Modem Integrator Algorithm (MODINT) User's Guide

SPIRIT CORP
DSP Software Source
www.spiritDSP.com/CST

Literature Number: SPRU636
March 2003

TEXAS INSTRUMENTS

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Preface

Read This First

About This Manual

For the purposes of this API description, the following abbreviations are used:

ADC	Analog-to-Digital Convertor
AFE	Analog Front End
DAC	Digital-to-Analog Convertor
HDLC	High-level Data Link Control protocol
MDP	V.22bis/V.32bis Modem Data Pump
PSTN	Public Switched Telephone Network
V42	V.42 algorithm library
V42B	V.42bis algorithm library
XDAS	TMS320 DSP Algorithm Standard

Related Documentation From Texas Instruments

Using the TMS320 DSP Algorithm Standard in a Static DSP System (SPRA577)

TMS320 DSP Algorithm Standard Rules and Guidelines (SPRU352)

TMS320 DSP Algorithm Standard API Reference (SPRU360)

Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers (SPRA579)

The TMS320 DSP Algorithm Standard (SPRA581)

Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface (SPRA716)

Related Documentation

ITU-T Recommendation V.32. A family of 2-wire, duplex modems operating at data signalling rates of up to 9600 bit/s for use on the general switched telephone network and on leased telephone-type circuits, 1993.

ITU-T Recommendation V.32bis. A duplex modem operating at data signalling rates of up to 14400 bit/s for use on the general switched telephone network and on leased point-to-point 2-wire telephone-type circuits, 1991.

ITU-T Recommendation V.22. 1200 bits per second duplex modem standardized for use in the general switched telephone network and on point-to-point 2-wire leased telephone-type circuits, 1993.

ITU-T Recommendation V.22bis. 2400 bits per second duplex modem using the frequency division technique standardized for use on the general switched telephone network and on point-to-point 2-wire leased telephone-type circuits, 1993.

ITU-T Recommendation V.25. Automatic answering equipment and general procedures for automatic calling equipment on the general switched telephone network including procedures for disabling of echo control devices for both manually and automatically established calls.

ITU-T Recommendation V.42. Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion

ITU-T Recommendation V.42bis. Data compression procedures for data circuit terminating equipment (DCE) using error correction procedures

Spirit Corp. *V.42/V.42bis User's Guide*, 2002

Spirit Corp. *Modem Data Pump User's Guide*, 2000

Trademarks

Software Copyright

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/products/index.htm
DSP Solutions	http://www.ti.com/dsp
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm
Microcontroller Home Page	http://www.ti.com/sc/micro
Networking Home Page	http://www.ti.com/sc/docs/network/nbuhomex.htm
Military Memory Products Home Page	http://www.ti.com/sc/docs/military/product/memory/mem_1.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580	
TI Literature Response Center U.S.A.	(800) 477-8924	
Software Registration/Upgrades	(972) 293-5050	Fax: (972) 293-5967
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285	
U.S. Technical Training Organization	(972) 644-5580	
Microcontroller Hotline	(281) 274-2370	Fax: (281) 274-4203 Email: micro@ti.com
Microcontroller Modem BBS	(281) 274-3700 8-N-1	
DSP Hotline		Email: dsph@ti.com
DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs		
Networking Hotline		Fax: (281) 274-4027
		Email: TLANHOT@micro.ti.com

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:		
Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32
Email: epic@ti.com		
Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	
Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/		

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com Email: micro@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

For product price & availability questions, please contact your local Product Information Center, or see www.ti.com/sc/support http://www.ti.com/sc/support for details.

For additional CST technical support, see the TI CST Home Page (www.ti.com/telephonyclientside) or the TI Semiconductor KnowledgeBase Home Page (www.ti.com/sc/knowledgebase).

If you have any problems with the Client Side Telephony software, please, read first the list of Frequently Asked Questions at http://www.spiritDSP.com/CST.

You can also visit this web site to obtain the latest updates of CST software & documentation.

Contents

Figures

Tables

Notes, Cautions, and Warnings

Chapter 1

Introduction to Modem Integrator (MODINT) Algorithms

This chapter briefly describes the Voice Activity Detector algorithms and supported products used with the TMS320C5400 platform.

For the benefit of users who are not familiar with the TMS320 DSP Algorithm Standard (XDAIS), brief descriptions of typical XDAIS terms are provided.

Topic **Page**

1.1 Introduction

This document describes Standard Modem Integrator developed by SPIRIT Corp. for TMS320C54xx platform. SPIRIT Modem Integrator is intended to be used in standalone modems, embedded equipment, etc.

This module is a service that integrates Modem Data Pump, V.42, V.42bis and provides missing functionality (such as V.14, retrain initiation logic, etc). The module does not include the referred algorithms and just links them.

TheSPIRIT Modem Integrator (MODINT) software is a fully TMS320 DSP Algorithm Standard (XDAIS) compatible, reentrant code. The MODINT interface complies with the TMS320 DSP Algorithm Standard and can be used in multitasking environments.

The TMS320 DSP Algorithm Standard (XDAIS) provides the user with object interface simulating object-oriented principles and asserts a set of programming rules intended to facilitate integration of objects into a framework.

The following documents provide further information regarding the TMS320 DSP Algorithm Standard (XDAIS):

- *Using the TMS320 DSP Algorithm Standard in a Static DSP System* (SPRA577)
- *TMS320 DSP Algorithm Standard Rules and Guidelines* (SPRU352)
- *TMS320 DSP Algorithm Standard API Reference* (SPRU360)
- *Technical Overview of eXpressDSP-Compliant Algorithms for DSP Software Producers* (SPRA579)
- *The TMS320 DSP Algorithm Standard* (SPRA581)
- *Achieving Zero Overhead with the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716)

However, if the user prefers to have non-XDAIS-compliant interface, for example, when a framework is not XDAIS-oriented (it usually means that dynamic memory management is not supported), the XDAIS interface can be omitted, as it is merely a wrapper for the original interface.

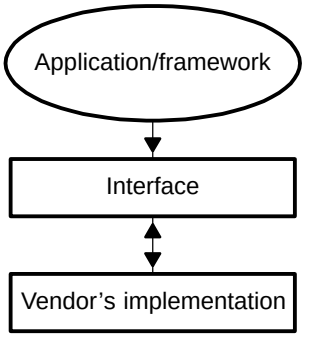
1.2 XDAIS Basics

This section instructs the user on how to develop applications/frameworks using the algorithms developed by vendors. It explains how to call modules through a fully eXpress DSP-compliant interface.

Figure 1-1 illustrates the three main layers required in an XDAIS system:

- Application/Framework layer
- Interface layer
- Vendor implementation. Refer to appendix A for a detailed illustration of the interface layer.

