Automatic Gain Control (AGC) Algorithm User’s Guide

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

Literature Number: SPRU631
March 2003
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

Copyright © 2003, Texas Instruments Incorporated
About This Manual

This user’s guide describes algorithms designed for Automatic Gain Control (AGC) for Voice Applications developed by SPIRIT™ Corp for the TMS320C54x platform. It provides instructions for implementation and integration with embedded voice modems, payphones, answering machines, and other telephone equipment.

The following abbreviations are used in this document:

- AGC: Automatic Gain Control
- PSTN: Public Switched Telephone Network
- VAD: Voice Activity Detector

How to Use This Manual

The contents of the Automatic Gain Control (AGC) Algorithm User’s Guide are as follows:

- Chapter 1, Introduction to AGC Algorithms, is a brief overview of the AGC and how it is used with the TMS320C54x platform. It also provides an overview of the TMS320 DSP Algorithm Standard (also known as XDAIS).
- Chapter 2, AGC Integration, provides descriptions, diagrams, and examples explaining the integration of the AGC with frameworks.
- Chapters 3, AGC API Descriptions, provides the user with an understanding of AGC algorithms and their implementation with XDAIS.
- Appendix A, Test Environment, describes the test environment for the AGC object.
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

Voice Activity Detector (VAD) Algorithm User’s Guide (SPRU635)

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
- DSP Solutions: http://www.ti.com/dsp
- 320 Hotline On-line™: http://www.ti.com/sc/docs/dsp/support.htm

North America, South America, Central America
- Product Information Center (PIC): (972) 644-5580
- Ti Literature Response Center: (800) 477-8924
- Software Registration/Upgrades: (972) 293-5050
- U.S.A. Factory Repair/Hardware Upgrades: (281) 274-2285
- U.S. Technical Training Organization: (972) 644-5580
- Microcontroller Hotline: (281) 274-2370
- Microcontroller Modem BBS: (281) 274-3700
- 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
  - Email: epic@ti.com
- Deutsch: +49 8161 80 33 11
- English: +33 1 30 70 11 68
- 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
- Hong Kong DSP Hotline: +852 2 956 7268
- Korea DSP Hotline: +82 2 551 2804
- Korea DSP Modem BBS: +82 2 551 2914
- Singapore DSP Hotline: +65 390 7179
- Taiwan DSP Hotline: +886 2 377 1450
- 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)
- DSP Hotline: +03-3457-0972 or (INTL) 813-3457-0972
- DSP BBS via Nifty-Serve: Type “Go TIASP”

Read This First
If You Need Assistance

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 AGC Algorithms ................................. 1-1
   This chapter is a brief introduction to AGC algorithms and their use with the TMS320C5400 platform. Also provided is a description of basic terms and examples for XDAIS.
   1.1 Introduction .......................................................... 1-2
   1.2 XDAIS Basics ......................................................... 1-3
       1.2.1 Application/Framework ....................................... 1-3
       1.2.2 Interface ....................................................... 1-4
       1.2.3 Application Development .................................... 1-5

2 AGC Integration ....................................................... 2-1
   This chapter provides descriptions, diagrams, and examples explaining the integration of the AGC with frameworks.
   2.1 Overview ............................................................. 2-2
   2.2 Integration Flow ..................................................... 2-3
   2.3 Example of a Call Sequence ....................................... 2-4

3 Automatic Gain Control (AGC) API Descriptions ................. 3-1
   This chapter provides the user with a clear understanding of AGC algorithms and their implementation with XDAIS.
   3.1 Standard Interface Structures ................................... 3-2
   3.2 Standard Interface Functions ..................................... 3-4
   3.3 Vendor-Specific Interface Functions ............................. 3-6

A Test Environment ..................................................... A-1
   A.1 Description of Directory Tree ................................... A-2
       A.1.1 Test Vectors Format ......................................... A-2
       A.1.2 Test Project ................................................ A-3
This chapter is a brief introduction to Automatic Gain Control (AGC) algorithms and their 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.
1.1 Introduction

This document describes implementation of Automatic Gain Control for Voice Applications developed by SPIRIT™ Corp. for the TMS320C5400 platform and is intended for integration with embedded voice modems, payphones, answering machines, and other telephone equipment.

