GaGe

CompuGen SDK User's Guide

For CompuGen 1100, CompuGen T30 and CompuGen 3250

Supporting:

- **CompuGen SDKs for DOS (SDK v3.00+)**
- **CompuGen SDKs for Win 95/98 and Win NT (SDK v1.00+)**
- **CompuGen SDKs for LabVIEW for Win 95/98 and Win NT (SDK v1.00+, LabVIEW 4.0+)**

With Sample Programs in C/C++ and Visual BASIC

P/N: 0045023
Reorder #: MKT-SWM-CGD06
0404

Third Edition (April 1999)

Please complete the following section and keep it handy when calling Gage for **technical support**:

Owned by: _____
Serial Number: _____
Purchase Date: _____
Purchased From: _____

You must also have the following information when you call:

- Brand name and type of computer
- Processor and bus speed
- Total memory size
- Contents of AUTOEXEC.BAT and CONFIG.SYS files
- Information on all other hardware in the computer

How to reach Gage Applied Technologies, Inc. for Product Support
Toll-free phone: (800) 567-GAGE
Toll-free fax: (800) 780-8411

To reach Gage from outside North America
Tel: (514) 633-7447
Fax: (514) 633-0770
E-mail: prodinfo@gage-applied.com
Website: http://www.gage-applied.com

Table of Contents

Preface

This manual is designed to describe the routines included in the **CompuGen 1100** and **CompuGen T30** Software Development Kits (SDKs) for the **Windows NT, Windows 98, Windows 95** and **DOS** environments.

It is assumed that the programmer is familiar with the concepts of Dynamic Link Libraries, Windows programming and the programming language in use. No description is included for these topics. If the programmer is not comfortable with any one of these topics, it is strongly recommended that a relevant reference manual be referred to before starting.

The SDK supports multiple CompuGen boards; however, from the programmer's point of view, only one board is accessible at any given time. The initialization routine reads a specially formatted array to determine where the user has installed the CompuGen cards and then tries to initialize each board, determine that it is indeed present and the then tests and sizes the memory on each board found. Another routine will read a binary disk file and initialize the special array, or the user can create the array with the format described in the initialization routine and pass it to the initialization routine. A routine is provided to select the desired active board and then all subsequent operations are applied to the active board, from data generation to configuration set-up.

The routine descriptions, listed alphabetically, describe the syntax. Also listed in the description will be a set of named constants that can be used with each routine. Every effort to use the name and not the equivalent numeric value should be made, as the numbers represented by the constants are subject to change, but the names of the constants are not.

The CompuGen SDK for DOS is a collection of C files that allow all of the DLL calls available in Windows NT and Windows 95/98 environments. Therefore any reference to a DLL call is still applicable in the DOS environment (see *Important Definitions*).

Introduction to the CompuGen Software Development Kits (SDKs)

CompuGen SDKs for Win NT

Windows NT is a classic operating system which traps any accesses made to system memory or I/O. An SDK has been written which allows the same calls to be made to the NT drivers as programmers have been used to with Windows 95/98. All compilers which can make calls to a Windows NT DLL are supported. These include Visual BASIC for Windows, Visual C++, Borland C++, etc.

CompuGen SDKs for Win 95/98

All Windows-based compilers use Gage's CompuGen DLL, which is based on the same source code as the SDK for DOS. A DLL is necessary to allow Windows to talk to a CompuGen card without making any direct memory accesses. The SDK for Win 95/98 supports CompuGen cards in single-card or Multi-Card Master/Slave configurations. All compilers which can make calls to a Windows 95/98 DLL are supported. These include Visual BASIC for Windows, Visual C++, Borland C++, etc.

CompuGen SDKs for DOS

CompuGen cards are supported by a high-performance Software Development Kit for DOS. This SDK consists of source code C drivers compatible with Microsoft C, Borland C and Watcom C. This SDK comprises subroutines which allow the programmer to initialize the hardware, set up all the relevant parameters, start a signal generation sequence, and transfer data from PC memory to on-board memory. This SDK is the most efficient way of programming Gage CompuGen cards.

Configuring your CompuGen Card

Now that you have fully installed CompuGen SDK for Win 95/98, Win NT or DOS, you should configure your CompuGen board(s) and test your SDK installation.

Configuration under Windows 95/98 and Windows NT

CompuGen 1100

To configure your CompuGen 1100 card under Windows, you should run CGWin.exe. CGWin is provided on separate installation disks. CGWin will create the configuration file GAGE_GEN.INC, which holds the I/O address and memory segment information for each CompuGen board in the system. The file should always be in your Windows directory (usually C:\WINDOWS). For details on installing and running CGWin, please refer to your *CompuGen for Windows Start-up Guide*.

CompuGen T30

To configure your CompuGen T30 card under Windows, you should run CGT30.exe. CGT30 is provided on separate installation disks. CGT30.exe will create the configuration file GAGE_GEN.INC, which holds the I/O address and memory segment information for each CompuGen board in the system. The file should always be in your Windows directory (usually C:\WINDOWS). For details on installing and running CGT30, please refer to your *CGT30 Software Start-up Guide*.

Configuration under DOS

To configure your CompuGen card under DOS, you should run CGINST.exe. CGINST is provided with the CompuGen SDK for DOS and is installed in the directory you chose during setup (C:\Gage\GDrivers by default). CGINST will create the configuration file GAGE_GEN.INC, which holds the I/O address and memory segment information for each CompuGen board in the system. Under DOS, GAGE_GEN.INC should go in the same directory as your application executable.

Memory Organization

Memory Architecture

The D/A speeds at which the CompuGen boards generate signals are too fast for the computer's ISA bus to handle (the maximum transfer rate for the ISA bus is approximately 2 Mbytes/s). As such, the CompuGen boards have high-speed on-board memory to store the digital data from the computer to generate waveforms/patterns.

Interface for the ISA Bus

In order to allow optimum data transfer rates from the PC memory or extended memory to the CompuGen memory, the on-board memory is mapped within the memory map of the 80x86 processor, between 640K and 1M (factory default is E000H - E1FFH for the CompuGen boards).

The CompuGen 1100 takes only 4 kilobytes of memory space between 640K and 1M, while the CompuGen T30 takes only 8 kilobytes of memory space in this same region. This memory address is configurable by writing to the on-board segment register, i.e. it is configured by software, not by DIP switches, etc. This small memory window and software configuration mean that there is very little chance of memory conflicts in any PC.

The CompuGen has memory depths much greater than just 4 KB or 8 KB. Therefore, the on-board memory is addressed in a segmented manner and is divided into 4 KB blocks.

ON-BOARD MEMORY ADDRESS	BLOCK NUMBER
0 to 4095	0
4096 to 8191	1
8192 to 12287	2
12288 to 16383	3
... ...	

Using the above method, any byte of on-board memory can be addressed using the BLOCK NUMBER and an OFFSET.

Memory Organization from a Programmer's Point of View

This section describes the memory management scheme for the CompuGen board and is provided for information purposes only. The implementation details are performed by the driver. Application programs do not have to handle these details.

The CompuGen memory is too large to fit into the memory map of the PC below the one megabyte assigned to add-in adapters on the ISA bus. Although the area above one megabyte could be used, only 15 megabytes of memory space is available and the memory can only be accessed via protected mode programs.

To simplify the access to the generation memory, a "paged" scheme was created. To avoid confusion with the typical PC/Intel terminology (that a page is 4 kilobytes of memory which is used when a paging memory mechanism is enabled inside the x86 CPU while in protected mode), Gage refers to these pages as "blocks."

Since the available memory is considerably larger than our block size, all of the onboard memory is not available to the programmer at one time. Rather, the memory is accessed via a movable block pointer (actually a register in the I/O map of the CompuGen card) that points to an area that is 4 kilobytes long.

The full memory space of the card is broken into an integer number of these 4 kilobyte blocks. The samples are organized as 16-bit words within each block. The block address pointer is then moved to the next block and data can be moved into the next section of memory.

This process is repeated until all of the memory that will contain the signal to be generated has been updated.

Accessing the Memory

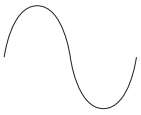
The data is organized as 16 bit samples. As such the memory is accessed on even addresses or, in other words, on word boundaries.

	Memory Address
Sample0	0
Sample1	2
Sample2	4
Sample3	8
..	..

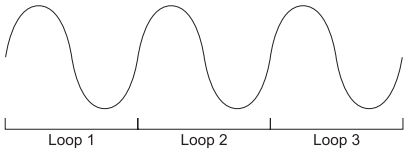
The start address must be specified using the appropriate I/O mapped register. Similarly, the end address must also be specified using an I/O mapped register for this purpose. The pattern loaded into memory can then output starting from the start address continuing on to the end address. This pattern may be output once (one shot mode), any preset number of times specified in the appropriate register (looping mode), or continuously.