Figure 1-1. XDAIS System Layers



1.2.1 Application/Framework

Users should develop an application in accordance with their own design specifications. However, instance creation, deletion and memory management requires using a framework. It is recommended that the customer use the XDAIS framework provided by SPIRIT Corp. in ROM.

The framework in its most basic form is defined as a combination of a memory management service, input/output device drivers, and a scheduler. For a framework to support/handle XDAIS algorithms, it must provide the framework functions that XDAIS algorithm interfaces expect to be present. XDAIS framework functions, also known as the ALG Interface, are prefixed with "ALG_". Below is a list of framework functions that are required:

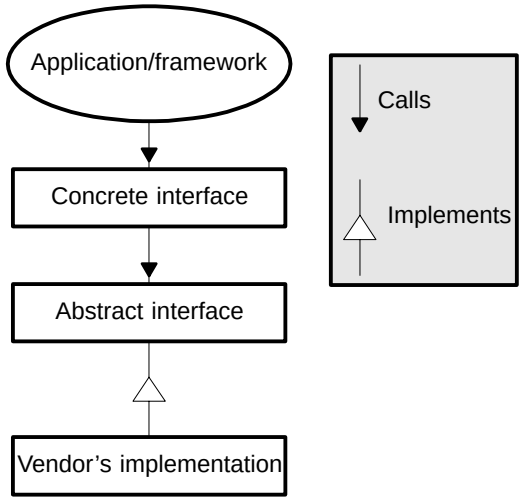
- `ALG_create` - for memory allocation/algorithm instance creation
- `ALG_delete` - for memory de-allocation/algorithm instance deletion
- `ALG_activate` - for algorithm instance activation

- ALG_deactivate - for algorithm instance de-activation
- ALG_init - for algorithm instance initialization
- ALG_exit - for algorithm instance exit operations
- ALG_control - for algorithm instance control operations

1.2.2 Interface

Figure 1-2 is a block diagram of the different XDAIS layers and how they interact with each other.

Figure 1-2. XDAIS Layers Interaction Diagram



1.2.2.1 Concrete Interface

A concrete interface is an interface between the algorithm module and the application/framework. This interface provides a generic (non-vendor specific) interface to the application. For example, the framework can call the function MODULE_apply() instead of MODULE_VENDOR_apply(). The following files make up this interface:

- Header file MODULE.h - Contains any required definitions/global variables for the interface.
- Source File MODULE.c - Contains the source code for the interface functions.

1.2.2.2 Abstract Interface

This interface, also known as the IALG Interface, defines the algorithm implementation. This interface is defined by the algorithm vendor but must comply with the XDAIS rules and guidelines. The following files make up this interface:

- Header file `iMODULE.h` - Contains table of implemented functions, also known as the IALG function table, and definition of the parameter structures and module objects.
- Source File `iMODULE.c` - Contains the default parameter structure for the algorithm.

1.2.2.3 Vendor Implementation

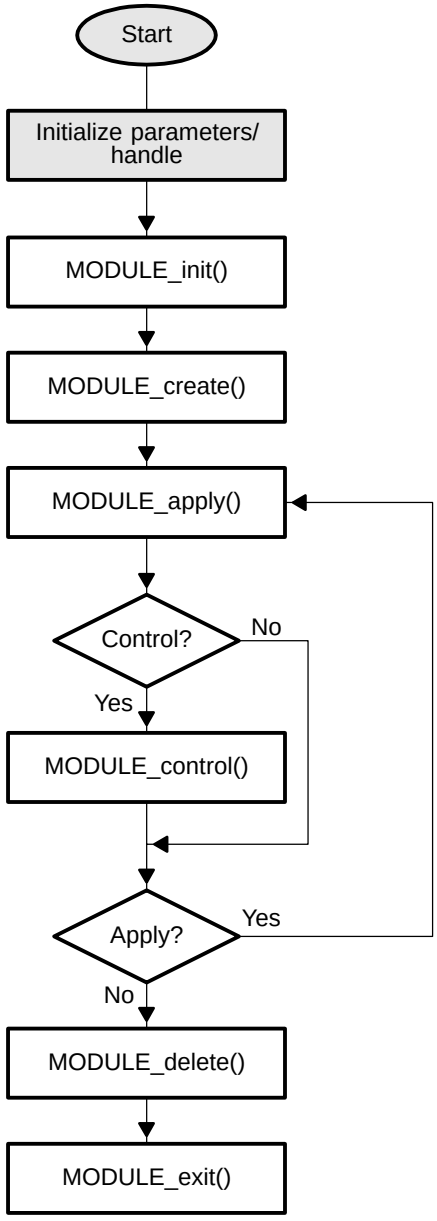
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is known as the function table. The function table allows the framework to invoke any of the algorithm functions through a single handle. The algorithm instance object definition is also done here. This instance object is a structure containing the function table (table of implemented functions) and pointers to instance buffers required by the algorithm.

1.2.3 Application Development

Figure 1-3 illustrates the steps used to develop an application. This flowchart illustrates the creation, use, and deletion of an algorithm. The handle to the instance object (and function table) is obtained through creation of an instance of the algorithm. It is a pointer to the instance object. Per XDAIS guidelines, software API allows direct access to the instance data buffers, but algorithms provided by SPIRIT prohibit access.

Detailed flow charts for each particular algorithm is provided by the vendor.

Figure 1-3. Module Instance Lifetime



The steps below describe the steps illustrated in Figure 1-3.

Step 1: Perform all non-XDAIS initializations and definitions. This may include creation of input and output data buffers by the framework, as well as device driver initialization.

Step 2: Define and initialize required parameters, status structures, and handle declarations.

Step 3: Invoke the `MODULE_init()` function to initialize the algorithm module. This function returns nothing. For most algorithms, this function does nothing.

Step 4: Invoke the `MODULE_create()` function, with the vendor's implementation ID for the algorithm, to create an instance of the algorithm. The `MODULE_create()` function returns a handle to the created instance. You may create as many instances as the framework can support.

Step 5: Invoke the `MODULE_apply()` function to process some data when the framework signals that processing is required. Using this function is not obligatory and vendor can supply the user with his own set of functions to obtain necessary processing.

Step 6: If required, the `MODULE_control()` function may be invoked to read or modify the algorithm status information. This function also is optional. Vendor can provide other methods for status reporting and control.

Step 7: When all processing is done, the `MODULE_delete()` function is invoked to delete the instance from the framework. All instance memory is freed up for the framework here.

Step 8: Invoke the `MODULE_exit()` function to remove the module from the framework. For most algorithms, this function does nothing.