SPIRIT AGC is designed specifically to amplify voice signal, which contains a non-stationary amplitude envelope. The AGC operates more efficiently in conjunction with the voice activity detector (VAD). The VAD algorithm alerts the AGC when there is no speech in the signal. This prevents the AGC from adaptation during these periods. For more information regarding the VAD, please refer to the Voice Activity Detector (VAD) Algorithm User’s Guide (SPRU635).

The SPIRIT AGC software is a fully expressDSP-compliant reentrant code. The AGC interface complies with the TMS320DSP Algorithm Standard (also known as XDAIS) and can be used in multitasking environments.

The TMS320 DSP Algorithm Standard 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.

AGC can be efficiently used in conjunction with following products of Spirit™ Corp.:

- Voice Activity Detector
- Vocoders (G.721.3, G.729, G.726, G.711, proprietary low bit rate)

Also, it can be easily supported by third-party software.
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
### 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.
AGC Integration

This chapter provides a brief overview and instructions for integrating the AGC into a specified framework.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Overview</td>
<td>2-2</td>
</tr>
<tr>
<td>2.2 Integration Flow</td>
<td>2-3</td>
</tr>
<tr>
<td>2.3 Example of a Call Sequence</td>
<td>2-4</td>
</tr>
</tbody>
</table>
2.1 Overview

Figure 2-1 shows a typical AGC integration diagram.

Figure 2-1. AGC Integration Diagram

The AGC object performs the following operations:

- Amplifies or attenuates input samples and adapts dynamic ranges of signals;
- Adapts gain coefficient according to input samples: To prevent adaptation on noise (silence) frames AGC must be used with voice activity detector (VAD), as shown in Figure 2-2.

Figure 2-2. AGC Integration With VAD
2.2 Integration Flow

In order to integrate the AGC control into a framework you must follow these steps:

**Step 1:** Create `AGC_Params` structure and initialize it with required values.

**Step 2:** Call `AGC_create()` to create the instance of control. There is no restriction on the maximum number of instances of the control to be created.

**Step 3:** Pass a stream with input samples (8 kHz, 16 bits) to `AGC_process()` routine.

**Step 4:** Delete the instance of the control by using `AGC_delete()`.

These steps are illustrated in Figure 2-3.

*Figure 2-3. Typical Control Integration Flow*

An example of this call sequence has been provided in section 2.3.
The example below demonstrates a typical call sequence for AGC. A similar example can be found in Src\FlexExamples\StandaloneXDas\AGC.

```c
#include <stdio.h>
#include "agc.h"
#include "iagc.h"
#include "agc_spCorp.h"

/* file I/O functions */
int fread16(const XDAS_Int16 *pData, size_t size, FILE *pFile);
int fwrite16(const XDAS_Int16 *pData, size_t size, FILE *pFile);

void main()
{
#define INPUT_FRAME 120
FILE *inFile, *outFile;
XDAS_Int16 pFrame[INPUT_FRAME];
XDAS_Int16 size;
XDAS_Int16 i;

AGC_Handle AGCInst;
AGC_Status AGCStatus;

printf("SPIRIT AGC\ncprocessing in progress...\n");

/* open input and out files */
inFile  = fopen("AGC_demo.pcm", "rb");
outFile = fopen("AGC_demo_out.pcm", "wb");
if(!inFile || !outFile)
{
    printf("file open error!\n");
    return;
}

/* creating AGC instance with default parameters */
AGCInst = AGC_create(&AGC_SPCORP_IAGC, NULL);

/* circle AGC processing */
while((size=fread16(pFrame, INPUT_FRAME, inFile)) == INPUT_FRAME)
{
    /* processing received samples */
    AGC_process(AGCInst, pFrame, INPUT_FRAME, TRUE);

    /* processing received samples */
    AGCStatus = AGC_getStatus(AGCInst);

    /* do something */
    fwrite16(pFrame, INPUT_FRAME, outFile);
}
}
```

2.3 Example of a Call Sequence
/* Deleting AGC instance */
AGC_delete(AGCInst);
if(size) fwrite16(pFrame, size, outFile);
printf ("\n...Finish\n");
/* closing all opening files */
close(inFile);
close(outFile);
}
This chapter provides the user with a clear understanding of Automatic Gain Control (AGC) algorithms.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Standard Interface Structures</td>
<td>3-2</td>
</tr>
<tr>
<td>3.2 Standard Interface Functions</td>
<td>3-4</td>
</tr>
<tr>
<td>3.3 Vendor-Specific Interface Functions</td>
<td>3-6</td>
</tr>
</tbody>
</table>
3.1 Standard Interface Structures

In this section, parameter structures are described. Table 3-1 summarizes the standard interface structures of AGC API.

