed prior to going through this note - see README.txt for more details).

Expert Linker Overview

The Expert Linker is a graphical tool that simplifies complex tasks such as memory map manipulation, code and data placement, overlay and shared memory creation, and C stack/heap usage. This tool provides a visualization capability enabling new users to take immediate advantage of the powerful LDF format flexibility in a very user-friendly way.

This note assumes a basic understanding of the Linker Description File as well as the way the linker utility (linker.exe) operates. For detailed information on this utility as well as the LDF, please use the VisualDSP++ on-line help. Also, refer to the *VisualDSP++™ 3.0 Linker and Utility Manual for TigerSHARC®, EE-69 "Understanding and Using Linker Description Files (LDFs)"* (for a general description on the LDF), and *EE-167* (for an explanation on the different multiprocessor linker commands).

Expert Linker LDF Wizard

The Expert Linker (EL) wizard is used to generate an LDF for new VisualDSP++ projects. However, the Expert Linker can also be used to view or modify an already existing LDF.

Open the project (*MP TS101.dpj*) attached to this note. The source code comes with no LDF, which will be created, step-by-step, through this note.

Please note that an *MPTS101_orig_ldf.txt* containing an already created LDF file is available as a reference.

Let's now get started with the creation of the LDF. First of all, to invoke the Expert Linker wizard choose from the pull-down menu as shown in Figure 1.

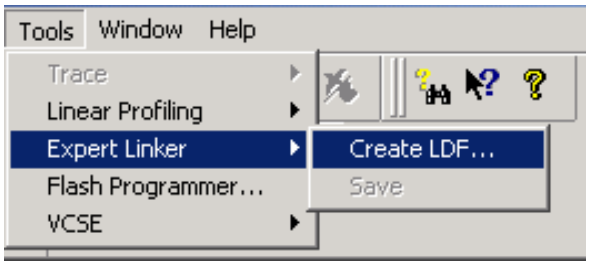

Figure 1 Invoking the Expert Linker Wizard

Figure 2 shows the start up window when first invoking the EL wizard.

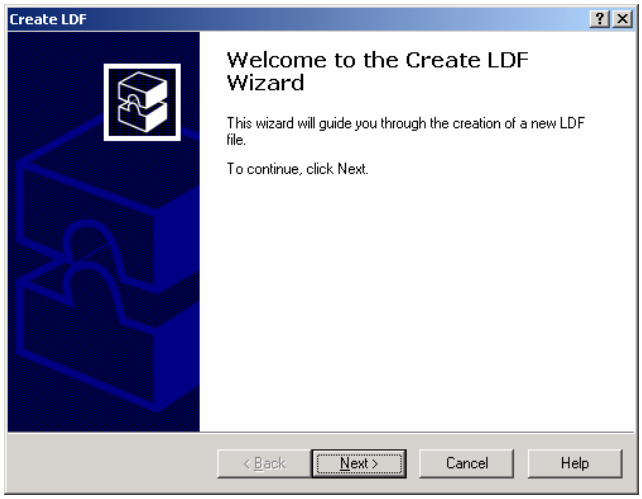

Figure 2 Expert Linker Wizard Start-up Window

Click *Next*.

Project type

At this stage, the user needs to specify the project information corresponding to the project type for which the LDF is being generated. As shown in Figure 3, the type can be C, C++, Assembly or VDK.

Note that in case a mix of assembly and C files, or any other combination, is used, the most abstract programming language should be selected. For example, for a project with C and assembly files, a C LDF should be selected. Similarly, for a C++ and C project the C++ LDF should be selected.

In this particular example, the files source code is assembly, and therefore the selected project type is also *Assembly*.

The LDF name is specified here as well, which by default uses the same as the project name.

Note that if an LDF file already exists, the user will be prompted whether to replace the existing file.

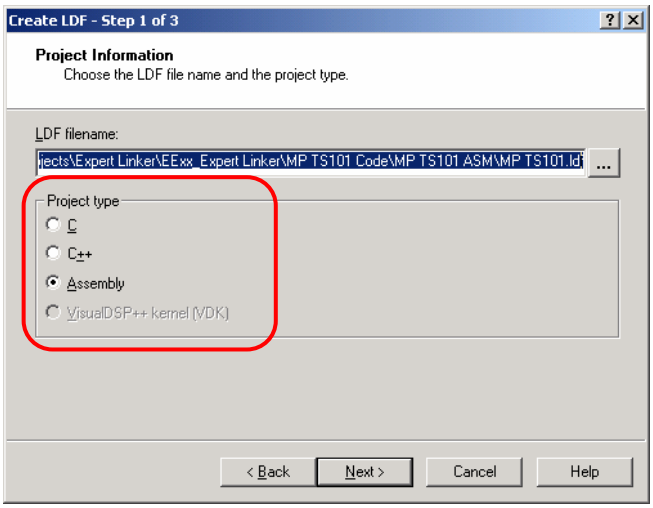

Figure 3 Project Type

Click *Next*.