Pattern Modes for CompuGen 1100

One Shot Mode



Looping Mode

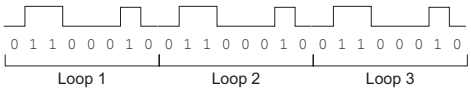


Continuous Mode

Pattern Modes for CompuGen T30

One Shot Mode

Looping Mode



Continuous Mode

Important Definitions

The following type definitions are used in the driver to simplify the code and reduce confusion. These definitions are used both in this manual and in the sample programs for all platforms provided on the release diskettes. Note that the terms “driver” and “DLL” are used interchangeably in this section.

```
*************************************************
*       Memory Type definitions                 *
*************************************************
typedef unsigned char             uInt8;
typedef unsigned short  uInt16;
typedef unsigned long             uInt32;
typedef signed char               int8;
typedef signed short              int16;
typedef signed long               int32;
typedef char far *                LPSTR;
```

If a routine fails, a global error function is available that describes the error which occurred and the board that caused the error. This function, called **ggen_get_error_code,** returns a value which is encoded with the high byte containing the board in error and the low byte set equal to the defined error constant. These constants are listed below.

```
***********************************************************
*           Error Type definitions                        *
***********************************************************
```

Constants	***value***
GGEN_NO_ERROR	0x00
GGEN_NO_SUCH_MODE	0x01
GGEN_NO_SUCH_SAMPLE_RATE	0x02
GGEN_INVALID_SAMPLE_RATE	0x03
GGEN_NO_SUCH_BOARD	0x04
GGEN_NO_SUCH_GAIN	0x05
GGEN_NO_SUCH_TRIG_SLOPE	0x06
GGEN_NO_SUCH_TRIG_SOURCE	0x07
GGEN_LOAD_INVALID_CHANNEL	0x08
GGEN_LOAD_INVALID_LENGTH	0x09
GGEN_NO_PATTERN_LOADED	0x0A
GGEN_NO_SUCH_SYNC_OUT	0x0B
GGEN_INVALID_CLK_SOURCE	0x0C
GGEN_NO_SUCH_TRIG_RANGE	0x0D
GGEN_NO_SUCH_FILTER	0x0E
GGEN_MISC_ERROR	0xff

If the DLL detects a board during initialization then one of the following "ORable" constants is set in the **ggen_board_location** status position. The board type definitions are:

```
************************************************************
* CompuGen board type                                      *
************************************************************
```

Constants	***value***
GGEN_ASSUME_NOTHING	0x0000
GGEN_ASSUME_CG1100	0x0400
GGEN_ASSUME_CGT30	0x1000
GGEN_ASSUME_ALL_BOARDS	(GGEN_ASSUME_CGT30 | GGEN_ASSUME_CGT30)

The GGEN_ASSUME_ALL_BOARDS constant is included to provide a quick means to determine if any CompuGen hardware was encountered during initialization.

```
************************************************************
* CompuGen operation mode constants.                       *
************************************************************
```

Constants	***value***	
GGEN_12BIT_MODE	2	for CompuGen 1100
GGEN_16BIT_MODE	4	for CompuGen T30

```
************************************************************
*  CompuGen trigger slope values.                          *
************************************************************
```

Constants	***value***
GGEN_POSITIVE	0
GGEN_NEGATIVE	1

```
************************************************************
*  CompuGen trigger source values.                         *
************************************************************
```

Constants	***value***
GGEN_EXTERNAL	0
GGEN_SOFTWARE	1

****
* CompuGen Rate Table. *
****

Constants	*value*	
GGEN_RATE_1	1	
GGEN_RATE_2	2	
GGEN_RATE_5	5	
GGEN_RATE_10	10	
GGEN_RATE_20	20	
GGEN_RATE_50	50	
GGEN_RATE_100	100	
GGEN_RATE_200	200	
GGEN_RATE_500	500	
GGEN_RATE_40	40	Only used with GGEN_MHZ
GGEN_RATE_80	80	Only used with GGEN_MHZ in CG1100 system

****
* CompuGen sample rate multiplier values. *
****

Constants	*value*
GGEN_HZ	1
GGEN_KHZ	2
GGEN_MHZ	3
GGEN_GHZ	4

****
* CompuGen clock values. *
****

Constants	*value*
GGEN_EXTERNAL_CLOCK	5
GGEN_SOFTWARE_CLOCK	6

****
* CompuGen filter. *
****

Constants	*value*	
GGEN_NO_FILTER	0	
GGEN_FILTER_20MHZ	1	
GGEN_FILTER_5MHZ	3	Only for CG1100

Since the C language does not have true Boolean types and constants, in the Pascal sense of the term, two constants have been provided for this purpose to synthesize the Boolean meanings.

```
*********************************************************
* Boolean constants.                                    *
*********************************************************
```

Constants	***value***	
TRUE	1	A true value.
FALSE	0	A false value.

```
*********************************************************
*  CompuGen gain values.                                *
*********************************************************
```

Constants	***value***	
GGEN_TIMES_5	1	± 5 volt range
GGEN_TIMES_2	2	± 2 volt range
GGEN_TIMES_1	3	± 1 volt range
GGEN_TIMES_0_POINT_5	4	± 500 mVolt range
GGEN_TIMES_0_POINT_2	5	± 200 mVolt range
GGEN_TIMES_0_POINT_1	6	± 100 mVolt range

The **ggen_driver_initialization** routine uses an array called **ggen_board_location** to pass the desired locations for the CompuGen hardware to the DLL code. The easiest method of setting up this array is to call the **ggen_read_config_file** routine that uses the same GAGE_GEN.INC file. The **ggen_board_location** array can also be initialized by calling the **ggen_set_records** routine. See the description for this routine for an example explaining the initialization of the **ggen_board_location** array. This array is declared (in C) as:

```
uInt16  ggen_board_location[GGEN_B_L_BUFFER_SIZE];
```

The first segment and I/O index locations are:

```
ggen_board_location[0];        memory segment for board 1
ggen_board_location[1];        I/O address for board 1
ggen_board_location[2];        memory segment for board 2
ggen_board_location[3];        I/O address for board 2
ggen_board_location[4];        memory segment for board 3
ggen_board_location[5];        I/O address for board 3
etc.
```

Currently, a maximum of 16 CompuGen boards can be used in one system.

When the **ggen_driver_initialize** routine encounters an error during initialization, it returns the error codes encountered via the **ggen_board_location** array. The error status for the first board is:

ggen_board_location[GGEN_B_L_STATUS_START];

where GGEN_B_L_STATUS_START is currently equal to 32.

The value returned can be one or more of the following constants. Note that these constants are individual bits and can be "ORed" together to form a mask. The CompuGen definitions for **ggen_board_location** status errors are:

GGEN_BAD_LSB_SEGMENT	0x0001	bad least significant bit segment
GGEN_BAD_MSB_SEGMENT	0x0002	bad most significant bit segment
GGEN_BAD_LSB_INDEX	0x0004	bad least significant bit index
GGEN_BAD_MSB_INDEX	0x0008	bad most significant bit index
GGEN_DETECT_FAILED	0x0010	
GGEN_MEMORY_FAILED	0x0020	
GGEN_BAD_MEMORY_SIZE	0x0040	

The driver is normally instructed to check the encountered CompuGen hardware for the size of memory installed. The method of operation can be overridden for special purposes. The GGEN_MEMORY_SIZE_TEST (0) constant is available to tell the driver to check the memory in the call to **ggen_driver_initialize**. If a non-zero value is used, the driver will assume that the value is the size of the installed CompuGen memory and will not perform a memory test.

An important structure, defined in GGEN_DRV.H, is **ggen_driver_info_type**. This structure is used to obtain information about the currently selected board by using the **ggen_get_driver_info** routine. The **ggen_driver_info** routine is explained more fully in the section explaining API routines. Below are the fields of the **ggen_driver_info_type** structure.

uInt16	index	base I/O port index of currently selected CompuGen, used to transfer data to the CompuGen card. The value is typically read from the configuration file GAGE_GGEN.INC. The CompuGen boards use 8 I/O ports starting from this address.
uInt16	segment	absolute address of memory segment, typically read from GAGE_GEN.INC.
uInt16	selector	selector to the memory segment.
uInt16	offset	offset from the base memory segment used. For example, if the memory segment used is 0xD200, the base segment is 0xD000 and the offset is 0x2000.
uInt8 far	*memptr	pointer to the segment:offset address of the start of CompuGen RAM.
int16	mode	mode of the currently selected board, GGEN_12BIT_MODE (CG1100) or GGEN_16BIT_MODE (CGT30), set by **ggen_generate_mode** routine.

int16	rate	set by **ggen_generate_mode** routine.
int16	multiplier	set by **ggen_generate_mode** routine. The constants are GAGE_HZ, GAGE_KHZ, GAGE_MHZ, GAGE_GHZ, GAGE_EXT_CLK and GAGE_SW_CLK.
int16	oneshot	flag the tells whether the CompuGen board is in one-shot mode or not.
int32	max_memory	memory size of CompuGen board (in samples).
int16	board_type	numeric constant representing the board type as defined in the GGEN_DRV.H file.
int16	o_range	value being used for the output range One of the following predefined constants defined in GGEN_DRV.H file should be used with the **ggen_output_control** routine to set this parameter: GGEN_TIMES_5, GGEN_TIMES_2, GGEM_TIMES_1, GGEN_TIMES_0_POINT_5, GGEN_0_TIMES_2 or GGEN_TIMES_0_POINT_1. These only have an effect on a CompuGen 1100. The output range for a CompuGen T30 is TTL.
int16	o_filter	value being used for the output filter on a CompuGen T30. One of the following predefined constants should be used with the **ggen_output_control** routine to set this parameter: GGEN_NO_FILTER, GGEN_FILTER_20MHZ or GGEN_FILTER_5MHZ.
int16	t_source	value being used for the trigger source. One of the following predefined constants should be used with the **ggen_trigger_control** routine to set this parameter: GGEN_EXTERNAL or GGEN_SOFTWARE.
int16	t_slope	value being used for the trigger slope. One of the following predefined constants should be used with the **ggen_trigger_control** routine to set this parameter: GGEN_POSITIVE or GGEN_NEGATIVE.
int16	t_range	value being used for the trigger range. This is set with the **ggen_trigger_control** routine. The predefined constant GGEN_TIMES_5 can be used to set the ±5 volt rage.

int16	t_level	value being used for the trigger level. The routine **ggen_trigger_control** is used to set this value. The value ranges from 0 to 255, where 0 is the most negative voltage in the trigger range and 255 the most positive. This value has no effect if the trigger source is software, or on a CompuGen T30 (which only has a TTL trigger range)
int16	ext_clk_50ohm	flag which tells if the external clock has been set to 50 ohm impedance or not. A non-zero value means that it has. The driver routine **ggen_ext_clock_ctrl_50ohm_on** should be used to set this flag.
int16	ext_trig_50ohm	flag which tells if the external trigger input has been set to 50 ohm impedance or not. A non-zero value means it has. The driver routine **ggen_ext_trig_ctrl_50ohm_on** should be used to set this flag.
int16	gate_lc_50ohm	flag which tells whether or not the gate input has been set to 50 ohm impedance. A non-zero value means it has. The driver routine **ggen_gate_lc_ctrl_50ohm_on** should be used to set this flag.
int16	inter_ctrl_50ohm	flag which tells whether or not the CompuGen board is using 50 ohm termination internally. A non-zero value means it has. The driver routine **ggen_internal_ctrl_50ohm_on** should be used to set this flag. Note: most application programs should have no need to change this flag from the default.