The integration flow of specific algorithms can be quite different from the sequence described above due to several reasons:

- Specific algorithms can work with data frames of various lengths and formats. Applications can require more robust and effective methods for error handling and reporting.
- Instead of using the `MODULE_apply()` function, SPIRIT Corp. algorithms use extended interface for data processing, thereby encapsulating data buffering within XDAIS object. This provides the user with a more reliable method of data exchange.

1.3 Supported Products

The Modem Integrator supports the following SPIRIT products:

- ❏ V.32bis/V.22bis Modem Data Pump
- ❏ V.42 error correction protocol
- ❏ V.42bis data compression
- ❏ Embedded V.14 based asynchronous-to-synchronous conversion

1.4 Compatibility

This implementation is fully compliant with the following ITU-T standards:

- ❏ V.14
- ❏ V.22bis/V.22
- ❏ V.25
- ❏ V.32bis/V.32
- ❏ V.42
- ❏ V.42bis

Chapter 2

Modem Integrator (MODINT) Integration

This chapter provides descriptions, diagrams, and example explaining the integration of the Modem Integrator (MODINT) with frameworks.

Topic **Page**

2.1 Overview

The Modem Integrator (MODINT) combines several independent XDAIS objects into what is essentially called a modem.

The Modem Integrator performs the following operations:

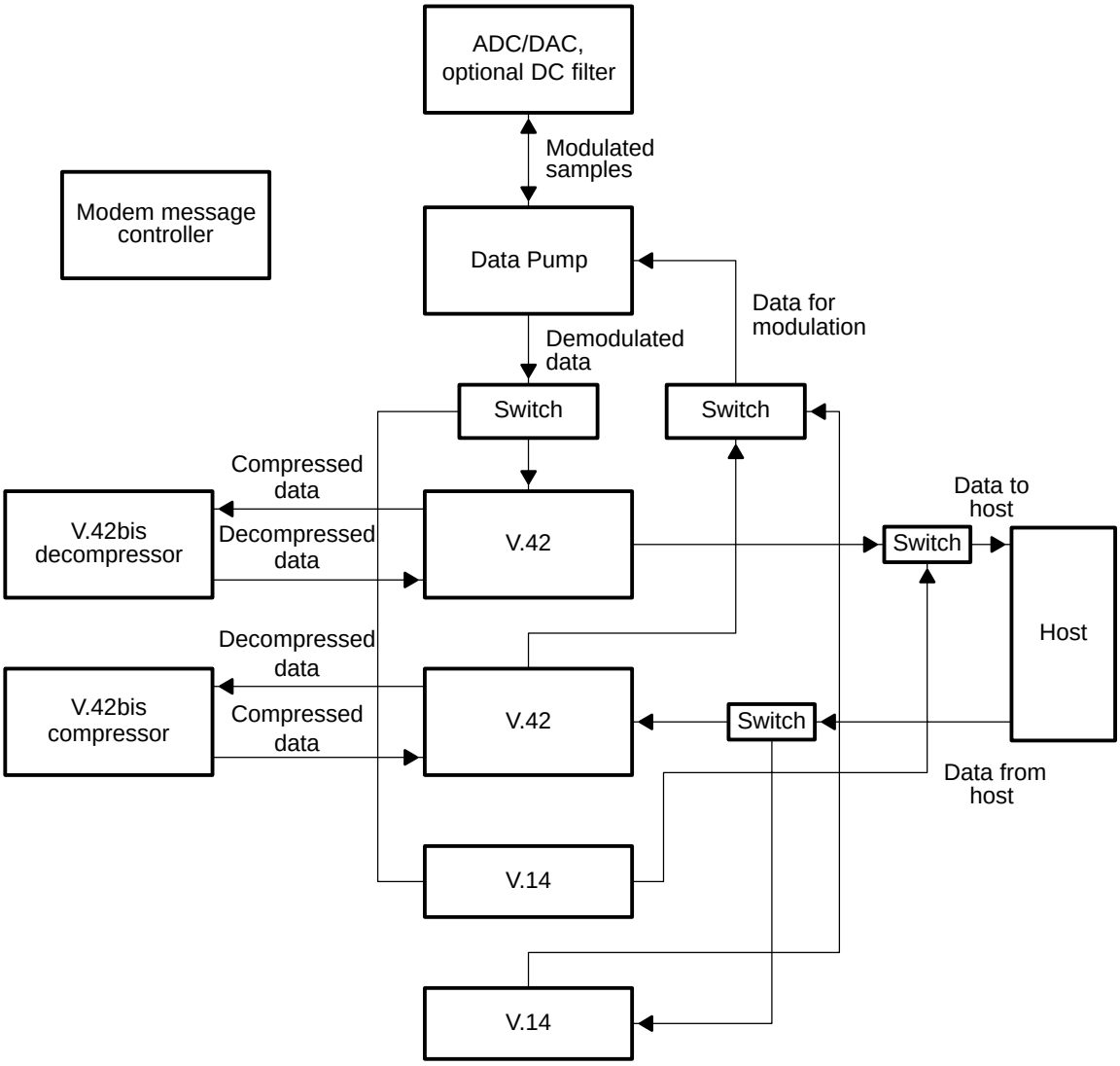
- Implicitly creates all required external objects and performs all required interconnections. Depending on parameters, three configurations are available: MDP + V.14 only, MDP + V.14/V.42 and MDP + V.14/V.42/V.42bis.
- Provides a unified and convenient user interface: Unified parameters, sample and data flows, extended status, etc.
- Modem Data Pump extra control
- V.14 based asynchronous-to-synchronous conversion
- V.14/V.42 switch, connect and disconnect condition report

Additional features:

- V.42bis can be configured both in symmetric (standard) mode and in asymmetric mode (some modems do not support asymmetric V.42bis properly)
- Both single and two-thread support
- Fast connect capability

Figure 2-1 represents typical Modem Integrator API integration.

Figure 2-1. Typical Modem Data Pump Integration



2.2 Integration Flow

In order to integrate Modem Integrator into a framework the user must follow these steps (Figure 2-2):

Step 1: Call `MODINT_SPCORP_init()` to register external algorithm libraries, which are MDP, V42/V42B. The absence (zero address) of V42 automatically disables support of appropriate capabilities.

Step 2: Implement three optional callback methods of the high-level client and place their addresses into the `IMODINT_ClientSubfxns` structure.

Step 3: Create and fill a structure of `IMODINT_Params` type and all referred structures.

Step 4: Call `MODINT_create` to create the modem instance.