Selecting an MP LDF

By default, the LDF is for single processors. Choose the *Multiprocessor* box for MP support (Figure 4).

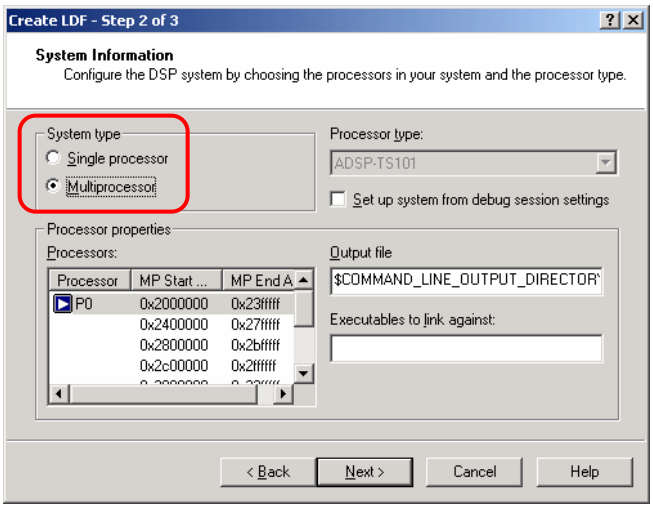

Figure 4 Multiprocessor LDF selection

Determining the Number of Processors and MP Memory Offset Values

Right click on the *Processor Properties* box to add the desired number of processors to be included in the LDF. For this particular example, a dual processor system is selected. Therefore, a

second processor (*P1*) needs to be added to the list.

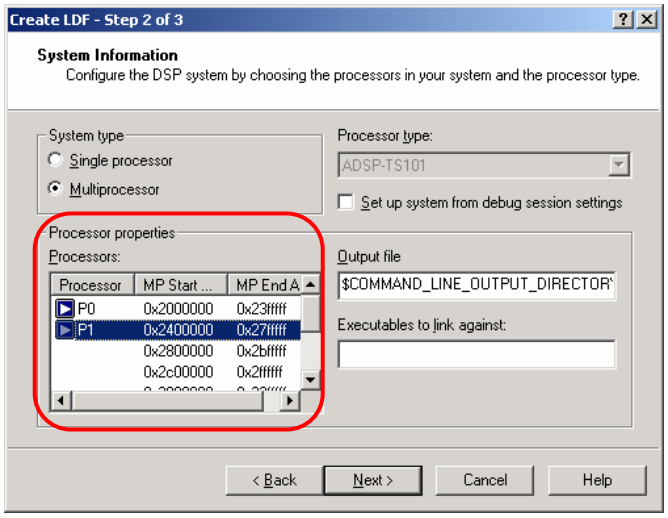

Figure 5 Processors and MMS Offset

As it can be seen in the *Processor* window (Figure 5), the multiprocessor memory space (MMS) offset value is automatically added in by the EL. This helps the user to avoid having to worry about specific MP addresses and memory offsets, making the use of MP commands much easier. This is an automatic replacement for the linker command *MPMEMORY* used in the LDF source file.

Linking Processors Executables

In the *Output File* box, the user can specify the name of the executable file for each processor in the system. By default, the EL selects the same name for the *.dxe* file as for the processor name.

In this case, *P0.dxe* and *P1.dxe* are selected as the names for the DSP executable files and are placed in the *Debug* folder within the project folder.

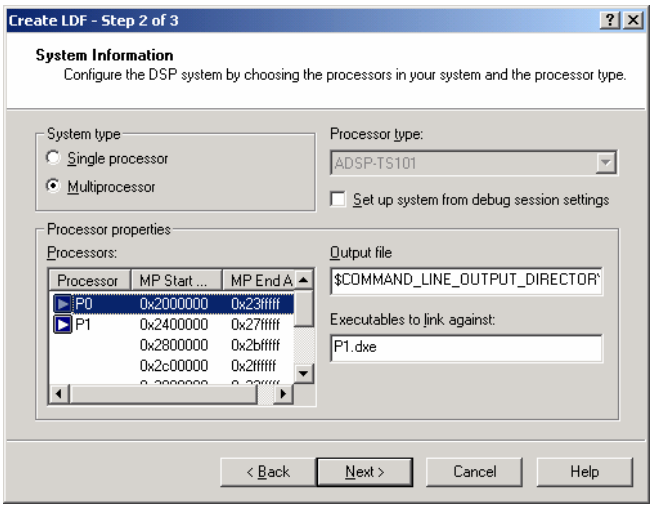

Figure 6 Executables to Link Against

As it would be done in the LDF source file with the LINK_AGAINST command, the EL allows the user to resolve symbols declared within MP space. This is done by simply specifying for each processor to which DSP to *link against*.

In this particular example, symbols referenced in P0 but declared in P1 can be resolved by the linker by adding *P1.dxe* to the *Executable to Link Against* box (Figure 6) for P0. Similarly, *P0.dxe* is added in for P1. In cases where more than one *.dxe* is added to this box, commas or spaces can be used as separators.

