TMS320 DSP Algorithm Standard Developer’s Guide

Literature Number: SPRU424C
October 2002

Printed on Recycled Paper

Preface

Read This First

About This Manual

This document is intended for DSP algorithm producers who want to take an existing algorithm software module and make it compliant to the TMS320 DSP Algorithm Standard (referred to as XDAIS throughout the rest of this document). The procedures outlined will show the algorithm producer how to prepare and/or rework the existing algorithm, create the standard interface files required for compliance using a Code Composer Studio™ plug-in tool, and then run the final deliverables through another pre-compliance Code Composer Studio plug-in tool for immediate feedback.

DSP system integrators or XDAIS consumers should not follow the procedures outlined in this document, as it only applies to producers of algorithm software components. XDAIS consumers should refer to the application note, *A Consumer's Guide to Using eXpressDSP-Compliant Algorithms* (SPRA810), instead of this user's guide.

Contents

Figures

Tables

TMS320 DSP Algorithm Standard Developer's Guide

This document, along with the latest set of *TMS320 DSP Algorithm Standard Rules and Guidelines*, (literature number SPRU352) and referred to as "XDAIS" throughout the document, will help assist the algorithm developer with implementing the XDAIS interface, as well as with creating a test application. When the XDAIS conversion is complete, the algorithm should conform to all the XDAIS rules. Also required is the Component Wizard plug-in for Code Composer Studio™, which greatly simplifies the XDAIS development effort. Each copy of Code Composer Studio IDE contains this tool.

The latest version of the standard is found in the TI DSP Developer's Village (http://dspvillage.ti.com). Follow the link to the TMS320 DSP Algorithm Standard, look in the documentation section, and click on the appropriate link.

The XDAIS development procedure consists of seven sections. The first section requires the developer to determine the present state of the algorithm, decide on module names, and answer some preliminary questions. The second, third, fourth, and fifth sections give the developer some detailed insight into the XDAIS implementation details by leading him/her through the process of acting as consumer, interface definer, and algorithm vendor. The fifth section describes how to create the XDAIS library deliverable. The sixth section describes how to prevalidate the XDAIS algorithm. Most of the procedures contain examples based on a G.729 encoder algorithm.

1 Preliminary XDAIS Work and Preparation

This part of the process focuses the algorithm developer on the operation of the algorithm and how the XDAIS interface 'fits' onto it. This section verifies that the algorithm is in compliance with Rules 1-10 of the *TMS320 DSP Algorithm Standard (XDAIS) Rules and Guidelines* document.

1) Is the testing environment already set up? There should be an application program that calls the algorithm to process a test vector(s) and generates an output vector(s). The output vector(s) is then compared to a known 'good' vector(s) for correctness. Make sure this test environment is set up and the algorithm successfully passes the test process before proceeding.

2) The original algorithm must conform to the first 10 rules stated in the XDAIS Rules and Guidelines document. It is a good idea to verify this step before proceeding, but it could be temporarily postponed for the sake of expediting the XDAIS process. Rule #8 (External Identifiers) can be skipped for now — it will be addressed later in the build procedure.

3) What are the names for the module, version (optional), vendor, architecture, and memory model? The MODULE and VENDOR names must be in all capital letters and should be as short as possible, since they both will be prefixed to all symbols visible to the client. Use Appendix A to record this information and refer to Table 1 for an example. For the C54x architecture, it is recommended to supply both near and far mode memory models. For the C6x architectures, all algorithms must, as a minimum, be supplied in little endian format, but it is recommended to supply a big endian model, as well. The different variants/models should not produce different interfaces for the same algorithm; the only difference will be the way the object files are compiled within each library variant/model (and the file name of each library).

4) Construct the Memory Table (memTab[]). This table describes the number of memory blocks, and the quality of each memory block required to run one instance of the algorithm. Go to Appendix C and create a table based on the Instance Memory Table (Table C-3, Rule 19). Each row, starting with memTab[1], should describe each block of memory needed to create one instance of the algorithm. Leave memTab[0] blank for now.

Table 1. Example Table for Naming the Module, Vendor, Variant, Architecture, and Memory Model

Module Name	Vendor Name	Variant	Architecture	Memory model (optional)
G729ENC	TI	None	[24 | 28 | 54 | 55 | 62 | 64 | 67]	[far (f) | big endian (e)]

2 Defining the Module-Specific Abstract Interface

If there are sample APIs in the XDAIS Developers Kit which exist for your algorithm, please use them (these will be the i<MODULE>.[ch] files in “c:\ti\<target>\xdais\src” directories). Some of the more common interfaces which TI has defined and recommended to be used include CPTD, DTMF, LEC, G.7XXDEC, and G.7XXENC modules. **Note:** <*target*> refers to C5400, C5500, C6000, etc. If not, you must act as the Interface Definer (Service Provider) for your algorithm and create these files which are used by the application. Determine what the extended methods of the IALG interface will be for this algorithm. These are typically the module-specific “process and/or control” functions. This interface must be extended by at least one process method to allow the algorithm to be called, but any number of them can be defined in this section. Typically, a generically named **apply** method is defined for each module; in this case we use **encode**. Use the following example as a guideline for naming the extended methods for the XDAIS algorithm. These module-specific methods, along with the traditional IALG methods, make up the service provider interface (SPI), which is supplied to the consumer/system integrator of the application.

Note:

If your core algorithm contains one or more initialization functions, please do not include these functions as part of the extended methods. The standard IALG interface already defines a “per instance initialization function,” so all internal initialization code should be manually placed within that function (the function stub will be created by the name of <MODULE>_<VENDOR>_initObj() after running the Component Wizard).

Table 2 shows an example of how to enter the newly named methods in Table 6 of Appendix A. Also, note that the *IALG_Handle* parameter does not need to be entered in this table (the code generation tool will add it to all functions automatically as the first parameter). XDAIS types are encouraged for the module-specific interface (most likely located in “c:\ti\<target>\xdais\include\xdas.h”).

Table 2. Example Table for Naming Extended Methods

Return Type	Method Name	Param1 Type and Name	Param2 Type and Name	Param3 Type and Name
XDAS_Int8	encode	XDAS_Int16 *in	XDAS_Int8 *out	N/A

Determine the interaction of the algorithm with the application (or framework). In particular, what are the instance creation parameters and what are the real-time status/control parameters? The instance creation parameters are values passed to the algorithm when it is instantiated (i.e., only one time) and the status/control parameters are algorithm status information that is read and/or written while the algorithm is in operation. The complex types that deal with these parameters are automatically generated by the Component Wizard using the names specified by the user. The source code listed below is an example output of the Component Wizard. The developer simply supplies the structure member names and determines the appropriate parameter names and enter Instance Creation Parameters in Table 4 and the Real-Time Status and Control Parameters in Table 5 in Appendix A. In both cases, the *size* parameter does not need to be entered in the table because it is automatically generated by the Component Wizard as the first field of all Params and Status structures.

The *size* field **must** be the first field of all Params and Status structures. This variable is typically used to determine whether a Params or Status structure has been extended by the Interface Definer. When Params and Status structures are extended (i.e., more fields are added), the original set of members must still work, so any IALG function will be able to tell which structure (original or extended) to use based on the value stored in the *size* field.

```
/* ig729enc.h */
#ifndef IG729ENC_
#define IG729ENC_
#include <ialg.h>
#include ”ig729.h“
/*
 *  ======== IG729ENC_Obj ========
 *  This structure must be the first field of all G729ENC instance objects.
 */
typedef struct IG729ENC_Obj {
    struct IG729ENC_Fxns *fxns;
} IG729ENC_Obj;
/*
 *  ======== IG729ENC_Handle ========
 *  This handle is used to reference a G729ENC instance object.
 */
typedef struct IG729ENC_Obj *IG729ENC_Handle;
/*
```

```
 *  ======== IG729ENC_Params ========
 *  This structure defines the creation parameters for all G729ENC instance ob-
jects.
 */
typedef struct IG729ENC_Params {
Int size;    /* MUST be the first field */
Int frameLen;
Int pfo;
Int vad;
} IG729ENC_Params;
/*
 *  ======== IG729ENC_Status ========
 *  This structure defines the status parameters or values that can be
 *  read and/or written while the algorithm is ‘live’.
 */
typedef struct IG729ENC_Status {
Int size;    /* MUST be the first field */
Int maxChannels; /* Can be read/write */
Int frameLen;    /* Can be read/write */
Int signalStatus;   /* Read only */
} IG729ENC_Status;
/*
 *  ======== IG729ENC_Fxns ========
 *  This structure defines all of the operations on G729ENC objects.
 */
typedef struct IG729ENC_Fxns {
    IALG_Fxns   ialg;
    XDAS_Int8 (*encode)(IG729ENC_Handle handle, XDAS_Int16 *in, XDAS_Int8 *out);
} IG729ENC_Fxns;
#endif  /* IG729ENC_ */
```

3 The eXpressDSP Component Wizard

The eXpressDSP Component Wizard from Hyperception is supplied as part of the TMS320 Algorithm Developers Kit. The Component Wizard simplifies the creation of eXpressDSP-compliant algorithms by generating the source, header, and library build files. The resulting code can be used to create algorithms that comply with the rules specified in the TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352). Please refer to TMS320 DSP Algorithm Standard Developer's Guide, (this document) to get further information on how to make an existing algorithm eXpressDSP-compliant.

The Component Wizard tool leads users through a series of step-by-step screens to get inputs, and produces a source code template and associated project files that can be compiled into vendor-specific eXpressDSP-compliant algorithm. Vendors need to supply the actual algorithm code to be implemented and compile the project.

The Component Wizard for eXpressDSP can either create a new eXpressDSP algorithm template from scratch or it can load an existing template file and allow you to make modifications. The resulting source code template and accompanying files will adhere to the Texas Instruments TMS320 DSP Algorithm Standard.

The Component Wizard also generates optional block component for Hyperception's VAB DSP design tool.