Application Development

In this section, a typical CompuGen application will be described. The following description applies to both the CompuGen 1100 and CompuGen T30 boards, as well as to all supported operating systems. Any differences will be noted in the description.

The description is written using C, but the same concepts apply regardless of the programming language used. The code fragments used are modified versions of the Windows sample program, CG_OUT, which is supplied on the Sample Source Code/SDK disk(s).

Initialization

Before the CompuGen hardware can be used, both it and the drivers must be initialized. The following code fragment can be used to initialize the hardware.

```
char     board_loc_file[260];       //path of config file
uInt16   ggen_board_location[64];
int initialize()
{
        int ret = 0;

        //Determine the complete path of the configuration file GAGE_GEN.INC
        ggen_get_config_filename(board_loc_file);

        //Reads the file and stores the data in the ggen_board_location array
        if ((ret = ggen_read_config_file ((board_loc_file), (uInt16 *)(ggen_board_location))) < 0)
                return -1;
        if (!ggen_driver_initialize((uInt16 *)(ggen_board_location),
                GGEN_MEMORY_SIZE_TEST))
                return -1;
        return 0;
}
```

The **ggen_get_config_filename** is used to read the configuration file, GAGE_GEN.INC, which holds the I/O address and memory segment information for each CompuGen board in the system. The configuration file can be created using the CGWIN.EXE (for CG1100 under Windows), CGT30.EXE (for CGT30 under Windows or the CGINST.EXE application (for both CG1100 and CGT30 under DOS) provided with the SDK.

In DOS, GAGE_GEN.INC should go in the same directory as your executable. Under Windows 95/98 or Windows NT, the file should go in the Windows directory. When the function returns, the board_loc_file parameter will hold the full path (including the filename) to GAGE_GEN.INC. In DOS, this will be the current working directory. In Windows 95 or Windows NT, this will be the Windows directory.

The **ggen_read_config_file** routine is used to read the configuration file and put the information into an array that is used by **ggen_driver_initialize**. It uses the location returned by **ggen_get_config_filename** to find the configuration file. Alternatively, the path can be used directly as the first parameter to the routine. This is useful if the GAGE_GEN.INC file is kept somewhere other than the default location. You can also fill the array passed to **ggen_driver_initialize** directly with the **ggen_set_records** routine. The array is filled up with the memory segment and I/O address for each board filling up the first half of the array. Unused portions are filled with zeros. The second half of the array is used to return status codes and board types upon return from the initialization routine.

The CompuGen hardware and driver are initialized by calling **ggen_driver_initialize**. The second parameter is a predefined constant (whose value is 0) that tells the routine to determine the memory size of the board(s). Each memory segment and I/O address pair found in the first parameter is examined to see if a CompuGen board (either a CG1100 or CGT30) is found there. If it is, a structure in the driver is created to represent it.

The return value from **ggen_driver_initialize** is the number of CompuGen boards found and successfully initialized. If a negative number is returned, not all the memory segment and I/O address pairs passed to the routine had CompuGen hardware that could be found. The negative number represents the number of boards that could not be detected and initialized.

Multiple boards in a Master/Slave configuration are automatically detected. If the boards are set up in a Multiple/Independent configuration (that is, not using a common clock or trigger), **ggen_set_independent_operation** should be called right after the boards are initialized to tell the drivers to use Multiple/Independent mode.

Initialization only has to be performed once. If not all of the boards are found, the usual cause of the error is a memory or I/O address conflict. See your CompuGen Hardware Manual for details on resolving a memory or I/O conflict.

Board Setup

Once the drivers are initialized, the hardware must be told what the generation settings are. These settings are the mode, sample rate, output range, trigger source, etc. The following code fragment shows how to do this:

```
ggen_generate_mode(struc->cgi_mode, struc->cgi_clock_rate, struc->cgi_clock_mult);
ggen_output_control(struc->cgi_gain, struc->cgi_filter);
ggen_trigger_control(struc->cgi_trigger_source, struc->cgi_slope, struc->cgi_gain, struc-
>cgi_trigger_level);
ggen_set_outer_loop_counter (0);
```

In the above code, **struc** is an application-defined structure for grouping the variables used to set the hardware. It can be replaced by constants or other variables. The order of the calls in the setup is important.

The routine **ggen_generate_mode** is used to send the mode and sample rate to the hardware. The mode can be either GGEN_12BIT_MODE (used for the CG1100) or GGEN_16BIT_MODE (used for the CGT30). The sample rate can be any valid sample rate broken up into rate and multiplier. The valid values for the rate are:

Constant	Value	
GGEN_RATE_1	1	
GGEN_RATE_2	2	
GGEN_RATE_5	5	
GGEN_RATE_10	10	
GGEN_RATE_20	20	
GGEN_RATE_40	40	Only valid with GAGE_MHZ
GGEN_RATE_50	50	
GGEN_RATE_80	80	Only valid with GAGE_MHZ and the CG1100
GGEN_RATE_100	100	
GGEN_RATE_200	200	
GGEN_RATE_500	500	

The valid multiplier values are:

Constant	Value
GGEN_HZ	1
GGEN_KHZ	2
GGEN_MHZ	3
GGEN_EXTERNAL_CLOCK	5

The rate and multiplier values are combined to form the sample rate. All rates are valid for both the CG1100 and the CGT30, except for 80 MHz, which is only valid on the CG1100. If an external clock is being used, then both the rate and multiplier should be set to GGEN_EXTERNAL_CLOCK.

The **ggen_output_control** routine is used to configure the output range and output filter of the CompuGen hardware. The available output ranges are:

Constant	Value	
GGEN_TIMES_5	1	±5 volts
GGEN_TIMES_2	2	±2 volts
GGEN_TIMES_1	3	±1 volt
GGEN_TIMES_0_POINT_5	4	±500 millivolts
GGEN_TIMES_0_POINT_2	5	±200 millivolts
GGEN_TIMES_0_POINT_1	6	±100 millivolts

The CGT30 only uses a TTL output (0 to 5 volts) regardless of what is used in the call to **ggen_output_control**.

On the CG1100, the above output ranges all assume a 50 ohm load on the output. If a 50 ohm load is not there, the output range is doubled.

The **filter** parameter is used to smooth out the signal above certain frequencies. The filter parameter applies only to the CG1100, not to the CGT30. Valid values are:

Constant	Value
GGEN_NO_FILTER	0
GGEN_FILTER_20MHZ	1
GGEN_FILTER_5MHZ	3

The **gage_trigger_control** routine sets up the trigger parameters of the hardware. The **source** parameter can be either GGEN_EXTERAL (0) or GGEN_SOFTWARE (1). If the source is set to external, data generation will not be started until an external trigger event is received. If the trigger source is software, data generation will begin when a software trigger is issued. This is done by calling **ggen_software_trigger**. The other parameters only have an effect if the trigger source is external.

The **trigger slope** can be either GGEN_POSITIVE (0) or GGEN_NEGATIVE (1).

The **trigger gain** can be GGEN_TIMES_5 (1) for the ±5 volt range for the CG1100. The CGT30 only has a TTL external trigger.

The **trigger level** is a uInt8 value between 0 and 255, where 0 represents the lowest value in the trigger range and 255 the highest. The formula to convert the trigger level to a voltage is;

((level - 128.0) / 128.0) * trigger range (where trigger range is 5 for the ±5 volt range)

The differential between the actual trigger signal and the trigger level should not be too large. For example, if the signal coming into the external trigger BNC was ±5 volts and the trigger level was set to 4 volts, the difference would between the minimum value of the external trigger signal and the level is nine volts.

The trigger level on the CGT30 cannot be changed.

The **ggen_set_outer_loop_counter** routine is used to tell the CompuGen hardware how many times to generate the pattern loaded into it. A zero value means to generate in continuous mode. In this mode the pattern is continuously generated until it is aborted. A positive, non-zero parameter will cause the pattern to be generated that number of times. Data generation will then stop.

Filling the Buffer

The on-board memory is filled by calling the routine **ggen_load_vram_from_buffer**. This routine loads the CompuGen memory with a user-supplied pattern. The following code fragment shows how to use this routine:

ggen_load_vram_from_buffer (length, buffer, loop_number, end_flag);

where:

length — the size of the buffer in 16 bit words. The length should be a multiple of 4. If it is not, the driver will round the length down to the nearest multiple.

buffer — a pointer to the user-supplied pattern.

loop_number how many times the pattern should be generated. This parameter is reserved for future use and should be set to 1 for the current drivers.

end_flag a flag to tell the driver which is the last pattern to be loaded. This parameter is reserved for future use and should be set to 1 for the current drivers.