Now that an MP LDF has been selected, the processors have been added to the list, and the relevant linker commands have been specified, the LDF is ready for completion.

Click *Next*.

Note that in the case where shared external memory is used (*shared.sm*), this would also need to be added to the *link against* command box. This is automatically handled by the EL and will be explained later on.

MP LDF Wizard Completion

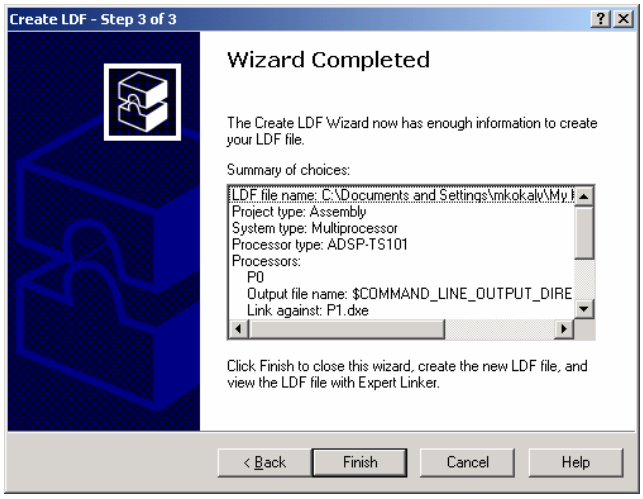

Figure 7 Expert Linker Wizard Completion

Click *Finish*.

Expert Linker Window

After completion of the Expert Linker wizard, the LDF graphical interface will open up (Figure 8).

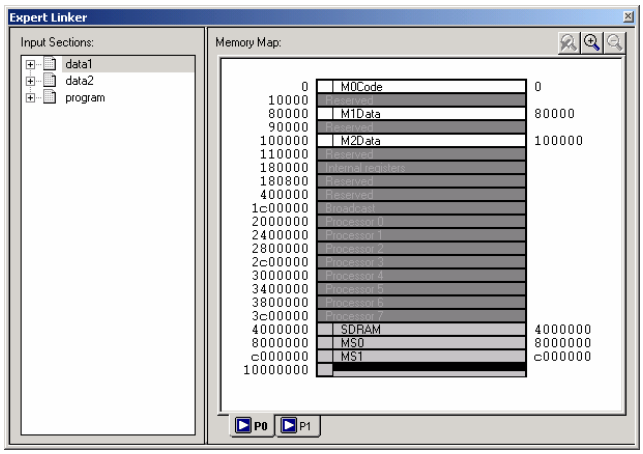

Figure 8 Expert Linker Window

The EL window has two panes: *Input sections* (displays a tree of all the projects input sections) and *Memory Map* (tree or graphical representation of each memory map).

For more details on the Expert Linker window and display options, please use the on-line help in VisualDSP++.

Adding Shared Memory Segments

In many DSP applications where large amounts of memory for multiprocessing tasks and sharing of data are required, an external resource in the form of shared memory may be desired.

To add a shared memory section to the LDF using the EL, the following steps should be followed:

1. Right click on the *Memory Map* pane
2. Select *New/Shared Memory*
3. Specify a name for the shared memory segment (*.SM*)
4. Select the DSPs that have access to this shared memory segment.

Click *OK*.

As shown in Figure 9, a new shared memory segment, visible to Processors P0 and P1, has been successfully added to the system.

Note that variables declared in the shared memory segment will be accessed by both processors in the system. In order for the linker to be able to correctly resolve these variables, the *link against* command should be used once again (see *Linking Processors Executables*).

The EL automatically does this, and therefore the user does not need to perform any additional modifications to the LDF.