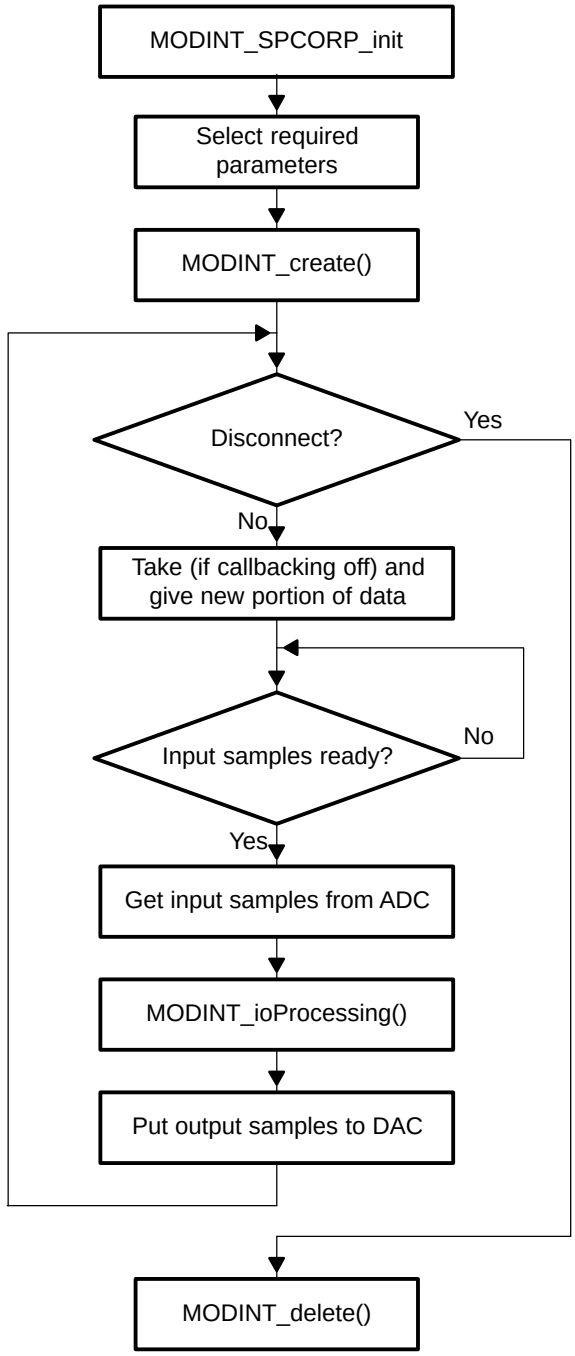
Step 5: While the modem is running, get a number of samples from ADC buffer and pass them (through DC filter or directly) to the modem by using the `MODINT_ioProcessing()` method. This produce the same number of output samples.

Step 6: Put output samples to DAC buffer.

Step 7: Periodically get (if callback mechanism is not used) and send data, control connect condition. These operations can be done using a low priority thread.

All linked objects can be directly controlled via their pointers contained within the Modem Integrator status structure.

Figure 2-2. Modem Integrator Flowchart



2.3 Example of a Call Sequence

The example below demostrates a typical call sequence used with the Modem Integrator API. Full sample code is placed in the file `Src/FlexExamples\StandaloneXDAS\MODINT`.

```
//V.42 related parameters
IMODINT_V42Params V42Params=
{
  1500,                     //heapSize
  15,                       //windowSize
  128,                      //n401
  5000,                     //t401
  3,                        //n400
  1000,                     //t400, 1 sec
  1,                        //UseFCS32
  //V.42bis related parameters
  {
    512,                      //dictionarySize
    32,                       //maxStringLength
    1,                        //isCompressor
    1                         //isDecompressor
  }
};
//Modem Data Pump related parameters
IMODINT_MdpParams MDPParams=
{
  14400,                    //maxSpeed
  0,                        //isFastConnect
  3000,                     //txLevel
  1,                        //isApp
  85,                       //afeDelay
  500,                      //maxRoudTripDelay, 500 millisec
  0,                        //isSpeedupDisabled
  0                         //isSlowdownDisabled
};
//My callback functions: set configuration for two-thread mode
```

```
  IMODINT_ClientSubfxns MySubfxns=
  {
    MyTransferData,             //pTransferData
    MyPreemptionControl,        //pPreemptionControl
    0                           //pIsRealtimeShortage
  };
  IMODINT_Params DMParams=
  {
    sizeof (IMODINT_Params),    /* sizeof the whole parameter struct */
    0,                          //pClient, unnecessary for single channel
applications
    0,                          //instanceID, unnecessary for single channel
applications
    &MySubfxns,                 //pClientSubfxns
    &MDPParams,                 //pMDPParams
    &V42Params,                 //pV42Params
    1,                          //isPreemption, enable two-thread mode
    0                           //isOriginator
  };
void main ()
{
  ...
  //Init the module: register Modem Data Pump and V.42 algorithms
  MODINT_SPCORP_init (&MDP_SPCORP_IMDP,&V42_SPCORP_IV42);
  //Create instance
  pMODINT=MODINT_create(&MODINT_SPCORP_IMODINT,&DMParams);
  if ( !pMODINT )  return;
  ...
}
//can be called from main loop or by DSP-BIOS
void MyHighPeriodicThread()
{
  MODINT_Status Status;
  ...
  MODINT_ioProcessing(pMODINT,pInputSamples,pOutputSamples,Count);
  Status=MODINT_getStatus(pMODINT);
```

```
  if (Status.connectCondition==IMODINT_DISCONNECT) {
    //Disconnect occurred
  }
  ...
}
//to be called from DSP-BIOS
void MyLowPeriodicThread()
{
  //If there is any data, convey it to the modem
  Count=MODINT_injectData(pMODINTHandle,pData,Count);
  pData+=Count;
  ...
  //Call low priority V.42/V.42bis operations
  MODINT_backGround(pMODINT);
}
//Take data from the modem (callback function)
int MyTransferData(void *pMyself,int MyProprietaryID,XDAIS_UInt8 *pData,int Count)
{
  //Copy pData[0.. Count-1]
  ...
  return Count; //All data have been stored
}
//Enable/disable low task interrupt to protect critical V.42/V.42bis operations
void MyPreemptionControl(XDAS_Bool isPermitted)
{
  if (isPermitted)
    SWI_enable();
  else
    SWI_disable();
}
```

Chapter 3

Modem Integrator (MODINT) API Descriptions

This chapter provides the user with a clear understanding of Modem Integrator (MODINT) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).

Topic **Page**

3.1 Standard Interface Structures

The section describes parameter structures for the Modem Integrator.

Table 3-1 lists the type and location of the Standard Interface structures.

