

Universal Multifrequency Tone Detector (UMTD) Algorithm User's Guide

SPIRIT CORP

DSP Software Source

www.spiritDSP.com/CST



Literature Number: SPRU638

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

Read This First

About This Manual

The following abbreviations are used in this document:

| | |
|-------|---|
| CPTD | Call Progress Tone (Detection) |
| CAS | Customer Alerting Signal |
| CMS | Composite Multitone Signal |
| DT-AS | Dual-Tone Alerting Signal |
| DTMF | Dual Tone Multifrequency (signaling) |
| MF | multifrequency (signaling) |
| UMTD | Universal Multifrequency Tone Detection |
| XDAIS | 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 E.180/Q.35. Tones in national signaling systems – Operation, numbering, routing and mobile services, 1998.

ITU-T Recommendation E.180, Supplement 2. Various tones used in national networks – Telephone network and ISDN. Operation, numbering, routing and mobile service, 1994.

ITU-T Recommendation Q.23. Technical features of push-button telephone sets – International automatic and semi-automatic working, 1993.

ITU-T Recommendation Q.24. Multifrequency push-button signal reception – International automatic and semi-automatic working, 1993.

ITU-T Recommendation Q.320, Signal code for register signalling – Specifications of signalling system R1, 1993.

ITU-T Recommendation Q.322, Multifrequency signal sender – Specifications of signalling system r1, 1993.

ITU-T Recommendation Q.323, Multifrequency signal receiving equipment – Specifications of signalling system R1, 1993.

ITU-T Recommendation Q.441, Signalling code – Specifications of signalling system R2, 1993.

ITU-T Recommendation V.8. Procedures for starting sessions of data transmission over the public switched telephone network – General, 1998.

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 – Interfaces and voiceband modems, 1996.

Public Switched Telephone Network (PSTN); Protocol over the local loop for display and related services; Terminal Equipment requirements; Part 1: Off-line data transmission, ETS 300 778-1, September 1997, DE/ATA-005062-1

Public Switched Telephone Network (PSTN); Protocol over the local loop for display and related services; Terminal Equipment requirements; Part 2: On-line data transmission, ETS 300 778-2, September 1997, DE/ATA-005062-2

Specification of Dual Tone Multi-Frequency (DTMF); Transmitters and Receivers; Part 3: Receivers, ETSI TS 101 235-3 v1.1.1 (2000-05)

Calling Line Identification Service, British Telecommunication plc, SIN227, Issue 03.

CCITT Recommendation V.23 (1988): "600/1 200-baud modem standardized for use in the general switched telephone network".

EIA/TIA-464-A. Private Branch Exchange (PBX) Switching Equipment for Voiceband Application. ANSI/EIA/TIA, February, 1989.

Trademarks

TMS320™ is a trademark of Texas Instruments.

SPIRIT CORP™ is a trademark of Spirit Corp.

All other trademarks are the property of their respective owners.

Software Copyright

CST Software Copyright © 2003, SPIRIT Technologies, Inc.

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

| | | |
|----------|---|------------|
| 1 | Introduction to Universal Multifrequency Tone Detector (UMTD) Algorithms | 1-1 |
| | <i>This chapter is a brief explanation of the Universal Multifrequency Tone Detector (UMTD) and its use with the TMS320C5400 platform.</i> | |
| 1.1 | Introduction | 1-2 |
| 1.2 | XDAIS Basics | 1-4 |
| 1.2.1 | Application/Framework | 1-4 |
| 1.2.2 | Interface | 1-5 |
| 1.2.3 | Application Development | 1-6 |
| 2 | Universal Multifrequency Tone Detector (UMTD) Integration | 2-1 |
| | <i>This chapter provides descriptions, diagrams, and examples explaining the integration of the Universal Multifrequency Tone Detector (UMTD) with frameworks.</i> | |
| 2.1 | Overview | 2-2 |
| 2.2 | Integration Flow | 2-4 |
| 2.3 | Signal Recognition | 2-6 |
| 2.3.1 | General | 2-6 |
| 2.3.2 | Frequency Selection | 2-7 |
| 2.3.3 | Amplitude Discrimination | 2-8 |
| 2.3.4 | Time Selection | 2-9 |
| 2.3.5 | Flow Control Options | 2-12 |
| 2.3.6 | Host Indication Options | 2-13 |
| 2.4 | Parameter Definition | 2-15 |
| 2.4.1 | Amplitude Discriminator Options | 2-20 |
| 2.4.2 | Time Slot Flags | 2-23 |
| 2.5 | Messages and Host Interface | 2-25 |
| 2.6 | Examples | 2-27 |
| 2.6.1 | Typical CPTD Settings for USA's PSTN | 2-27 |
| 2.6.2 | Typical DTMF Settings | 2-29 |
| 2.6.3 | CPTD Settings for United Arab Emirates | 2-34 |
| 3 | Universal Multifrequency Tone Detector (UMTD) API Descriptions | 3-1 |
| | <i>This chapter provides the user with a clear understanding of Universal Multifrequency Tone Detector (UMTD) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).</i> | |
| 3.1 | Standard Interface Structures | 3-2 |

| | | |
|----------|---|------------|
| 3.1.1 | Instance Creation Parameters | 3-2 |
| 3.1.2 | Status Structure | 3-2 |
| 3.2 | Standard Interface Functions | 3-3 |
| 3.2.1 | Algorithm Initialization | 3-3 |
| 3.2.2 | Algorithm Deletion | 3-3 |
| 3.2.3 | Instance Creation | 3-4 |
| 3.2.4 | Instance Deletion | 3-4 |
| 3.3 | Vendor-Specific Interface Functions | 3-5 |
| 3.3.1 | Process Detection | 3-5 |
| 3.3.2 | Reset Detector Status | 3-6 |
| 3.3.3 | Get Actual Detector Status | 3-6 |
| A | Test Environment | A-1 |
| A.1 | Description of Directory Tree | A-2 |
| A.1.1 | Test Project | A-3 |

Figures

| | | |
|-----|---|------|
| 1-1 | XDAIS System Layers | 1-4 |
| 1-2 | XDAIS Layers Interaction Diagram | 1-5 |
| 1-3 | Module Instance Lifetime | 1-7 |
| 2-1 | UMTD Integration Diagram | 2-2 |
| 2-2 | Special Information Tone Used in Most of the Countries | 2-3 |
| 2-3 | Typical Detector Integration Flow | 2-5 |
| 2-4 | Simplified UMTD Program Flow | 2-6 |
| 2-5 | Normalized Filter Frequency Response | 2-7 |
| 2-6 | Host Indication Example | 2-14 |
| 2-7 | UMTD Thresholds | 2-17 |
| 2-8 | Bit Field Position illustration | 2-20 |
| 2-9 | Messages Sequence for DTMF Detection (With Early Detection) | 2-33 |

Tables

| | | |
|------|--|------|
| 2-1 | Amplitude Discrimination Options Summary | 2-8 |
| 2-2 | Time Selection Options Summary | 2-10 |
| 2-3 | Flow Control Options Summary | 2-12 |
| 2-4 | Host Indication Options Summary | 2-13 |
| 2-5 | General Parameters for Recognized Signal Series | 2-15 |
| 2-6 | Parameters for Recognized Signal Series | 2-17 |
| 2-7 | Pointers to Series Parameters | 2-18 |
| 2-8 | Common UMTD Parameters | 2-18 |
| 2-9 | CMS Recognition Parameters | 2-19 |
| 2-10 | UMTD Time Slots | 2-19 |
| 2-11 | Bit Field Positions in the Recognition Flags | 2-20 |
| 2-12 | DTMF Signal Mask for IUUMTD_TSUSEDTMFMASK Option | 2-21 |
| 2-13 | Amplitude Discrimination Options | 2-22 |
| 2-14 | Time Selection Options | 2-23 |
| 2-15 | Flow Control Options | 2-24 |
| 2-16 | Host Indication Options | 2-24 |
| 2-17 | UMTD Detector Messages Summary | 2-25 |
| 2-18 | Message Structure | 2-26 |
| 2-19 | Host Controller | 2-26 |
| 3-1 | UMTD Detector Real-Time Status Parameters | 3-2 |
| 3-2 | UMTD Detector Standard Interface Functions | 3-3 |
| 3-3 | Detector-Specific Interface Functions | 3-5 |
| A-1 | Test Files for UMTD | A-2 |

Notes, Cautions, and Warnings

| | |
|---------------------------------|-----|
| Test Environment Location | A-1 |
| Test Duration | A-3 |

Introduction to Universal Multifrequency Tone Detector (UMTD) Algorithms

This chapter is a brief explanation of the Universal Multifrequency Tone Detector (UMTD) and its use 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 | 1-2 |
| 1.2 XDAIS Basics | 1-4 |

1.1 Introduction

This document describes the implementation of the Universal Multifrequency Tone Detector (UMTD) algorithms developed by SPIRIT Corp. for the TMS320C54xx platform and is intended for integration into embedded devices for the decoding of various telephone service tones including:

- Standard CPTD tones:
 - Busy
 - Dial
 - Ringback
 - Reorder
- Extended set of CPTD tones for majority of countries:
 - Recall dial tone
 - Special ringback tone
 - Intercept tone
 - Call waiting tone
 - Busy verification tone
 - Executive override tone
 - Confirmation tone
- DTMF signaling
- MF-R1, MF-R2 signalling
- Caller ID CAS tone for various standards
- Modem specific tones:
 - Bell 103 answer tone
 - V.23 forward/backward mark bit
 - CED
 - CNG
 - ANS
 - ANSam, etc.

UMTD can be used as a simple spectrum analyser for custom applications since it provides low MIPS consumption (approx. 0.06 MIPS per tone) and good dynamic range (at least 60 dB).

The detector can be configured easily to the most country specific CPTD standards.

The SPIRIT UMTD software is a fully TMS320 DSP Algorithm Standard (XDAIS) compatible, reentrant code. The UMTD 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.

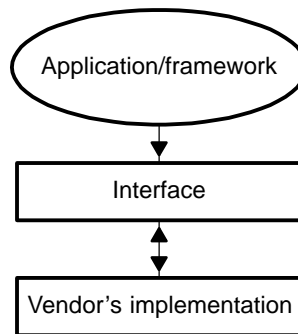
1.2 XDAIS Basics

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:

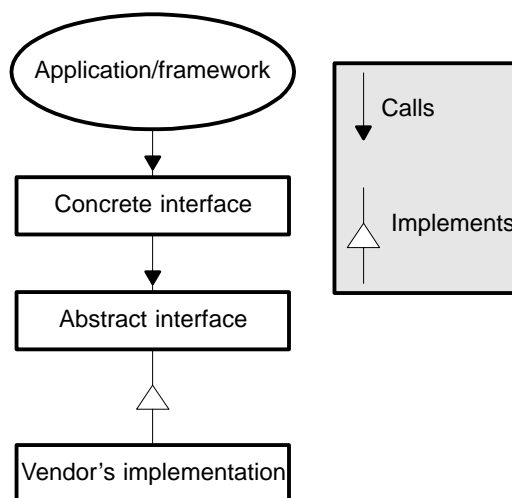
- ALG_create - for memory allocation/algorithm instance creation
- ALG_delete - for memory de-allocation/algorithm instance deletion
- ALG_activate - for algorithm instance activation

- ❑ 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*

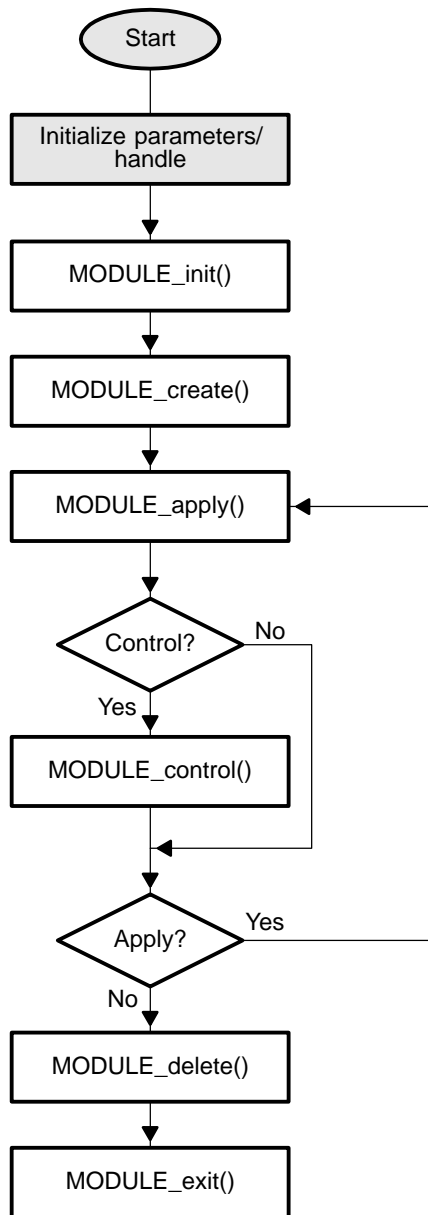
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is 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.