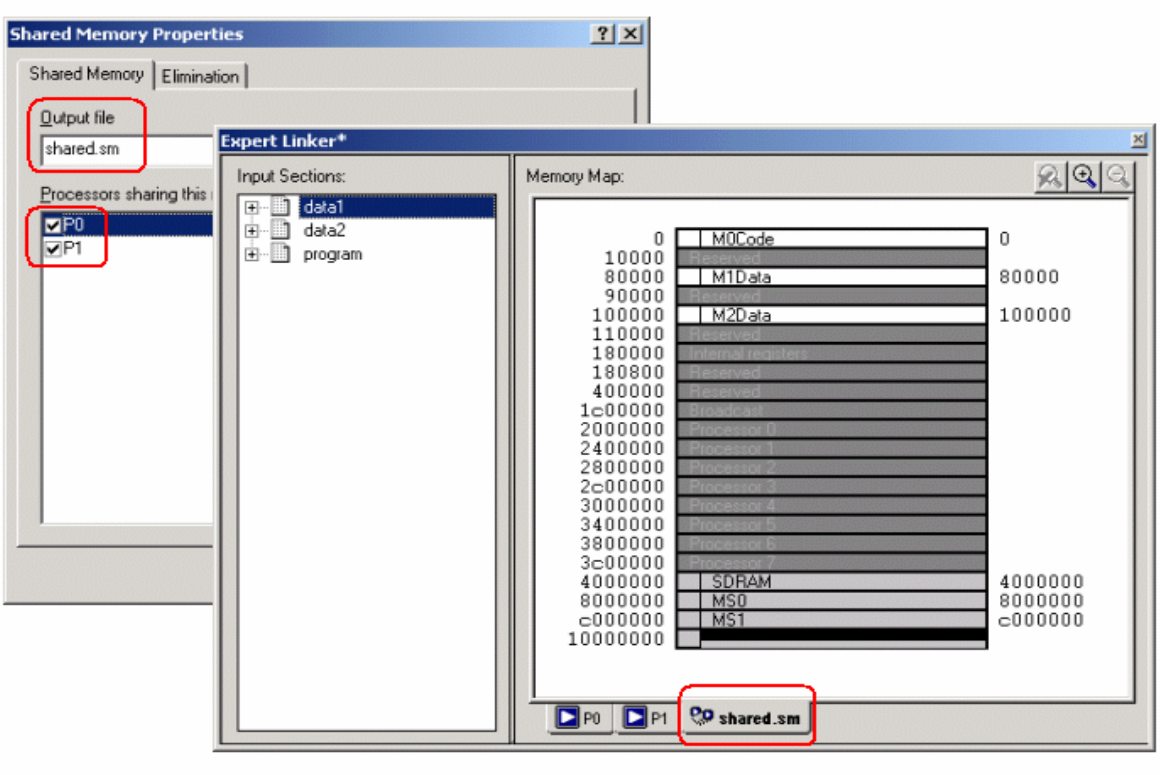

Figure 9 Shared Memory Segment

The user can confirm that the EL has correctly added the *.sm* file to the link against command line by simply viewing the *Memory Map* pane properties:

1. Right click on the *Memory Map pane*
2. Select *View Global Properties*
3. Click on the *Processor* tab

Shared.sm should now be contained in the *Executables to Link Against* box for each processor as illustrated in Figure 10.

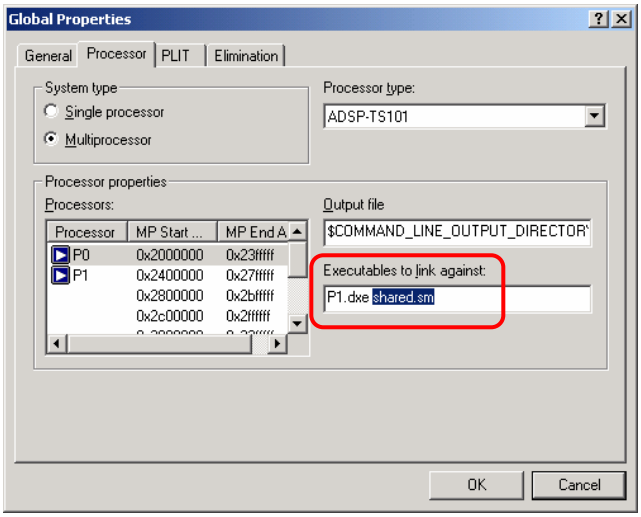

Figure 10 Adding "shared.sm" to the "Executables to link against" box.

Detection of Non-Linked input sections

By default, the LDF contains *data1*, *data2* and *program* input sections for each processor as well as for the shared memory segment.

In the scenario where the user declares in his code an input section different to any of the three mentioned above, the EL will detect it and it will mark it with a red cross as a "non-linked" input section (Figure 11).