Table 3-1. Standard Interface Structures Summary

Parameters	Located in Table...
Modem Integrator Interface Creation	Table 3-2
Modem Data Pump (MDP) Interface	Table 3-3
V.42 Interface Configuration	Table 3-4
V.42bis Interface Configuration	Table 3-5
Status Structure returned by Modem Integrator	Table 3-6
High Level Data Controls	Table 3-7
Particular Connect Conditions	Table 3-8

3.1.1 Instance Creation Parameters

Description This structure defines the creation parameters for the algorithm. A default parameter structure is defined in "`iMODINT.c`".

Structure Definition

Table 3-2. Modem Integrator Interface Creation Parameters

```
typedef struct IMODINT_Params {
```

Params Type	Params Name	Description
`XDAS_Void*`	`pClient`	Pointer to/handle of high algorithm instance usually called as a "client" (see Note 1)
`Int`	`instanceID`	An auxiliary ID in case if pClient value is not sufficient for identification (see Note 1)
`IMODINT_ClientSubfxns*`	`pClientSubfxns`	Optional pointer to a part of client's "Fxns" structure containing pTransferData and pPreemptionControl optional methods (see Table 3-11) (see Note 1)
`IMODINT_MdpParams*`	`pMDPParams`	Effective parameters of Modem Data Pump
`IMODINT_V42Params*`	`pV42Params`	Effective V.42 parameters (including V.42bis) (see Note 2)

Table 3-2. Modem Integrator Interface Creation Parameters (Continued)

Params Type	Params Name	Description
XDAS_Bool	isPreemption	Two-thread mode support enable (see Note 3)
XDAS_Bool	isOriginator	Configure Modem Data Pump and V.42 as an originator (calling modem)

```
} IMODINT_Params;
```

Notes: 1) Modem Integrator supports three optional callback methods needed for two-thread running and more effective data pumping out. The client submits these methods to the modem via ClientSubfxns structure (see Table 3-11). Zero address in pClientSubfxns or in either field means disabling this particular option.

2) Zero address of pV42Params automatically disables V.42/V.42bis. pMDPParams must be non-zero.

3) Two-thread mechanism is enabled only if both isPreemption and pPreemptionControl method are non-zero.

Type IMODINT_Params is defined in "iMODINT.h".

The following data types are used above:

IMODINT_ClientSubfxns - defined section NO TAG.

3.1.1.1 Parameters of Modem Data Pump (MDP) Interface

Table 3-3. Partial Parameters for MDP Algorithm Configuration

```
typedef struct IMODINT_MdpParams {
```

Params Type	Params Name	Description
Int	maxSpeed	Maximal permitted speed (BPS) (default value 14400)
XDAS_Bool	isFastConnect	Disable automodem (skip answer tone) and reduce V.32 training duration to the minimum (default value 0)
XDAS_Int16	txLevel	Output signal amplitude in absolute units
XDAS_Bool	isApp	If true, adaptive phase predictor is enabled(default value 1)
XDAS_Int16	afeDelay	Total number of 8KHz samples in AFE buffers between modem pump and analog I/O.
XDAS_Int16	maxRoudTripDelay	Maximal supported round trip delay in milliseconds.
XDAS_Bool	isSpeedupDisabled	Disable initiation of speedup procedure
XDAS_Bool	isSlowdownDisabled	Disable initiation of recovery procedure

```
} IMODINT_MdpParams;
```

Maximal Operating Speed (`maxSpeed`)

Selects maximal permitted speed for Data Pump. Has an implicit influence on operating modem protocol. If `maxSpeed` is set to 2400 or 1200 only V.22 protocol is enabled.

Enable Fast Connect (`isFastConnect`)

Disable automodem (skip answer tone) and reduce V.32 training duration to the minimum.

Output Signal Level (`txLevel`)

txLevel selects amplitude for tone generation (e.g. answer tone). Output power of QAM signal will be equal to the power of a single tone. Do not set it to a value greater than 12000, otherwise it can overflow 16-bit limit (32767) during QAM signal generation.

Enable Adaptive Phase Predictor (`isApp`)

Adaptive Phase Predictor is a dedicated unit intended for phase jitter compensation. According to impairment parameters for TAS “USA Worst Case”, phase jitter can reach up to 23 deg. If phase jitter is not compensated, its level is high enough to make V.32 14400/12000 reception impossible. Therefore it is recommended to have this option enabled. However, disabling this unit allows saving around 0.5 MIPS, and slightly improves data pump quality on very noisy channels without phase jitter.

Delay Introduced by AFE (`afeDelay`)

Total number of 8KHz samples in AFE buffers between modem pump and analog I/O (needed for correct positioning of Near Echo Canceller and correct measurement of round trip delay). In other words, a total delay of near echo, measured in 8KHz samples.

Maximal Round Trip Delay (`maxRoudTripDelay`)

Maximal round trip delay in milliseconds supported by V.32 echo canceller (influences the amount of memory reserved for Far Echo bulk delay).

Disable Initiation of Speedup Procedure (`isSpeedupDisabled`)

Disables self-initiated rate renegotiation up. That means, the Modem Integrator will never try to speed modem up, regardless stable excellent line condition.

Disable Initiation of Recovery Procedure (`isSlowdownDisabled`)

Disables self-initiated rate renegotiation down or retrain. That means, the Modem Integrator will never try to speed modem down or retrain, regardless stable intolerable line condition.

For more on Modem Data Pump, see *Spirit Corp. Modem Data Pump User's Guide*.

3.1.1.2 Parameters of V.42/V.42bis Interface

Table 3-4. Partial Parameters for V.42 Algorithm Configuration

```
typedef struct IMODINT_V42Params {
```

Params Type	Params Name	Description
XDAS_UInt16	heapSize	Typical value is 1500. It cannot be less than 1000 words
XDAS_Int16	windowSize	Window size (default value 15)
XDAS_Int16	n401	Maximum number of octets in the information field (maximum value 128)
XDAIS_Uint32	t401	Acknowledgement timer (in msec) (default value 5000)
XDAS_Int16	n400	Maximum number of retransmissions (default value 3)
XDAIS_Int32	t400	V.42 handshake timer, in msec. (default value 750)
XDAS_Bool	UseFCS32	Allow to use 32-bit FCS (default value true)
IMODINT_V42bisParams	V42bisParams	V.42bis configuration

```
} IMODINT_V42Params;
```

Table 3-5. Partial Parameters for V.42bis Algorithm Configuration

```
typedef struct IMODINT_V42BisParams {
```

Params Type	Params Name	Description
Int	dictionarySize	Desired dictionary size (512,1024,2048)
Int	maxStringLength	Maximal string length (6…32)
XDAS_Bool	isCompressor	Enable compression
XDAS_Bool	isDecompressor	Enable decompression

```
} IMODINT_V42BisParams;
```

Heap Size (heapSize)

Heap size is chosen according to the system needs. Longer heap provides more robust performance for outgoing traffic (lower delays and better outgoing throughput). Upper boundary for heap size can be estimated as:

```
heapsize = windowSize*(12+n401/2) + 550
```

Heap size must be not lesser than 1000.