Universal Multifrequency Tone Detector (UMTD) Integration

This chapter provides descriptions, diagrams, and example explaining the integration of the Universal Multifrequency Tone Detector (UMTD) with frameworks.

| Topic | Page |
|--|-------------|
| 2.1 Overview | 2-2 |
| 2.2 Integration Flow | 2-4 |
| 2.3 Signal Recognition | 2-6 |
| 2.4 Parameter Definition | 2-15 |
| 2.5 Messages and Host Interface | 2-25 |
| 2.6 Examples | 2-27 |

2.1 Overview

Universal Multifrequency Tone Detector (UMTD) is designed to recognize the set of Composite Multitone Signals (CMS or composite signals later). A composite signal represents a sequence of partial multitone signals that can be either successive or divided by pauses. Each partial multitone signal can be a weighted sum of spectral components in a limited bandwidth. The sequence can be either recurrent or standalone.

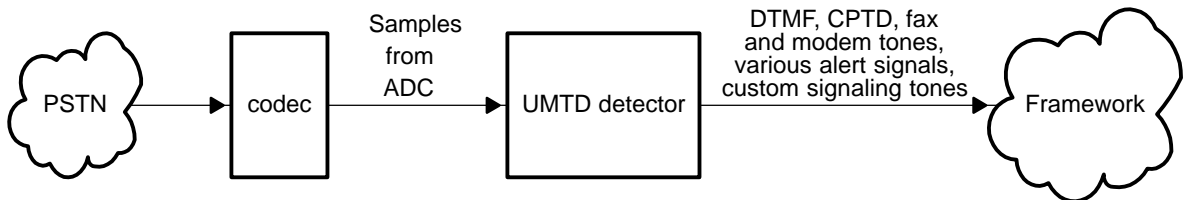
The UMTD detector performs the following operations:

- provides frequency selection by filtering input samples and estimates spectrum of input signal at a given set of frequencies;
- makes user-defined amplitude discrimination defining whether or not a partial multitone signal is present currently;
- makes user-defined time selection (e.g. verifies the cadences and pauses) to make the decision about presence of a composite signaling tone.

A large selection set of options allows the user to control the recognition on all layers mentioned above.

Figure 2-1 illustrates a typical UMTD integration diagram.

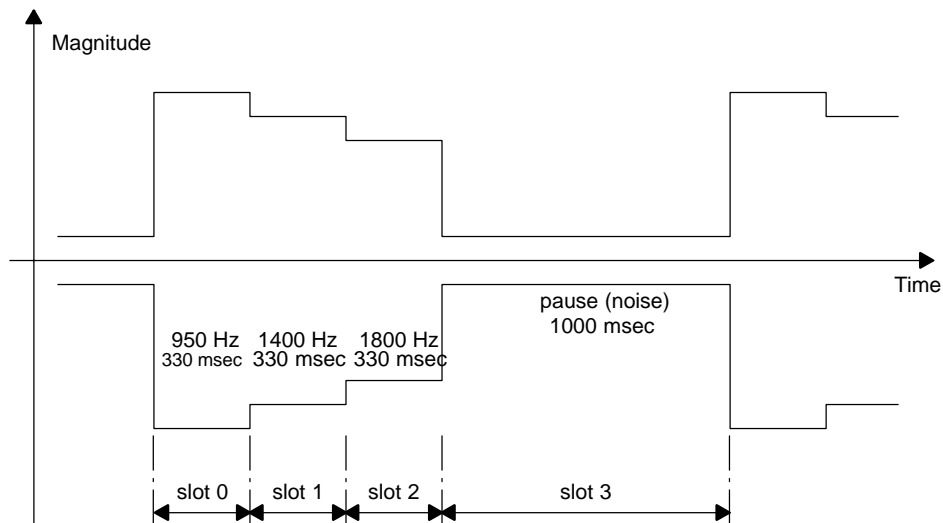
Figure 2-1. UMTD Integration Diagram



This definition covers the majority of alert signals used in the telephone services.

Figure 2-2 illustrates a typical composite signal.

Figure 2-2. Special Information Tone Used in Most of the Countries

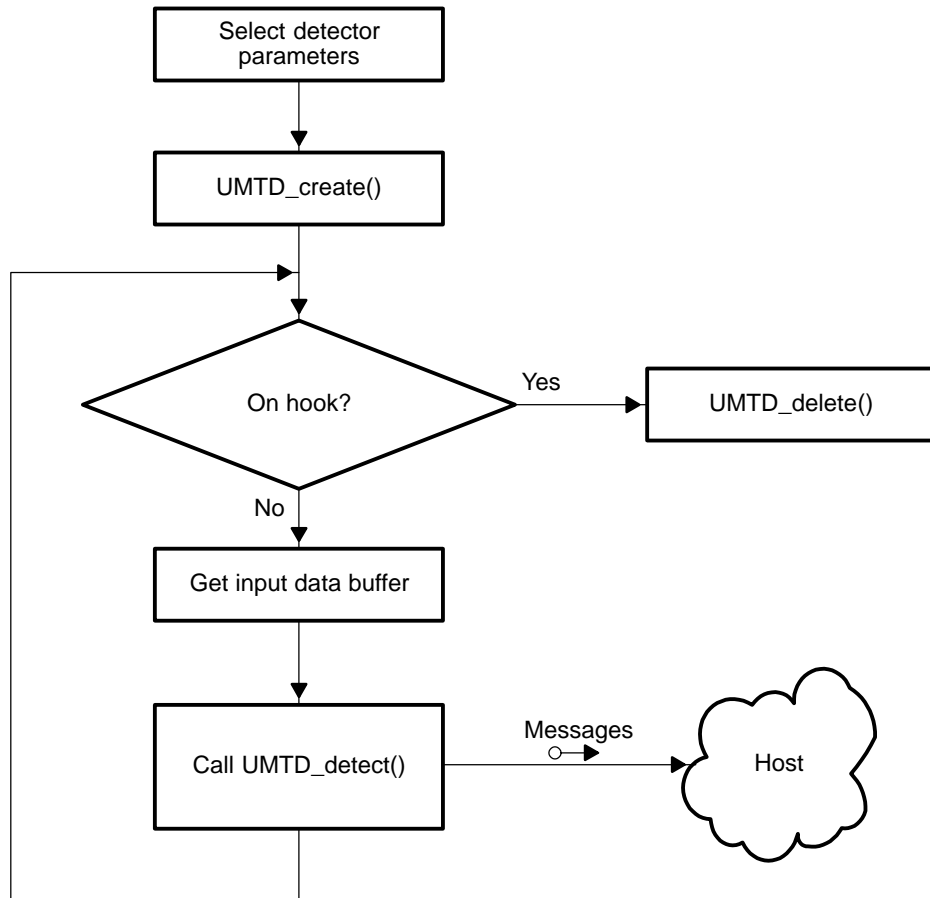


2.2 Integration Flow

In order to integrate the UMTD detector into a framework, the user should (see Figure 2-3):

- Step 1:** Create a handler that will accept messages from a number of UMTD instances
- Step 2:** Create a `UMTD_Params` structure and initialize it with required values.
- Step 3:** Call `UMTD_create()` to create an instance of detector. There are no restrictions on the maximum number of detector instances created.
- Step 4:** Pass a stream with input samples (8 kHz, 16 bits) to `UMTD_detect()` routine. When detector decides that one of the tones is valid or some significant event arises, the detector sends an appropriate message to the host.
- Step 5:** Delete the detector by using `UMTD_delete()` when on hook state is detected or for any other reason depending on your application.

Figure 2-3. Typical Detector Integration Flow



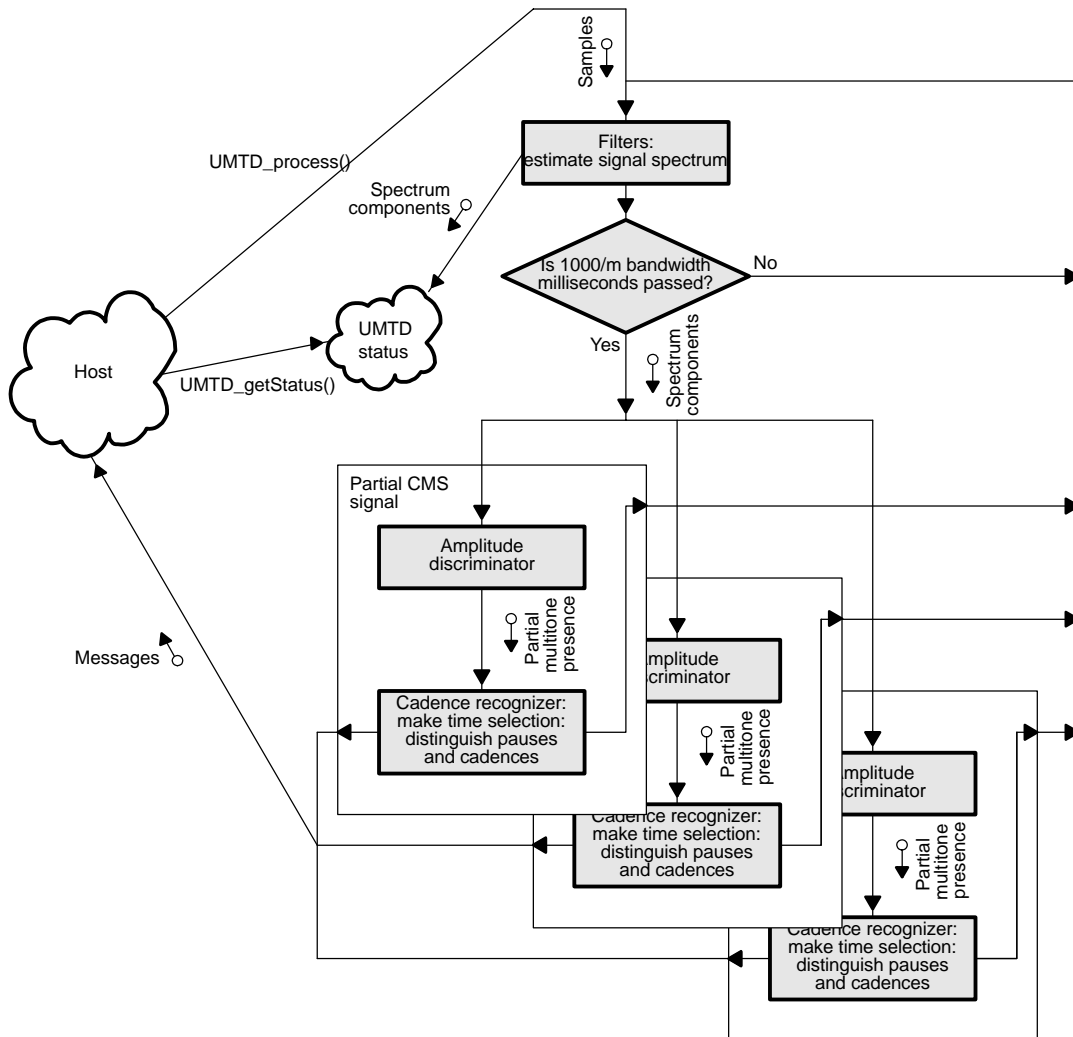
2.3 Signal Recognition

2.3.1 General

Each CMS is recognized independently from each other, however, since the different analyzed signals can share frequencies to be taken into account, only one common frequency list is defined and used for all signals in current group. The size of this list is the main factor influencing detector's MIPS consumption. Typically, 1 MIPS is needed for 16 spectrum components.

