Automatic Gain Control (AGC) Algorithm User's Guide

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

Literature Number: SPRU631
March 2003

TEXAS INSTRUMENTS

Preface

Read This First

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

Software Copyright

If You Need Assistance . . .

❑ World-Wide Web Sites

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

❑ North America, South America, Central America

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

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

Networking Hotline		Fax: (281) 274-4027 Email: TLANHOT@micro.ti.com

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

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

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

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

❑ **Documentation**

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

Mail:	Texas Instruments Incorporated Technical Documentation Services, MS 702 P.O. Box 1443 Houston, Texas 77251-1443	Email: dsph@ti.com Email: micro@ti.com

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

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

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

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

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

Contents

Figures

Tables

Notes, Cautions, and Warnings

Chapter 1

Introduction to AGC Algorithms

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.

Topic **Page**

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.

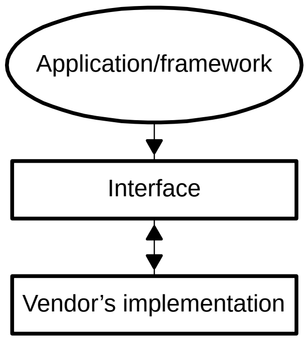
1.2 XDAIS Basics

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:

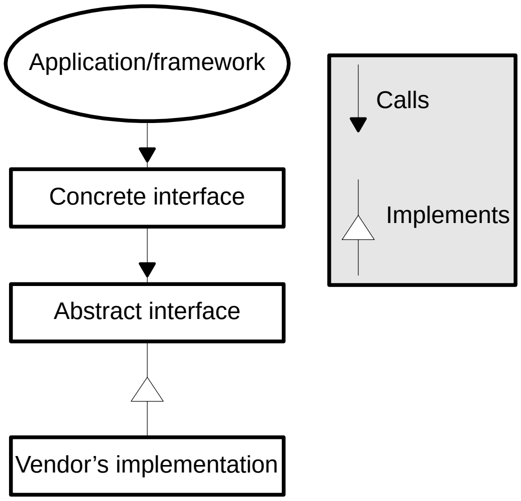
- ❑ `ALG_create` - for memory allocation/algorithm instance creation
- ❑ `ALG_delete` - for memory de-allocation/algorithm instance deletion
- ❑ `ALG_activate` - for algorithm instance activation

- ❑ `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

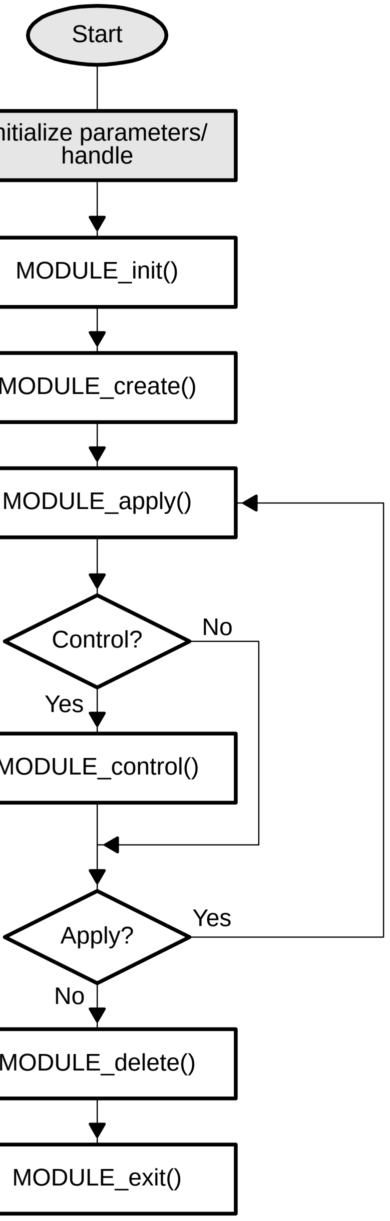
Vendor implementation refers to the set of functions implemented by the algorithm vendor to match the interface. These include the core processing functions required by the algorithm and some control-type functions required. A table is built with pointers to all of these functions, and this table is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

Chapter 2

AGC Integration

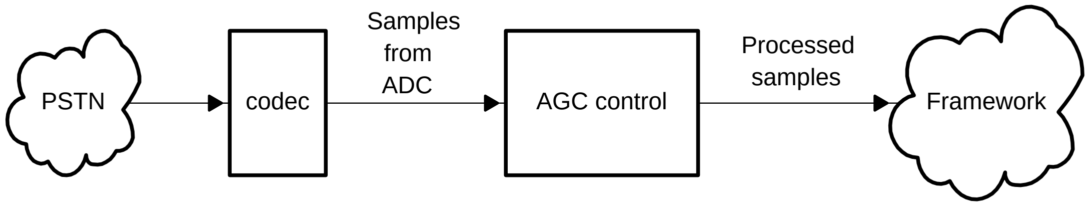
This chapter provides a brief overview and instructions for integrating the AGC into a specified framework.

Topic **Page**

2.1 Overview

Figure 2-1 shows a typical AGC integration diagram.

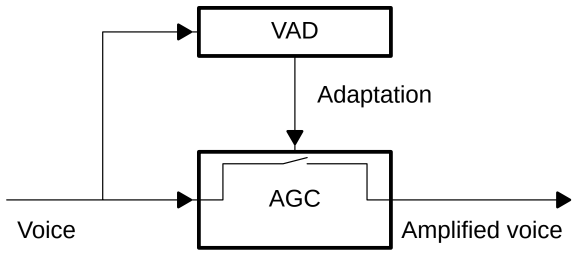
Figure 2-1. AGC Integration Diagram



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.