Window Size (windowSize)

This parameters governs the maximum number of information frames that a V.42 instance can have unacknowledged. To provide compatibility with ITU-T standard, user should set this parameter to 15 or greater.

Window size is set during negotiation/indication procedure and can not be less than 15. However, real window size can be less than declared value, and be defined dynamically according to the actual heap size.

Maximum Number of Octets in an Information Field (n401)

Parameter `n401` governs the maximum number of octets that can be carried in the information field of an information frame transmitted by V.42 instance. The typical value of `n401` is 128.

In very noisy environment (e.g. when datapump provides BER>10-4) this parameter can be reduced, but it is not recommended to set it below 30.

Acknowledgement Timer (t401)

The acknowledgement timer governs the amount of time that V.42 instance will wait for an acknowledgement for previously initiated operation. Two V.42 instances may operate with different values of T401 timer. Typically, modems use values in range 0.5 … 5 seconds.

In nonstandard implementations user must take into account following factors that affect the optimal value of the timer:

- ❏ overall propagation delay between modems – (T_a)
- ❏ maximum time allowed to complete transmission of frames in the remote V.42's "transmit queue" – (T_b)
- ❏ time needed for the V.42 instances to process the received frame – (T_c)
- ❏ time needed to transmit the acknowledging frame – (T_d)

Correspondingly, value of the acknowledgement timer used by the V.42 instance should be set as follows:

$t401 \geq 2VT_a + T_b + 2VT_c + T_d$

Maximum Number of Retransmissions (n400)

Parameter mN400 governs the maximum number of times that a V.42 instance will re-attempt a procedure requiring a response. The two V.42 instances may operate with a different value of mN400.

Typically, user can select this parameter from the range 1…3.

Handshake Timer (t400)

The handshake timer governs the amount of time that V.42 instance will continue protocol detection (physical handshake) phase (see Table 3-4). Recommended value is 750 msec, but, typically, modems use values in range 0.75 … 5 seconds.

Basically, longer timer is needed when data pumps are connected via protocol, which does not define strictly the end of data-pump handshake, for example V.22.

Using of 32-Bit FCS (mUseFCS32)

V.42 can optionally use 32-bit frame check sequences (FCS) instead of 16-bit for better data transfer reliability. Since FCS32 is optional, it will be used only if both modems supports this capability. Default value for mUseFCS32 is 1; resetting it to 0 disables indication of this capability during negotiation/indication stage.

V.42bis Dictionary Size (dictionarySize)

The size of V.42bis dictionary affects memory requirements and compression rate. Using large dictionaries allows better compression. Extending the dictionary size from 512 to 1024 entries increases average compression rate approximately for 15%; extending from 1024 to 2048 entries additionally grants about 5%. The value of mV42bis_P1 shall only be 512, 1024 or 2048; any other values are invalid.

V.42bis Maximum String Length (maxStringLength)

Maximum string length has an influence on maximum compression rate. Using large string lengths allows to achieve better compression rates on very regular blocks of data but practically has no effect on more or less heterogeneous data. Allowed values for mV42bis_P2 are 6..32. Using of default value (32) is recommended.

Enable V.42bis Compressor (`isCompressor`)

This parameter sets capabilities of V.42bis compression algorithm. During negotiation/indication stage, V.42bis capabilities will be indicated according to this parameter. The value of this parameter is also affects the amount of required memory.

Enable V.42bis Decompressor (`isDecompressor`)

This parameter sets capabilities of V.42bis decompression algorithm. During negotiation/indication stage, V.42bis capabilities will be indicated according to this parameter. The value of this parameter is also affects the amount of required memory.

For more on V.42/V.42bis algorithm see *Spirit Corp. V.42/V.42bis User's Guide*.

3.1.2 Status Structure

Description This structure defines the status parameters for the algorithm. Status structure is used for control purposes. The status can be received via function `MODINT_getStatus()` at any time during the instance lifetime.

Structure Definition

Table 3-6. Status Structure Being Returned by Modem Integrator

```
typedef struct IMODINT_Status {
```

Status Type	Status Name	Description
`IMODINT_High-LevelProcotol`	`highLevelProcotol`	High level data control (see Table 3-7)
`IMODINT_ModemConnectCondition`	`connectCondition`	Set of particular connect conditions (see Table 3-8)
`XDAS_UInt32`	`connectCondDuration`	Duration of the current connect condition in 8kHz ticks
`IMDP_Status`	`mdpStatus`	Modem Data Pump status (see S*pirit Corp. Modem Data Pump User's Guide*)
`IV42_Status`	`v42Status`	V42 status (see *Spirit Corp. V.42/V.42bis User's Guide*)
`IMDP_Handle`	`pMDP`	Pointer to MDP instance
`IV42_Handle`	`pV42`	Pointer to V42 instance if created
`IV42B_Handle`	`pV42BisCompressor`	Pointer to V42B compressor if created
`IV42B_Handle`	`pV42BisDecompressor`	Pointer to V42B decompressor if created

```
} IMODINT_Status;
```

Type IMODINT_Status defined in "iMODINT.h".

The following types are used above:

Table 3-7. Set of High-Level Data Controls

```
typedef enum IMODINT_HighLevelProcotol {
```

Value	Name	Description
0	IMODINT_NA	Modem has not been connected
1	IMODINT_V14	Standard connect without error correction
2	IMODINT_V42	Pure V.42 connect established
3	IMODINT_V42BIS	V.42bis connect established. It also can be asymmetric V.42bis

```
} IMODINT_HighLevelProcotol;
```

Table 3-8. Set of Particular Connect Conditions

```
typedef enum IMODINT_ModemConnectCondition {
```

Value	Name	Description
0	IMODINT_WAITING	Idle or automodem state
1	IMODINT_TRAINING	Modem Data Pump is switched to certain signal protocol training (also retraining etc)
2	IMODINT_DATA	Modem Data Pump has finished all training stages. Data is going through
4	IMODINT_DISCONNECT	Modem is disconnected now

```
} IMODINT_ModemConnectCondition;
```

3.2 Standard Interface Functions

Table 3-9 summarizes the standard interface functions of the Modem Integrator API. Only one override is available: MODINT_SPCORP_init.

`MODINT_apply()` and `MODINT_control()` are optional, but neither are supported by Spirit Corp.