The buffer should be filled with 16 bit values between 0 and 4095 when using a CG1100. The most positive voltage in the current output range corresponds to 0 and the least positive voltage corresponds to 4095. For example, in the ±1 volt range, -1 volt is represented by 4095 and +1 volt is represented by 0. The following code fragment can be used to fill the buffer with an application generated sine wave. This buffer would then be passed to the **ggen_load_vram_from_buffer** routine.

```
void    sinewave (uInt16 *buffer, int32 *length)
{
        int16   x, value,           offset = 0, amplitude = 2048;
        double  a, pi = 3.141592;

        for (x = 0 ; x < *length ; x++)  {
                a = -(sin (2 * pi * ((double)(x) / (double)(*length))));
                value = (2048 - offset) + (int16)(a * amplitude);
                if (value > 4095)
                        value = 4095;
                if (value < 0)
                        value = 0;
                buffer[x] = value;
        }
}
```

This routine creates a ±1 volt sine wave. Remember that the CG1100 drivers assume a 50 ohm load on the output, so it may appear as a ±2 volt sine wave if there is not a 50 ohm load.

The CGT30 can be filled either with a buffer of 16 bit values or a buffer of 32 bit values. In either case, the length parameter must be the number of 16 bit values in the buffer. If a buffer of 16 bit values is used, each two consecutive 16 bit values are treated as the low-high halves of a 32 bit word. For example, a counter pattern can be loaded as follows:

Value	Binary (32 bits)
0	00000000000000000000000000000000
1	00000000000000000000000000000001
2	00000000000000000000000000000010
3	00000000000000000000000000000011
4	00000000000000000000000000000100
5	00000000000000000000000000000101
6	00000000000000000000000000000110
7	00000000000000000000000000000111

In this case, the length parameter would be 16 (eight 32 bit values = sixteen 16 bit values).

Value	Binary (16 bits)
0	0000000000000000
1	0000000000000000
2	0000000000000001
3	0000000000000000
4	0000000000000010
5	0000000000000000
6	0000000000000011
7	0000000000000000
8	0000000000000100
9	0000000000000000
10	0000000000000101
11	0000000000000000
12	0000000000000110
13	0000000000000000
14	0000000000000111
15	0000000000000000

In this case, the buffer is filled with 16 bit values and the length is also 16.

Generating the Data

Once the pattern has been loaded, the **ggen_dump_data** routine should be called to arm the board. After the board has been armed, the data will be generated when a trigger event is received. The following code fragment shows this:

```
ggen_dump_data ();
if (struc->cgi_trigger_source == GGEN_SOFTWARE)
        ggen_software_trigger ();
```

The **ggen_software_trigger** routine is used to immediately generate a software trigger.

In non-continuous mode, **ggen_dump_data** does not have to be called to generate data again if the pattern has not been changed.

Output Frequency

CompuGen 1100

The output frequency is determined by the sample rate and the number of samples being generated. For example, if one cycle of a sine wave is created using 1000 samples and generated at a sample rate of 40 MHz, the output frequency would be:

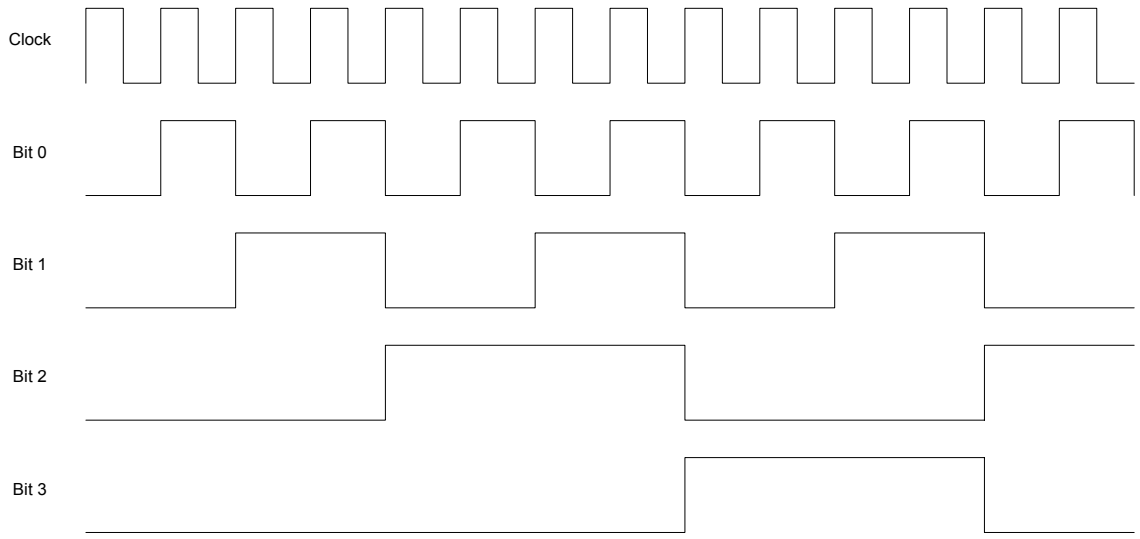
sample rate / number of samples = output frequency
40,000,000 / 1000 = 40,000 = 40 kHz

If the same sine wave was created with 100 points, the output frequency would be:

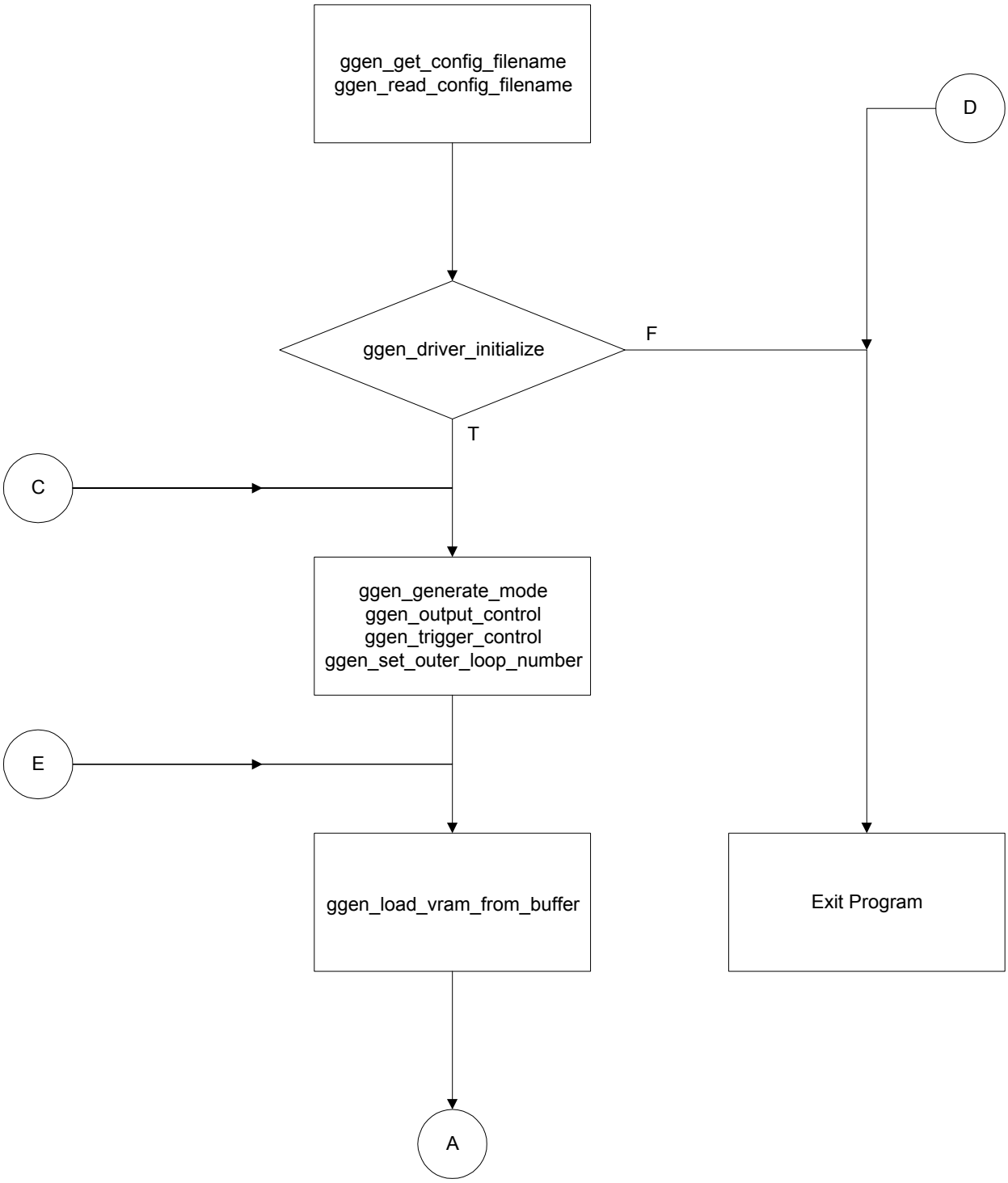
40,000,000 / 100 = 400,000 = 400 kHz

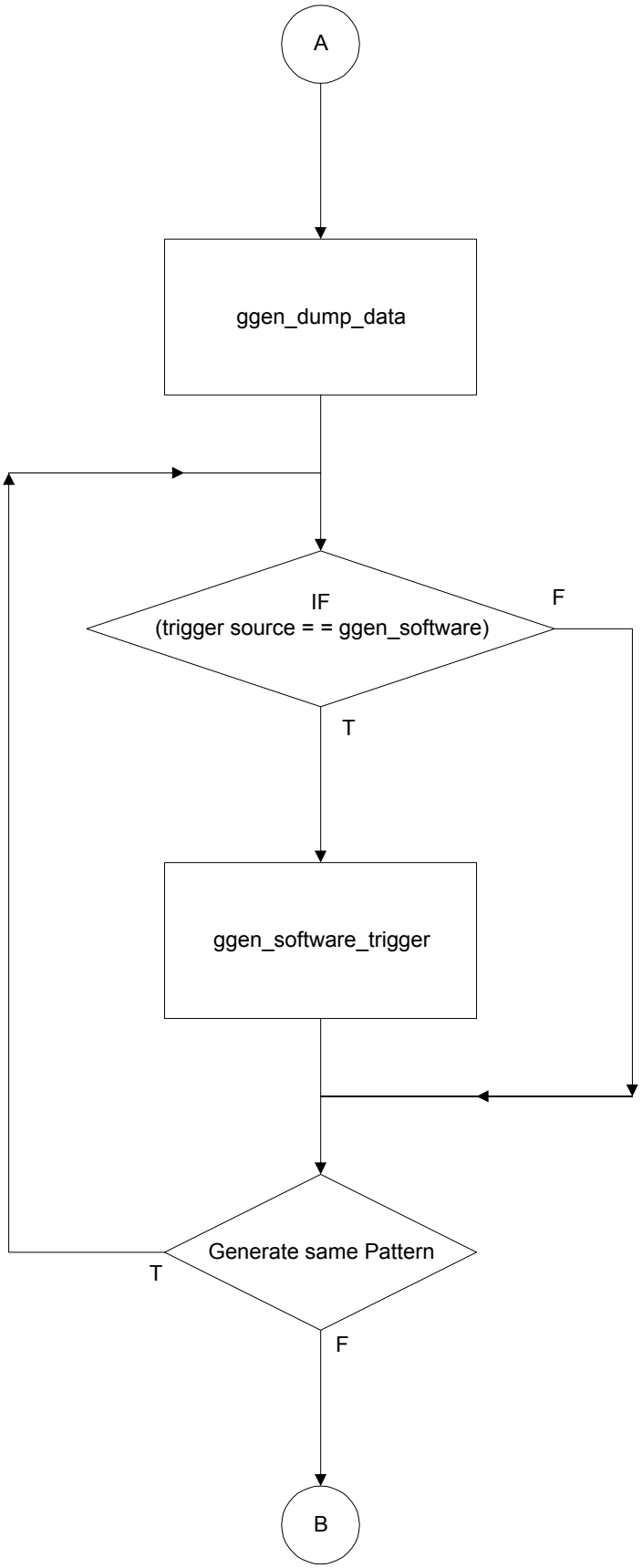
CompuGen T30

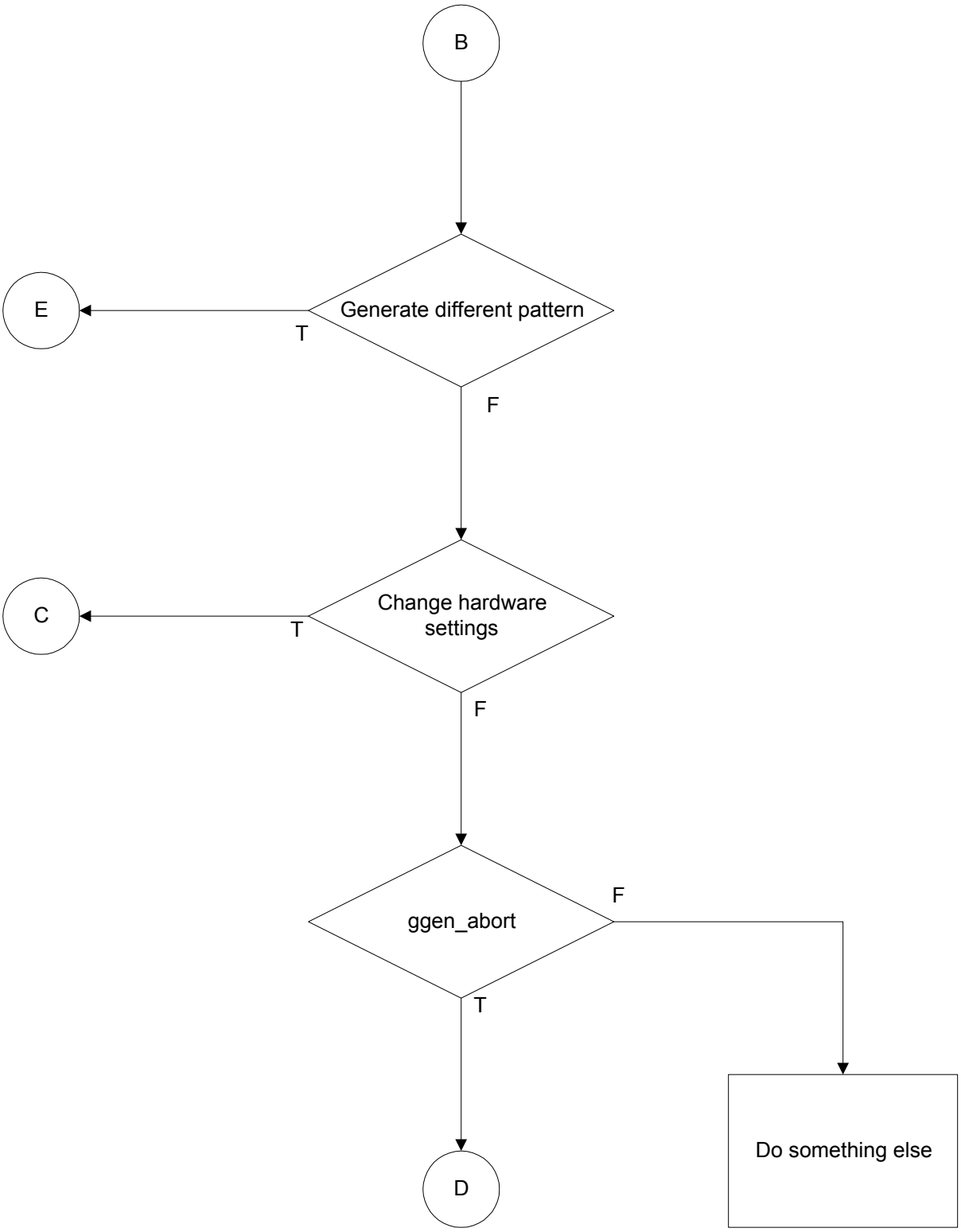
Either a low (0) or high (1) is generated whenever a clock occurs. If a counter is being generated at 40 MHz, the first few bits of the pattern would look like this:



Program Execution




A
ggen_dump_data
IF
(trigger source = = ggen_software)
F
T
ggen_software_trigger
Generate same Pattern
T
F
B


B
Generate different pattern
E
T
F
Change hardware settings
C
T
F
ggen_abort
F
T
Do something else
D

Sample Programs

This section describes two of the sample programs available in the CompuGen SDK for Windows 95/98, one for C and one for Visual BASIC. Note that the availability and number of sample programs may change after the publication of this manual. The sample programs are meant to show how to program the CompuGen 1100 and CompuGen T30 using the provided API routines. These API routines are used the same way for both boards and under all platforms (Windows 95/98, Windows NT and DOS) unless otherwise stated in this manual.

CompuGen SDK Basics: C

The simplest way to use the API routines in your own C programs is to link your program with the provided CGWINDLL.LIB. This will give your application program access to all the CompuGen DLL functions. You can do this by including CGWINDLL.LIB in your project file.

Note that the CGWINDLL.DLL and CGWINDLL.LIB files have been compiled with Microsoft Visual C++. Because of this, the CGWINDLL.LIB may (depending on the version of the compiler you are using) be incompatible with Borland C. To use the Borland C compiler with the DLL, you will need to recreate the CGWINDLL.LIB file. This can be done using the IMPLIB.EXE tool that comes with your Borland C compiler. The command line syntax is:

```
implib cgwindll.lib cgwindll.dll
```

This will create a new CGWINDLL.LIB file that is compatible with Borland compilers.

The Sample Program CG_OUT: C for Windows

The sample program CG_OUT was created using the design tools that are bundled with the Microsoft Visual C package. It features discrete and continuous data generation. Several routines and data structures have been added to the sample program that will allow the programmer to get started quickly.

The program reads in a response file (the default name is DEFAULT.CGI) which contains the board settings to use for data generation. The response file specifies a GageScope SIG file to read in and generate. If no response file is found, or if there is no SIG file mentioned in the response file, a 100-sample sine wave is generated. If no CGI file is found, default settings are used for the board parameters. Otherwise, the settings in the CGI file are used. The CGI file should be mentioned on the command line of the program, i.e. CG_OUT DEFAULT.CGI.

The program will work in the same way for either a CompuGen 1100 or a CompuGen T30.

Data Structures

The first data structure defines variables that are used to fill the CGI structure prior to setting the hardware parameters. An entry for each of the relevant controls is present. The data structure is used by the "Generate" routine, which sends the value of each parameter to the hardware.

All constants mentioned can be found in the file OFILES.H or GGEN_DRV.H.

The following is a description of the main structure used in the program. Valid values for each of the parameter fields can be found in the description of the GGEN_DRIVER_INFO_TYPE structure elsewhere in this manual.

STRUCTURE OF THE CGI FILE

typedef struct {

char cgi_filename[30];	The name of the GageScope SIG file to read and generate. If no SIG file is found or specified, a 100-sample sine wave is generated.
int16 cgi_mode;	The operating mode to use. This should be GGEN_12BIT_MODE for a CG1100 and GGEN_16BIT_MODE for a CGT30.
int16 cgi_clock_mult;	The sample rate multiplier to use. This value and the cgi_clock_rate are used to form the sample rate.
int16 cgi_quietmode;	Whether or not to show progress of program.
int16 cgi_loop_count;	How many times to generate the pattern. A 0 means continuous mode.
int16 cgi_trigger_source;	The trigger source.
int16 cgi_trigger_level;	The trigger level.
int16 cgi_clock_rate;	The rate to use for the sample rate.
int16 cgi_filter;	Which output filter to use.
int16 cgi_gain;	Which output range to use.
int16 cgi_slope;	The trigger slope.
int16 cgi_ext_clock;	A flag to tell the program if external clock is being used or not.

