Using External References in Algorithms Compliant with the TMS320 DSP Algorithm Standard

Abhinaba Basu *XDAIS Engineering Team*

ABSTRACT

The TMS320 DSP Algorithm Standard (referred to as XDAIS) allows algorithms to make external references to functions implemented either in standard libraries or in other eXpressDSP-compliant libraries. However, in some cases, algorithms may be required to access specific functionalities that can be accomplished only in frameworks or in other externally linked modules.

This document discusses how eXpressDSP-complaint algorithms can access functions implemented in externally linked modules using registered function pointers. This mechanism of function calls is known as *callbacks*. This document also contains code snippets to show how algorithms and frameworks can use callback functions that are registered, by using the instance creation parameter.

Contents

List of Tables

1 Introduction

TMS320 DSP Algorithm Standard specification currently allows algorithms to call functions defined either in standard libraries like rts, dsplib, imglib, acpy, etc. or in other eXpressDSP-compliant libraries. In some cases, the algorithms are required to access specific functionalities that can be accomplished only by frameworks (e.g., to actively acquire data) or may be too specific to get included in any standard library.

These functions may either be implemented in the framework itself or may come as a part of another library that is linked to create the final application. The algorithms must access these functions in such a way that name-space pollution is avoided, and such that the same functions can be shared across multiple algorithms or multiple instances of the same algorithm.

2 Callback − Making References to Externally Implemented Functions Through Pointers

Algorithms may access externally defined functions using the following steps:

1. The algorithm documents the functionality it requires
2. The algorithm's client implements the function or links in a library, which implements it
3. The client passes the function's pointer to the algorithm
4. The algorithm stores the pointer internally and makes calls to the function using the pointer

This mechanism of function calls is known as *callbacks*. Callbacks provide a solution for an algorithm to make references to externally linked modules without causing name-space pollution, and at the same time allows sharing of functions across multiple algorithms.

The client of an eXpressDSP-complaint algorithm can register function pointers with the algorithm in several ways. In the following sections, we discuss how the instance creation parameter may be used to register callback functions.

3 Registering Function Pointers Through an Instance Creation Parameter

The algorithm may define members of the instance creation parameter structure I<MOD>_Params that are actually functions pointers. The algorithm must document its callback functionality requirements using the following table.

Table 1. Callback Requirement Characterization

Structure Member	Prototype of the Function	Pre-condition	Post-condition	Mandatory	Description

The framework implements the callback functions based on the algorithm's documentation and then sets the relevant members of the instance creation parameter to the function pointers before calling algAlloc(). The framework passes the same parameters to algInit(). The algorithm in algInit copies these function pointers from the instance creation parameter to corresponding members in the algorithm instance object. After the algInit() phase, it makes calls to the functions through these pointers.

Table 2. Example of Callback Requirement Characterization

Structure Member	Prototype of the Function	Pre-condition	Post-condition	Mandatory	Description
pStatusFunc	XDAS_Bool pStatusFunc (XDAS_UInt16 currSize)	The parameter currSize contains the number of bytes of data already copied by the copy method. This function will be called after every 256 bytes of data copy.	If the function returns TRUE, copy will continue else copying will be aborted.	No	This callback is used as an event notification to indicate data copy progress.
pMemcpyFunc	XDAS_Void pMemcpyFunc (XDAS_Void *s1, const XDAS_Void *s2, XDAS_UInt32 n);	s1 and s2 are valid pointers to non−overlapping memory locations	n bytes have been copied from s1 to s2	Yes	This function is used for copying data from one memory location to another.