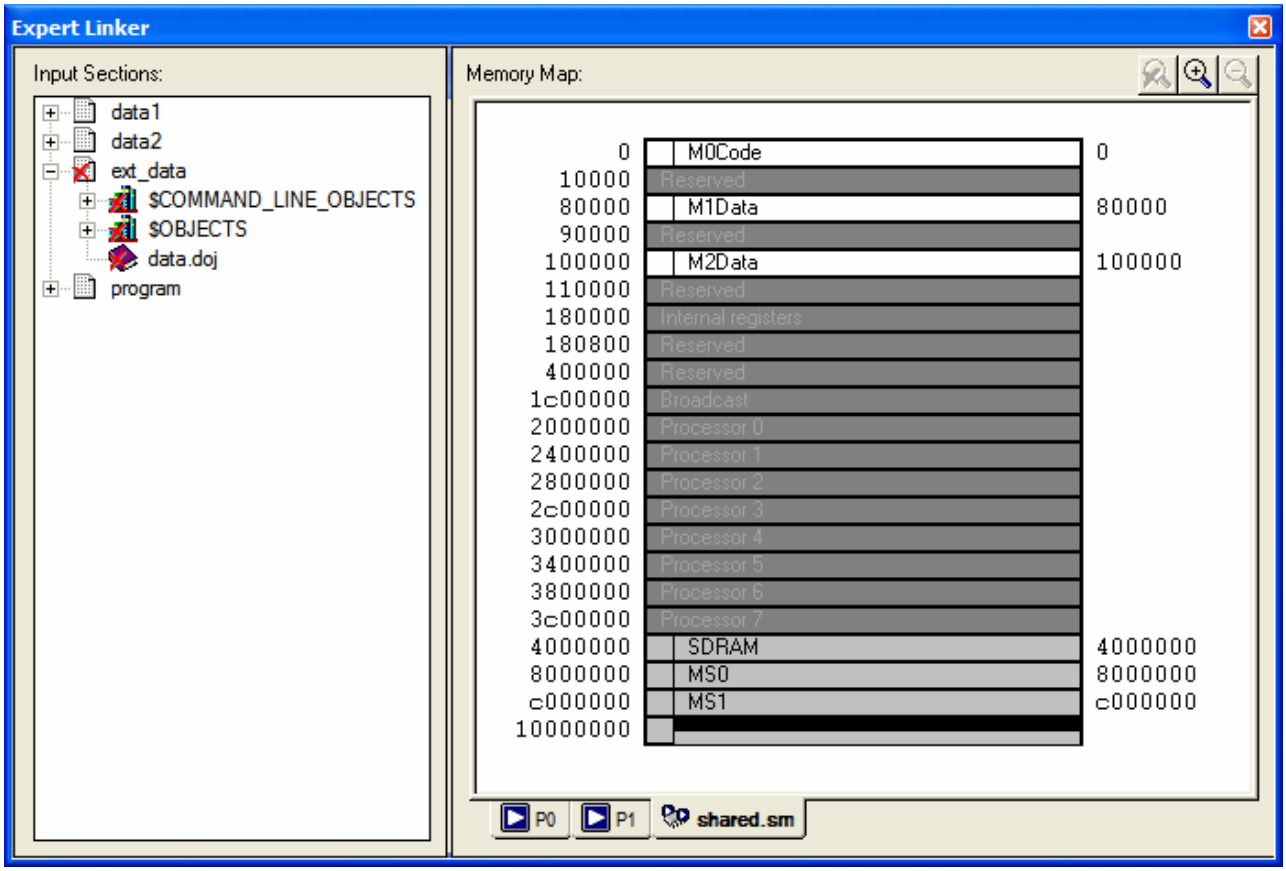

Figure 11 Detection of non-linked input sections

An example of "non-linked" section is provided in the source code (*ext_data)*. Press the *Rebuild All* button and update the contents of the EL window (double click on the LDF file in the project window).

Figure 11 shows how the linker has detected this "non-linked" input section. In this case, it corresponds to a variable declared in external SDRAM memory, which belongs to the shared memory segment.

Note that at this stage, the linker will generate some errors when building the project. This is due to the fact that the output sections have not been properly configured (object files not linked yet).

Linking Object Files

Now that both processors and the shared memory segments have been properly configured, and the EL has detected all input sections, the next step is to link the object files from these different input sections to their corresponding memory sections.

First of all, sort the left pane of the Expert Linker window by *LDF Macros* instead of *Input Sections* (default setting). This can be done by right clicking on the left pane and selecting *Sort by/LDF Macros*.

Then, right click on the *LDF Macro* window and add a new macro for *P0* (*Add/LDF Macro*). For example, *$OBJECTS_P0*. Repeat the same step for *P1* and *shared.sm* (Figure 12).

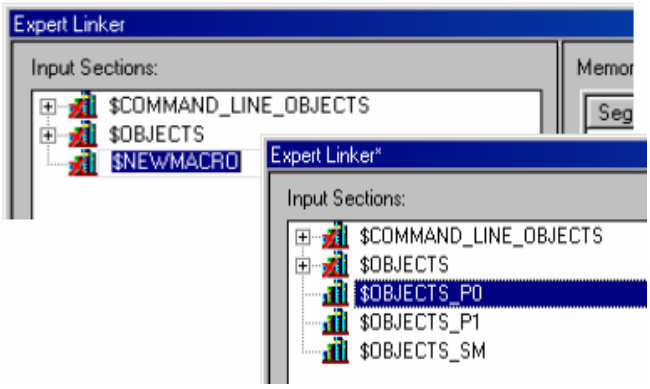

Figure 12 Creating LDF Macros

The next step is to add the object files (*.doj*) that correspond to each processor as well as to the shared memory segment. This is done by right clicking on each recently created LDF Macro and then selecting *Add/Object/Library File*. Figure 13 shows the objects files added to each LDF Macro.

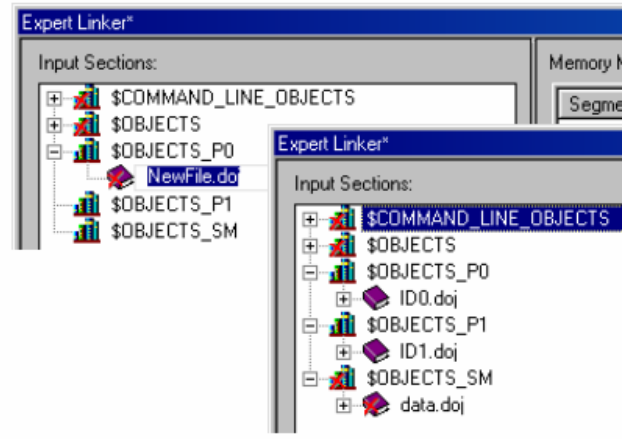

Figure 13 Adding Object Files

The use of LDF macros becomes extremely useful in systems where there is more than one *.doj* file per processor or shared memory segment, in which case the same step previously explained should be followed for each *.doj* file.