3.1 Why use the eXpressDSP Component Wizard?

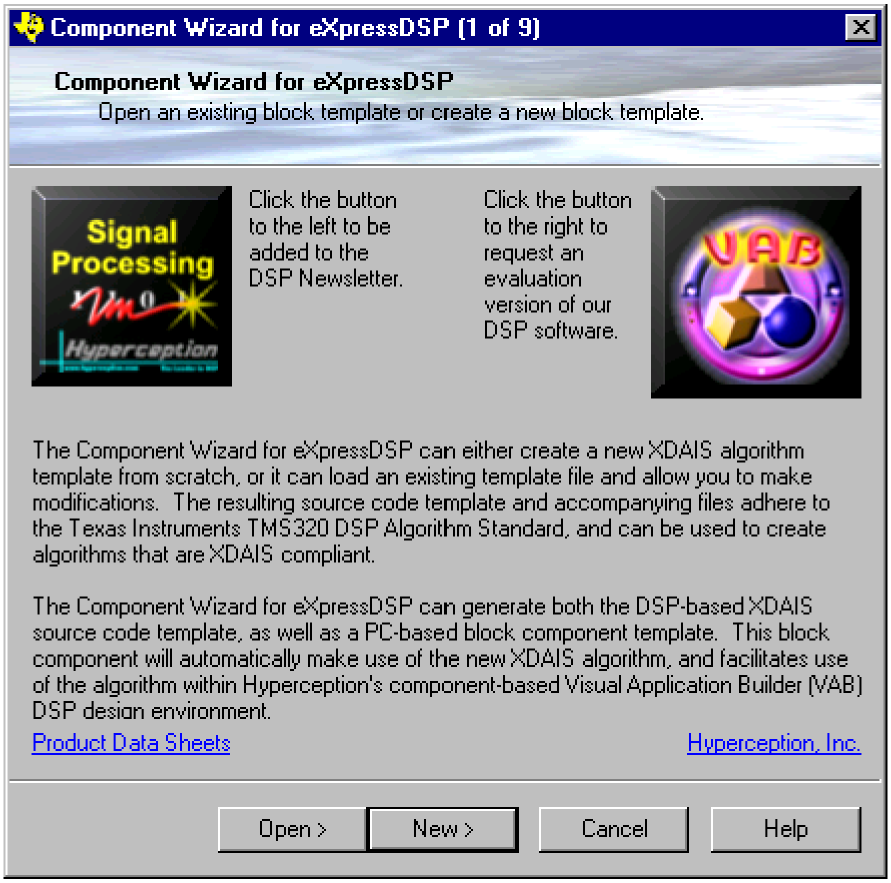
The Component Wizard greatly simplifies the creation of your eXpressDSP-compliant algorithm. You are free to concentrate on your algorithm as opposed to having to delve into all of the inner workings of the TMS320 DSP Algorithm Standard. All aspects of the standard are handled automatically by the Component Wizard, and are based upon data that you provide in a series of comfortable, easy-to-follow, setup screens. Implementation time and design complexity is greatly reduced.

An added benefit of using the eXpressDSP Component Wizard is that you will be able to automatically produce a block component for the VAB DSP design tool. You will be able to harness the power of your eXpressDSP algorithm by simply placing a block component onto a worksheet and using it interactively with hundreds of other block components. Entire DSP applications can be created graphically by simply placing these block components onto a worksheet and establishing a data flow with a few mouse clicks.

3.2 eXpressDSP Component Wizard Initial Screen

To open the tool within the Code Composer Studio environment, follow this link on the Code Composer Studio Toolbar: **Tools** → **Algorithm Standard** → **Component Wizard**. When you first run the eXpressDSP Component Wizard you should see a screen similar to the one shown below:

Figure 1. eXpressDSP Component Wizard Initial Screen



Component Wizard for eXpressDSP

Open an existing block template or create a new block template.

Click the button to the left to be added to the DSP Newsletter.

Click the button to the right to request an evaluation version of our DSP software.

The Component Wizard for eXpressDSP can either create a new XDAIS algorithm template from scratch, or it can load an existing template file and allow you to make modifications. The resulting source code template and accompanying files adhere to the Texas Instruments TMS320 DSP Algorithm Standard, and can be used to create algorithms that are XDAIS compliant.

The Component Wizard for eXpressDSP can generate both the DSP-based XDAIS source code template, as well as a PC-based block component template. This block component will automatically make use of the new XDAIS algorithm, and facilitates use of the algorithm within Hyperception's component-based Visual Application Builder (VAB) DSP design environment.

Product Data Sheets

Hyperception, Inc.

The Component Wizard is capable of producing two types of source code templates: a "DSP-side", and a "PC-side." The DSP-side template is the DSP source code that will ultimately be compiled into your own eXpressDSP algorithm. This is where your vendor algorithm will reside. The resulting code is executed on a target DSP, and will conform to the Texas Instruments TMS320 DSP Algorithm Standard.

Please refer to the document <ccsinstallpath>/plugins/xdais/cw/docs/compwiz.pdf for further information on help.

3.2.1 Creating a New Template

By selecting the **New** button from the initial screen you are choosing to create a new template. This will cause the Component Wizard to initialize all subsequent screen entries to empty, and you will need to supply any pertinent information as you step through the Component Wizard screens.

3.2.2 Opening an Existing Template

By selecting the Open button you are choosing to open up an existing template. This will cause the Component Wizard to re-establish all of the setup information based upon the template you have selected. Using this option is often useful for going back at a later date to adjust the parameters of your algorithm and then recreate your eXpressDSP source code template.

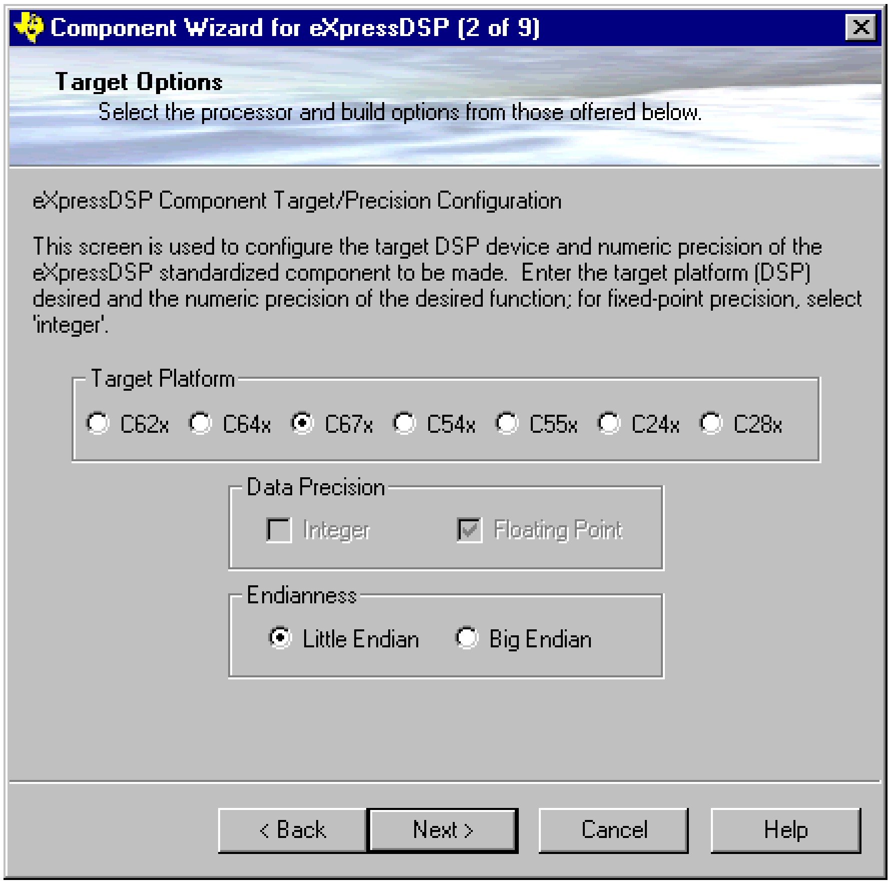
When you select to open an existing template the Component Wizard will display a dialog box labeled Open that will allow you to Browse for the desired template file. Template file extensions are .HBT, and have been created from a previous usage of the Component Wizard. That is, the very first time you use the Component Wizard you will need to use the New button and proceed to enter information. Upon completion of a Component Wizard session a .HBT file is generated.

3.3 Selecting a DSP Target Platform

The Component Wizard screen shown in Figure 2 will allow you to select some information about the target DSP on which the algorithm is intended to run. You will be able to select the DSP architecture (e.g., C62, C67, C54, etc.), as well as data precision and model endianness.

Target Platform	Specifies the DSP architecture that will be used
Data Precision	Specifies the data precision
Endianness (6x)	Specifies either little endian or big endian mode
Memory Model (5x)	Specifies either far or near memory model

Figure 2. Selecting DSP Target Platform Information



Component Wizard for eXpressDSP (2 of 9)

Target Options

Select the processor and build options from those offered below.

eXpressDSP Component Target/Precision Configuration

This screen is used to configure the target DSP device and numeric precision of the eXpressDSP standardized component to be made. Enter the target platform (DSP) desired and the numeric precision of the desired function; for fixed-point precision, select 'integer'.