}cgi;

Function Prototypes

The following functions have been written to help speed up the development process for the programmer using the CompuGen SDK and the CompuGen series of high-speed data generation cards.

int initialize(int quietMode);

This routine performs all the necessary calls in the proper order to initialize the driver. Certain modifications to this routine will be necessary if the programmer does not want the progression messages displayed.

cgi *open_Fcgi(int quietMode, LPSTR lpszCmdLine);

This routine opens the CGI file mentioned in the command line and fills the structure that will be used to set up the CompuGen hardware. If no CGI file is mentioned, the structure is filled with default values.

uInt16 *openSigFile(int quietMode, cgi *ret_struc, int32 *length);

This routine opens and reads the GageScope signal file (.SIG) mentioned in the CGI file. The data in the signal file is read into a buffer, which is used by the **generate** routine.

int generate (cgi *ret_struc, uInt16 *file_buf, int32 *length);

This routine uses the "gendef" structure named "board," which was previously loaded with the desired generation parameters, and generates the waveform. The sample program initializes these parameters

statically; it is the programmer's responsibility to perform the same task prior to generating data from the CompuGen card. Note the order in which these routines are called, as the order is important for the proper operation of the CompuGen hardware and for the CompuGen DLL driver to maintain correct information as to the size of memory available, the various waveform parameters, etc.

void sinewave (uInt16 *buffer, int32 *length);

This routine generates a sine wave mathematically and scales it to the proper values for the CG1100. The sine wave is stored in a buffer which is passed to the CompuGen via the **ggen_load_vram_from_buffer** DLL call in the **Generate** routine.

CompuGen SDK Basics: Visual BASIC

This basic introduction to the CompuGen SDK shows how to include the DLL in your code and then how to make use of these routines.

The simplest method of using the CompuGen SDK routines is to add the following DLL import code to a separate global module in your program. For example:

Declare Sub ggen_dump_data Lib "ggwindll.dll" ()

These commands can also be found as part of the file GLOBAL.BAS in the included sample program. Also, the file CGWINDLL.DLL should be located in your Windows directory (usually C:\WINDOWS) and the runtime Visual BASIC DLL, VBRUNxxx.DLL, should be in your Windows\System directory (usually C:\WINDOWS\SYSTEM).

When a CompuGen DLL routine is required, the name of the routine is used just as in a regular Visual BASIC subroutine call.

The Sample Program CG_OUT: Visual BASIC

The Visual BASIC sample program CG_OUT was created using the design tools that are available in Microsoft Visual BASIC 5.0. It features discrete and continuous data generation. Several routines and data structures have been added to the sample program that will allow the programmer to get started quickly.

Data Structures

The first data structure defines variables that are used prior to setting the hardware parameters. An entry for each of the relevant controls is present. The data structure is used by the **SetBoard** routine, which sends the value of each parameter to the hardware.

All constants mentioned can be found in the file GLOBAL.BAS.

```
Type gendef
        opmode          As Integer      Operating mode of the board.
        srindex         As Integer      Index to a sample rate table.
        output_range    As Integer      The output range to set.
        output_filter   As Integer      The output filter to set.
        generate_once   As Integer      One shot or continuous generation.
        t_source        As Integer      The trigger source to set.
        t_slope         As Integer      The trigger slope to set.
        t_level         As Integer      The trigger level to set.
        t_range         As Integer      The trigger range to set.
End Type
```

The sample rate table structure is not needed, but it is included in the file GLOBAL.BAS and initialized in the **Initialize** subroutine as a convenience to the programmer.

```
Type srtype
        rate    As Integer      Rate used for setting the CompuGen.
        mult    As Integer      Multiplier used for setting the CompuGen.
        flag    As Integer      Flag to indicate which board supports which sample rate.
        calc    As Long         Time between samples (in ns).  Used in calculating the number of
                                points.
        text    As String       Text associated with the current settings of the CompuGen.
End Type
```

Global srtable As srtype

Contents of the srtable data structure for the sample program.

index	rate	mult	flag	calc	text
00	GEN_RATE_1	GGEN_HZ	&H0003	1000000000	1 Hz
01	GGEN_RATE_2	GGEN_HZ	&H0003	500000000	2 Hz
02	GEN_RATE_5	GGEN_HZ	&H0003	200000000	5 Hz
03	GGEN_RATE_10	GGEN_HZ	&H0003	100000000	10 Hz
04	GGEN_RATE_20	GGEN_HZ	&H0003	50000000	20 Hz
05	GGEN_RATE_50	GGEN_HZ	&H0003	20000000	50 Hz
06	GGEN_RATE_100	GGEN_HZ	&H0003	10000000	100 Hz
07	GEN_RATE_200	GGEN_HZ	&H0003	5000000	200 Hz
08	GGEN_RATE_500	GGEN_HZ	&H0003	2000000	500 Hz
09	GGEN_RATE_1	GGEN_KHZ	&H0003	1000000	1 kHz
10	GGEN_RATE_2	GGEN_KHZ	&H0003	500000	2 kHz
11	GGEN_RATE_5	GGEN_KHZ	&H0003	200000	5 kHz
12	GGEN_RATE_10	GGEN_KHZ	&H0003	100000	10 kHz
13	GGEN_RATE_20	GGEN_KHZ	&H0003	50000	20 kHz
14	GGEN_RATE_50	GGEN_KHZ	&H0003	20000	50 kHz
15	GGEN_RATE_100	GGEN_KHZ	&H0003	10000	100 kHz
16	GGEN_RATE_200	GGEN_KHZ	&H0003	5000	200 kHz
17	GGEN_RATE_500	GGEN_KHZ	&H0003	2000	500 kHz
18	GGEN_RATE_1	GGEN_MHZ	&H0003	1000	1 MHz
19	GGEN_RATE_2	GGEN_MHZ	&H0003	500	2 MHz
20	GGEN_RATE_5	GGEN_MHZ	&H0003	200	5 MHz
21	GGEN_RATE_10	GGEN_MHZ	&H0003	100	10 MHz
22	GGEN_RATE_20	GGEN_MHZ	&H0003	50	20 MHz
23	GGEN_RATE_40	GGEN_MHZ	&HC003	25	40 MHz

Function Prototypes

The following functions have been written to help speed up the development process for the programmer using the CompuGen SDK and the CompuGen series of high-speed data generation cards.

Sub SetDefaultBoardLocation(ByVal seg As Integer, ByVal ind As Integer)

This routine is called if the DLL routine **ggen_read_config_file** returns false, indicating the board location configuration file is corrupt or missing. A default segment and index are passed to this routine and the global data structure **ggen_board_location** is updated prior to calling the DLL routine **ggen_driver_initialize**. The routine also converts the two constants, defined in GLOBAL.BAS, passed to it and returns the text equivalent for the type and memory size of the CompuGen board found.

function InitBoard () As Integer

This routine performs all the necessary calls in the proper order to initialize the driver. Some modification to this routine will be necessary if the programmer does not want the progression messages displayed. As written, the sample program will abort if no CompuGen board is found or if **ggen_select_board** fails. The probable cause of this is an incorrect segment or index value in the configuration file. The problem can be corrected by running the CGWin.exe or CGT30.exe utility. The routine also converts two constants, defined in GLOBAL.BAS, representing the name and memory size of the CompuGen board into their text equivalents.

Private Sub cmdGenerate_Click()

This routine uses the "CGI" structure, which was previously read from the text file and loaded with the desired capture parameters, and generates the waveform. The sample program initializes these parameters statically; it is the programmer's responsibility to perform the same task prior to generating data from the CompuGen card. Note the order in which these routines are called, as the order is important for the proper operation of the CompuGen hardware and for the CompuGen DLL driver to maintain correct information as to the size of memory available to each channel, the various waveform parameters, etc.

Function Sinewave1100 () As Integer

This routine is an example of generating a waveform and storing it in a buffer. The contents of the buffer are passed to the CompuGen via the subroutine AnalogGenerateStart.

Global Routines

The following section provides an alphabetical listing of all of the Global Routines, complete with syntax, remarks, return value, other routines and sample programs to consult for additional information, and examples.

Global Routines: Variable Definitions for Examples

This section lists the definitions assumed to be present for the various examples listed with the descriptions for the Global Routines.

```
Global Const POINTS1 = 16383

Global i                          As Integer
Global expected                   As Integer
Global boards                     As Integer
Global driver_info                As ggen_driver_info_type
Global address                    As Integer
Global analog                     As Integer
Global digital                    As Integer
Global a_buffer (POINTS1+1)       As Integer
Global ggen_board_location (GGEN_B_L_BUFFER_SIZE)       As Integer
Global ret                        As Integer
Global tempstr                    As String
```

The following definitions are used for some of the examples which have their roots in the GGDLLDEM.BAS program. The additional "types" are defined in the section above, S*ample Programs.*

Initialization of the board structure. The board structure is of type **gendef**.

```
board.opmode          =  GGEN_DUAL_MODE
board.srindex         =  21
board.output_range    =  GGEN_TIMES_1
board.output_filter   =  1   ' True
board.generate_once   =  0  ' False
board.t_source        =  GGEN_SOFTWARE
board.t_slope         =  GGEN_POSITIVE
board.t_level         =  128
board.t_range         =  GGEN_TIMES_1


board_type            = GGEN_ASSUME_NOTHING
board_memory          = 0
wave_length           = 0
Static    wave_buffer (GGEN_DOUBLE_DEPTH) As Integer
```

ggen_abort

Syntax

C
#include <ggen_drv.h>
void ggen_abort (void);

Visual BASIC
sub ggen_abort ()

Remarks

ggen_abort is used to regain control of the CompuGen board, primarily in the event that a trigger event never occurs. This routine forces the board to a non-busy state, thus allowing the board to be re-configured, rearmed and/or the memory to be accessed. This routine is also used to stop data generation at the end of a program or to enable a new pattern to be loaded and generated.