As shown in Figure 14, the LDF macro *$COMMAND_LINE_OBJECTS* must be deleted from the *$OBJECTS* macro to avoid duplicate of object files during the linking process.

The $COMMAND_LINE_OBJECTS macro contains the *.doj* files that correspond to every source file used in the project (in this case *ID0.doj, ID1.doj* and *data.doj*). If this macro is left in, the linker will automatically map the *.doj* files for both processors into each processor's memory map, i.e. *M0Code/code* will contain *ID0.doj(program)* and *ID1.doj(program)*. This is obviously wrong, since there is no need to map any of *ID1.doj* code into processor *P0*.

Therefore, right click on the *$COMMAND_LINE_OBJECTS* macro and select *Remove*.

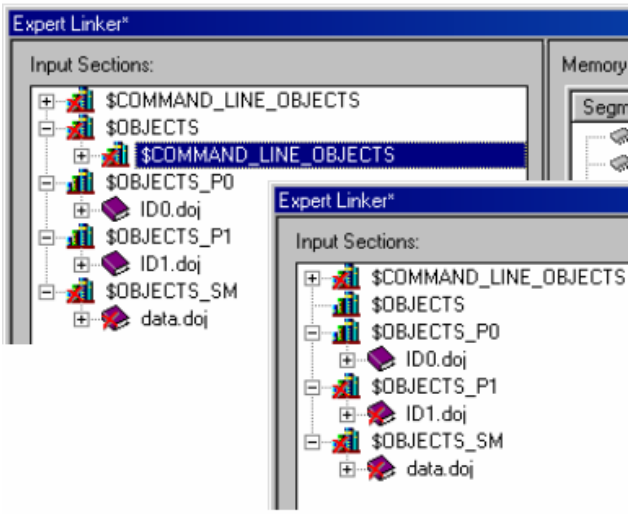

Figure 14 Deleting the $COMMAND_LINE_OBJECTS LDF Macro

The next step is to map the new macros into memory. This is done by placing each macro into its corresponding memory section.

Before this can be done, the left pane needs to be sorted by *Input Sections* instead of *LDF macros*.

Thus, right click on the left pane and select *Sort by/Input Sections*.

Additionally, change in the right pane the *Memory Map View Mode* from *Graphical* to *Tree* mode. Right click on the *Memory Map* window, select *View Mode* and then *Memory Map Tree*.

Now select one of the processors by clicking on the processor's name tab. In this case *P0* is selected first. Then, place (drag and drop) the recently created LDF macro, *$OBJECTS_P0*, in its corresponding memory segment. These steps are shown in Figure 15.