Target Platform: C62x [✓ C67x] C64x C54x C55x C24x C28x

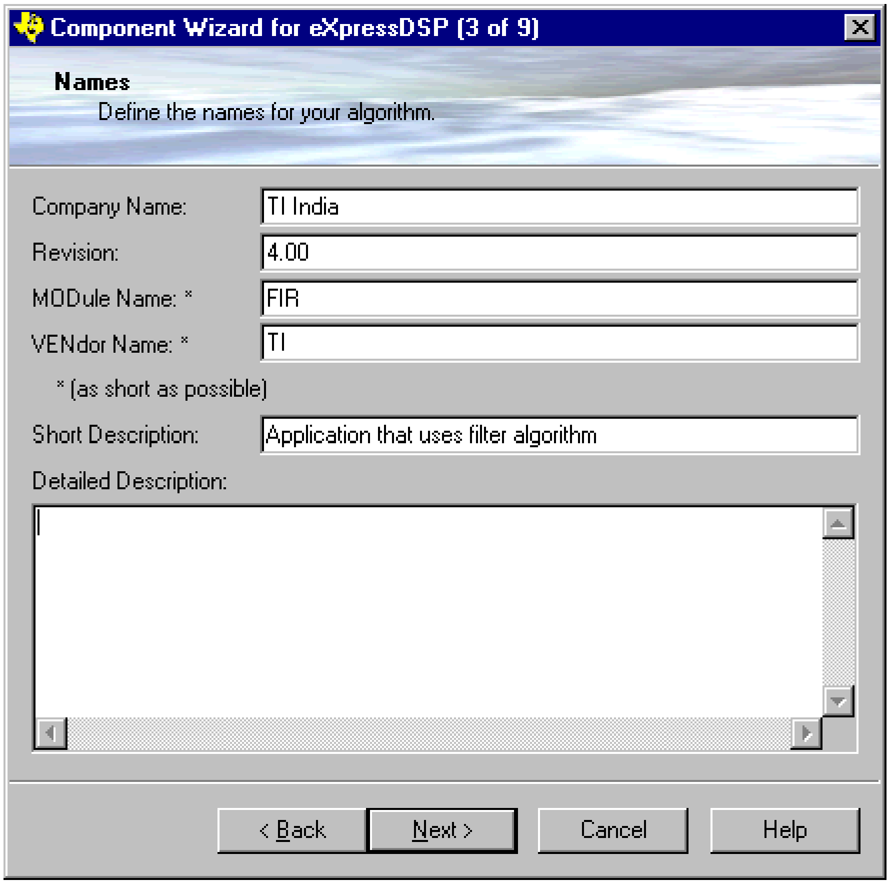
Data Precision: Integer [✓ Floating Point]

Endianness: [✓ Little Endian] Big Endian

3.4 Choosing an Algorithm and Vendor Name

You will be able to choose your algorithm’s name and enter your vendor-specific name with this Component Wizard screen. A screen similar to the one shown in Figure 3 will allow you to enter this information.

Figure 3. Choosing an Algorithm Name and Vendor ID



Component Wizard for eXpressDSP (3 of 9)

Names

Define the names for your algorithm.

Company Name:	TI India
Revision:	4.00
MODule Name: *	FIR
VENdor Name: *	TI
* (as short as possible)	
Short Description:	Application that uses filter algorithm
Detailed Description:	

Company Name	Specifies your company name for “bookkeeping” purposes (not required by XDAIS specification)
Revision	Specifies the revision or version of the algorithm/module
Algorithm/Module Name	Specifies the XDAIS algorithm/module name (3-5 characters suggested)
Vendor Name	Your unique vendor name (3-5 characters suggested)

Short description	A short one-line description of the module (not required by XDAIS specification)
Detailed description	Detailed description of the module (not required by XDAIS specification)

All fields marked with a * are mandatory and should be filled.

3.5 Determine the Number of Input and Output Arrays and Parameters Variables Used by Your Algorithm

3.5.1 Describing Your Algorithm's Input and Output

Your XDAIS algorithm will need to be described in order for the Component Wizard to properly generate a source code template that will "fit" your algorithm. This screen will provide you with the means of being able to do this. But before you begin you will need to have some information available on the algorithm that you are creating.

You will need to define how your algorithm will operate functionally. For example, maybe your algorithm takes-in a single input frame of data, performs some operation based upon parameter variable information, and then outputs a single frame of data. Or perhaps it requires multiple input frames of data, performs the intended algorithm with the input frames of data, and then outputs a single frame of data. There are many combinations to consider, but you may find that typically your algorithm will operate on a single frame of data and output a single frame of data.

There are a few important items that must be addressed before you can properly fill in the requested information in this screen. The number of input and output frames of data required for your algorithm must be defined before you should proceed.

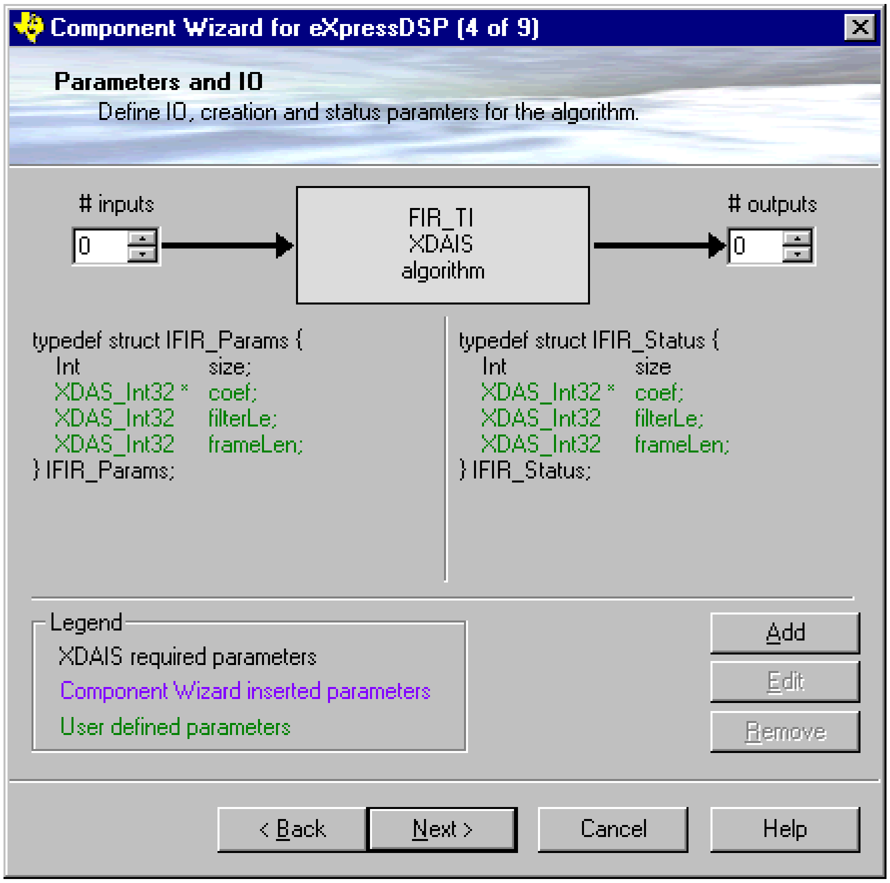
The two entry fields listed below will allow you to determine the number of Input Frames, and Output Frames that will be used by your algorithm. Don't worry about defining what these will be (e.g., choosing names and data types) in this Component Wizard screen. There will be time to address this in the subsequent Component Wizard screens!

Inputs	Specify the number of algorithm input data frames
Outputs	Specify the number of algorithm output data frames

Your algorithm may require several variables that must be supplied to the algorithm in order for it to operate properly. You'll need to decide how many variables your algorithm will need.

If you have determined that your algorithm will make use of parameter variables then you can add them via a Component Wizard screen similar to the one shown below:

Figure 4. Describing Your Algorithm’s Input and Output


Component Wizard for eXpressDSP (4 of 9)
Parameters and IO
Define IO, creation and status paramters for the algorithm.
# inputs
0
FIR_TI
XDAIS
algorithm
# outputs
0
Legend
XDAIS required parameters
Component Wizard inserted parameters
User defined parameters
Add
Edit
Remove
< Back
Next >
Cancel
Help

```
typedef struct IFIR_Params {
    Int             size;
    XDAS_Int32 *    coef;
    XDAS_Int32      filterLe;
    XDAS_Int32      frameLen;
} IFIR_Params;
```

```
typedef struct IFIR_Status {
    Int             size
    XDAS_Int32 *    coef;
    XDAS_Int32      filterLe;
    XDAS_Int32      frameLen;
} IFIR_Status;
```

3.5.2 Defining your Algorithm’s Parameter Variables

Notes:

All parameter variables listed will always be made available as “in-scope” variables to the XDAIS algorithm. These variables do NOT have to be individually passed to the algorithm through an extended algorithm method.

All parameter variables listed will also be available from the PC-side VAB block component, and can be used to directly access the variables of the XDAIS algorithm.

You will need to take four steps for each and every variable that appears in the Component Parameter listbox. These steps are outlined below:

1) Rename the variable
2) Select the data type for the variable
3) Select the default value for the variable
4) Choose variable to be instance creation, status/control, or both

3.5.3 Renaming the Variable

Each parameter variables will initially contain a generic name such as “parameter0,” “parameter1,” etc. You will want to change these generic names to something more descriptive for your algorithm. To change a variable name, simply highlight the variable name of interest and then choose the Edit button and then type the name of the variable in the box labeled Name.

3.5.4 Selecting the Data Type for the Variable

First, highlight the variable name of interest. Then choose the Edit button and select the appropriate data type by choosing one of the XDAIS data types presented. In addition to the XDAIS data types, variables can also be chosen from several other types

3.5.5 Selecting the Default Value for the Variable

First, highlight the variable name of interest. Then select the Edit button and enter the desired default value for the variable.

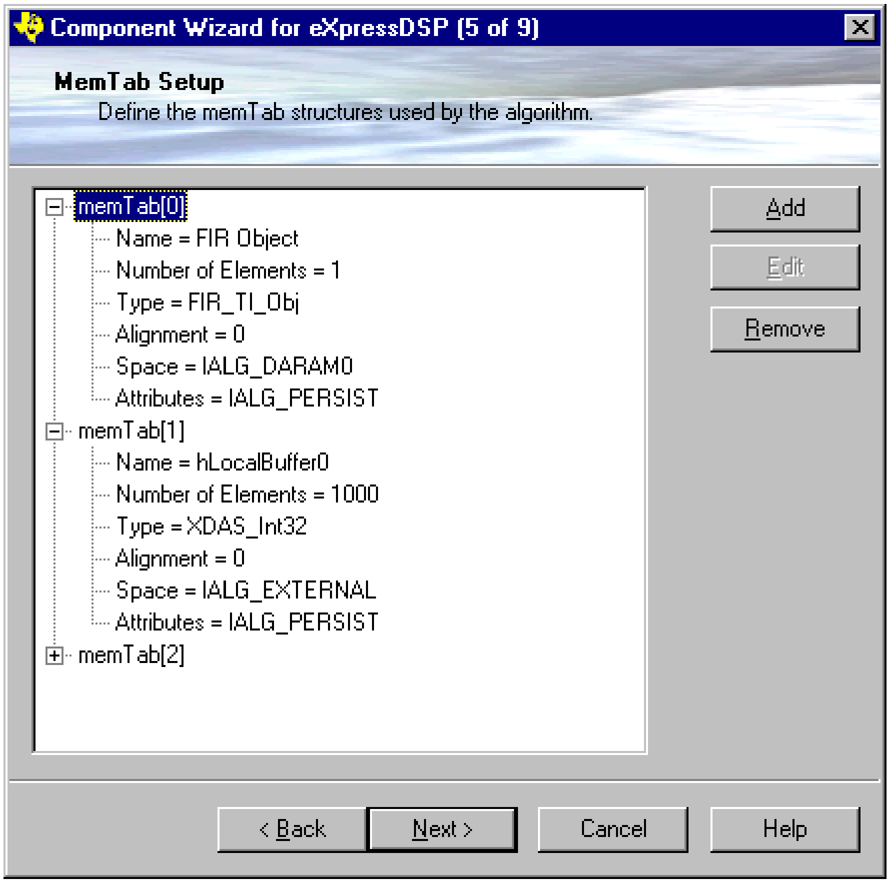
3.5.6 Choosing Instance Creation, Status/Control, or Both

Instance creation parameter variables are values passed to the algorithm when it is instantiated. Status/control variables contain algorithm status information that can be read and written as the algorithm is running. The Component Wizard for eXpressDSP allows you to decide if your parameter variables will be an instance creation variable, a status/control variable, or both. Simply highlight the parameter variable name, and then choose the Edit button to select the desired parameter types.