Example COPY_TI Algorithm Code

```
/*
 * ======================================================================================
 * Instance creation parameters in icopy.h
*/
typedef struct ICOPY_Params {
    Int size;
    XDAS_Void *(*pMemcpyFunc)(XDAS_Void *s1, const XDAS_Void *s2, XDAS_UInt32 n);
    XDAS_Bool (*pStatusFunc) (XDAS_UInt16 currSize); /* Callback pointer */
} ICOPY_Params;
/*
 * ======================================================================================
 * Instance object defined in copy_ti_ialg.c
*/
typedef struct COPY_TI_Obj {
    IALG_Obj        alg;
    XDAS_Void *(*pMemcpyFunc)(XDAS_Void *s1, const XDAS_Void *s2, XDAS_UInt32 n);
    XDAS_Bool  (*pStatusFunc) (XDAS_UInt16 currSize);
} COPY_TI_Obj;
/*
 * ======================================================================================
 * algInit implementation in copy_ti_ialg.c
*/
Int COPY_TI_initObj(IALG_Handle handle,
```

```
                const IALG_MemRec memTab[], IALG_Handle p, const
                IALG_Params *COPYParams)
{
    COPY_TI_Obj *COPY = (Void *)handle;
    const ICOPY_Params *params = (Void *)COPYParams;
    /* Set default parameters if none is given */
    if(params == NULL){
        params = &ICOPY_PARAMS;
    }
    /* If mandatory callback function pointer is not given the fail init */
    if (params->pMemcpyFunc == NULL)
       return (IALG_EFAIL);

    /* Store callback function pointer in instance object */
    COPY->pMemcpyFunc = params->pMemcpyFunc;
    COPY->pStatusFunc = params->pStatusFunc;
    return (IALG_EOK);
}
/*
 * ======================================================================
 * Make callback thru pointer stored in instance object in an IMOD method
*/
XDAS_UInt16 COPY_TI_copy(ICOPY_Handle handle, XDAS_Void * inBuf, XDAS_Void * outBuf,
XDAS_UInt16 bufLen)
{
    COPY_TI_Obj *COPY = (Void *)handle;
    /* Call mandatory callback function pointer pMemcpyFunc without verifying */
    COPY->pMemcpyFunc(...);

    /* Call non-mandatory pStatusFunc after verifying that it is not NULL */
    if (COPY->pStatusFunc != NULL){
        COPY->pStatusFunc (...);
    }
    /* do other processing ... */
}
Example Framework Code For the COPY_TI Algorithm
/*
 * ======================================================
 * Implement callback function
```

```
*/
XDAS_Bool StatusFunc (XDAS_UInt16 currSize)
{
    /* StatusFunc processing code ...  */
    return TRUE;
}
Void main()
{
    COPY_Params copyParams;
    COPY_Handle copyHandle;
    /* Do other variable declarations and initializations ... */
    COPY_init();
    /* Initialize instance creation params with function ptr */
    copyParams = ICOPY_PARAMS;
    copyParams.pMemcpyFunc = memcpy;      /* callback function from external library */
    copyParams.pStatusFunc = StatusFunc; /* client implemented callback function */
    /* Create algorithm instance */
    if((copyHandle = COPY_create (&COPY_TI_ICOPY, &copyParams)) != NULL) {
        COPY_copy(copyHandle, inBuff, outBuff, BUFFLEN ); /* run IMOD methods */
        COPY_delete(copyHandle);
    }
    COPY_exit();
}
```

3.1 Default Instance Creation Parameters

If the framework does not pass an instance creation parameter pointer as an argument to algAlloc or algInit, then the algorithm uses the default instance creation parameter I<MOD>_PARAMS. The algorithm vendor provides I<MOD>_PARAMS either in a separate C file or in the algorithm archive. The algorithm can provide a default implementation of the callback function and initialize the callback-function pointer member in I<MOD>_PARAMS to it, or it can choose to initialize it to NULL.

COPY_TI Algorithm's Default Instance Creation Parameter

```
/*
 * ======================================================================
 * Default implementation of callback function *
*/
XDAS_Bool StatusFunc (XDAS_UInt16 currSize)
{
    /* StatusFunc default implementation code ... */
}
```

```
/*
* ========================================================================
 * Default instance creation parameter
*/
ICOPY_Params ICOPY_PARAMS = {
    sizeof(ICOPY_Params),
    StatusFunc,
    /* Initialize other members ... */
};
```

3.2 NULL Callback Function Pointer

It is not mandatory for application frameworks to implement and give valid callback function pointers to algorithms. The framework may initialize callback-function pointer member of an instance creation parameter to NULL.

If the callback function is essential for the algorithm to work, then it must fail in algInit by returning IALG_EFAIL. If the algorithm uses the callback functions for non-essential functionality such as event notification for real-time analysis/debugging, and can work without it, then it may choose to proceed by returning IALG_OK. The algorithm informs the framework whether a callback function is essential or not through the “Mandatory” field in the “Callback Requirements Characterization” (see Table 1).

Note: The algorithm should not make calls to callback function pointers set to NULL.

3.3 Characterizing Algorithm Methods That Use Callbacks

If an algorithm uses callbacks in a method, then its characterization data (execution time, interrupt latency) becomes dependent of the callback function. In such cases, the algorithm should provide characterization data for the method in such a way that the relationship between the method and the callback functions become evident.

COPY_TI Algorithm Code

```
XDAS_UInt16 COPY_TI_copy(ICOPY_Handle handle, XDAS_Void * inBuf, XDAS_Void * outBuf,
XDAS_UInt16 bufLen)
{
   COPY_TI_Obj *COPY = (Void *) handle;
   if(COPY->pStatusFunc)
       COPY->pStatusFunc (...);
   COPY->pMemcpyFunc (...);
   /* Other processing code ... */
}
```

Table 3. Execution Time Characterization for COPY_TI Algorithm Using Callback

Operation	Period	Worst-Case Interrupt Lat. (Instr. Cycle)	Worst-Case Cycles/Period
copy	2250ms	0 + pStatusFunc + pMemcpyFunc	26700 + pStatusFunc + pMemcpyFunc

3.4 Changing Callback Functions

In this method,once an algorithm instance has been initialized, the callback function cannot be changed. This is a limitation of using an instance creation parameter to register callback functions.

4 References

1. *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352)
2. *TMS320 DSP Algorithm Standard API Reference* (SPRU360)

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265