Figure 2-4 illustrates a simplified UMTD program flow.

Figure 2-4. Simplified UMTD Program Flow



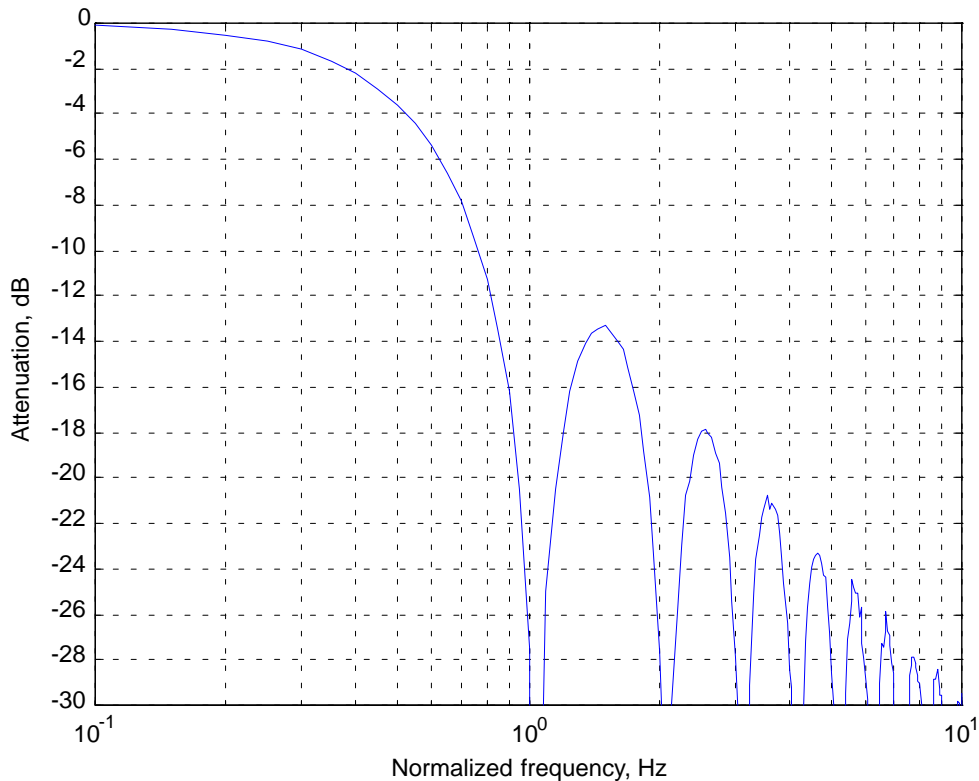
2.3.2 Frequency Selection

The user can specify any number of frequencies to be analyzed simultaneously. When the user wants to increase this value, several UMTD detector instances can be created simultaneously, providing the user with enhanced signal detection.

Simple and fast first order IIR filters are used for the selection. Filtering bandwidth (`bandwidth`, see Table 2-8) controls frequency recognition. Also, it defines the interval for amplitude discriminator.

Filter frequency response is shown in Figure 2-5. Normalized frequency is the ratio of frequency deviation to filtering bandwidth. For large deviations, frequency response has approximately 6 dB per octave slope.

Figure 2-5. Normalized Filter Frequency Response



2.3.3 Amplitude Discrimination

Amplitude discrimination is invoked with the interval defined by filtering bandwidth, i.e., once per 1000/bandwidth milliseconds. This decimation reduces average MIPS consumption with an insignificant loss of quality.

Each partial multitone signal to be recognized is supplied with the options that control its amplitude discrimination.

The amplitude discriminator outputs a binary decision whether a particular multitone signal is present at the moment. summarizes available discriminator options.

The output of amplitude discriminator is passed to the cadence recognizer (see section 2.3.4) for further processing.

Table 2-1. Amplitude Discrimination Options Summary

| Name | Description |
|---------------------|---|
| IUMTD_ADIGNORE | Amplitude of given frequency component is ignored. |
| IUMTD_ADEXIST | Given frequency component must be greater than threshold level <code>signalThreshold</code> (see Table 2-8). |
| IUMTD_ADCANEXIST | Given frequency component can be either greater or less than threshold level <code>signalThreshold</code> (see Table 2-8). However, at least one of spectrum components marked by this option must exist. |
| IUMTD_ADPAUSE | Given frequency component must be less than pause threshold level <code>pauseThreshold</code> (see Table 2-8). |
| IUMTD_ADSPURIOUS | Given frequency component must be less than the spurious level <code>spuriousLevel</code> regarding the maximum spectrum component (see Table 2-8). |
| IUMTD_ADDTMFROW | Given frequency component is treated as DTMF 'row' tone. Row/column twist ratio is verified. |
| IUMTD_ADDTMFCOL | Given frequency component is treated as DTMF 'column' tone. Column/row twist ratio is verified. |
| IUMTD_ADDTMF2HARM | Given frequency component is treated as DTMF second harmonics tone. 2nd harmonics test is performed. |
| IUMTD_ADLOWER | Given frequency component must be lower than the maximum spectrum component. |
| IUMTD_ADSPURIOUSROW | Given frequency component must be less than the spurious level <code>spuriousRow</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFROW</code> (see Table 2-8). |

Table 2-1. Amplitude Discrimination Options Summary (Continued)

| Name | Description |
|----------------------|---|
| IUMTD_ADSPURIOUSCOL | Given frequency component must be less than the spurious level <code>spuriousCol</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFCOL</code> (see Table 2-8). |
| IUMTD_ADSPURIOUSROW2 | Given frequency component must be less than the spurious level <code>spuriousRow2</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFROW</code> (see Table 2-8). |
| IUMTD_ADSPURIOUSCOL2 | Given frequency component must be less than the spurious level <code>spuriousCol2</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFCOL</code> (see Table 2-8). |

2.3.4 Time Selection

The final step of CMS analysis is time selection made by cadence recognizer. It distinguishes each CMS by the state machine controlled by the list of time slots defined in Table 2-10.

A time slot is considered “failed” when it is shorter than given duration (including tolerance). In this case, the detector issues message `IUMTD_MFAIL` (see Table 2-17) and state machine starts execution from the first time slot.

Cadence recognizer options are shown in Table 2-2.

Table 2-2. Time Selection Options Summary

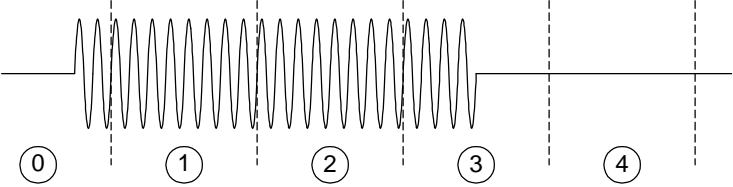
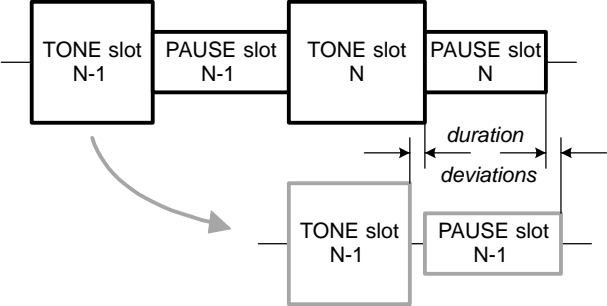
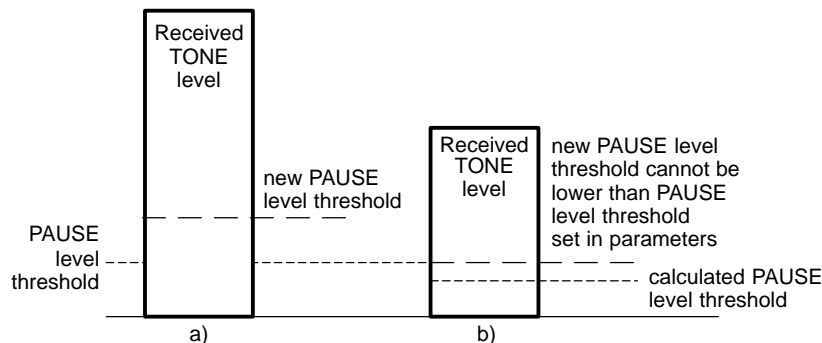
| Name | Description |
|-------------------|---|
| IUMTD_TSGUARDTIME | <p>Enables transition process at the beginning of this time slot. State machine ignores signal spectrum during the first frame of this time slot.</p>  <p>Frame 0 and 3 will not be detected as tone time slots. When frame 0 is processed state machine is not in detected state and no state switching is performed. Setting <code>IUMTD_TSGUARDTIME</code> option for the first time slot will not have effect.</p> <p>But when frame 3 is processed it cannot be recognized not as tone time slot and state machine will switch continue detection with next time slot (pause slot in this case). Pause time slot will not be detected too, so state machine will finish detection and initiate the message <code>IUMTD_FAIL</code> in case option <code>IUMTD_TSGUARDTIME</code> is not set for the pause timeslot. Otherwise state machine ignores frame 3 and continues signal detection from frame 4. If pause will not be detected on frame 4 when signal detection will be finished with message <code>IUMTD_FAIL</code>.</p> |
| IUMTD_TSQUALITY | <p>Enables checking of DTMF signal level to whole signal level.</p> <p>This option is used to improve talkoff performance. UMTD calculates whole signal power, valid signal power and then calculates the ratio:</p> $ratio = \frac{POWER_{valid}}{POWER_{whole}}$ <p>When <i>ratio</i> is more than <code>signalQuality</code> parameter value (see Table 2-5) UMTD decides that it is a valid signal.</p> |
| IUMTD_TSPOROSITY | <p>Verifies tone and pause duration derivations for signals with two cadences.</p>  <p>Duration deviations cannot be more than 1.5 frame length.</p> |

Table 2-2. Time Selection Options Summary (Continued)

| Name | Description |
|----------------------|---|
| IUMTD_TSMINTIME | Verifies duration of tone using only <code>minTime</code> limit mentioned in the . After <code>minTime</code> was reached, waits for input signal changes and then goes to the next time slot. When option <code>IUMTD_GONEXT</code> was specified also UMTD does not wait for signal changes and switches to the next time slot immediately after <code>minTime</code> was reached. |
| IUMTD_TSQ35BUSY | Makes duration check specifically for Q.35 busy tone. |
| IUMTD_TSINTERRUPTION | Enables signal interruptions. The number of frames for which signal can be interrupted is set in parameters structure (see Table 2-5). This option can be used only with the <code>IUMTD_TSMINTIME</code> option. |
| | <p>Diagram illustrating signal interruption. A horizontal bar represents a signal frame. A shaded region in the middle indicates an interruption. A double-headed arrow labeled $\geq \text{minTime}$ spans from the start of the frame to the start of the interruption. Another double-headed arrow labeled $\leq \text{interruptCount}$ spans the duration of the interruption. A third double-headed arrow labeled $\geq 1 \text{ frame}$ spans from the end of the interruption to the end of the frame. Text annotations explain that "signal interruption property turns on only after <code>minTime</code> period", "<code>interruptCount</code> is amount of frames what can be not detected (interrupted)", and "at least one frame should be detected after <i>interrupted</i> period and before next time slot".</p> |
| | Signal Interruption is disappearing of the signal for some short interval. According to some standards, if a signal (DTMF, f.e.) disappears for less than 5 ms and then re-appears, it still has to be considered as one signal (one digit). |
| IUMTD_TSUSEDTMFMASK | Removes peak MIPS on DTMF signals detection. This option can only be used when lower 8 bits of <code>signalID</code> (see Table 2-9) have values described in Table 2-12. |

Table 2-2. Time Selection Options Summary (Continued)

| Name | Description |
|--------------------|---|
| IUMTD_TSADAPTPAUSE | Enables pause level calculation during tone time slots. |



New pause level calculated using formula:

$$PAUSE_{adapted} = \max[PAUSE_{threshold} (TONE_{level} \times adapt_coef)].$$

where

$$PAUSE_{adapted} = \text{new pause threshold}$$

$$PAUSE_{threshold} = mPause \text{ field value (see Table 2-5)}$$

$$TONE_{level} = \text{received tone level}$$

$$adapt_coef = mPauseAdapt \text{ field value (see Table 2-5)}$$

2.3.5 Flow Control Options

Additionally, time slot flags contain fields that provide flow control and host indication (see Table 2-3). They are used together with time selection options.