3.6 Defining Local Arrays or Buffers (memTab Memory Descriptors) Used By an Instance of Your Algorithm

Additionally, you should decide whether your algorithm requires the use of any local data array storage (e.g., delay elements for a filter). If you have determined that your algorithm will be using local memory arrays in the form of MemTAB memory descriptors then you can add them via a Component Wizard screen that looks similar to the one shown below:

Figure 5. Defining memTab Memory Descriptors



Component Wizard for eXpressDSP (5 of 9)

MemTab Setup

Define the memTab structures used by the algorithm.

```
memTab[0]
    Name = FIR Object
    Number of Elements = 1
    Type = FIR_TI_Obj
    Alignment = 0
    Space = IALG_DARAM0
    Attributes = IALG_PERSIST
memTab[1]
    Name = hLocalBuffer0
    Number of Elements = 1000
    Type = XDAS_Int32
    Alignment = 0
    Space = IALG_EXTERNAL
    Attributes = IALG_PERSIST
memTab[2]
```

You will need to follow the steps that are listed below in order to configure properly the local array memory descriptors. To select individual memory descriptors and then make unique changes for each you will need to first highlight a memory descriptor name. Then you can modify the content by double-clicking the name and then selecting the appropriate information for it (e.g., you can change its name by typing in a new one, or select the data type, etc.).

1) Rename the memory descriptor

2) Define the number of elements

Note:

If your algorithm requires a block of memory whose size is dependent upon parameter variables (e.g., filter length, or input framesize) you can double-click on the Number of Elements field and enter an equation using the parameter variable of interest; in some cases you may need to modify the resulting vendor-specific source code file so that the number of elements used will be based upon the proper parameter variable. The name of this file will be the one that ends with an “_ialg.c”.

3) Select the data type for the elements

4) Select memory alignment

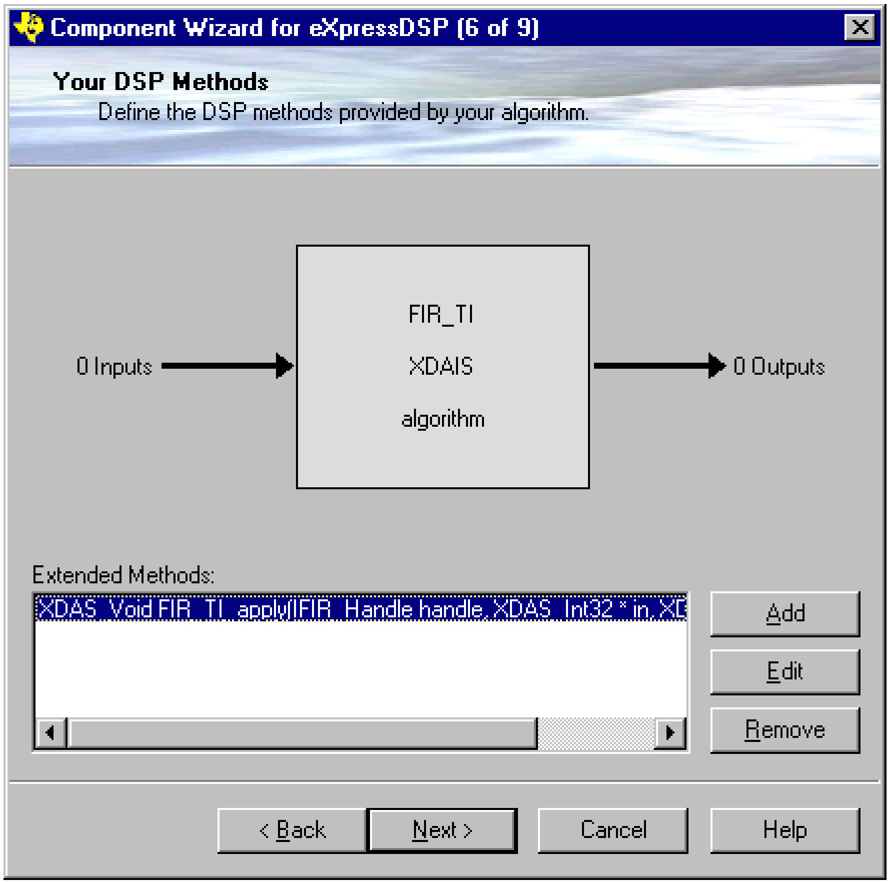
5) Select the memory space that will be used

6) Select the required memory attributes

3.7 Algorithm’s Extended Method(s)

You can add as many extended algorithm methods to your algorithm, as you would like by simply using the Add button located directly to the right of the Algorithm Methods list box. Selecting the Add button will cause the Function Prototype Wizard to appear. This Wizard will allow you to individually specify each function parameter of your extended method. The extended Algorithm Method setup screen will appear similar to the one shown in Figure 6.

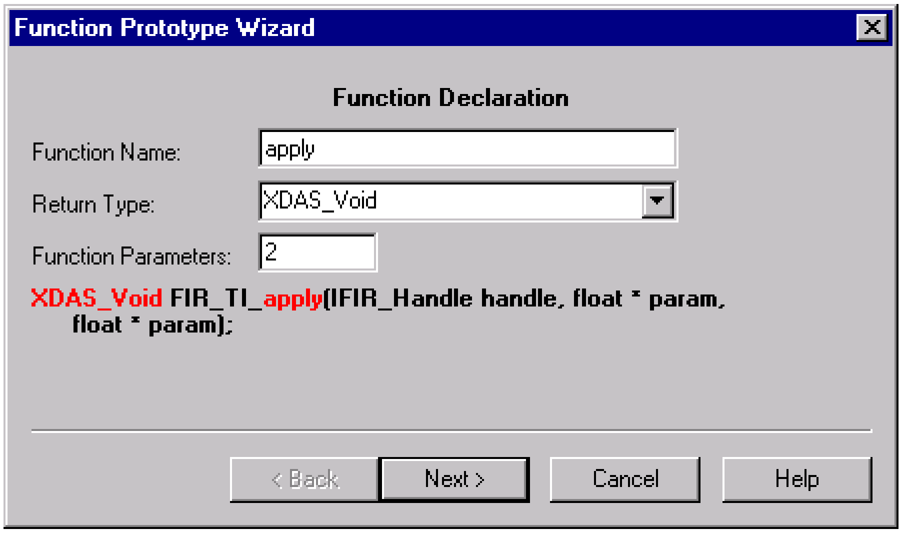
Figure 6. Extended Algorithm Method Entry Screen


Component Wizard for eXpressDSP (6 of 9)
Your DSP Methods
Define the DSP methods provided by your algorithm.
FIR_TI
XDAIS
algorithm
0 Inputs
0 Outputs
Extended Methods:
XDAS_Void FIR_TI_apply(IFIR_Handle handle, XDAS_Int32 * in, XD
Add
Edit
Remove
< Back
Next >
Cancel
Help

3.7.1 *Adding a New Extended Algorithm Method*

You can add as many extended algorithm methods to your algorithm as you would like by simply using the Add button located directly to the right of the Algorithm Methods list box. Selecting the Add button will cause the Function Prototype Wizard to appear. This Wizard will allow you to individually specify each function parameter of your extended method. The Function Prototype Wizard is shown in Figure 7.

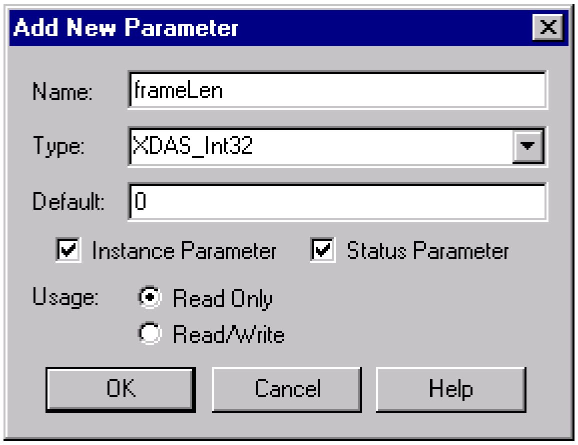
Figure 7. Function Declaration



Function Name: apply
Return Type: XDAS_Void
Function Parameters: 2

```
XDAS_Void FIR_TI_apply(IFIR_Handle handle, float * param,
    float * param);
```

Figure 8. Function Prototype Wizard Parameter Definition



Name: frameLen
Type: XDAS_Int32
Default: 0
[✓] Instance Parameter [✓] Status Parameter
Usage: [✓ Read Only] Read/Write

Function Name	Specify the extended algorithm method’s name
Return Type	Specify the function return type
Function Parameters	Specify the number of function parameters that are passed

The first step in creating a new extended algorithm method is to provide a name for the function. You will want to select an appropriate name for the function (e.g., “apply”, ”filter”, etc.), and type this name into the Function Name list box provided in the Function Prototype Wizard screen.

The next step will be to determine what sort of function return type will be used. You can select the appropriate return type from the Return Type combination box provided in the Function Prototype Wizard screen.

Next, you will need to enter the number of function parameters that your algorithm will require. The default number of parameters will be based upon the number of input and output data frames that you selected in an earlier Component Wizard screen. You can change the number of function parameters by simply changing the number shown in the Function Parameters edit control box provided in the Function Prototype Wizard screen.