Table 3-9. Modem Integrator Standard Interface Functions

Functions	Description	See Page...
`MODINT_init`	Algorithm initialization. Overrridden by MODINT_SPCORP_init	3-11
`MODINT_exit`	Algorithm deletion	3-12
`MODINT_create`	Instance creation	3-12
`MODINT_delete`	Instance deletion	3-13

3.2.1 Algorithm Initialization

MODINT_SPCORP_init *Initializes the MODINT algorithm through table registration*

Description Instead of generic `MODINT_init`, function `MODINT_SPCORP_init` shall be called. This function initializes the `MODINT` algorithm through external function table registration.

Function Prototype

```
MODINT_SPCORP_init
(IMDP_Fxns *pIMDPFxns,
 IV42_Fxns *pIV42Fxns);
```

Arguments

`pIMDPFxns` pointer to MDP function table (see *Spirit Corp. Modem Data Pump User's Guide*)

`pIV42Fxns` pointer to V42 function table (see *Spirit Corp. V.42/V.42bis User's Guide*)

Return Value none

3.2.2 Algorithm Deletion

MODINT_exit *Calls the framework exit function to remove the MODINT algorithm*

Description This function calls the framework exit function, `ALG_exit()`, to remove the algorithm `MODINT`. In respect of Modem Integrator this function does nothing. It can be skipped and removed from the target code according to *Achieving Zero Overhead With the TMS320 DSP Algorithm Standard IALG Interface* (SPRA716).

Function Prototype

```
void MODINT_exit()
```

Arguments none

Return Value none

3.2.3 Instance Creation

MODINT_create *Calls the framework create function to create a new object instance*

Description In order to create a new MODINT object, `MODINT_create` function should be called. This function calls the framework create function, `ALG_create()`, to create new object instance and perform memory allocation task. Global structure `MODINT_SPCORP_IMODINT` contains MODINT virtual table supplied by SPIRIT Corp.

Function Prototype

```
MODINT_Handle MODINT_create
(const IMODINT_Fxns *fxns,
 const MODINT_Params *prms);
```

Arguments

`fxns` Pointer to vendor's functions (Implementation ID). Use reference to `MODINT_SPCORP_IMODINT` virtual table supplied by SPIRIT Corp.

`prms` Pointer to Parameter Structure (see Table 3-2).

Return Value

`MODINT_Handle` Defined in file "`MODINT.h`". This is a pointer to the created instance object.

3.2.4 Instance Deletion

MODINT_delete *Calls the framework delete function to delete an instance object*

Description This function calls the framework delete function, `ALG_delete()`, to delete the instance object and perform memory de-allocation task.

Function Prototype `void MODINT_delete (MODINT_Handle handle)`

Arguments `MODINT_Handle` Instance's handle obtained from `MODINT_create()`

Return Value none

3.3 Vendor-Specific Interface

This section describes the vendor-specific functions in the SPIRIT's algorithm implementation and interface (extended IALG methods).

Table 3-10 summarizes SPIRIT's API functions of Modem Integrator.

The whole interface is placed in header files `iMODINT.h`, `MODINT.h`, `MODINT_spcorp.h`.

Table 3-10. Modem Integrator-Specific Interface Functions

Functions	Description	See Page...
`MODINT_ioProcessing`	Processes input samples and generates output samples.	3-15
`MODINT_backGround`	In two-thread mode, performs low priority operations and can be interrupted by `MODINT_ioProcessing` (also see `pPreemptionControl` method in Table 3-11).	3-15
`MODINT_disconnect`	Safe disconnect request.	3-16
`MODINT_returnData`	Returns a number of received data bytes.	3-16
`MODINT_injectData`	Takes a number of data bytes to be transmitted.	3-17
`MODINT_getStatus`	Returns the Modem Integrator status that reports the current connection state and contains information about integrated algorithms.	3-17

3.3.1 I/O Processing

MODINT_io-Processing *Processes count input samples and generates count output samples*

Description Processes `count` input samples and generates `count` output samples. In single thread mode, also performs low priority control operations.

Function Prototype

```
XDAS_Void MODINT_ioProcessing
(MODINT_Handle handle,
 XDAS_Int16 in[],
 XDAS_Int16 out[],
 Int count)
```

Arguments

`handle`	Pointer to Modem Integrator object
`in`	Array of input samples (sampling rate 8kHz)
`out`	Array of output samples (sampling rate 8kHz)
`count`	Number of samples to be processed

Return Value none

Note: Maximum Echo Delay

Maximal V.32 local echo delay is approx. ten milliseconds. Thus, I/O buffers should not exceed this limitation.

3.3.2 Low-Level Control Operation

MODINT_back-Ground *Performs low priority operations in two-thread mode*

Description In two-thread mode, performs low priority operations and can be interrupted by `MODINT_ioProcessing()` (also see `pPreemptionControl` method in Table 3-11). In single thread mode, it does nothing.

Function Prototype

```
XDAS_Void MODINT_backGround (MODINT_Handle handle);
```

Arguments `MODINT_Handle` Instance's handle obtained from `MODINT_create()`

Return Value none

3.3.3 Safe Disconnect

MODINT_disconnect *Safe disconnect request*

Description The purpose of this operation is an attempt to release V.42 connection correctly. Modem integrator indicates real disconnect in its status flags (see Table 3-6 and Table 3-8).

Function Prototype

```
XDAS_Void MODINT_disconnect
(MODINT_Handle handle,
 XDAS_UInt16 HDLCTimer);
```

Arguments

`handle`	Pointer to Modem Integrator object
`HDLCTimer`	Time in milliseconds to complete V.42 release procedure. A typical value is 1000.

Return Value none

3.3.4 Modem Integrator Data Flows

MODINT_returnData *Returns a number of received data bytes*

Description Modem Integrator supports single mechanism to pump data in (`MODINT_injectData` method) and two mechanisms to pump out (either `MODINT_returnData` method or callback). If an optional callback `pTransferData` method is not set (see Table 3-11), the pump out is performed in the same manner as the pump in, i.e. via standard method which is described below.

Checking the number of actually transferred bytes provides the flow control in both directions. If `MODINT_injectData(), MODINT_returnData()` or appropriate callback handler returns 0, no data is transferred. Host must control returned value and store the remaining data for further use.

Function Prototype

```
Int MODINT_returnData
(MODINT_Handle handle,
 XDAS_UInt8 *pBuffer
 Int maxCount);
```

Arguments

`handle`	Pointer to Modem Integrator object
`pBuffer`	Receive buffer
`maxCount`	Maximum length of receive buffer

Return Value Number of written bytes. Can not exceed `maxCount`.

MODINT_injectData *Takes a number of data bytes to be injected*

Function Prototype

```
Int MODINT_injectData
(MODINT_Handle handle,
 XDAS_UInt8 *pBuffer,
 Int count);
```

Arguments

`handle`	Pointer to Modem Integrator object
`pBuffer`	Transmit buffer
`count`	Number of bytes in transmit buffer

Return Value

Number of bytes that was actually taken by Modem Integrator. When this value is less than count, the buffer is not transferred completely and host has to store the rest for further transactions.