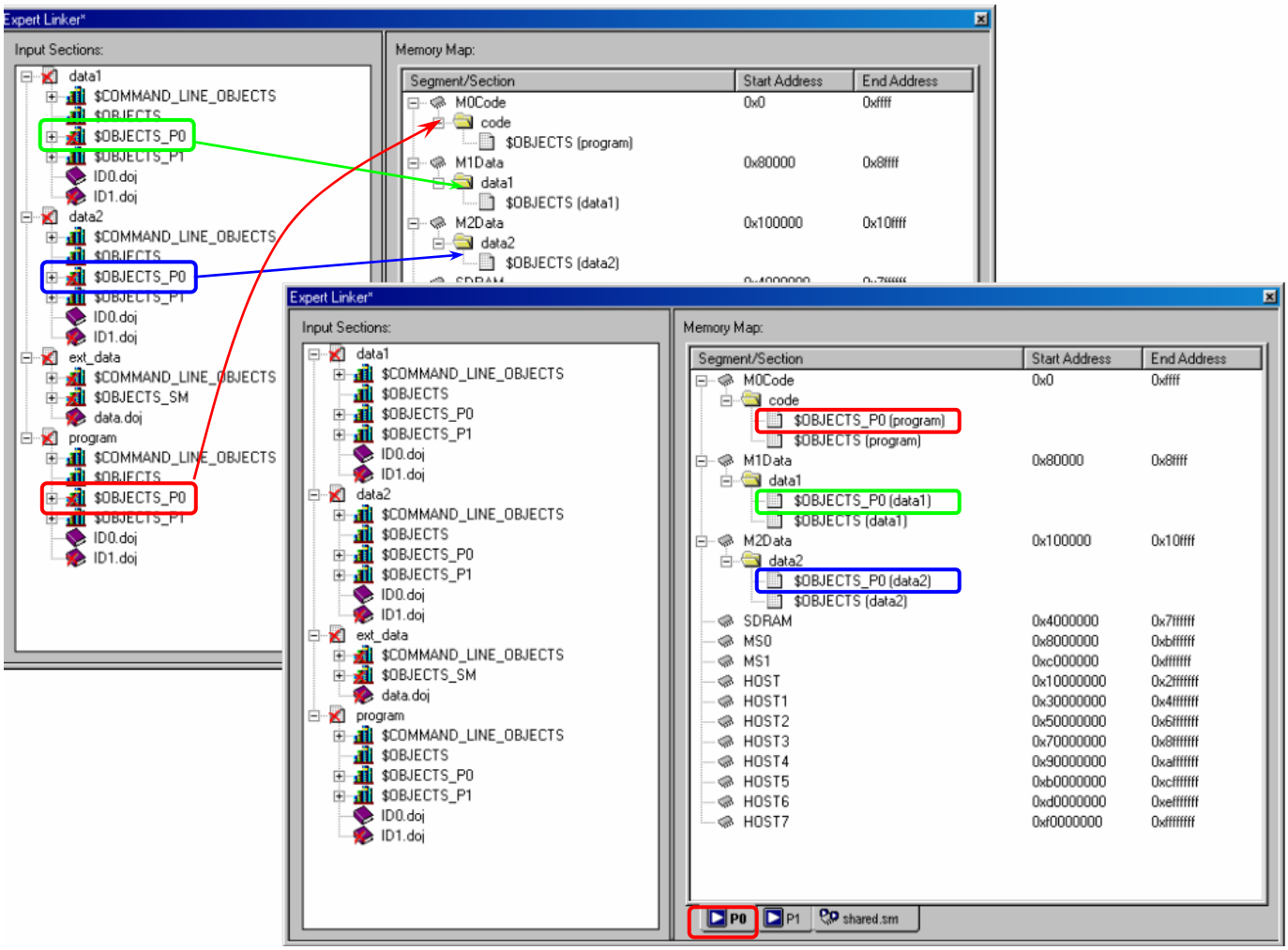

Figure 15 Linking Object Files Using LDF Macros

Repeat the same steps for processor *P1 ($OBJECTS_P1)* and for the shared memory segment, *shared.sm* (place *$OBJECTS_SM* in the *SDRAM* section).

Press *Rebuild All*.

As it can be seen in Figure 16, the red crosses denoting the "non-linked" sections have disappeared, indicating that the input sections have been properly mapped into memory.