Note: You are free to add any number of function parameters that you require for your algorithm. However, keep in mind that the parameter variables that you specified in a previous Component Wizard screen are already provided as “in-scope” variables to your XDAIS algorithm. You will not need to pass them individually.

When all three of the above-referenced items have been entered then select the Next button. This will cause the Function Prototype Wizard to proceed to individual screens in which you will be able to select the Parameter Name and Parameter Type for each function parameter. For example, if you selected four function parameters, then you will need to setup the parameter name and parameter type four times. This is accomplished by simply typing in the desired name for the parameter, and selecting the parameter type from a pull-down combination box. When you have selected the name and data type, then you can select the screen’s Next button. This will take you to the next function parameter. When you have entered appropriate information for all function parameters then the Function Prototype Wizard will close and you will have successfully entered your own extended algorithm method into the Component Wizard for eXpressDSP. To make any changes to an extended algorithm method simply double-click on the desired method that is listed and this will bring up the Function Prototype Wizard so that you can make modifications.

3.7.2 Removing an Extended Method

You can remove an extended method from the Extended Method list box by simply highlighting the function name with a mouse click and then selecting the Remove button. Once you have removed the extended method you will not be able to undo the operation.

3.8 VAB Specific Information

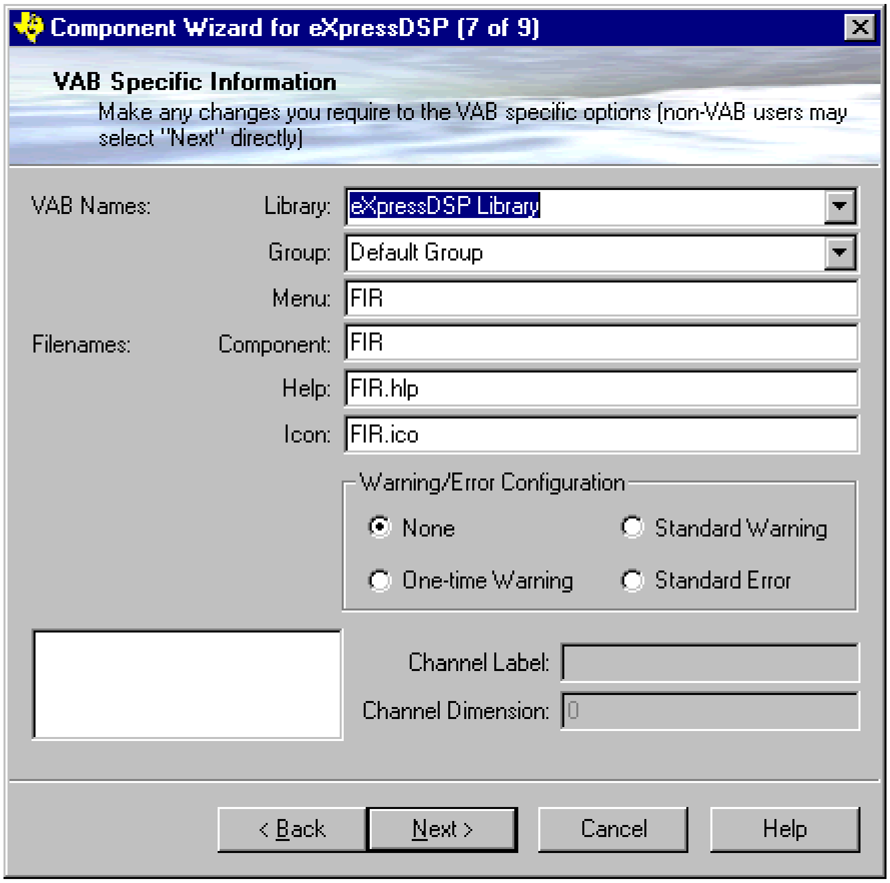
Note: This Component Wizard screen will allow you to enter some “bookkeeping” information that will be used by the VAB software program. The information entered here is not used in any way for the subsequent DSP source code generation, and has no bearing on the TMS320 DSP Algorithm Standard. Non-VAB users may select the Next button directly.

Remember, the resulting VAB block component that is produced will make use of your XDAIS algorithm. But you do not need to use this VAB block in order to use the eXpressDSP-compliant DSP algorithm that has been produced by the Component Wizard (but it does make it convenient to do so).

If you decide to generate a PC-based VAB block component in addition to the DSP source code template, you will need to fill in the information requested by this screen. This information will determine how your VAB block component will be categorized when you want to select it for use within the VAB environment. VAB arranges block components into separate function Libraries (e.g., PC-based, DSP-based, eXpressDSP, or image processing, etc). Each library contains multiple Groups that hold like-minded functions (e.g., DSP functions, logical functions, etc.). Within each group the individual block component is referenced by a menu name (e.g., FFT, magnitude, etc.).

The entry fields shown in Figure 9 are used by VAB only and have no bearing on the TMS320 DSP Algorithm Standard.

Figure 9. VAB Specific Information


Component Wizard for eXpressDSP (7 of 9)

VAB Specific Information

Make any changes you require to the VAB specific options (non-VAB users may select "Next" directly)

VAB Names:

Library: eXpressDSP Library

Group: Default Group

Menu: FIR

Filenames:

Component: FIR

Help: FIR.hlp

Icon: FIR.ico

Warning/Error Configuration

[✓ None] Standard Warning

One-time Warning Standard Error

Channel Label:

Channel Dimension: 0

3.8.1 Entering VAB Names

Library	Specifies the library in which the resulting block component will appear
Group	Specifies the group in which the resulting block component will appear
Menu	Specifies the menu name by which the resulting block component is listed

3.8.2 Entering Filenames

The information entered in this screen is used by the Component Wizard to allow it to generate files in a directory of your choosing. The Component Filename is used to specify the name of the PC-based VAB block component that will be produced. The Help and Icon filename are also associated with the PC-based VAB block component. The Directory is used by the Component Wizard to determine where to create the source code template files.

Component	Specifies the VAB block component name
Help	Specifies the VAB block component help filename
Icon	Specifies the VAB block component icon filename

The source code templates produced by the Component Wizard for eXpressDSP are stored as follows.

3.8.3 Warning/Error Configuration

If you decide to compile the PC based source code template and create a block component for use in the VAB environment, you will have a choice as to how that block handles warnings/error conditions. Selecting one of the listed types will allow your block component to determine how it will report anu algorithm warnings or errors to the VAB program.

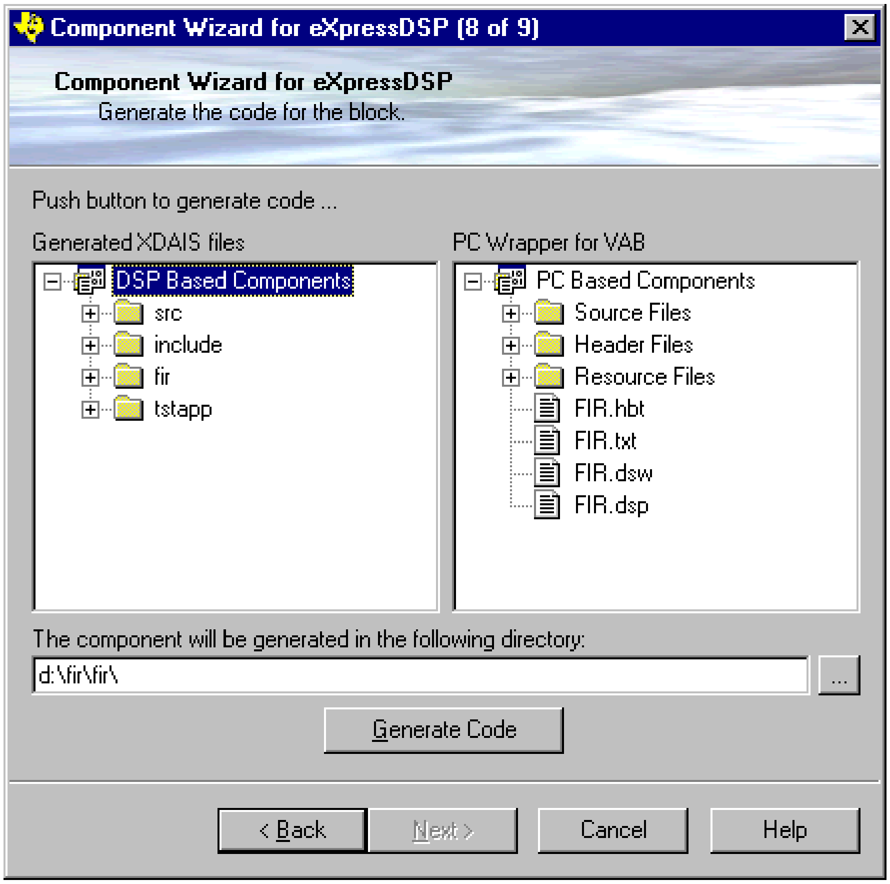
3.8.4 Channel Label and Dimension

These channel labels corresponds to the input and output frames of data and are only used in the PC side of the source code template. The channel labels field allows you to select a name that you would like to see when using the block component in a worksheet.

3.9 Generating the Source Code Template

At this point, you have entered all of the required information so that the Component Wizard for eXpressDSP will be able to generate the source code template for your XDAIS algorithm. You will see a screen similar to the one shown in Figure 10. Do you notice how there are two columns represented? The column on the right lists those source code files that will be generated for the PC-based VAB block component that can make use of your XDAIS algorithm. The column on the left lists those source code files that will be generated for the DSP-based XDAIS algorithm. It is with the DSP-based files that you will be adding your algorithm and source code changes.