Table 3-1. AGC Standard Interface Structures

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Located in Table...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface Creation</td>
<td>Table 3-2</td>
</tr>
<tr>
<td>Real-Time Status</td>
<td>Table 3-3</td>
</tr>
</tbody>
</table>

Instance Creation Parameters

Description

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

Structure Definition

Use structure IAGC_Params to provide each instance with parameters.

Table 3-2. Interface Creation Parameters

typedef struct IAGC_Params {

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDAS_Int16</td>
<td>speedGrow</td>
<td>Adaptation speed coefficient when gain is increasing</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>speedFall</td>
<td>Adaptation speed coefficient when gain is decreasing</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>reference</td>
<td>Output reference amplitude</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>maxOutputValue</td>
<td>Max value of output sample that AGC will output</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>initGainCoef</td>
<td>High 16 bit of GainCoef</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>safeOutputValue</td>
<td>Safe value of output sample. If output value will be higher than this threshold, AGC will start decreasing its gain faster than usual. This value is convenient to use to avoid overamplification effects on some wide-range signals, such as speech.</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>gainMax</td>
<td>Maximal gain value</td>
</tr>
<tr>
<td>XDAS_Int16</td>
<td>gainMin</td>
<td>Minimal gain value</td>
</tr>
</tbody>
</table>

} IAGC_Params
Standard Interface Structures

**Type**

IAGC_Params is defined in "iAGC.h".

<table>
<thead>
<tr>
<th>Gain Parameter Value</th>
<th>Gain Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0001</td>
<td>1/256</td>
</tr>
<tr>
<td>0x0002</td>
<td>1/128</td>
</tr>
<tr>
<td>0x0004</td>
<td>1/64</td>
</tr>
<tr>
<td>0x0008</td>
<td>1/32</td>
</tr>
<tr>
<td>0x0010</td>
<td>1/16</td>
</tr>
<tr>
<td>0x0020</td>
<td>1/8</td>
</tr>
<tr>
<td>0x0040</td>
<td>1/4</td>
</tr>
<tr>
<td>0x0080</td>
<td>1/2</td>
</tr>
<tr>
<td>0x0100</td>
<td>1</td>
</tr>
<tr>
<td>0x0200</td>
<td>2</td>
</tr>
<tr>
<td>0x0400</td>
<td>4</td>
</tr>
<tr>
<td>0x0800</td>
<td>8</td>
</tr>
<tr>
<td>0x1000</td>
<td>16</td>
</tr>
<tr>
<td>0x2000</td>
<td>32</td>
</tr>
<tr>
<td>0x4000</td>
<td>64</td>
</tr>
</tbody>
</table>

**Status Structure**

**Description**

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

**Structure Definition**

*Table 3-3. AGC Real-Time Status Parameters*

typedef struct IAGC_Status {

    XDAS_Int16 gainCoefHigh
        High 16 bit of GainCoef
    XDAS_Int16 isClipping
        True if signal was clipped

} IAGC_Status;

**Type**

IAGC_Status defined in "iAGC.h".
AGC_init

3.2 Standard Interface Functions

The following functions are required when using the algorithm AGC. AGC_apply() and AGC_control() are optional, but neither are supported by Spirit Corp.

Table 3-4 summarizes standard Interface functions of AGC API.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGC_init</td>
<td>Calls the framework initialization function (Algorithm initialization)</td>
<td>3-4</td>
</tr>
<tr>
<td>AGC_exit</td>
<td>Calls the framework exit function (Algorithm deletion)</td>
<td>3-4</td>
</tr>
<tr>
<td>AGC_create</td>
<td>Calls the framework creation function (Instance creation)</td>
<td>3-5</td>
</tr>
<tr>
<td>AGC_delete</td>
<td>Calls the framework deletion function (Instance deletion)</td>
<td>3-5</td>
</tr>
</tbody>
</table>

Algorithm Initialization

**AGC_init**  
*Calls the framework initialization function*

Function AGC_init

Description This function calls the framework initialization function, ALG_init(), to initialize the algorithm. For AGC object, this function does nothing. It can be skipped and removed from the target code.

Function Prototype void AGC_init();

Arguments none

Return Value none

Algorithm Deletion

**AGC_exit**  
*Calls the framework exit function*

Function AGC_exit

Description This function calls the framework exit function, ALG_exit(), to remove the algorithm. For AGC object, this function does nothing. It can be skipped and removed from the target code.