Table 2-3. Flow Control Options Summary

| Name | Description |
|------------------|--|
| IUMTD_TSGONEXT | Executes next time slot once this time slot is finished. Has effect if used together with IUMTD_TSMINTIME option. |
| IUMTD_TSREPEAT | Repeats execution from the first time slot once this time slot is finished. |
| IUMTD_TSBREAK | Forces detector to consider the CMS detection as “failed” once this time slot is finished, and repeats execution from the first time slot. |
| IUMTD_TSGOCOMMON | Continues execution from common time slot pointed by field pCommonSlot (see Table 2-5). |

Execution always starts from the first time slot if current time slot fails.

2.3.6 Host Indication Options

The moment when the detector signals to the host about detection of time slot is defined in Host indication options in the time slot flags (see Table 2-4).

Table 2-4. Host Indication Options Summary

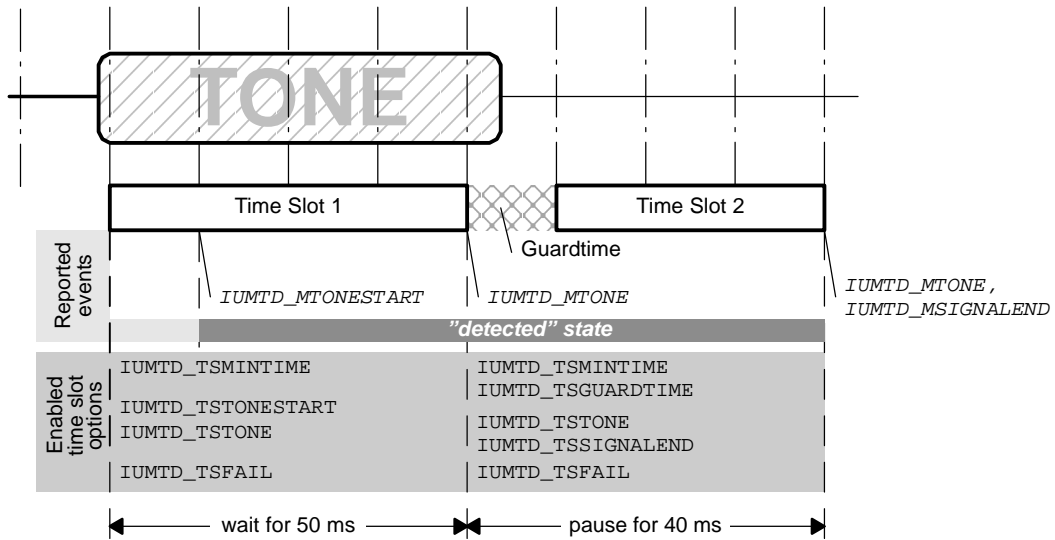
| Name | Description |
|-------------------|--|
| IUMTD_TSTONESTART | Enables UMTD to indicate/report the transition to the “detected” state for this signal. |
| IUMTD_TSTONE | Enables UMTD to indicate the successful time slot detection |
| IUMTD_TSDIGNALEND | Enables UMTD to indicate the successful detection of last time slot. |
| IUMTD_TSFALL | Enables UMTD to indicate the detection failure (end of the “detected” state for this time slot). |

The host can receive 4 messages from UMTD: IUMTD_MTONESTART, IUMTD_MTONE and IUMTD_MSIGNALEND and IUMTD_TSFALL.

- Message IUMTD_MTONESTART indicates successful first frame detection and the fact that recognizer fell into “*detected*” state for this signal. This message is useful for signal *early detection*.
- Message IUMTD_MTONE indicates successful time slot detection, but it is issued every time when a slot with enabled IUMTD_TSTONE option is processed and the detector is in “detected” state.
- Message IUMTD_MSIGNALEND indicates successful detection of last time slot.
- Message IUMTD_MFALL indicates the end of the “*detected*” state. For the first time slot this message is issued only if IUMTD_TSTONESTART and IUMTD_TSFALL are present. For other time slots only option IUMTD_TSFALL is needed.

Figure 2-6 shows the UMTD configuration to recognize a 40 ms (or more) “tone” and following “pause” for at least 20 ms and send the first message about tone detection after 20 ms.

Figure 2-6. Host Indication Example



2.4 Parameter Definition

Table 2-5. General Parameters for Recognized Signal Series

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|-----------------------------|------------|---------------|------------------|--|
| signalThreshold | XDAS_Int16 | -30 | | Detector threshold, dBm |
| pauseThreshold | XDAS_Int16 | -35 | <signalThreshold | Pause threshold, dBm. Used by amplitude discriminator when option IUMTD_ADPAUSE (see Table 2-13) is enabled |
| pauseAdapt | XDAS_Int16 | .25 | <1 | Signal/Pause adaptation coefficient. Used when option IUMTD_TSADAPTPAUSE (see Table 2-14) is set. |
| signalQuality | XDAS_Int16 | .80 | <1 | Valid signal to whole signal power ratio. Signal/Pause adaptation coefficient. Used when option IUMTD_TSQUALITY (see Table 2-14) is set. |
| interruptCount | XDAS_Int16 | 0 | 0...100 | The number of frames for which signal can be interrupted. Used when option IUMTD_TSINTERRUPTION (see Table 2-14) is set. |
| spuriousLevel | XDAS_Int16 | -10 | -12 ...-6 | Spurious (out-band) signal level, dBc. Used by amplitude discriminator when option IUMTD_ADSPURIOUS (see Table 2-13) is enabled. |
| spuriousRow spuriousRow2 | XDAS_Int16 | -10 | -12 ...-6 | Spurious (out-band) signal level, dBc. Used by amplitude discriminator when option IUMTD_ADSPURIOUSROW (see Table 2-13) is enabled. |

Table 2-5. General Parameters for Recognized Signal Series (Continued)

| Name | Type | Typical Value | Limits | Description |
|------------------------------|----------------------|---------------|-----------|---|
| spuriousCol1 spuriousCol2 | XDAS_Int16 | -10 | -12 ...-6 | Spurious (out-band) signal level, dBc. Used by amplitude discriminator when option IUMTD_AD-SPURIOUSCOL (see Table 2-13) is enabled. |
| dtmfTwist | XDAS_Int16 | -5 | | Spurious (out-band) signal level, dBc. Used by amplitude discriminator when options IUMTD_ADDTMFROW and IUMTD_ADDTMFCOL (see Table 2-13) is enabled. |
| dtmf2harm | XDAS_Int16 | -3 | | Spurious (out-band) signal level for *2 harm. In MF signal, dBc. Used by amplitude discriminator when option IUMTD_ADDTMF2HARM (see Table 2-13) is enabled. |
| nFrequencies | XDAS_Int16 | N/A | < 20 | Number of frequencies to be processed simultaneously |
| nSignals | XDAS_Int16 | N/A | > 0 | Number of CMS signals to be recognized |
| pSignalList | const UMTD_Signal* | N/A | N/A | Recognized CMS signals (see Table 2-9) |
| pCommonSlot | const UMTD_TimeSlot* | N/A | N/A | Pointers to common time slot. Used if flag IUMTD_TSGOCOMMON in time slot parameters is enabled (see Table 2-10). |

```

}
IUMTD_GeneralSeriesParams;

```

Figure 2-7 illustrates typical UMTD thresholds.

Figure 2-7. UMTD Thresholds

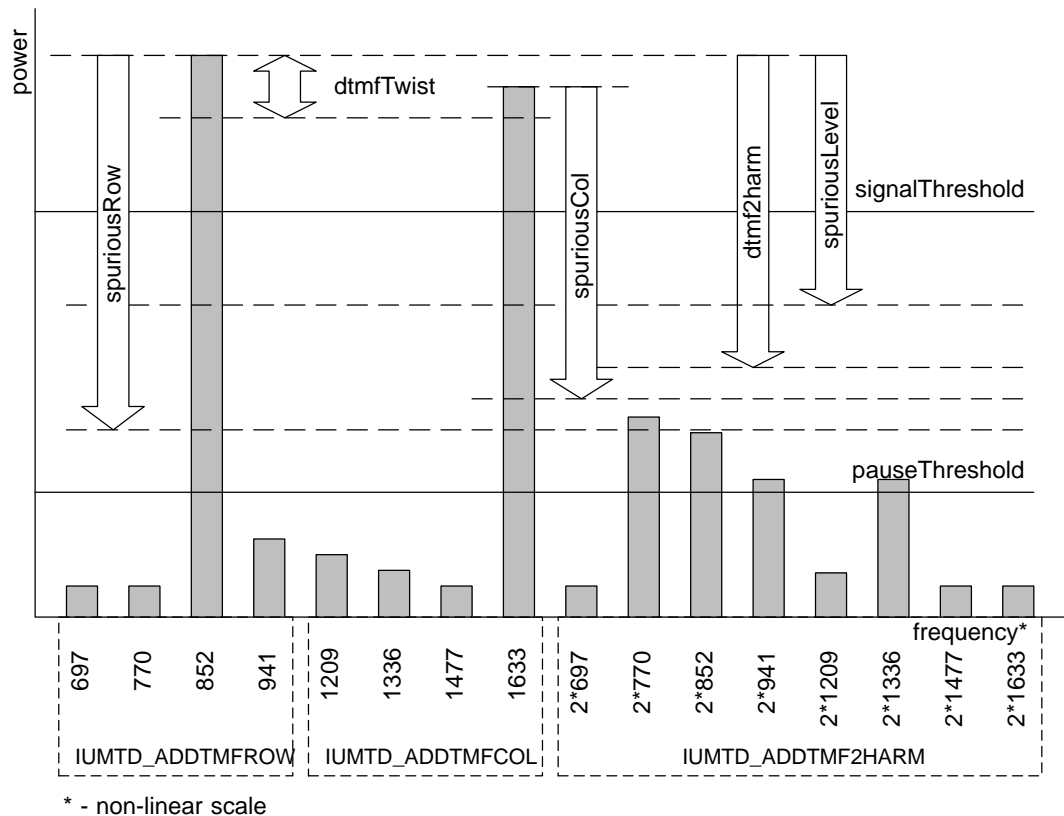


Table 2-6. Parameters for Recognized Signal Series

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|----------------|-----------------------------------|---------------|--------|---|
| pFrequencyList | const XDAS_Int16* | N/A | <340 | Frequency planner. Frequencies are in Hz |
| bandwidth | XDAS_Int16 | 80 | >0 | Filtering bandwidth, Hz |
| parameters | const UMTD_GeneralSeriesParams | N/A | N/A | General series parameters described above (see Table 2-5) |

```
}
IUMTD_SeriesParams;
```

Table 2-7. Pointers to Series Parameters

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|-------------|------------------------------|---------------|--------|---|
| pSerieParam | const IUMTD_SeriesParams* | N/A | N/A | Pointer to array containing pointers to parameters for recognized signal series (see Table 2-6) |

```
}
IUMTD_ptrSerParam;
```