Once a pattern has been generated in continuous mode, it will continue until it has been aborted or the computer is reset.

Return Value

None.

See also

ggen_busy

Examples

C
ggen_abort ();

Visual BASIC
ggen_abort

ggen_driver_initialize

Syntax

C
#include <ggen_drv.h>
int16 ggen_driver_initialize (uInt16 far *records, uInt16 memory);

Visual BASIC
Function ggen_driver_initialize (records As Integer, ByVal memory As Integer) As Integer

Remarks

ggen_driver_initialize will fully configure each board found in the system. From then on a call to **ggen_select_board** will be required to access any board .

The **records** parameter is assumed to be an uninitialized uInt16 array supplied by the application program. This array must be **GGEN_B_L_BUFFER_SIZE** (48) bytes long. The format of the array is that the first **GGEN_B_L_STATUS_START** (32) uInt16s are for the board segment and index values, each pair occupies **GGEN_B_L_ELEMENT_SIZE** (2) uInt16s, for each of the possible **GGEN_B_L_MAX_CARDS** (16) boards.

A status field is provided for each potential board location, which is **GGEN_B_L_STATUS_SIZE** (1) uInt16s in length. The values for the status field are constants that correspond to bit positions in the status field and must be masked to determine which errors occurred when initializing the board. The low nibble is for problems with the segment and index.

CompuGen **ggen_board_location** (defined in GGEN_DRV.H file) array "pseudo structure"

array index:	0	1	2	3	...	30	31	32	33	34	35	...	46	47
meaning:	S1	I1	S2	I2	...	S16	I16	E1	E2	E3	E4	...	E15	E16

where: Sx = segment for board x,
Ix = index for board x,
Ex = returned board type or error status for board x,

The possible error codes for the status fields are:

GGEN_BAD_LSB_SEGMENT (0x0001) means that the low order byte of the segment was not equal to zero.

GGEN_BAD_MSB_SEGMENT (0x0002) is used when the segment is either less than A000 hex or greater than DF00 hex (the valid area in the memory map reserved for slot resources is 0xA0000 to 0xDFFFF).

GGEN_BAD_LSB_INDEX (0x0004) is set when the least significant bit of the index is not zero.

GGEN_BAD_MSB_INDEX (0x0008) is used when the high order byte of the index is either equal to 00 hex or greater than 03 hex (the valid area in the I/O map reserved for slot resources is 0100 hex to 03ff hex).

GGEN_DETECT_FAILED (0x0010) is set when no CompuGen board is detected at the specified I/O address. The usual cause is either an I/O conflict or the wrong address being specified in the **GAGE_GEN.INC** file.

GGEN_MEMORY_FAILED (0x0020) is set when a CompuGen board was detected but the memory test fails. The usual cause is a memory conflict. This can often be resolved by booting clean or reserving the memory segment you wish to use with a memory manager. For example, EMM386.EXE will often reserve all memory between the top of DOS and 1 Megabyte for itself. Using the following line in your config.sys file will prevent this from happening:

```
DEVICE=C:\WIN95\EMM386.EXE NOEMS X=D000-D1FF
```

This tells the memory manager not to use expanded memory and not to touch the memory region 0xD000 to 0xD1FF. The example above assumes that you are in Windows 95 and that the GAGE_GEN.INC file is set to use memory address 0xD000.

Once the board has been initialized, the board type can be found by calling **ggen_driver_info** and reading the **board_type** field. Possible values for the board type are:

GGEN_ASSUME_CG1100	0x0400
GGEN_ASSUME_CGT30	0x1000

The **memory** parameter allows the memory self-test to be disabled by supplying the size of the memory in kilobytes of the installed board(s). The constant **GGEN_MEMORY_SIZE_TEST** (0) will force the memory test to be performed. Note that the memory size for all installed boards must be the same; otherwise a conflict can occur and the data returned may be invalid for the boards with the incorrect memory size assigned.

If a memory size is used instead, the driver will assume that the installed board has that amount of memory. Specifying the wrong size can lead to improper results from the driver.

If an incorrect memory size is found, then the status field for the segment and index record in question will have the **GGEN_BAD_MEMORY_SIZE** bit set. If the status is zero and the corresponding segment and index record are non-zero, then this particular board was properly initialized. If, however, the status is zero and the segment and index record are also zero, then the "board" is the premature end of the **records** array. By default the first board found will be selected.

A local data structure in the DLL is created for each board. To allow access to this structure, the routine **ggen_get_driver_info** has been implemented. This routine queries the DLL about information on the current selected board. It is advised that this method be maintained for compatibility with future releases of the CompuGen SDKs rather than using the internal DLL variables directly.

Return Value

The return value is the number of CompuGen boards found and initialized. If a negative number is returned, then that number of boards was found in the **GAGE_GEN.INC** file but could not be initialized. The routine **ggen_get_error_code** can then be called to determine why.

See also

ggen_read_config_file, ggen_get_error_code, ggen_get_driver_info and ggen_select_board

Examples

C

```
ggen_driver_initialize ((uInt16 far *)ggen_board_location, GGEN_MEMORY_SIZE_TEST);
```

Visual BASIC

```
ggen_driver_initialize(ggen_board_location(0), ms)
```

ggen_dump_data

Syntax

C
#include <ggen_drv.h>
void ggen_dump_data (void);

Visual BASIC
Sub ggen_dump_data ()

Remarks

This routine operates on the current board only. Different boards are selected by calling the **ggen_select_board** routine. It sets the CompuGen hardware to allow data generation as soon as a trigger has been received. Note that this routine alone will not cause data to be generated—a trigger event, either hardware or software, must occur before the data is generated. The routine must be called for each CompuGen board in the system that is required to generate data.

Return Value

None.

See also

ggen_busy

Examples

C
ggen_dump_data ();

Visual BASIC
ggen_dump_data

ggen_ext_clock_ctrl_50ohm_on

Note: This routine is for boards with the External Clock option only.

Syntax

C
#include <ggen_drv.h>
void ggen_ext_clock_ctrl_50ohm_on (int16 on);

Visual BASIC
Sub ggen_ext_clock_ctrl_50ohm_on (ByVal on As Integer)

Remarks

The **ggen_ext_clock_ctrl_50ohm_on** routine is used to tell the driver that the external clock will be 50 ohm AC terminated. Normally, the CompuGen expects a TTL level external clock. If the **on** parameter is non-zero, the driver will assume that any external clock is 50 ohm terminated. If the parameter is 0, the driver will assume that it is not 50 ohm terminated. The rate and multiplier must be set to External Clock in the call to **ggen_generate_mode** for this routine to have any effect. This call must be made before the call to **ggen_generate_mode**.

The default setting is 50 ohm termination is off.

Note that you must have the External Clock option for this routine to have any effect.

Return Value

None

See also

ggen_generate_mode

Examples

C
ggen_ext_clock_ctrl_50ohm_on (1);

Visual BASIC
ggen_ext_clock_ctrl_50_ohm_on (1)

ggen_ext_trig_ctrl_50ohm_on

Syntax

C
#include <ggen_drv.h>
void ggen_ext_trig_ctrl_50ohm_on (int16 on);

Visual BASIC
Sub ggen_ext_trig_ctrl_50ohm_on (ByVal on As Integer)

Remarks

The **ggen_ext_trig_ctrl_50ohm_on** routine is used to tell the driver that the external trigger will be 50 ohm terminated. If the **on** parameter is non-zero, the driver will assume that any external trigger is 50 ohm terminated. If the parameter is 0, the driver will assume that it is not 50 ohm terminated. The trigger source must be set to external trigger in the call to **ggen_trigger_control** for this routine to have any effect. This call must be made before the call to **ggen_trigger_control**.

The default is 50 ohm termination is off.

Return Value

None

See also

ggen_trigger_control

Examples

C
ggen_ext_trig_ctrl_50ohm_on (1);

Visual BASIC
ggen_ext_trig_ctrl_50_ohm_on (1)

ggen_force_pattern

Syntax

C
#include <ggen_drv.h>
void ggen_force_pattern (int32 offset, void* buffer, uInt32 size);

Visual BASIC
Sub ggen_force_pattern (ByVal offset As Long, buffer As Integer, By Val length As Long)

Remarks

The **ggen_force_pattern** routine can access the on-board memory directly. It is used to modify a part of a previously loaded pattern directly without the need to reload the pattern. It is useful when only a small part of the pattern needs to be changed. The **ggen_dump_data** routine must be called in order to make the change effective.

The **offset** parameter is the offset in bytes from the beginning of the on-board memory in which to put the new pattern. The **buffer** parameter is a pointer to the new pattern used to modify the original pattern. The **size** parameter is the size in bytes of the new pattern.

Return Value

None

See also

ggen_load_vram_from_buffer

Examples

C
ggen_force_pattern (200, pBuf, 16);

Visual BASIC
ggen_force_pattern (200, pBuf(0), 16)

ggen_gate_lc_ctrl_50ohm_on

Note: This routine is for boards with the Gated Generation option only.

Syntax

C
#include <ggen_drv.h>
void ggen_gate_lc_ctrl_50ohm_on (int16 on);

Visual BASIC
Sub ggen_gate_lc_ctrl_50ohm_on (ByVal on As Integer)

Remarks

The **ggen_gate_lc_ctrl_50ohm_on** routine is used to tell the driver that a gate signal will be 50 ohm terminated. If the **on** parameter is non-zero, the driver will assume that any external gate is 50 ohm terminated. If the parameter is 0, the driver will assume that it is not 50 ohm terminated.

The default setting is 50 ohm termination is off.

Note that you must have the Gated Generation option for this routine to have any effect. The Gated Generation option allows the user to control the analog output by gating the clock that transfers data from the memory to the DAC.