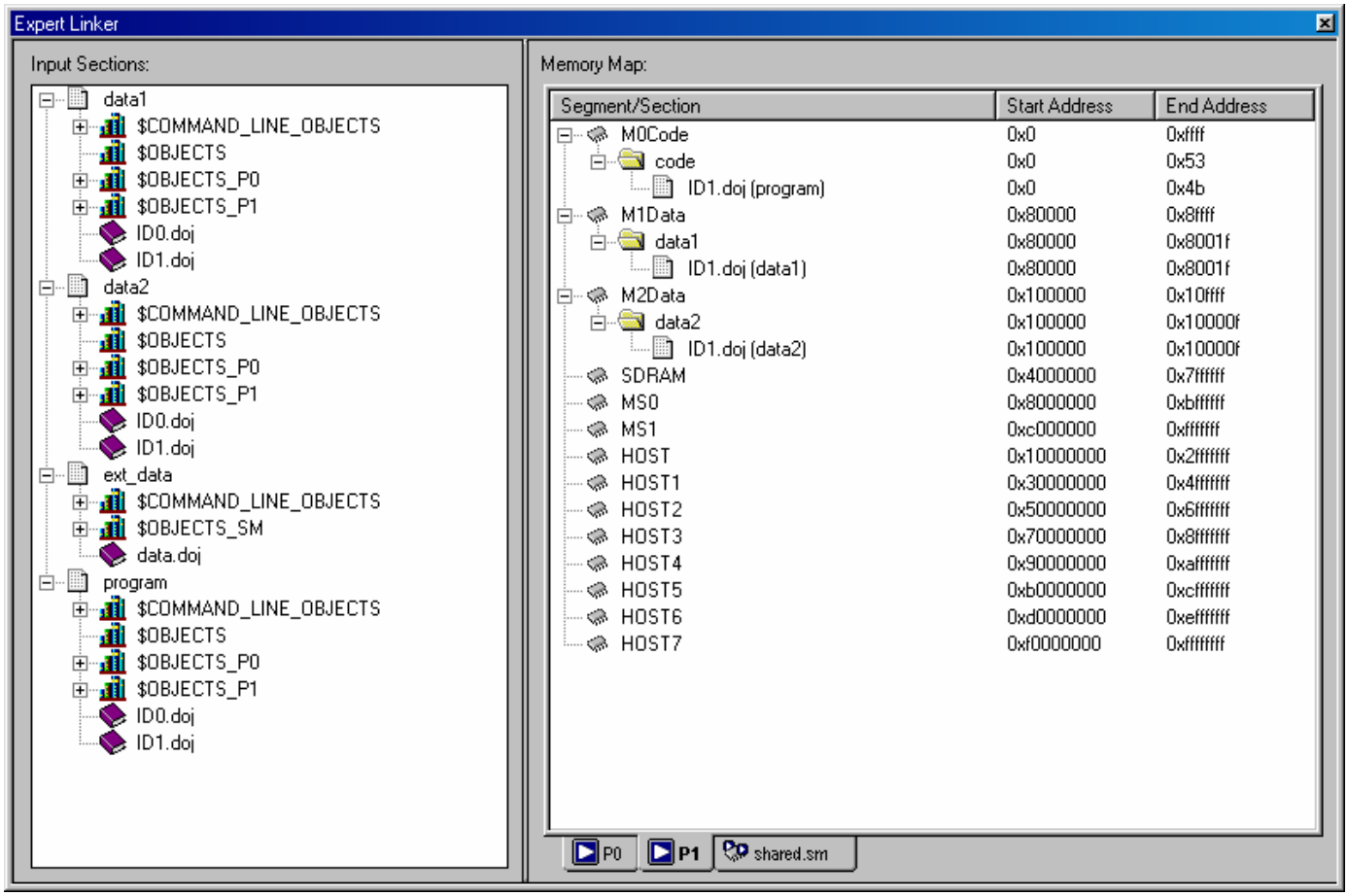

Figure 16 Expert Linker Multiprocessor LDF

Also, note that the LDF macros that were moved from the *Input Sections* window (left pane) to their corresponding sections in the *Memory Map* window (right pane) have been automatically replaced during linking process with the actual object files (*.doj*) used by the linker.

Expert Linker Multiprocessor LDF Source Code

The LDF is now complete! Figure 17 illustrates the generated LDF in the *Source Code View* mode.

As shown in Figure 17, the multiprocessor linker commands, *MPMEMORY, SHARED MEMORY* and *LINK AGAINST,* as well as the corresponding *LDF Macros*, have been successfully generated by the *Expert Linker* in a way absolutely transparent to the user.

The complete project is now ready to be built. Once again, perform a *Rebuild All* and safely start debugging with the application code.

```
ARCHITECTURE(ADSP-TS101)
SEARCH_DIR( $ADI_DSP\TS\lib )

  $OBJECTS = ;
  $OBJECTS_SM = data.doj;
  $OBJECTS_P1 = ID1.doj;
  $OBJECTS_P0 = ID0.doj;

[...]   // Simplified MP LDF version

MPMEMORY {  P0 { START(0x2000000) }
            P1 { START(0x2400000) }}

PROCESSOR P0
{
    LINK_AGAINST(p1.dxe, shared.sm)
    OUTPUT( $COMMAND_LINE_OUTPUT_DIRECTORY\P0.dxe )
    SECTIONS{
        code
        {   FILL(0xb3c00000)
            INPUT_SECTION_ALIGN(4)
            INPUT_SECTIONS( $OBJECTS_P0(program) $OBJECTS(program) )
            // The next line adds 8 nops to the end of the code section.
            // This is required on TS101 to prevent uninitialised memory
            // getting into the pipeline.
            . = . + 8;  } >M0Code
        data1
        {   INPUT_SECTIONS( $OBJECTS_P0(data1) $OBJECTS(data1) )
        } >M1Data

        data2
        {   INPUT_SECTIONS( $OBJECTS_P0(data2) $OBJECTS(data2) )
        } >M2Data   }}

PROCESSOR P1{

    LINK_AGAINST(p0.dxe, shared.sm)
    OUTPUT($COMMAND_LINE_OUTPUT_DIRECTORY\P1.dxe)
    SECTIONS{
        code
        {   FILL(0xb3c00000)
            INPUT_SECTION_ALIGN(4)
            INPUT_SECTIONS($OBJECTS_P1(program) $OBJECTS(program))
            . = . + 8;  }>M0Code
        data1
        {   INPUT_SECTIONS($OBJECTS_P1(data1) $OBJECTS(data1))
        }>M1Data

        data2
        {   INPUT_SECTIONS($OBJECTS_P1(data2) $OBJECTS(data2))
        }>M2Data    }}

SHARED_MEMORY {

    OUTPUT(shared.sm)
    SECTIONS{
        SDRAM{
            INPUT_SECTIONS($OBJECTS_SM(ext_data))
        }>SDRAM     }}
```

Figure 17 Expert Linker Multiprocessor LDF Source code

References

[1] ADSP-TS101 TigerSHARC® Processor Hardware Reference.
First Edition, March 2003. Analog Devices, Inc.

[2] VisualDSP++™ 3.0 Linker and Utility Manual for TigerSHARC®.
Rev. 1.0, October 2002. Analog Devices, Inc.

[3] Understanding and Using Linker Description Files (LDFs) (EE-69).
August 1999. Analog Devices, Inc.

[4] Introduction to TigerSHARC® Multiprocessor Systems Using VisualDSP++™ (EE-167).
April 2003. Analog Devices, Inc.

Document History

Version	Description
July 17th, 2003 by Maikel Kokaly-Bannourah	Initial Release