Table 2-8. Common UMTD Parameters

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|---------------|--------------------|---------------|--------|---|
| host | UMTD_Host | N/A | N/A | Host controller (see section 2.5) |
| dBm0 | XDAS_Int16 | 5000 | | Reference level of 0 dBm signal |
| maxLevel | XDAS_Int16 | 8000 | | Maximum level of input signal, dBm |
| pSerParamList | IUMTD_ptrSerParam* | N/A | N/A | Pointer to structures containing data for signal series recognition |
| seriesCount | XDAS_Int16 | N/A | >0 | Number of structures to be analyzed during the recognition |
| maxBufferLen | XDAS_Int16 | N/A | >=0 | Size of input data. Setting this parameter to size of your input frame will allow you to reduce peak MIPS. Set 0 to adjust parameter value automatically. |

```
}
IUMTD_Params;
```

Table 2-9. CMS Recognition Parameters

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|------------|----------------------|---------------|--------|---|
| signaled | XDAS_Int16 | N/A | N/A | Enumerator of CMS signal. Each signal has to be provided with unique identifier. This value shall be used by a host to distinguish between different CMS signals. |
| nTimeSlots | XDAS_Int16 | N/A | >0 | Number of time slots to be analyzed during the recognition |
| pTimeSlots | const UMTD_TimeSlot* | N/A | N/A | Time slots to be analyzed during the recognition of this CMS signal |

```

}
UMTD_Signal;

```

Table 2-10. UMTD Time Slots

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|--------------|--|---------------|--------|---|
| slotFlags | XDAS_Int16 (bitfield, see section 2.4.2) | N/A | N/A | Flags that control actions associated with this time slot (see Table 2-14, Table 2-15, and Table 2-16) |
| minTime | XDAS_Int16 | N/A | >0 | Minimum duration of this time slot, in tens of milliseconds. Interpreted in accordance with flags (see Table 2-14) |
| maxTime | XDAS_Int16 | N/A | >0 | Duration of this time slot, in tens of milliseconds. Interpreted in accordance with flags (see Table 2-14) |
| aADOptions[] | XDAS_UInt16[N] (bitfield, see section 2.4.1) | N/A | N/A | Amplitude discriminator options that control the recognition of this CMS. Format of bit fields is defined in Table 2-11 and Table 2-13. |

```

}
UMTD_TimeSlot;

```

2.4.1 Amplitude Discriminator Options

These flags control the recognition of partial multitone signal in the current time slot. This member consists of twenty 4-bit fields, each corresponding to an appropriate frequency in the common frequency list (see Table 2-11 and Table 2-13).

Figure 2-8. Bit Field Position illustration

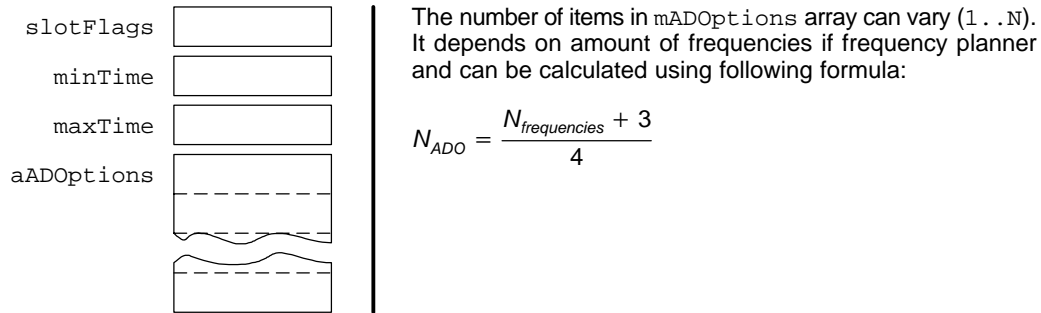


Table 2-11. Bit Field Positions in the Recognition Flags

| Bit Numbers | Word Number | Description |
|-------------|-------------|-------------------------------------|
| 0-3 | 0 | options for frequency with index 0 |
| 4-7 | 0 | options for frequency with index 1 |
| 8-11 | 0 | options for frequency with index 2 |
| 12-15 | 0 | options for frequency with index 3 |
| 0-3 | 1 | options for frequency with index 4 |
| 4-7 | 1 | options for frequency with index 5 |
| 8-11 | 1 | options for frequency with index 6 |
| 12-15 | 1 | options for frequency with index 7 |
| 0-3 | 2 | options for frequency with index 8 |
| 4-7 | 2 | options for frequency with index 9 |
| 8-11 | 2 | options for frequency with index 10 |
| 12-15 | 2 | options for frequency with index 11 |
| 0-3 | 3 | options for frequency with index 12 |
| 4-7 | 3 | options for frequency with index 13 |
| 8-11 | 3 | options for frequency with index 14 |

Table 2-11. Bit Field Positions in the Recognition Flags (Continued)

| Bit Numbers | Word Number | Description |
|-------------|-------------|-------------------------------------|
| 12-15 | 3 | options for frequency with index 15 |
| 0-3 | 4 | options for frequency with index 16 |
| 4-7 | 4 | options for frequency with index 17 |
| 8-11 | 4 | options for frequency with index 18 |
| 12-15 | 4 | options for frequency with index 19 |

Table 2-12. DTMF Signal Mask for IUUMTD_TSUSEDTMFMASK Option










| signalID | DTMF Signal | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|---|---|---|---|------|------|-----------|---|---|---|--|-------------|-------------|-------------|------|-----------|-----|----------------|---|---|---|---|-----|---|---|---|---|---|-----|---|---|---|---|---|-----|---|---|---|---|---|--------------|---|---|---|---|--|
| 0x..00 | 1 | <table border="1"> <tr> <td></td> <td>col</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>row</td> <td></td> <td>1209</td> <td>1336</td> <td>1477</td> <td>1633</td> <td>row index</td> </tr> <tr> <td>697</td> <td>1</td> <td>2</td> <td>3</td> <td>A</td> <td>0</td> </tr> <tr> <td>770</td> <td>4</td> <td>5</td> <td>6</td> <td>B</td> <td>1</td> </tr> <tr> <td>852</td> <td>7</td> <td>8</td> <td>9</td> <td>C</td> <td>2</td> </tr> <tr> <td>941</td> <td>*</td> <td>0</td> <td>#</td> <td>D</td> <td>3</td> </tr> <tr> <td>column index</td> <td>0</td> <td>1</td> <td>2</td> <td>3</td> <td></td> </tr> </table> | | col | | | | | | row | | 1209 | 1336 | 1477 | 1633 | row index | 697 | 1 | 2 | 3 | A | 0 | 770 | 4 | 5 | 6 | B | 1 | 852 | 7 | 8 | 9 | C | 2 | 941 | * | 0 | # | D | 3 | column index | 0 | 1 | 2 | 3 | |
| | col | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| row | | | 1209 | 1336 | 1477 | 1633 | row index | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 697 | 1 | | 2 | 3 | A | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 770 | 4 | | 5 | 6 | B | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 852 | 7 | | 8 | 9 | C | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 941 | * | | 0 | # | D | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| column index | 0 | | 1 | 2 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..01 | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..02 | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..10 | 4 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..11 | 5 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..12 | 6 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..20 | 7 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..21 | 8 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..22 | 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..31 | 0 | <table border="1"> <thead> <tr> <th colspan="4">mEnumerator</th> </tr> </thead> <tbody> <tr> <td>0x</td> <td></td> <td></td> <td></td> </tr> <tr> <td></td> <td>any digit -</td> <td>any digit -</td> <td>row index -</td> </tr> <tr> <td></td> <td></td> <td></td> <td>column index -</td> </tr> </tbody> </table> | mEnumerator | | | | 0x |  |  |  | | any digit - | any digit - | row index - | | | | column index - | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| mEnumerator | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x |  | |  |  | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | any digit - | | any digit - | row index - | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | column index - | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..03 | A | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..13 | B | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..23 | C | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..33 | D | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..30 | * | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 0x..32 | # | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Table 2-13. Amplitude Discrimination Options

```
typedef enum
{
```

| Name | Value | Description |
|----------------------|-------|---|
| IUMTD_ADIGNORE | 0x00 | Amplitude of given frequency component is ignored |
| IUMTD_ADPAUSE | 0x01 | Given frequency component must be lower than pause threshold level <code>pauseThreshold</code> (see Table 2-8) |
| IUMTD_ADSPURIOUS | 0x03 | Given frequency component must be less than the spurious level <code>spuriousLevel</code> regarding the maximum spectrum component (see Table 2-8). |
| IUMTD_ADCANEXIST | 0x04 | Given frequency component can be either greater or less than threshold level <code>signalThreshold</code> (see Table 2-8). However, at least one of spectrum components marked by this option must exist. |
| IUMTD_ADEXIST | 0x05 | Given frequency component must be greater than threshold level <code>signalThreshold</code> (see Table 2-8) |
| IUMTD_ADDTMFROW | 0x06 | Given frequency component is treated as DTMF 'row' tone. Row/column twist ratio is verified |
| IUMTD_ADDTMFCOL | 0x07 | Given frequency component is treated as DTMF 'column' tone. Column/row twist ratio is verified |
| IUMTD_ADDTMF2HARM | 0x08 | Given frequency component is treated as DTMF second harmonics tone. 2-nd harmonics test is performed. |
| IUMTD_ADLOWER | 0x09 | Given frequency component must be lower than the maximum spectrum component. |
| IUMTD_ADSPURIOUSROW | 0x0A | Given frequency component must be less than the spurious level <code>spuriousRow</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFROW</code> (see Table 2-8). |
| IUMTD_ADSPURIOUSCOL | 0x0B | Given frequency component must be less than the spurious level <code>spuriousCol</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFCOL</code> (see Table 2-8). |
| IUMTD_ADSPURIOUSROW2 | 0x0C | Given frequency component must be less than the spurious level <code>spuriousRow2</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFROW</code> (see Table 2-8). |
| IUMTD_ADSPURIOUSCOL2 | 0x0D | Given frequency component must be less than the spurious level <code>spuriousCol2</code> regarding the spectrum component marked with <code>IUMTD_ADDTMFCOL</code> (see Table 2-8). |

```
}
IUMTD_AOptions;
```

2.4.2 Time Slot Flags

These flags control the action of recognition state machine that must be performed at current time slot. They are represented by bit fields in the time slot flags `slotFlags` (see Table 2-10).

This member consists of a number of bit fields defined in Table 2-14, Table 2-15, and Table 2-16.

2.4.2.1 Time Selection

Time selection options control the cadence recognizer.

Table 2-14. Time Selection Options

| Name | Value | Duration | Description |
|----------------------|-------|----------------|--|
| IUMTD_TSGUARDTIME | 0x01 | 1 frame | Enables transition process at the beginning of this time slot. State machine ignores signal spectrum during the first frame of time slot. |
| IUMTD_TSQUALITY | 0x40 | 1 time slot | Enables checking ratio of DTMF signal level to whole signal level. |
| IUMTD_TSPOROSITY | 0x02 | | Enables checking of tone and pause duration changes. NOTE: This option is to used only for signals with two cadences. |
| IUMTD_TSMINTIME | 0x04 | \geq minTime | Verifies whether tone duration is within the limits of <code>minTime</code> , specified in the . |
| IUMTD_TSQ35BUSY | 0x08 | see Note | Makes duration check specifically for Q.35 busy tone. |
| IUMTD_TSINTERRUPTION | 0x10 | 1 frame | Enables signal interruption, but not more than one frame per time slot. |
| IUMTD_TSUSEDTMFMASK | 0x20 | | Removes peak MIPS on DTMF signals detection. NOTE: This option can only be used when lower 8 bits of <code>signalID</code> (see Table 2-9) have values described in Table 2-12. |
| IUMTD_TSADAPTPAUSE | 0x80 | | Enables pause level calculations during tone time slots. NOTE: This option is to used only for tone time slots. |

Note: For IUMTD_Q35BUSY, UMTD verifies the ratio of the tone period to the silent period as well as the pause duration. The ratio mentioned above should be between 0.67 and 1.5, and tone period may be up to 500 milliseconds shorter than the silent period. Under no circumstances should the tone period be shorter than 100 milliseconds.