Function Prototype void AGC_exit();

Arguments none

Return Value none
Instance Creation

**AGC_create**  
*Calls the framework creation function*

<table>
<thead>
<tr>
<th>Function</th>
<th>AGC_create</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>In order to create a new AGC control object, AGC_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 AGC_SPCORP_IAGC contains AGC virtual table supplied by SPIRIT Corp.</td>
</tr>
</tbody>
</table>
| Function Prototype | AGC_Handle AGC_create(  
|                  |   const IAGC_Fxns *fxns,  
|                  |   const AGC_Params *prms  
|                  | )  |
| Arguments     | IAGC_Fxns * Pointer to vendor’s functions (Implementation ID). Use reference to AGC_SPCORP_IAGC virtual table supplied by SPIRIT Corp.  
|               | AGC_Params * Pointer to Parameter Structure. Use NULL pointer to load default parameters.  |
| Return Value  | AGC_Handle defined in file “AGC.h”. This is a pointer to the created instance.  |

Instance Deletion

**AGC_delete**  
*Calls the framework deletion function*

<table>
<thead>
<tr>
<th>Function</th>
<th>AGC_delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This function calls the framework delete function, ALG_delete(), to delete the instance object and perform memory de-allocation tasks.</td>
</tr>
<tr>
<td>Function Prototype</td>
<td>void AGC_delete (AGC_Handle handle);</td>
</tr>
<tr>
<td>Arguments</td>
<td>AGC_Handle Instance’s handle obtained from AGC_create()</td>
</tr>
<tr>
<td>Return Value</td>
<td>none</td>
</tr>
</tbody>
</table>
3.3 Vendor-Specific Interface Functions

The following are vendor-specific functions in the SPIRIT's algorithm implementation and interface (extended IALG methods) API.

Table 3-5 summarizes SPIRIT's API functions of AGC control.

The whole interface is placed in header files iAGC.h, AGC.h, AGC_spcorp.h.

Table 3-5. AGC-Specific Interface Functions

<table>
<thead>
<tr>
<th>Functions</th>
<th>Description</th>
<th>See Page...</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGC_reInit</td>
<td>Reinitializes an object with specified parameters</td>
<td>3-6</td>
</tr>
<tr>
<td>AGC_setParameters</td>
<td>Allows the change of an actual loop gain and reference amplitude</td>
<td>3-7</td>
</tr>
<tr>
<td>AGC_process</td>
<td>Processes input samples through attenuation, amplification, etc.</td>
<td>3-7</td>
</tr>
<tr>
<td>AGC_getStatus</td>
<td>Returns the current AGC status</td>
<td>3-8</td>
</tr>
</tbody>
</table>

Reinitialization

**AGC_reInit**

*Reinitializes an object with specified parameters*

**Function**

AGC_reInit

**Description**

Performs reinitialization of an object with specified parameters and set an AGC object to its initial state.

**Function Prototype**

```c
XDAS_Void AGC_reInit(
    AGC_Handle handle,
    const AGC_Params *params
);
```

**Arguments**

- handle: Pointer to AGC instance
- params: Pointer to structure with parameters (see Table 3-2)

**Return Value**

none

**Restrictions**

none
AGC_process

Changing AGC parameters

**AGC_setParameters**

*Allows the change of an actual loop gain and reference amplitude*

**Function**

AGC_setParameters

**Description**

Allows a change of an actual loop gain \(\text{speedGrow}, \text{speedFall}\) and reference amplitude \(\text{reference}\) for AGC. Current state of AGC object remains unchanged unlike function \(\text{AGC_reInit()}\).

Parameters \(\text{maxOutputValue}, \text{initGainCoef}\) in \(\text{params}\) are ignored.

**Function Prototype**

\[
\text{XDAS_Void AGC_setParameters(}
\text{  AGC_Handle handle,}
\text{  const AGC_Params *params}
\text{);}
\]

**Arguments**

- **handle** Pointer to AGC instance
- **params** Pointer to structure with parameters (see Table 3-2)

**Return Value**

none

**Restrictions**

Parameters \(\text{maxOutputValue}, \text{initGainCoef}\) in \(\text{params}\) can not be set and shall be updated by using \(\text{AGC_reInit()}\) function.