Figure 10. Generating a Source Code Template

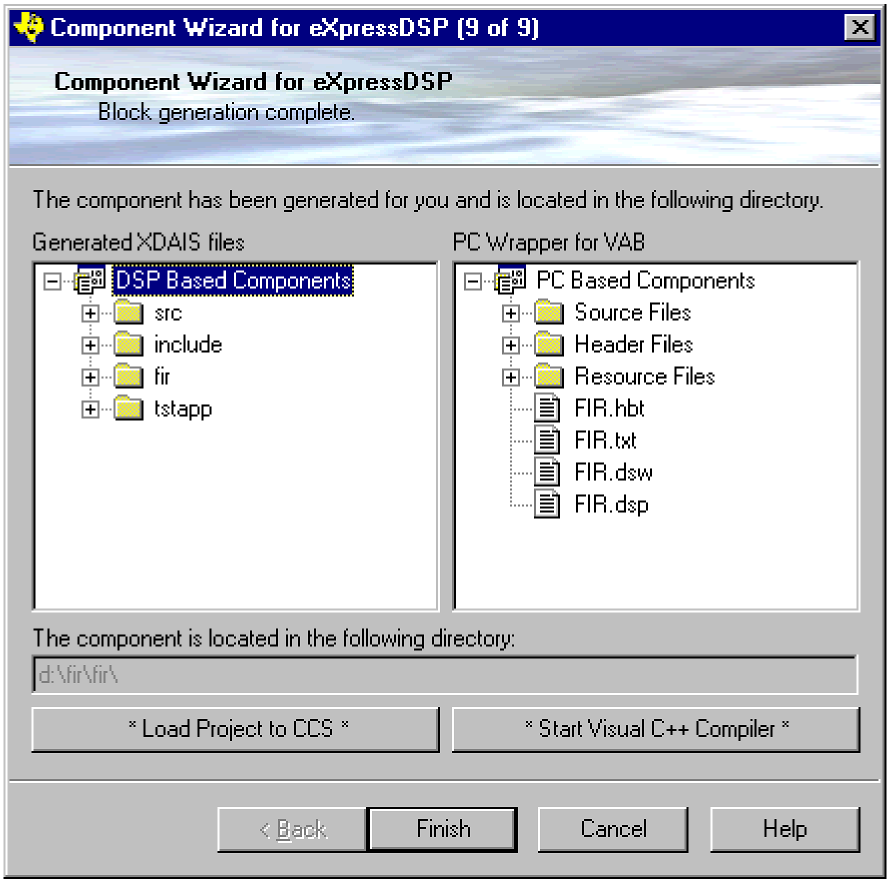


Selecting the Generate Code button on this screen will cause the Component Wizard for eXpressDSP to begin generating the source code files. The file location for these generated files has been determined by the Directory information that you selected in an earlier Component Wizard screen. You can use the Back button to go back a few screens and review this specified file location in the event that you have forgotten exactly where it was that you wanted to place the files. Or, if you'd rather you can simply select the Generate Code button to proceed to the next screen where the project file information will be displayed.

3.10 Launching the Compiler

After you have generated the source code you will see an Component Wizard for eXpressDSP screen similar to the one shown in Figure 11.

Figure 11. Generating Source Code Template


Component Wizard for eXpressDSP (9 of 9)

Component Wizard for eXpressDSP
Block generation complete.

The component has been generated for you and is located in the following directory.

Generated XDAIS files
- DSP Based Components
  - src
  - include
  - fir
  - tstapp

PC Wrapper for VAB
- PC Based Components
  - Source Files
  - Header Files
  - Resource Files
  - FIR.hbt
  - FIR.txt
  - FIR.dsw
  - FIR.dsp

The component is located in the following directory:
d:\fir\fir\

As in the previous screen you will notice that there are two columns represented: the PC-side, and the DSP-side. Each column will list the actual source code files that have been generated. At this point you have several options available to you.

Note: Selecting the Finish button will cause the Component Wizard for eXpressDSP to close. If you do this before selecting the Start Code Composer Studio button then the Component Wizard will close without creating a project file for Code Composer Studio.

3.10.1 Load Project to Code Composer Studio

Selecting the Load Project To Code Composer Studio button will cause the Component Wizard for eXpressDSP to automatically create a Code Composer Studio project file with all of the required files for your XDAIS algorithm and load it in Code Composer Studio. You will need to modify some of the source code files to add your algorithm (see Adding Your Algorithm to the XDAIS Source Code Template).

3.10.2 Starting Visual C/C++ Compiler

Selecting the Start Visual C/C++ Compiler button will cause the Component Wizard for eXpressDSP to automatically launch the Microsoft Visual C/C++ compiler and automatically create a project workspace for you. All that remains to do is Build the project workspace with Visual C/C++; no source code changes are required. The resulting block component file can be used by the VAB software program and allow your new XDAIS algorithm to be used directly within the graphical design environment that VAB provides you.

3.10.3 Closing the eXpressDSP Component Wizard

Now that you have successfully generated the XDAIS source code template with the Component Wizard for eXpressDSP, and generated a Code Composer Studio project and loaded it by selecting the Load Project To Code Composer Studio button you can close the Component Wizard by selecting the Finish button.

3.11 Adding Your Algorithm to the XDAIS Source Code Template

Now that you have successfully generated the XDAIS source code template with the Component Wizard for eXpressDSP, and generated a Code Composer Studio project by selecting the Start Code Composer Studio button, you are ready to begin adding your algorithm.

What File Should I Modify?

Of the source code files generated you will want to concentrate on the vendor-specific C source file. This will be the file that ends with an _ialg.c (e.g., G729ENC_HYP_ialg.c listed above). It is in this file that you will find the extended algorithm method shell(s) that you defined earlier in the Component Wizard.

Where Should I Add My Algorithm?

Once you have opened up the vendor-specific (_ialg.c) source file you should look for your extended algorithm method towards the end of the source file.

You will notice that the comments provided show you which variables are available for use within your extended algorithm method. You will have access to in-scope variables as well as to any input/output data pointers. You should add your own modifications to this source code template as required by your algorithm.

Also, if you have chosen to use MemTab memory descriptors (local arrays) for your algorithm you may need to make some adjustments to the memory allocation. You will need to do this if your MemTab array is size-dependent upon some parameter variable (e.g., a filter length). The Component Wizard produces a template that is based upon a fixed length, and must be user-modified if a variable length is required.

Hint: You may find it helpful as a starting point to simply echo the input data back to the output data and then proceed with compiling and testing. Once you have verified that the project is compiled correctly and runs as you expect, you can go back and add your algorithm.

What Compiler Settings Should I Use?

You will need to make some modifications to the Code Composer Studio compiler options before compiling your project. For example, you will need to make sure that you select the proper endianness is selected for your C6x algorithm. Also, please keep in mind the XDAIS rules listed below:

- [C6x] XDAIS Rule 26: “All C6x algorithms must access all static and global data as far data.” This is accomplished by setting the Code Composer Studio compiler options to use the -ml3 switch.
- [C54x] XDAIS Rules 28-30: If a ‘far’ model is desired, you must select the “Use Far Calls” option (-mf) and select “548” for the Processor Version field. Also verify that *each individual* OBJ file that makes up the algorithm is less than 32K words in size.
- [C55x] XDAIS Rule 32: “All C55x algorithms must access all static and global data as far data; also, the algorithms should be instantiable in a large memory model”. This is accomplished by setting the Code Composer Studio compiler options to use the [–ml] switch.

Once you have selected the appropriate Code Composer Studio compiler options, you can build the project. This will result in all source files being compiled into DSP object files, and an example test program (.out) file being produced.

4 Testing the Algorithm and the Newly Generated XDAIS Interface Code

You can test your algorithm in Code Composer Studio IDE by simply loading the DSP executable program and running it (either a simulation or on a target DSP).

The Component Wizard also creates a sample “main.c” file that you can use as a baseline to implement your test vectors. Refer to this file to see how a consumer of your XDAIS algorithm writes the code to properly integrate the algorithm into a DSP application. You will need to modify the supplied “main.c” file so that it better represents a true test for your algorithm. Run the program with the same test vectors normally used to properly exercise the algorithm. The newly eXpressDSP-compliant algorithm should execute properly and yield the same results as before.

```
#include <std.h>
#include ”g729enc.h”
#include ”g729enc_hyp.h”
int PtrIn0[256];
int PtrOut0[256];

void main()
{
    G729ENC_Handle  handle;
    IG729ENC_Fxns   fxns;
    G729ENC_Params  params;

    fxns = G729ENC_HYP_IG729ENC;
    params = G729ENC_PARAMS;
    params.framesizeIn0 = 256;
    params.framesizeOut0 = 256;
    G729ENC_init();
    if((handle = G729ENC_create(&fxns, &params)) != NULL)
    {
        // Call the control function to modify parameters as the algo runs
        G729ENC_Status  status;
        // get current status
        G729ENC_control(pPtr->handle, G729ENC_GETSTATUS, &status);
        // modify/check status parameters before setting the new status info
        G729ENC_control(pPtr->handle, G729ENC_SETSTATUS, &status);

        /* Call XDAIS algorithm specific routine(s) */
        /* For example: */

        G729ENC_apply(handle, PtrIn0, PtrOut0);

        G729ENC_delete(handle);
    }
    G729ENC_exit();
}
```

Note:

If the new algorithm does not run properly, it is most likely that the algorithm instance was not created successfully. Since the sample <MODULE>_create() function calls the ALG_create() function which dynamically allocates the required memory blocks to create a new algorithm instance, the ALG_create() function will return a NULL handle if the system failed to allocate any of the memory blocks requested by the algorithm. Increase the size of the system heap to faciliate the algorithm's total memory requirement so that the <MODULE>_create() function returns a valid pointer to the algorithm instance object.

5 XDAIS Library Creation

5.1 Creating the XDAIS Library File

When you have determined that the XDAIS algorithm is executing properly, you will want to produce a vendor-specific library file to satisfy the XDAIS packaging convention. This is easily accomplished by running the included makelib.bat file from a DOS prompt (this batch file is located in the same directory as the source code template, but should be moved to the directory where the OBJ files reside). Using the makelib batch file will result in two library files being created. There is your vendor-specific file required for eXpressDSP compliance, as well as an optional library file that allows the XDAIS algorithm to be called through the sample framework files generated by Component Wizard (<MODULE.>.c and <MODULE>.h.

Note: The Component Wizard expects all libraries to be built in Release mode; therefore, the algorithm object files will reside in the \Release folder of the project directory. For the batch file to run successfully, it is necessary to build the algorithm test project in Release mode. This way, the library build scripts can find all the various algorithm object files to build the final library deliverable.

5.2 Testing the XDAIS Library File

To test the newly created XDAIS library file, remove all the source files from the project which make up the library file. Add the library file directly to the Code Composer Studio project just as you would add a source file (i.e., **Project** → **Add Files to Project**). Rebuild the project and run the program. The program should still execute with the same results as expected.