See more information on these options in Table 2-2.

2.4.2.2 Flow Control Options

Flow control options control state machine operation.

Normally, execution goes slot-by-slot unless any slot fails. Some flags can change this behavior.

Execution is always started from the first time slot if current time slot fails.

Table 2-15. Flow Control Options

| Name | Value | Description |
|------------------|--------|---|
| IUMTD_TSGONEXT | 0x0100 | Execute next time slot once this time slot is finished. |
| IUMTD_TSREPEAT | 0x0200 | Repeat execution from the first time slot once this time slot is finished. |
| IUMTD_TSBREAK | 0x0400 | Force detector to consider the CMS detection as “failed” once this time slot finishes, and repeat execution from the first time slot. |
| IUMTD_TSGOCOMMON | 0x0800 | Continue execution from the common time slot pointed by field <code>pCommonSlot</code> (see Table 2-5). |

2.4.2.3 Host Indication Options

The moment and type of indication that detector signals to the host about the detection of time slot is defined in Host indication options in the time slot flags `slotFlags` (see Table 2-10). Read section 2.3.6 for more information on these options.

Table 2-16. Host Indication Options

| Name | Value | Description |
|-------------------|--------|--|
| IUMTD_TSTONESTART | 0x1000 | Enables UMTD to indicate/report transition to the “detected” state for this signal by sending message <code>IUMTD_MTONESTART</code> (see Table 2-17) to the host when this time slot is finished. This message is issued after successful detection of first frame in time slot. |
| IUMTD_TSTONE | 0x2000 | Enables UMTD to send message <code>IUMTD_MTONE</code> (see Table 2-17) to the host when this time slot is finished and detector is in “detected” state. |
| IUMTD_TSSIGNALEND | 0x4000 | Enables UMTD to send message <code>IUMTD_MSIGNALEND</code> (see Table 2-17) to the host when last (this) time slot is finished. |
| IUMTD_TSFAIL | 0x8000 | Enables UMTD to send message <code>IUMTD_MFAIL</code> (see Table 2-17) to the host when “detected” state of CMS signal is finished. It is means - detection failed. |

2.5 Messages and Host Interface

UMTD detector informs the host about its state by issuing the messages. The host is attached to the detector on object creation (see section 3.2.3). The proprietary messaging service is implemented for host interface. The detector always sends messages to the host, so it is a one-way message flow.

The user can change message flow and receive messages from `UMTD_detect` function directly (as return value). In this case `pfnHandler` field of `host` structure should be set to `NULL`.

Host identifies actual UMTD signals by field `sender` in the message. This field is filled by value `signalID` (see Table 2-17) corresponding to the UMTD signal related to this message.

The field `type` identifies the bit mask of message type. All message types are summarized in the .

Table 2-17. UMTD Detector Messages Summary

```
typedef enum
{
```

| Enumeration Constant | Value | Action |
|----------------------|--------|---|
| IUMTD_MNONE | 0x0000 | Not a valid message. Message of this type is never issued by UMTD to the host, but is returned by <code>UMTD_detect()</code> function when there is no message available at the moment. |
| IUMTD_MTONESTART | 0x1000 | Indicates successful detection of the signal for the first frame and the fact that the recognizer fell into the “detected” state for this signal. |
| IUMTD_MTONE | 0x2000 | Indicates successful time slot detection of time slot, but it is issues as many times as slot with enabled option <code>IUMTD_TSTONE</code> is processed. |
| IUMTD_MSIGNALEND | 0x4000 | Indicates successful detection of last time slot, it is issues only for last times slot with enabled option <code>IUMTD_TSSIGNALEND</code> . |
| IUMTD_MFAIL | 0x8000 | Indicates the end of the “detected” state. If no “detected” state was established earlier, this message is not issued. |

```
}
IUMTD_MessageType;
```

Table 2-18. Message Structure

```
typedef struct
{
```

| Type | Name | Comment |
|-------------------|--------|---|
| XDAS_Int16 | sender | Enumerator of CMS signal. Is filled by signalID field of structure UMTD_Signal (see Table 2-9). |
| IUMTD_MessageType | type | Message type. Must be one of enums defined in Table 2-17. |

```
}
IUMTD_Message;
```

Table 2-19. Host Controller

```
typedef struct
{
```

| Name | Type | Typical Value | Limits | Description |
|------------|--|---------------|--------|--|
| pInstance | Void* | N/A | N/A | Internal host instance handle. It is always used as the first parameter in the function defined below. |
| pfnHandler | XDAS_Void (*)(Void*, const IUMTD_Message*) | N/A | N/A | Host callback function to be invoked when message is sent. Messages of type UMTD_Message are defined in . UMTD uses field mpInstance defined above as a first parameter for this function. |

```
}
UMTD_Host;
```

2.6 Examples

2.6.1 Typical CPTD Settings for USA's PSTN

2.6.1.1 Header File

```
#ifndef UMTD_CPTD_H___
#define UMTD_CPTD_H___ 1
#include "iumtd.h"
#include "comtypes.h"
extern const IUMTD_SeriesParams CPTD_series;
#endif //UMTD_CPTD_H___
```

2.6.1.2 Source File

```
#include "CPTD_usa.h"
#define CPTD_TONE_FLAGS      IUMTD_TSFAIL
#define CPTD_PAUSE_FLAGS    IUMTD_TSDIGNALEND|IUMTD_TSFAIL|IUMTD_TSGUARDTIME
//-----
// Frequency planner . Frequencies are in Hz
//-----
static const XDAS_Int16 CPTD_frequencies [] =
{
    330,    350,    370,    440,
    460,    480,    500,    600,
    620,    640
};
//-----
// Typical recognition parameters for USA busy tone
//-----
static const IUMTD_TimeSlot busySlots [] =
{ {{0x5555, 0x9919, 0x5591, 0x5555, 0x5555}, IUMTD_TSLIMITS | CPTD_TONE_FLAGS,
  50,  550}, // 480+620 Hz, 0,5 sec
  {{0x3333, 0x3333, 0x3333, 0x3333, 0x3333}, IUMTD_TSQ35BUSY | CPTD_PAUSE_FLAGS,
  50,  550} // pause, indicate once
};
//-----
// Typical recognition parameters for USA dial tone
//-----
```

```

static const IUMTD_TimeSlot dialSlots [] =
{ {{0x1919, 0x5559, 0x5555, 0x5555, 0x5555}, IUMTD_TSLIMITS | CPTD_TONE_FLAGS,
  800, 10100}, // 350+440 Hz, 1.3 sec, indicate each time
  {{0x3333, 0x3333, 0x3333, 0x3333, 0x3333}, IUMTD_TSLIMITS | CPTD_PAUSE_FLAGS,
  50, 0x7fff} // pause,
};

//-----
// Typical recognition parameters for USA ringback tone
//-----

static const IUMTD_TimeSlot ringbackSlots [] =
{ {{0x1555, 0x5919, 0x5555, 0x5555, 0x5555}, IUMTD_TSLIMITS | CPTD_TONE_FLAGS,
  200, 2500}, // 440+480 Hz, 1 sec
  {{0x3333, 0x3333, 0x3333, 0x3333, 0x3333}, IUMTD_TSLIMITS | CPTD_PAUSE_FLAGS,
  300, 0x7fff} // pause, 3 sec, indicate once
};

//-----
// Recognized CPTD signals: busy, ringback, dial tone
//-----

static const IUMTD_Signal CPTD_signals[] =
{
  AddSignal(0x2001, busySlots),
  AddSignal(0x2002, dialSlots),
  AddSignal(0x2003, ringbackSlots)
};

const IUMTD_SeriesParams CPTD_series =
{
  CPTD_frequencies, // pFrequencyList */
  20, // bandwidth, Hz */
  {
    (XDAS_Int16)(0x8000*0.1189), /* (-18.5 dBm) signalThreshold */
    (XDAS_Int16)(0x8000*0.0631), /* (-24 dBm) pauseThreshold */
    (XDAS_Int16)(0x8000*0.2500), /* (-12 dBm) pauseAdapt */
    (XDAS_Int16)(0x8000*0.700), /* signalQuality */
    (XDAS_Int16)(0x8000*0.5000), /* (-6 dBc) spuriousLevel */
    0, /* ( dBc) spuriousCol */
    0, /* ( dBc) spuriousRow */
    0, /* ( dBc) spuriousCol2 */
  }
};

```



```

0,                /* (   dBc) spuriousRow2 */
0,                /* (   dBc) dtmfTwist */
0,                /* (   dBc) dtmf2harm */
sizeof(CPTD_frequencies)/sizeof(XDAS_Int16), /* nFrequencies */
sizeof(CPTD_signals)/sizeof(IUMTD_Signal), /* nSignals */
(IUMTD_Signal*)CPTD_signals,                /* pSignalList */,
NULL
}
};

```

2.6.2 Typical DTMF Settings

2.6.2.1 Header File

```

#ifndef UMTD_DTMF_H___
#define UMTD_DTMF_H___ 1
#include "iumtd.h"
extern const IUMTD_SeriesParams DTMF_series;
#endif /* UMTD_DTMF_H___ 1 */

```

2.6.2.2 Source File

```

#include "DTMF_usa.h"
#define DTMF_TONE_MIN_LEN 20
#define DTMF_TONE_MAX_LEN 0x7fff
#define DTMF_PAUSE_MIN_LEN 20
#define DTMF_PAUSE_MAX_LEN 0x7fff
#if 1
// Standard settings
#define DTMF_TS_FLAGS    IUMTD_TSUSEDTMFMASK|IUMTD_TSMINTIME|IUMTD_TSTONES-
TART|IUMTD_TSGOCOMMON|IUMTD_TSFAIL|IUMTD_TSTONE |IUMTD_TSINTERRUPTION
#define DTMF_PS_FLAGS    IUMTD_TSMINTIME|IUMTD_TSTONE|IUMTD_TSFAIL|IUMTD_TSSIGNA-
LEND|IUMTD_TSGUARDTIME
#else
// Robust settings
#define DTMF_TS_FLAGS    IUMTD_TSUSEDTMFMASK|IUMTD_TSMINTIME|IUMTD_TSTONES-
TART|IUMTD_TSGOCOMMON|IUMTD_TSFAIL|IUMTD_TSTONE |IUMTD_TSINTERRUPTION
#define DTMF_PS_FLAGS    IUMTD_TSMINTIME|IUMTD_TSTONE|IUMTD_TSFAIL|IUMTD_TSSIGNA-
LEND|IUMTD_TSGUARDTIME|IUMTD_TSGUARDTIME2|IUMTD_TSINTERRUPTION
#endif