3.3.5 Get Modem Integrator Status

MODINT_getStatus *Returns the Modem Integrator status*

Description

Returns Modem Integrator status (see `IMODINT_Status` Table 3-6) that informs about current connection state and summarizes information about integrated algorithms.

Function Prototype

```
IMODINT_Status MODINT_getStatus
(MODINT_Handle handle)
```

Arguments

`MODINT_Handle` Instance's handle obtained from `MODINT_create()`

Return Value

Current Modem Integrator status.

3.4 High-Level Client Interface

Modem Integrator supports two optional methods needed for two-thread running and more effective data pumping out. It also supports one optional method used in single-threaded environment.

Table 3–11. Client Methods Given to the Modem Integrator

```
typedef struct IMODINT_ClientSubfxns {
```

Params Type	Params Name	Description
`Int (*) (XDAS_Void *pClient, Int instanceID, XDAS_UInt8* pBuffer, Int count)`	`pTransferData`	By calling this method, Modem Integrator transfers to the client a new portion of demodulated data bytes.
`Int (*) (XDAS_Void *pClient, Int instanceID, XDAS_Bool isPermitted)`	`pPreemptionControl`	By calling this method, Modem Integrator sends a request to lock or unlock task switching on the fly.
`XDAS_Bool (*) (XDAS_Void *pClient, Int instanceID)`	`pIsRealtimeShortage`	By calling this method, Modem Integrator (V.42bis) asks to let it continue. This query is issued in single thread mode only, and only by V.42bis unit. True result suspends V.42bis processing to be continued at next time.

```
} IMODINT_ClientSubfxns;
```

Table 3–12. Parameter Description for Client Methods

Params Type	Params Name	Description
`XDAS_Void *`	`pClient`	Pointer to the client given for Modem Integrator at initialization as `pClient` parameter (see parameter `pClient` in `IMODINT_Params`, Table 3–2).
`Int`	`instanceID`	An auxiliary ID for case when `pClient` value is not sufficient for the identification (see parameter `instanceID` in `IMODINT_Params`, Table 3–2).
`XDAS_UInt8*`	`pBuffer`	Pointer to the source memory with data bytes. The pointed byte is first in time.
`Int`	`count`	Number of data bytes.
`XDAS_Bool`	`isPermitted`	If `isPermitted` value is zero, `MODINT_backGround` method cannot be interrupted by `MODINT_ioProcessing`.

3.4.1 Client Methods

Transfer Data

Description When implemented, Modem Integrator calls this method to transfer data (incoming traffic) to the client. This provides more effective processor usage than equivalent polling method `IMDP_returnData()`.

Function Prototype

```
Int (*pTransferData)
(XDAS_Void*,Int instanceID,XDAS_UInt8* pBuffer,Int
count)
```

Parameters

`pClient`	Internal instance pointer. Data pump sets it with `IMODINT_Params::pClient` value.
`instanceID`	Optional instance sub-item. Data pump sets it with `IMODINT_Params::instanceID` value.
`pBuffer`	Internal receive buffer.
`count`	Number of data bytes to be transferred.

Return Value Client shall return the number of taken bytes.

Preemption Allowing

Description Modem Integrator calls this method in two-thread mode to protect preemption sensitive operation. This message is invoked during low-priority processing or other V.42 functions that require data locking. Host must lock both Modem Integrator threads (disable task preemption) when this message with parameter `isPermitted` is equal to 0, and restore thread state when `isPermitted` is equal to `1`. When host uses DSP/BIOS, functions `SWI_disable()/SWI_enable()` can be used for this purpose. Otherwise, when host does not use DSP/BIOS and provides custom multitasking, disabling/enabling interrupts can be used also. It is guaranteed that these messages are always issued in pairs.

In single-threaded applications, framework can ignore this message.

Function Prototype

```
Int (*pPreemptionControl)
(XDAS_Void* pClient,Int instanceID,XDAS_Bool
isPermitted)
```

Parameters

`pClient`	Internal instance pointer. Data pump sets it with `IMODINT_Params::pClient` value.
`instanceID`	Optional instance sub-item. Data pump sets it with `IMODINT_Params::instanceID` value.
`isPermitted`	`==0`, when thread preemption should to be disabled and, correspondingly, `MODINT_backGround` method shall not be interrupted by `MODINT_ioProcessing` `==1`, restore thread status (is always issued after `isPermitted == 0`)

Return Value Ignored (obsolete)

Real-Time Control for Single-Threaded Environment

Description Modem Integrator calls this function in one-threaded mode to prevent real-time shortage. It calls the function before starting V.42bis time-consuming operations. True result suspends V.42bis processing to be continued at next time. User can make a decision based on e.g., how far is DAA buffer from collapse.

Function Prototype

```
Int (*pIsRealtimeShortage)
(XDAS_Void* pClient,Int instanceID);
```

Parameters

`pClient`	Internal instance pointer. Data pump sets it with `IMODINT_Params::pClient` value.
`instanceID`	Optional instance sub-item. Data pump sets it with `IMODINT_Params::instanceID` value.

Return Value Non-zero value corresponds to real-time shortage alarm.

Appendix A

Test Environment

Note: Test Environment Location

This chapter describes test environment for the MI object.

For TMS320C54CST device, test environment for standalone MI object is located in the Software Development Kit (SDK) in `Src\FlexExamples\StandaloneXDAS\MODINT`.

Topic **Page**

A.1 Description of Directory Tree

The SDK package includes the test project “test.pjt” and corresponding reference test vectors. The user is free to modify this code as needed, without submissions to SPIRIT Corp.

Table A-1. Test Files for MI

File	Description
main.c	Test file
FileC5x.c	File input/output functions
..\ROM\CSTRom.s54	ROM entry address
Test.cmd	Linker command file
Vectors\output.pcm	Reference output test vectors

A.1.1 Test Project

To build and run a project, the following steps must be performed:

Step 1: Open the project: `Project\Open`

Step 2: Build all necessary files: `Project\Rebuild All`

Step 3: Initialize the DSP: `Debug\Reset CPU`

Step 4: Load the output-file: `File\Load program`

Step 5: Run the executable: `Debug\Run`

Once the program finishes testing, the file *Output.pcm* will be written in the current directory. Compare this file with the reference vector contained in the directory *Vectors*.

Note: Test Duration

Since the standard file I/O for EVM is very slow, testing may take several minutes. Test duration does not indicate the real algorithm’s throughput.

Index

I

L

M

N

O

P

R

S

T

U

V

W

X