6 Pre-Validation of Algorithms Using QualiTI (Pre-compliance Tool)

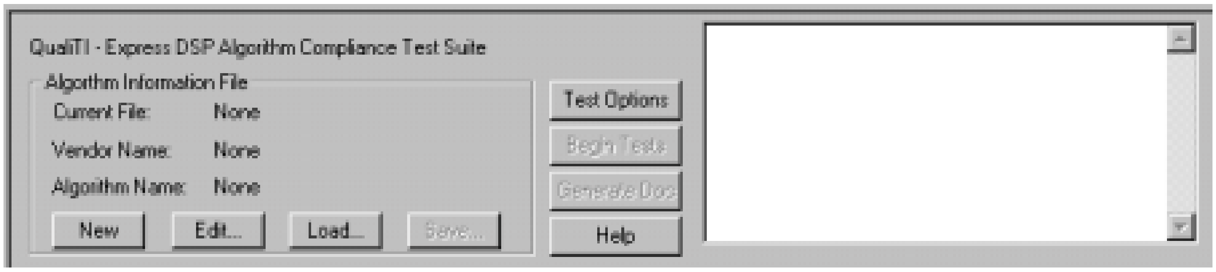
Before submitting the algorithm files for official testing, it can be pre-tested using a Code Composer Studio plug-in, QualiTI. An algorithm submitted for compliance testing consists of an algorithm library to be tested, all associated header files, and documentation. QualiTI operates on the algorithm library and the header files by running various automated tests to check for compliance based on TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352).

The compliance test is comprised of both static and dynamic tests. Static tests are those that do not involve execution of the algorithm. Dynamic tests involve linking the algorithm with a test framework and running the resulting executable on a standalone simulator to determine the PASS/FAIL result.

QualiTI targets both algorithm developers and algorithm testers. The algorithm developers may use the tool to perform pre-compliance of the algorithms before they make a final submission to Texas Instruments (TI) for compliance certification. This helps the algorithm developers to obtain a compliance certificate on the first submission and eliminates the cycle time of resubmitting/retesting an algorithm in case of non-compliance. The algorithm testers (third-party testers employed by TI) use the tool to certify compliance for the algorithms submitted for compliance testing. The algorithm tester's intervention is limited to providing the algorithm information such as Archive, Header files, Vendor Name, Module Name, etc., that are provided in the documentation associated with the algorithm being tested.

6.1 Invoking the Plug-In

To invoke QualiTI, you first need to start Code Composer Studio by double clicking on the Code Composer Studio Icon on your desktop or by selecting Code Composer Studio from the Start menu. Once Code Composer Studio is open, QualiTI can be started through the menu item **Tools** → **Algorithm Standard** → **QualiTI** or by clicking on the QualiTI icon in the tool bar.


QualiTI - Express DSP Algorithm Compliance Test Suite

Algorithm Information File

Current File: None

Vendor Name: None

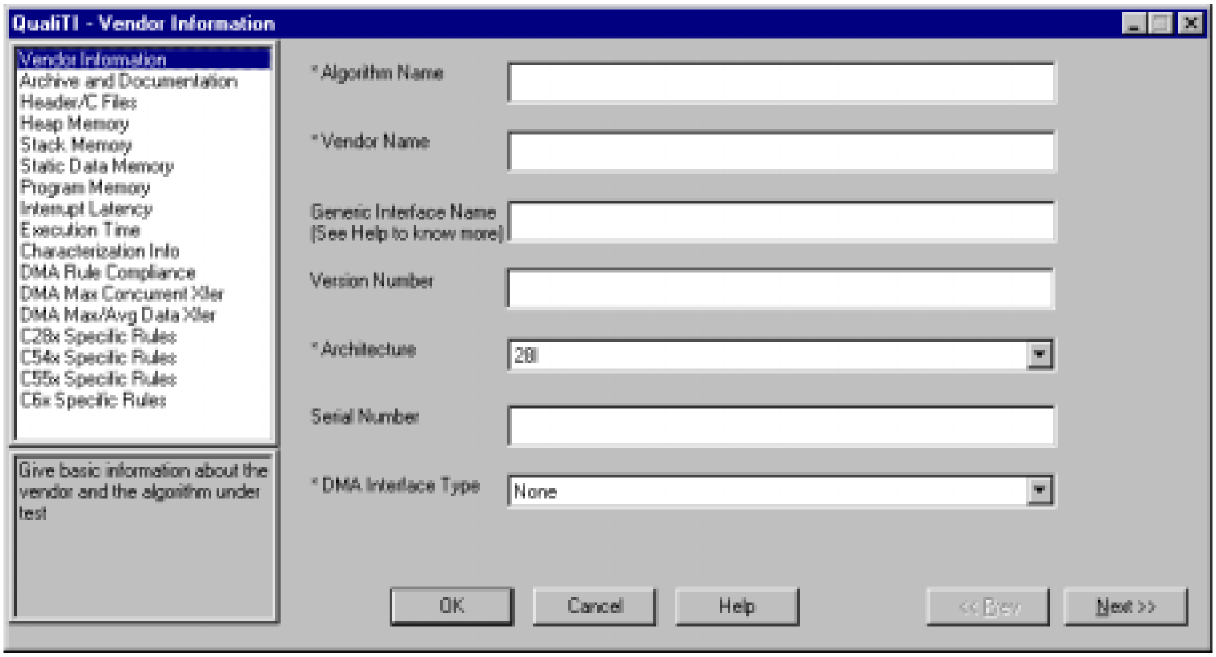
Algorithm Name: None

6.2 Entering the Data to Create an Algorithm Information File

Once QualiTI is started, it appears with its interface docked to the lower edge of Code Composer Studio. The QualiTI interface has two distinct sections: the Algorithm Information File and the Output Window.

The Algorithm Information File section gives the user basic information about the algorithm being tested. The Current File field indicates the algorithm information file being used. This file contains detailed information about the algorithm being tested, such as vendor name, algorithm name, algorithm archive, header files, and other characterization data as explained below. The **New** button deletes a previously created algorithm information database, creates a new one, and takes you directly to the algorithm information dialog box. The user is prompted to save if the loaded database has been changed.

Figure 12. Algorithm Information Dialog Box



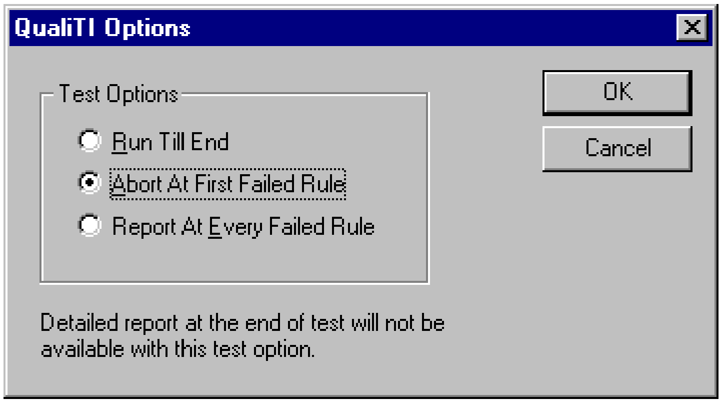
Give basic information about the vendor and the algorithm under test

* Algorithm Name	
* Vendor Name	
Generic Interface Name (See Help to know more)	
Version Number	
* Architecture	28l
Serial Number	
* DMA Interface Type	None

When the QualiTI plug-in is opened for the first time, it displays an empty algorithm information database that can be edited by using the **Edit** button. All the data entered through the QualiTI interface can be saved as an XML file with the extension .alg. After entering the data through the interface, use the **Save** button to save the file for later retrieval. This feature eliminates repetitive entry of algorithm information that can occur when the algorithm is being retested after fixing non-compliances from a previous test run. A file saved through QualiTI can be retrieved by using the **Load** button.

6.3 Executing the Test

After the necessary information is entered and saved in the algorithm information pages, you can proceed to the test step. Optionally, before the tests can begin, you can choose any of the Test Options from the plug-in opening screen to control the test flow. By default, the **Run Till End** option is selected and this makes QualiTI test all the rules regardless of whether each rule passes or fails. If the **Abort At First Failed Rule** option is selected, when the test encounters its first failed rule, QualiTI will stop testing as soon as a rule fails. If you wish to be prompted at every failed rule, then you can choose the option **Report At Every Failed Rule**. You cannot control the order in which the rules are tested. Once you decide on the Test Options you can click on the **Begin Test** button, which will initiate testing. You will then be able to view the complete test report.


QualiTI Options

Test Options
Run Till End [✓ Abort At First Failed Rule] Report At Every Failed Rule

Detailed report at the end of test will not be available with this test option.

6.4 The Document Generator

QualiTI must have detailed information about an algorithm before testing it. This information can be input manually through the QualiTI GUI or can be loaded via an algorithm information file (.alg file). QualiTI also generates data about the algorithm in terms of heap data memory requirements, static data memory requirements, and program data memory usage. All of this information can be saved in the form of an easily readable HTML document by clicking on the **Generate Doc** button (QualiTI, Opening screen). This document should be sent to TI as the associated document for the algorithm to be tested. This helps maintain uniformity across documents submitted from various algorithm vendors. Make sure to review the generated HTML file for accuracy and completeness before submitting for compliance.

6.5 Submitting Your Algorithm for eXpressDSP-Compliance Testing

Texas Instruments has a program in place that allows you to submit your newly created XDAIS algorithm for compliance testing. In addition to the vendor-specific library file, you will need to supply a document file that contains information about your algorithm. Contact Texas Instruments for additional details about submitting your algorithm for compliance testing.

7 Conclusion

This document helps the XDAIS algorithm developer through the entire XDAIS conversion process. When completed, the algorithm should be submitted immediately for compliance testing. Make sure all header files that are submitted (file names and contents) follow the XDAIS naming conventions and header file rules. The following files should be submitted:

- <MODULE>_<VENDOR>_<ARCH>.html (normally generated automatically by QualiTI)
- The XDAIS Library file
- <MODULE>_<VENDOR>.h (and any other header files included in this file)
- i<MODULE>.h (and any other header files included in this file)
- i<MODULE>.c
- <MODULE>.h
- <MODULE>.c
- Any additional header files inherently included by the above header files
- The actual linker command file used in your test application
- An algorithm User's Guide (optional, but highly recommended)

Appendix A

XDAIS Name Tables

Please refer to Chapter 1, Section 2, *Module-Specific Abstract Interface*, for instructions on how to enter information into the following tables.