```

```
//-----  
// Frequency planner . Frequencies are in Hz  
//-----  
const XDAS_Int16 DTMF_frequencies[] =  
{  
    697,    770,    852,    941,  
    1209,   1336,   1477,   1633,  
    1406,   1555,   1711,   1279,  
    1209*2, 1336*2, 1477*2, 1633*2,  
    1134,   1266,    617,   1030  
};  
//-----  
// Typical recognition parameters for DTMF signals  
//-----  
static const IUMTD_TimeSlot DTMF_Tone1slot [] =  
{ {{0x33c6, 0x3337, 0xb888, 0x8888, 0x3c9b}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,  
DTMF_TONE_MAX_LEN },  
};  
static const IUMTD_TimeSlot DTMF_Tone2slot [] =  
{ {{0x33c6, 0x3373, 0x988b, 0x8888, 0x3cb3}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,  
DTMF_TONE_MAX_LEN },  
};  
static const IUMTD_TimeSlot DTMF_Tone3slot [] =  
{ {{0x33c6, 0x3733, 0x38bb, 0x8888, 0x3c33}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,  
DTMF_TONE_MAX_LEN },  
};  
static const IUMTD_TimeSlot DTMF_Tone4slot [] =  
{ {{0x3c6c, 0x3337, 0xb888, 0x8888, 0x339b}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,  
DTMF_TONE_MAX_LEN },  
};  
static const IUMTD_TimeSlot DTMF_Tone5slot [] =  
{ {{0x3c6c, 0x3373, 0x988b, 0x8888, 0x33b3}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,  
DTMF_TONE_MAX_LEN },  
};  
static const IUMTD_TimeSlot DTMF_Tone6slot [] =  
{ {{0x3c6c, 0x3733, 0x38bb, 0x8888, 0x3333}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,  
DTMF_TONE_MAX_LEN },  
};
```

```

static const IUMTD_TimeSlot DTMF_Tone7slot [] =
{ {{0xa6a3, 0x3337, 0xb888, 0x8888, 0x339b}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_Tone8slot [] =
{ {{0xa6a3, 0x3373, 0x988b, 0x8888, 0x33d3}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_Tone9slot [] =
{ {{0xa6a3, 0x3733, 0x38bb, 0x8888, 0x3333}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_Tone0slot [] =
{ {{0x6a33, 0x3373, 0x988b, 0x8888, 0xc3d3}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_ToneAslot [] =
{ {{0x33c6, 0x7333, 0x3bb8, 0x8888, 0x3c33}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_ToneBslot [] =
{ {{0x3c6c, 0x7333, 0x3bb8, 0x8888, 0x3333}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_ToneCslot [] =
{ {{0xa6a3, 0x7333, 0x3bb8, 0x8888, 0x3333}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_ToneDslot [] =
{ {{0x6a33, 0x7333, 0x3bb8, 0x8888, 0xa333}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

static const IUMTD_TimeSlot DTMF_ToneStarslot [] =
{ {{0x6a33, 0x3337, 0xd888, 0x8888, 0xc39b}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

//c 99

static const IUMTD_TimeSlot DTMF_ToneGridslot [] =
{ {{0x6a33, 0x3733, 0x38bb, 0x8888, 0xa333}, DTMF_TS_FLAGS, DTMF_TONE_MIN_LEN,
DTMF_TONE_MAX_LEN },
};

```

```
};

static const IUMTD_TimeSlot DTMF_PauseSlot [] =
{ {{0x1111, 0x1111, 0x1111, 0x1111, 0x1111}, DTMF_PS_FLAGS, DTMF_PAUSE_MIN_LEN,
DTMF_PAUSE_MAX_LEN }
};

//-----
// Recognized DTMF signals
//-----

static const IUMTD_Signal DTMF_signals[] =
{
    ADD_UMTD_SIGNAL(0x4000, DTMF_Tone1slot),
    ADD_UMTD_SIGNAL(0x4001, DTMF_Tone2slot),
    ADD_UMTD_SIGNAL(0x4002, DTMF_Tone3slot),
    ADD_UMTD_SIGNAL(0x4010, DTMF_Tone4slot),
    ADD_UMTD_SIGNAL(0x4011, DTMF_Tone5slot),
    ADD_UMTD_SIGNAL(0x4012, DTMF_Tone6slot),
    ADD_UMTD_SIGNAL(0x4020, DTMF_Tone7slot),
    ADD_UMTD_SIGNAL(0x4021, DTMF_Tone8slot),
    ADD_UMTD_SIGNAL(0x4022, DTMF_Tone9slot),
    ADD_UMTD_SIGNAL(0x4031, DTMF_Tone0slot),
    ADD_UMTD_SIGNAL(0x4003, DTMF_ToneAslot),
    ADD_UMTD_SIGNAL(0x4013, DTMF_ToneBslot),
    ADD_UMTD_SIGNAL(0x4023, DTMF_ToneCslot),
    ADD_UMTD_SIGNAL(0x4033, DTMF_ToneDslot),
    ADD_UMTD_SIGNAL(0x4030, DTMF_ToneStarslot),
    ADD_UMTD_SIGNAL(0x4032, DTMF_ToneGridslot),
};

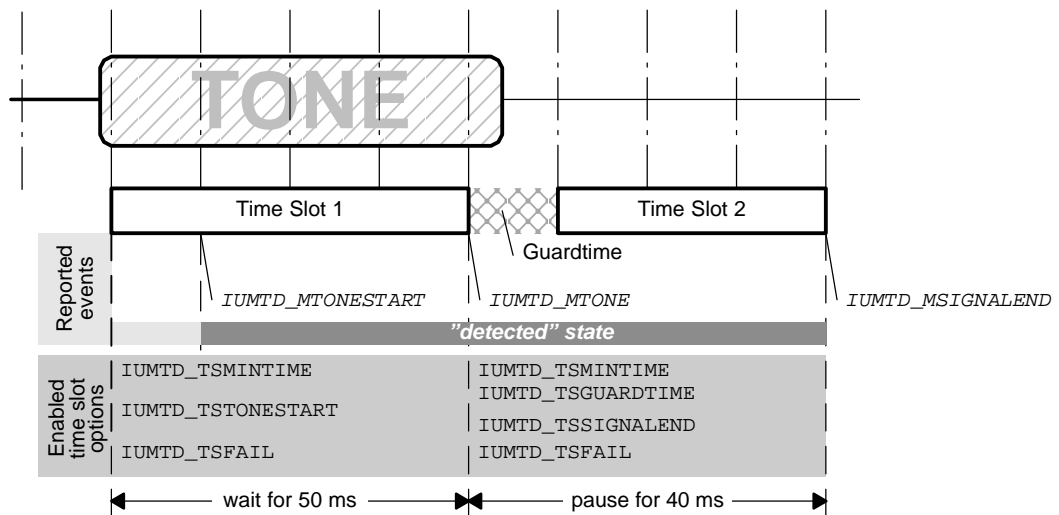
const IUMTD_SeriesParams DTMF_series =
{
    DTMF_frequencies,           // pFrequencyList
    80,                         // bandwidth
    {
        (XDAS_Int16)(0x8000*0.0150), // (-36.5 dBm) signalThreshold
        (XDAS_Int16)(0x8000*0.0080), // (-42 dBm) pauseThreshold
        (XDAS_Int16)(0x8000*0.2500), // (-12 dBm) pauseAdapt
        (XDAS_Int16)(0x8000*0.7000), // signalQuality
    }
};
```

```

        2, // interruptCount
(XDAS_Int16)(0x8000*0.4250), // (-7.4 dBc) spuriousLevel
(XDAS_Int16)(0x8000*0.8800), // (-1.11 dBc) spuriousCol "b"
(XDAS_Int16)(0x8000*0.3000), // (-10.5 dBc) spuriousRow "a"
(XDAS_Int16)(0x8000*0.9200), // (-0.72 dBc) spuriousCol2 "d"
(XDAS_Int16)(0x8000*0.3430), // (-9.3 dBc) spuriousRow2 "c"
(XDAS_Int16)(0x8000*0.2800), // (-11.1 dBc) dtmfTwist
(XDAS_Int16)(0x8000*0.707), // (-3 dBc) dtmf2harm
sizeof(DTMF_frequencies)/sizeof(XDAS_Int16), // nFrequencies
sizeof(DTMF_signals)/sizeof(IUMTD_Signal), // nSignals
DTMF_signals, // pSignalList
DTMF_PauseSlot // pCommonSlot
}
};

```

Figure 2-9. Messages Sequence for DTMF Detection (With Early Detection)



2.6.3 CPTD Settings for United Arab Emirates

2.6.3.1 Header File

```
#ifndef UMTD_CPTD_ARAB_H___
#define UMTD_CPTD_ARAB_H___ 1
#include "iumtd.h"
extern const IUMTD_SeriesParams CPTD_arab_series;
#endif //UMTD_CPTD_ARAB_H___
```

2.6.3.2 Source Signal

```
#include "CPTD_arab.h"
#define CPTD_ARAB_TONE_FLAGS      IUMTD_TSFAIL
#define CPTD_ARAB_PAUSE_FLAGS    IUMTD_TSSIGNALEND | IUMTD_TSFAIL | IUMTD_TSGUARDTIME
//-----
// Frequency planner . Frequencies are in Hz
//-----
static const XDAS_Int16 CPTD_arab_frequencies [] =
{
    300,    350,    375,    400,
    425,    440,    450
};
//-----
// Typical recognition parameters for UAE busy tone
//-----
static const IUMTD_TimeSlot arab_busySlots [] =
{ {{0x2955, 0x0592, 0x0000, 0x0000, 0x0000}, IUMTD_TSLIMITS |
  CPTD_ARAB_TONE_FLAGS, 300, 450}, // 400/425 Hz, 0.375 sec
  {{0x3333, 0x0333, 0x0000, 0x0000, 0x0000}, IUMTD_TSLIMITS |
  CPTD_ARAB_PAUSE_FLAGS, 300, 450} // .375 pause, indicate once
};
//-----
// Typical recognition parameters for UAE dial tone
//-----
static const IUMTD_TimeSlot arab_dialSlots [] =
{ {{0x5515, 0x0219, 0x0000, 0x0000, 0x0000}, IUMTD_TSLIMITS |
  CPTD_ARAB_TONE_FLAGS, 800, 10100}, // 350+440 Hz, 1.3 sec, indicate each time
  {{0x3333, 0x0333, 0x0000, 0x0000, 0x0000}, IUMTD_TSMINTIME |
  CPTD_ARAB_PAUSE_FLAGS, 50, 0x7fff} // pause,
```

```

};
//-----
// Typical recognition parameters for UAE ringback tone
//-----
static const IUMTD_TimeSlot arab_ringbackSlots [] =
{
  {{0x1955, 0x0129, 0x0000, 0x0000, 0x0000}, IUMTD_TSLIMITS |
  CPTD_ARAB_TONE_FLAGS, 350, 450}, // 400+450/425 Hz, 0.4 sec
  {{0x3333, 0x0333, 0x0000, 0x0000, 0x0000}, IUMTD_TSLIMITS |
  CPTD_ARAB_PAUSE_FLAGS, 145, 220}, // pause, 0.2 sec
  {{0x1955, 0x0129, 0x0000, 0x0000, 0x0000}, IUMTD_TSLIMITS |
  CPTD_ARAB_PAUSE_FLAGS, 350, 450}, // 400+450/425 Hz, 0.4 sec
  {{0x3333, 0x0333, 0x0000, 0x0000, 0x0000}, IUMTD_TSMINTIME |
  CPTD_ARAB_PAUSE_FLAGS, 1900, 2100}, // pause, 2.0 sec
};
//-----
// Recognized CPTD signals: busy, ringback, dial tone
//-----
static const IUMTD_Signal CPTD_arab_signals[] =
{
  AddSignal(0x2001, arab_busySlots),
  AddSignal(0x2002, arab_dialSlots),
  AddSignal(0x2003, arab_ringbackSlots),
};
const IUMTD_SeriesParams CPTD_arab_series =
{
  CPTD_arab_frequencies, /* pFrequencyList */
  20, /* bandwidth */
  {
    (XDAS_Int16)(0x8000*0.1189), /* (-18.5 dBm) signalThreshold */
    (XDAS_Int16)(0x8000*0.0631), /* (-18 dBm) pauseThreshold */
    (XDAS_Int16)(0x8000*0.2500), /* (-12 dBm) pauseAdapt */
    (XDAS_Int16)(0x8000*0.7000), /* signalQuality */
    0, /* interruptCount */
    (XDAS_Int16)(0x8000*0.5000), /* (-6 dBc) spuriousLevel */
    0, /* ( dBc) spuriousCol */
    0, /* ( dBc) spuriousRow */
    0, /* ( dBc) spuriousCol2 */
  }
};

```

```
0,                /* ( dBc) spuriousRow2 */
0,                /* ( dBc) dtmfTwist */
0,                /* ( dBc) dtmf2harm */
sizeof(CPTD_arab_frequencies)/sizeof(XDAS_Int16), /* nFrequencies */
sizeof(CPTD_arab_signals)/sizeof(IUMTD_Signal), /* nSignals */
(IUMTD_Signal*)CPTD_arab_signals, /* pSignalList */
NULL
}
};
```


Universal Multifrequency Tone Detector (UMTD) API Descriptions

This chapter provides the user with a clear understanding of Universal Multifrequency Tone Detector (UMTD) algorithms and their implementation with the TMS320 DSP Algorithm Standard interface (XDAIS).

| Topic | Page |
|--|-------------|
| 3.1 Standard Interface Structures | 3-2 |
| 3.2 Standard Interface Functions | 3-3 |
| 3.3 Vendor-Specific Interface Functions | 3-5 |

3.1 Standard Interface Structures

In this section, Standard interface structures for the UMTD are described.