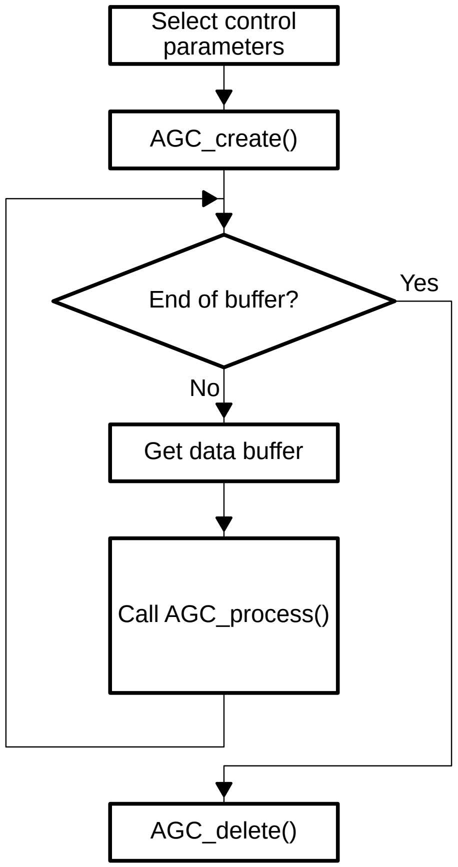
Step 2: Call `AGC_create()` to create the instance of control. There is no restriction on the maximum number of instances of the control to be created.

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.

2.3 Example of a Call Sequence

The example below demonstrates a typical call sequence for AGC. A similar example can be found in `Src\FlexExamples\StandaloneXDAS\AGC`.

```
#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\nprocessing 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);
  }/* while */
  /* Deleting AGC instance */
  AGC_delete(AGCInst);

  if(size) fwrite16(pFrame, size, outFile);

  printf (”\n...Finish\n”);

  /* closing all opening files */
  fclose(inFile);
  fclose(outFile);
}
```

Chapter 3

Automatic Gain Control (AGC) API Descriptions

This chapter provides the user with a clear understanding of Automatic Gain Control (AGC) algorithms.

Topic **Page**

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

Parameters	Located in Table...
Interface Creation	Table 3-2
Real-T ime Status	Table 3-3

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 {
```

Parameter Type	Parameter Name	Description
`XDAS_Int16`	`speedGrow`	Adaptation speed coefficient when gain is increasing
`XDAS_Int16`	`speedFall`	Adaptation speed coefficient when gain is decreasing
`XDAS_Int16`	`reference`	Output reference amplitude
`XDAS_Int16`	`maxOutputValue`	Max value of output sample that AGC will output
`XDAS_Int16`	`initGainCoef`	High 16 bit of GainCoef
`XDAS_Int16`	`safeOutputValue`	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.
`XDAS_Int16`	`gainMax`	Maximal gain value
`XDAS_Int16`	`gainMin`	Minimal gain value

```
} IAGC_Params
```

Type `IAGC_Params` is defined in “`iAGC.h`”.

Gain Parameter Value	Gain Coefficient
0x0001	1/256
0x0002	1/128
0x0004	1/64
0x0008	1/32
0x0010	1/16
0x0020	1/8
0x0040	1/4
0x0080	1/2
0x0100	1
0x0200	2
0x0400	4
0x0800	8
0x1000	16
0x2000	32
0x4000	64

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 {
```

Status Type	Status Name	Description
`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`”.

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 3-4. AGC Control Standard Interface Functions

Functions	Description	See Page...
`AGC_init`	Calls the framework initialization function (Algorithm initialization)	3-4
`AGC_exit`	Calls the framework exit function (Algorithm deletion)	3-4
`AGC_create`	Calls the framework creation function (Instance creation)	3-5
`AGC_delete`	Calls the framework deletion function (Instance deletion)	3-5

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*

Function

AGC_create

Description

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.

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*

Function

AGC_delete

Description

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

Function Prototype

```
void AGC_delete (AGC_Handle handle);
```

Arguments

AGC_Handle	Instance's handle obtained from `AGC_create()`

Return Value

none

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

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

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

```
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

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 (`speedGrow, speedFall`) and reference amplitude (`reference`) for AGC. Current state of AGC object remains unchanged unlike function `AGC_reInit()`.

Parameters `maxOutputValue`, `initGainCoef` in `params` are ignored.

Function Prototype

```
XDAS_Void AGC_setParameters(
     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 Parameters `maxOutputValue`, `initGainCoef` in `params` can not be set and shall be updated by using `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 `maxOutputValue`;
- Estimates power of signal and adapts loop gain according to `speed-Grow, speedFall`;
- Changes loop gain so overall signal magnitude will be approximately equal to output reference amplitude (`reference`, see Table 3-2). Adaptation can be disabled by clearing parameter `isAdapting` (see function parameters below).

Function Prototype

```
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

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

Arguments

handle	Pointer to AGC instance

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

Restrictions none

Appendix A

Test Environment

C54CST

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`.

Topic **Page**

A.1 Description of Directory Tree

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

Table A-1. Test Files for AGC

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 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 dBF

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