Table 3. Name Selection

Module Name	Vendor Name	Variant	Architecture	Memory Model (optional)
			Cores: '24', 28', 54', '55', '62', '64', '67'	'f' = far calls/returns 'e' = big endian 'm' = mixed calls

Table 4. Instance Creation Parameters

```
typedef struct I<MODULE>_Params {
```

No.	Params Type	Params Name
1.		
2.		
3.		
4.		

```
} I<MODULE>_Params;
```

Table 5. Real-Time Status and Control Parameters

```
typedef struct I<MODULE>_Status {
```

No.	Status/Control Type	Status/Control Name
1.		
2.		
3.		
4.		

```
} I<MODULE>_Status;
```

Table 6. Extended IALG (IMODULE) Methods

No.	Return Type	Method Name	Param1 Type and Name[†]	Param2 Type and Name	Param3 Type and Name
1					
2					
3					
4					
5					

[†] The IALG_Handle parameter does not need to be specified because the Component Wizard will insert it by default.

Template File Descriptions

The Component Wizard automatically generates most of the following files that make up the XDAIS layers of abstraction:

```
*APPLICATION FRAMEWORK*          Concrete Interface (API)
                                 (framework-specific set of sample
                                 functions for framework to manage
                                 algorithm instance objects)
<MODULE>.c (implementation of client API functions)
<MODULE>.h (client API interface definitions)
```

```
*MODULE -- PUBLIC*               Module-Specific Interface (IMODULE)
                                 (abstract interface)
i<MODULE>.c (definition of default parameter structure settings)
i<MODULE>.h (abstract interface definition header -- PUBLIC data types & methods)
```

```
*ALGORITHM -- PRIVATE*              Vendor Specific Interface (IALG)
                                    (algorithm-specific)
<MODULE> <VENDOR>.h (vendor implementation header file; used by application)
<MODULE> ialg.c (vendor-specific algorithm functions)
<MODULE> ialgvt.c (function v-table definitions)
```

Appendix C

XDAIS Memory and Performance Characterization

This appendix contains a set of tables that help characterize the TMS320 DSP Algorithm Standard (known as XDAIS) algorithm. This information can be extremely useful for integrating algorithms into a system that has limited memory. Refer to the *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352) document for details on how to complete the following tables with the correct information.

Table 7. Module

Module	Vendor	Variant	Arch	Model	Version	Doc Date	Library Name
G729ENC	TI	none	54	far	none	05.05.2000	g729enc_ti.l54f

Table 8. ROMable (Rule 5)

Yes	No
X	

Table 9. Heap Data Memory (Rule 19)

memTab	Attribute	Size (bytes)	Align (MAUs)	Space
0	Persist	54	0	External
1	Scratch	256	32	DARAM0
2	Scratch	130	4	SARAM0
3	Persist	712	0	External
4	Scratch	656	64	DARAM1
5	Persist	256	16	SARAM1

Note: The unit for size is (8–bit) byte and the unit for align is Minimum Addressable Unit (MAUs).

Table 10. Stack Space Memory (Rule 20)

	Size (bytes)	Align (MAUs)
Worst Case	256	0

Table 11. Static Data Memory (Rule 21)

.data					.bss				
Object File	Size (bytes)	Align (MAUs)	Read/ Write	Scratch	Object File	Size (bytes)	Align (MAUs)	Read/ Write	Scratch
g729encTI.obj	325	0	R	No	g729enc_ti_ ialgvt.obj	22	0	R	No

Table 12. Program Memory (Rule 22)

	Code	
Code Sections	Size (bytes)	Object File
.text	7,782	g729encTI.obj
.text:algAlloc	128	g729encTI.obj
.text:algInit	209	g729encTI.obj
.text:algFree	86	g729encTI.obj
.text:algActivate	7	g729encTI.obj
.text:algMoved	9	g729encTI.obj
.text:algDeactivate	7	g729encTI.obj
.text:init	3	g729encTI.obj
.text:exit	3	g729encTI.obj
.cinit	24	g729encTI.obj

Table 13. Interrupt Latency (Rule 23)

Operation	Typical Call Frequency (microsec)	Worst-Case (Instruction Cycles)
encode()	2,000	50

Table 14. Period / Execution Time (Rule 24)

Operation	Typical Call Frequency (microsec)	Worst-Case Cycles/Period
encode()	2,000	16,000

Table 15. C55x Stack Configuration (Rule 31)

This algorithm was based on the following stack configuration:

x Dual 16-bit stack with fast return

Dual 16-bit stack with slow return

32-bit stack with slow return (default at hardware reset)

Table 16. C55x B-Bus Data Access (Rule 34)

Number of memTab [] b locks that are accessed by the B-bus	**Block Numbers**
2	1, 2

Data section names that are accessed by the B-bus	**Block Numbers**
.data	
.coefwords	

Appendix D

eXpressDSP-Compliance Report

This section contains the official checklist used during compliance testing. It is recommended to go through the entire checklist before submission.

TMS320 DSP Algorithm Standard (XDAIS)

XDAIS Compliance Testing

Compliance Test Report (preliminary)

Note: All references to Rules and Guidelines are from the February 2000 revision of SPRU352.

Date:

Vendor:

Algorithm

Incoming Inspection

documentation:

header file(s):

Rule 1: All algorithms must follow the run-time conventions imposed by TI's implementation of the C programming language.'

Report: Vendor should supply statement that this is correct.

Rule 2: All algorithms must be reentrant within a preemptive environment (including time-sliced preemption).

Report: Vendor must state that algorithm is reentrant according to the definition of the DSP Algorithm Standard

Rule 3: All algorithm data references must be fully relocatable (subject to alignment requirements). That is, there must be no "hard coded" data memory locations.

Report: Vendor should supply statement that this is correct.

Rule 4: All algorithm code must be fully relocatable. That is, there can be no hard coded program memory locations.

Report: Vendor should supply statement that this is correct.

Rule 5: Algorithms must characterize their ROM-ability; i.e., state whether they are ROM-able or not.

Report: Vendor must document that their code is ROM-able or not.

Rule 6: Algorithms must never directly access any peripheral device. This includes but is not limited to on-chip DMAs, timers, I/O devices, and cache control registers.

Report: Vendor should supply statement that this is correct.

Rule 7: All header files must support multiple inclusions within a single source file.

Report:

Rule 8: All external definitions must be either API identifiers or API and vendor prefixed.

Report:

Rule 9: All undefined references must refer either to the operations specified in Appendix B (a subset of C runtime support library functions and the DSP/BIOS) or other eXpressDSP-compliant modules.

Report:

Rule 10: All modules must follow the naming conventions of the DSP/BIOS for those external declarations disclosed to the client.

Report: Vendor should supply statement that this is correct.

Rule 11: All modules must supply an initialization and finalization method.

Report:

Rule 12: All algorithms must implement the IALG interface.

Report:

Rule 13: Each of the IALG methods implemented by an algorithm must be independently relocatable.

Report:

Rule 14: All abstract algorithm interfaces must derive from the IALG interface.

Report:

Fxns:

Params:

Status:

Rule 15: Each eXpressDSP-compliant algorithm must be packaged in an archive which has a name that follows a uniform naming convention.

Note: Refer to section 3.4 of the TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352) for further guidance on file naming conventions.

Report:

Rule 16: Each eXpressDSP-compliant algorithm header must follow a uniform naming convention.

Header file name should be of the form:
<module><vers>_<vendor>_<variant>.h

Report:

Rule 17: Different versions of an eXpressDSP-compliant algorithm from the same vendor must follow a uniform naming convention.

Report:

Rule 18: If a module's header includes definitions specific to a "debug" variant, it must use the symbol _DEBUG to select the appropriate definitions;

_DEBUG is defined for debug compilations and only for debug compilations.

Report:

Rule 19: All algorithms must characterize their worst-case heap data memory requirements (including alignment).

Report:

Rule 20: All algorithms must characterize their worst-case stack space memory requirements (including alignment).

Report:

Rule 21: Algorithms must characterize their static data memory requirements.

Report:

Rule 22: All algorithms must characterize their program memory requirements.

Report:

Rule 23: All algorithms must characterize their worst-case interrupt latency for every operation.

Report:

Rule 24: All algorithms must characterize the typical period and worst-case execution time for each operation.

DSP-Specific Rules

Rule 25: All C6x algorithms must be supplied in little endian format.

Report: C6x Vendors should supply a statement that this is correct.

Rule 26: All C6x algorithms must access all static and global data as far data.

Report: C6x Vendors should supply a statement that this is correct.

Rule 27: C6x algorithms must never assume placement in on-chip program memory; i.e., they must properly operate with program memory operated in cache mode.

Report: C6x Vendors should supply a statement that this is correct.

Rule 28: On processors that support large program model compilation, all core run-time support functions must be accessed as far functions; for example, on the C54x, the calling function must push both the XPC and the current PC.

Report: C54x Vendors should supply a statement that this is correct.

Rule 29: On processors that support large program model compilation, all algorithm functions must be declared as far functions; for example, on the C54x, callers must push both the XPC and the current PC and the algorithm functions must perform a far return.

Report: C54x Vendors should supply a statement that this is correct.

Rule 30: On processors that support an extended program address space (paged memory), the code size of any object file should never exceed the code space available on a page when overlays are enabled.

Report: C54x Vendors should supply a statement that this is correct.

Rule 31: All C55x algorithms must document the content of the stack configuration followed.

Report:

Rule 32: All C55x algorithms must access all static and gobal data as far data; also, the algorithms should be instantiable in a large memory model.

Report: C54x Vendors should supply a statement that this is correct.

Rule 33: C55x algorithms must never assume placement in on-chip memory; i.e., they must properly operate with program memory operated in instruction cache mode.

Report: C55x Vendors should supply a statement that this is correct.

Rule 34: All that access data by B-bus must document: (1) the instance number of the IALG_MemRec structure that is accessed by the B-bus (heap data), and (2) the data section name that is accessed by the B-bus (static data).