Processing input samples

**AGC_process**

*Processes input samples through attenuation, amplification, etc.*

**Function**

AGC_process

**Description**

Processes samples by the following way:

- Attenuates or amplifies signal;
- Validates output magnitude and clips signal when its absolute value is greater then \(\text{maxOutputValue}\);
- Estimates power of signal and adapts loop gain according to \(\text{speedGrow}, \text{speedFall}\);
- Changes loop gain so overall signal magnitude will be approximately equal to output reference amplitude \(\text{reference}\), see Table 3-2. Adaptation can be disabled by clearing parameter \(\text{isAdapting}\) (see function parameters below).
AGC_getStatus

Function Prototype

```c
XDAS_Void AGC_process(
    AGC_Handle handle,
    XDAS_Int16 *pBuffer,
    XDAS_Int16 size,
    XDAS_Bool isAdapting
);
```

Arguments

- **handle**: Pointer to AGC instance
- **pBuffer**: Pointer to buffer with samples to be processed. Output samples in `pBuffer` replaces input samples.
- **size**: Size of input buffer
- **isAdapting**: Determines if adaptation will be performed. When false, adaptation is disabled and loop gain remains unchanged.

Return Value

none

Restrictions

none

Getting Actual AGC Status

AGC_getStatus

*Returns the current AGC status*

Function

AGC_getStatus

Description

Returns the current AGC status.

Function Prototype

```c
IAGC_Status AGC_getStatus(
    AGC_Handle handle
);
```

Arguments

- **handle**: Pointer to AGC instance

Return Value

Actual detector status (see Table 3-3)

Restrictions

none
Note: Test Environment Location

This chapter describes test environment for the AGC object.

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

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.c</td>
<td>Test file</td>
</tr>
<tr>
<td>FileC5x.c</td>
<td>File input/output functions</td>
</tr>
<tr>
<td>..\ROM\CSTRom.s54</td>
<td>ROM entry address</td>
</tr>
<tr>
<td>Test.cmd</td>
<td>Linker command file</td>
</tr>
<tr>
<td>Vectors\output.pcm</td>
<td>Reference output test vectors</td>
</tr>
</tbody>
</table>

A.1.1 Test Vectors Format

All test vectors are raw PCM files with the following parameters:

- **Bits per sample**: 16, Mono  
- **Word format**: Intel PCM (LSB goes first)  
- **Encoding**: uniform  
- **Level**: -2 dBFS
A.1.2 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

A
AGC
   See also Automatic Gain Control (AGC)
   Call sequence example 2-4
AGC_create 2-3, 3-5
AGC_delete 2-3, 3-5
AGC_exit 3-4
AGC_getStatus 3-8
AGC_init 3-4
AGC_PARAMS 2-3
AGC_process 2-3, 3-7
AGC_reInit 3-6
AGC_setParameters 3-7
ALG, interface 1-3
ALG_activate 1-3
ALG_control 1-4
ALG_create 1-3
ALG_deactivate 1-4
ALG_delete 1-3
ALG_exit 1-4
ALG_init 1-4
Algorithm Deletion 3-4
Algorithm Initialization 3-4
Application Development 1-5
   steps to creating an application 1-7
Application/Framework 1-3
Automatic Gain Control (AGC), introduction 1-2

C
Call Sequence, example of 2-4
Changing AGC parameters 3-7

D
Directory Tree A-2

E
Environment, for testing A-2

F
Framework 1-3
   functions
      standard 3-4
      vendor-specific 3-6

G
Getting Actual AGC Status 3-8

H
Header file
   for abstract interfaces 1-5
   for concrete interfaces 1-4

I
IALG 1-5
   Instance Creation 3-5
   Instance Creation Parameters 3-2
   Instance Deletion 3-5
   Integration, with frameworks 2-2
   Integration Flow, steps for AGC integration 2-3
   Interface 1-4
      abstract 1-5
      concrete 1-4
      vendor implementation 1-5

Index-1
M
Module Instance Lifetime. See Application Development

P
Parameters
  instance creation 3-2
  Real-time status 3-3
  Processing input samples 3-7

R
Reinitialization 3-6

S
Source file
  for abstract interfaces 1-5
  for concrete interfaces 1-4
Status Structure 3-3
Structures, standard 3-2
Instance Creation Parameters 3-2

T
Test
  files A-2
  format A-2
  project A-3
Test Environment A-2

V
VAD 2-2
  See also Voice Activity Detector
Voice Activity Detector 1-2

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