Table 3-1 lists the UMTD Detector Real-time Status parameters.

3.1.1 Instance Creation Parameters

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

Structure Definition Use structure IUMTD_Params (see Table 2-8) to provide each instance with parameters.

Type IUMTD_Params is defined in "iUMTD.h".

3.1.2 Status Structure

Description This structure defines the status parameters for the algorithm. Detector status structure is used for control purposes. Status can be received by function UMTD_getStatus().

Structure Definition

Table 3-1. UMTD Detector Real-Time Status Parameters

```
typedef struct IUMTD_Status
{
```

| Status Type | Status Name | Description |
|-------------|--------------|--|
| XDAS_Int16 | nFrequencies | Length of array pMagnitudes specified below |
| XDAS_Int16* | pMagnitudes | Pointer to the array of signal magnitudes at given frequencies |

```
}
IUMTD_Status;
```

Type IUMTD_Status defined in "iUMTD.h".

3.2 Standard Interface Functions

The following functions are all required when using the UMTD algorithm.

Table 3-2 summarizes standard interface functions of UMTD detector API.

UMTD_apply() and UMTD_control() are optional, but neither are supported by Spirit Corp.

Table 3-2. UMTD Detector Standard Interface Functions

| Functions | Description | See Page... |
|-------------|--------------------------|-------------|
| UMTD_init | Algorithm initialization | 3-3 |
| UMTD_exit | Algorithm deletion | 3-3 |
| UMTD_create | Instance creation | 3-4 |
| UMTD_delete | Instance deletion | 3-4 |

3.2.1 Algorithm Initialization

UMTD_init *Calls the framework initialization function to initialize an algorithm*

Description

This function calls the framework initialization function, ALG_init(), to initialize the algorithm. For UMTD detector, 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 UMTD_init ()
```

Arguments

none

Return Value

none

3.2.2 Algorithm Deletion

UMTD_exit *Calls the framework exit function to remove an algorithm instance*

Description

This function calls the framework exit function, ALG_exit(), to remove an instance of the algorithm. For UMTD detector, 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 UMTD_exit ()
```

Arguments

none

Return Value

none

3.2.3 Instance Creation

UMTD_create *Calls the framework create function to create an instance object*

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

Function Prototype

```
UMTD_Handle UMTD_create
    (const IUMTD_Fxns *fxns,
     const UMTD_Params *prms);
```

Arguments

`IUMTD_Fxns *` Pointer to vendor's functions (Implementation ID). Use reference to `UMTD_SPCORP_IUMTD` virtual table.

`UMTD_Params *` Pointer to Parameter Structure. Use `NULL` pointer to load default parameters.

Return Value `UMTD_Handle` is defined in file "UMTD.h". This is a pointer to the created instance.

3.2.4 Instance Deletion

UMTD_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 tasks.

Function Prototype

```
void UMTD_delete (UMTD_Handle handle)
```

Arguments `UMTD_Handle` Instance's handle obtained from `UMTD_create()`

Return Value none

3.3 Vendor-Specific Interface Functions

In this section, functions in the SPIRIT's algorithm implementation and interface (extended IALG methods) are described.

Table 3-3 summarizes SPIRIT's API functions of the UMTD detector.

The whole interface is placed in the header files `iUMTD.h`, `UMTD.h`, `UMTD_spcorp.h`.

Table 3-3. Detector-Specific Interface Functions

| Functions | Description | See Page... |
|-----------------------------|---|-------------|
| <code>UMTD_detect</code> | Sends samples to the detector and process detection | 3-5 |
| <code>UMTD_reset</code> | Resets actual detector status for all signals | 3-6 |
| <code>UMTD_getStatus</code> | Returns current detector status | 3-6 |

3.3.1 Process Detection

| | | | | | | | |
|---------------------------|--|---------------------|--------------------------|-----------------|---|--------------------|-----------------------------------|
| UMTD_detect | <i>Returns valid call progress tones or special notification messages</i> | | | | | | |
| Description | Returns valid call progress tones or special notification messages. | | | | | | |
| Function Prototype | <pre>IUMTD_Message UMTD_detect (UMTD_Handle handle, const XDAS_Int16 in[], XDAS_Int16 count)</pre> | | | | | | |
| Arguments | <table> <tr> <td><code>handle</code></td> <td>Pointer to UMTD instance</td> </tr> <tr> <td><code>in</code></td> <td>Array of input samples at sample rate 8 kHz</td> </tr> <tr> <td><code>count</code></td> <td>Number of samples to be processed</td> </tr> </table> | <code>handle</code> | Pointer to UMTD instance | <code>in</code> | Array of input samples at sample rate 8 kHz | <code>count</code> | Number of samples to be processed |
| <code>handle</code> | Pointer to UMTD instance | | | | | | |
| <code>in</code> | Array of input samples at sample rate 8 kHz | | | | | | |
| <code>count</code> | Number of samples to be processed | | | | | | |
| Return Value | <p>Returns bit mask with messages from UMTD when the host is not attached to the detector on UMTD creation (field <code>mpHost</code> in <code>IUMTD_Params</code> is <code>NULL</code>, see Table 2-5).</p> <p>When no message is available or message is processed by the host, field <code>mType</code> of <code>IUMTD_Message</code> is set to <code>IUMTD_MNONE</code>.</p> | | | | | | |
| Restrictions | none | | | | | | |

3.3.2 Reset Detector Status

UMTD_reset *Resets the current detector status for all signals*

Description Resets the current detector status for all signals.

Function Prototype `Void UMTD_reset
(UMTD_Handle handle)`

Arguments `handle` Pointer to UMTD instance

Return Value none

Restrictions none

3.3.3 Get Actual Detector Status

UMTD_getStatus *Returns the current detector status*

Description Returns current detector status. Just copies internal state variables into status structure. Can be used to analyze magnitudes of all frequency components.

Function Prototype `Void UMTD_getStatus
(UMTD_Handle handle, XDAS_Int16 index, IUMTD_Statu-
s* pStatus)`

Arguments `Handle` Pointer to UMTD instance
`Index` Series index
`PStatus` Pointer to the status structure to be read

Return Value Actual detector status (see Table 3-1).

Restrictions none

Test Environment



Note: Test Environment Location

This chapter describes test environment for the UMTD object.

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

| Topic | Page |
|--|-------------|
| A.1 Description of Directory Tree | A-2 |

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 UMTD

| 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.

A

- ALG, interface 1-4
- ALG_activate 1-4
- ALG_control 1-5
- ALG_create 1-4
- ALG_deactivate 1-5
- ALG_delete 1-4
- ALG_exit 1-5
- ALG_init 1-5
- Algorithm Deletion 3-3
- Algorithm Initialization 3-3
- Amplitude Discrimination 2-8
- Amplitude Discrimination Options 2-22
- Amplitude Discriminator Options 2-20
- Application Development 1-6
 - steps to creating an application 1-8
- Application/Framework 1-4

B

- Bit Field, positions and flags table 2-20

C

- CMS Recognition Parameters 2-19
- Common UMTD Parameters 2-18
- CPTD Settings for United Arab Emirates 2-34
- CPTD Settings for USA's PSTN, example of 2-27

D

- Detector Messages 2-25
- Directory Tree A-2

- DTMF Detection, Message sequence
 - illustration 2-33
- DTMF Settings 2-29
- DTMF Signal Mask for IUMTD_TSUSEDTMFMASK Option 2-21

E

- Environment, for testing A-2
- Examples 2-27
 - CPTD Settings for United Arab Emirates 2-34
 - DTMF Settings 2-29
 - Typical CPTD Settings for USA's PSTN 2-27

F

- Flow Control Options 2-12, 2-24
- Formulas, for Bit fields 2-20
- Framework 1-4
- frequency planner, formula for 2-20
- Frequency Selection 2-7
- Functions
 - standard 3-3
 - vendor-specific 3-5

G

- Get Actual Detector Status 3-6

H

- Header File
 - CPTD Settings for United Arab Emirates 2-34
 - DTMF Settings 2-29
 - for CPTD settings for USA's PSTN 2-27
- Header file
 - for abstract interfaces 1-6

- for concrete interfaces 1-5
- Host Controller 2-26
- Host Indication Example 2-14
- Host Indication Options 2-13, 2-24

I

- IALG 1-6
- Instance Creation 3-4
- Instance Creation Parameters 3-2
- Instance Deletion 3-4
- Interface 1-5
 - abstract 1-6
 - concrete 1-5
 - vendor implementation 1-6

M

- Message Structure 2-26
- Messages and Host Interface 2-25
 - Host Controller 2-26
 - Message structure 2-26
 - UMTD Detector Messages Summary 2-25
- Module Instance Lifetime. See Application Development

O

- Options
 - Amplitude Discrimination 2-22
 - amplitude discriminator 2-20
 - DTMF Signal Mask for IUMTD_TSUSEDTMFMASK 2-21
 - flow control 2-12
 - host indication 2-13
 - time selection 2-23

P

- Parameter, definitions 2-15
- Parameter Definition 2-15
 - amplitude discriminator options 2-20
 - Time Slot Flags 2-23
- Parameters
 - CMS Recognition 2-19

- Common UMTD 2-18
 - for Recognized Signal Series 2-17
 - Pointers to Series 2-18
 - recognized signal series 2-15
 - UMTD Time Slots 2-19

- Pointers to Series Parameters 2-18
- Process Detection 3-5

R

- Recognition Flags, for Bit Field Positions 2-20
- Recognized Signal Series 2-17
- Reset Detector Status 3-6

S

- Signal recognition 2-6
 - amplitude discrimination 2-8
 - flow control options 2-12
 - frequency selection 2-7
 - general 2-6
 - host indication 2-13
 - time selection 2-9
- signals, composite, illustration of 2-2
- Source File
 - DTMF Settings 2-29
 - for CPTD settings for USA's PSTN 2-27
- Source file
 - for abstract interfaces 1-6
 - for concrete interfaces 1-5
- Source Signal, CPTD Settings for United Arab Emirates 2-34
- Status Structure 3-2
- Structures, standard 3-2

T

- Test
 - files A-2
 - project A-3
- Test Environment A-2
- Time Selection 2-9
- Time Selection Options 2-23
- Time Slot Flags 2-23
 - flow control options 2-24
 - time selection 2-23

U

UMTD. See Universal Multifrequency Tone Detector
UMTD Detector Messages 2-25
UMTD Detector Real-Time Status Parameters 3-2
UMTD Thresholds 2-17
UMTD Time Slots 2-19
UMTD_apply 3-3
UMTD_control 3-3
UMTD_create 3-4
UMTD_delete 3-4
UMTD_detect 3-5
UMTD_exit 3-3
UMTD_getStatus 3-6
UMTD_init 3-3
UMTD_reset 3-6

Universal Multifrequency Tone Detector
Examples 2-27
Messages and Host Interface 2-25
overview 2-2
Parameter Definition 2-15
signal recognition 2-6
steps to integrating into a framework 2-4

X

XDAIS
Application Development 1-6
Application/Framework 1-4
basics 1-4
Interface 1-5
related documentaion 1-3
System Layers, illustration of 1-4