Return Value

None

See also

ggen_trigger_control

Examples

C
ggen_gate_lc_ctrl_50ohm_on (1);

Visual BASIC
ggen_gate_lc_ctrl_50_ohm_on (1)

ggen_generate_mode

Syntax

C
```
#include <ggen_drv.h>
int16 ggen_generate_mode (int16 op_mode, int16 rate, int16 multiplier);
```

Visual BASIC
```
Function ggen_generate_mode (ByVal op_mode As Integer,
                             ByVal s_rate As Integer,
                             ByVal multiplier As Integer) As Integer
```

Remarks

This routine sets up the operating mode, rate and multiplier values for a CompuGen board. The **op_mode** sets the operating mode for the CompuGen board. The value should be **GGEN_12BIT_MODE** for the CompuGen 1100 and **GGEN_16BIT_MODE** for the CompuGen T30.

The **rate** and **multiplier** parameters together set the sample rate for the CompuGen. Allowable values for the **rate** are:

Constant	Value	
GGEN_RATE_1	1	
GGEN_RATE_2	2	
GGEN_RATE_5	5	
GGEN_RATE_10	10	
GGEN_RATE_20	20	
GGEN_RATE_50	50	
GGEN_RATE_100	100	
GGEN_RATE_200	200	
GGEN_RATE_500	500	
GGEN_RATE_40	40	Only used with GGEN_MHZ.
GGEN_RATE_80	80	Only used with GGEN_MHZ and CG1100

The sample rates increase in a 1, 2, 5 pattern (i.e. 1 kHz, 2 kHz, 5 kHz, 10 kHz, 20 kHz, etc.) up to 40 MHz. A sample rate of 80 MHz is only valid on the CompuGen 1100.

Allowable values for the **multiplier** are:

Constant	Value
GGEN_HZ	1
GGEN_KHZ	2
GGEN_MHZ	3
GGEN_GHZ	4
GGEN_EXTERNAL_CLOCK	5
GGEN_SOFTWARE_CLOCK	6

If external clock is used, both the **rate** and **multiplier** should be set to **GGEN_EXTERNAL_CLOCK**.

The **GGEN_SOFTWARE_CLOCK** multiplier can be used to single-step through a pattern on the CompuGen T30.

Return Value

A non-zero (True) value if the call was successful and a zero (False) if an error occurred. The type of error can be determined by calling **ggen_get_error_code**.

See also

ggen_output_control and ggen_trigger_control

Examples

C

```
ret = ggen_generate_mode (GGEN_12BIT_MODE,  GGEN_RATE_40, GGEN_MHZ);
```

Visual BASIC

```
ret = ggen_generate_mode (GGEN_12BIT_MODE, GGEN_RATE_40,  GGEN_MHZ)
```

ggen_get_boards_found

Syntax

C
#include <ggen_drv.h>
int16 ggen_get_boards_found (void);

Visual BASIC
Function ggen_get_boards_found () As Integer

Remarks

After calling **ggen_driver_initialize** to configure all board found in the system, **ggen_get_boards_found** will return the number of CompuGen boards found in the system.

Return Value

The number of CompuGen boards currently installed by the DLL.

See also

ggen_driver_initialize

Examples

C
boards_found = ggen_get_boards_found ();

Visual BASIC
boards_found = ggen_get_boards_found

ggen_get_config_filename

Syntax

C
#include <ggen_drv.h>
int16 ggen_get_config_filename (LPSTR cfgfn);

Visual BASIC
Function ggen_get_config_filename (ByVal board_loc_file As String) As Integer

Remarks

ggen_get_config_filename determines the complete path to the configuration file GAGE_GEN.INC. This file contains the memory segment and I/O address pair used by each CompuGen board in the system. By default, the DOS CompuGen drivers look for the GAGE_GEN.INC file in the current working directory. The Windows drivers expect to find the GAGE_GEN.INC file in the Windows directory.

cfgfn is a string variable that must be long enough to hold the returned path.

The GAGE_GEN.INC file can be created with either the CGINST.EXE utility (for CG1100 and CGT30 under DOS), CGWIN.EXE (for CG1100 under Windows) or CGT30.EXE (for CGT30 under Windows)

Return Value

A true value is returned, if the routine successfully returned the configuration filename.

See also

ggen_read_config_file

Examples

C
ret = ggen_get_config_filename ((LPSTR)(board_loc_file));

Visual BASIC
i = ggen_get_config_filename (board_loc_file)

ggen_get_driver_info

Syntax

C
#include <ggen_drv.h>
void ggen_get_driver_info ((ggen_driver_info_type far*)(driver_info));

Visual BASIC
Sub ggen_get_driver_info (driver_info As ggen_driver_info_type)

Remarks

ggen_get_driver_info fills a structure or record with the relevant information from the driver, with variables as to the current settings of the current CompuGen. This structure is a subset of the data structure used internally by the driver and includes only those values that have meaning to the control program. The structure returned from this routine is used to determine the settings of the currently selected CompuGen board. It should **not** be used to change the settings of the driver. If driver settings need to be changed, they should be done so by calling the appropriate DLL routine.

The ggen_driver_info_type structure is shown below. A similar structure is defined for Visual BASIC.

```
typedef struct  {
        uInt16          index;
        uInt16          segment;
        uInt16          selector;
        uInt16          offset;
        uInt8   far     *memptr;
        int16           mode;
        int16           rate;
        int16           multiplier;
        int16           oneshot;    /* endless or not endless */
        int32           max_memory;
        int16           board_type;
        int16           o_range;
        int16           o_filter;
        int16           t_source;
        int16           t_slope;
        int16           t_range;
        int16           t_level;
        int16           ext_clk_50ohm;
        int16           ext_trig_50ohm;
        int16           gate_lc_50ohm;
        int16           inter_ctrl_50ohm;
}       ggen_driver_info_type;
```

Return Value

None, but the structure or record is filled with the proper information from the DLL's structure.

See also

Description of the structure on page 16.

Examples

C

```
ggen_driver_info_type driver_info
ggen_get_driver_info (&driver_info);
```

Visual BASIC

```
ggen_get_driver_info driver_info
```

ggen_get_error_code

Syntax

C
#include <ggen_drv.h>
int16 ggen_get_error_code (void);

Visual BASIC
Function ggen_get_error_code () As Integer

Remarks

ggen_get_error_code returns the error code associated with the last call to the CompuGen DLL.

Return Value

The error that occurred and the board that caused the error. This function returns a value, which is encoded with the high byte containing the board in error, and the low byte is set to a defined error constant. These constants are:

Constant	Value (in hex)
GGEN_NO_ERROR	0x00
GGEN_NO_SUCH_MODE	0x01
GGEN_NO_SUCH_SAMPLE_RATE	0x02
GGEN_INVALID_SAMPLE_RATE	0x03
GGEN_NO_SUCH_BOARD	0x04
GGEN_NO_SUCH_GAIN	0x05
GGEN_NO_SUCH_TRIG_SLOPE	0x06
GGEN_NO_SUCH_TRIG_SOURCE	0x07
GGEN_LOAD_INVALID_CHANNEL	0x08
GGEN_LOAD_INVALID_LENGTH	0x09
GGEN_NO_PATTERN_LOADED	0x0A
GGEN_NO_SUCH_SYNC_OUT	0x0B
GGEN_INVALID_CLK_SOURCE	0x0C
GGEN_NO_SUCH_TRIG_RANGE	0x0D
GGEN_MISC_ERROR	0xff

For example, an error code of 0x105 signifies that the output gain for board 1 is invalid.

In addition to these errors, there are other errors that can occur during initialization. These errors can occur after calling ggen_driver_initialize:

Constant	Value (in hex)
GGEN_BAD_LSB_SEGMENT	0x0001
GGEN_BAD_MSB_SEGMENT	0x0002
GGEN_BAD_LSB_INDEX	0x0004
GGEN_BAD_MSB_INDEX	0x0008
GGEN_DETECT_FAILED	0x0010
GGEN_MEMORY_FAILED	0x0020
GGEN_BAD_MEMORY_SIZE	0x0040

See also

Nothing.

Examples

C

```
ret = ggen_get_error_code ();
```

Visual BASIC

```
ret = ggen_get_error_code()
```

ggen_load_vram_from_buffer

Syntax

C

```
#include <ggen_drv.h>
int16 ggen_load_vram_from_buffer (int32 length,  uInt16 far *buffer, int32  loop_number,  int16 end_flag);
```

Visual BASIC

```
Sub ggen_load_vram_from_buffer (ByVal length As Long, ptr As Integer, ByVal loop_number As Long,
                ByVal end_flag As Integer) As Integer
```

Remarks

This routine is used to load the CompuGen internal buffer with the data passed to it. The **length** parameter is the number of int16 values in the user-supplied buffer. The **buffer** parameter is a uInt16 buffer which holds the pattern to be generated. The **loop_number** is used to tell the driver how many times to generate the current pattern. This parameter is for future use and should currently be set to 1. The number of times to generate should be controlled by using the **ggen_set_outer_loop_counter** routine. The **end_flag** parameter is a flag used to tell the driver which is the last pattern to be loaded. This parameter should usually be set to 1. If you have a larger pattern, it can be faster to load it into the CompuGen memory in chunks. In this case, each chunk should have an end flag of 0, except for the last chunk, which should have an end flag of 1.

The CompuGen 1100 uses a uInt16 buffer to load the pattern onto the on-board memory. The CompuGen T30 can use either a 32 bit buffer or a 16 bit buffer. Regardless of which is used, the **length** parameter is the number of 16 bit values in the buffer. If a 16 bit buffer is used with the CompuGen T30, the data is loaded in high word, low word format.

The CG1100 expects a buffer with values ranging from 0 to 4095. A 0 value represents the highest voltage in the current output range. A value of 4095 represents the lowest voltage in the current output range.

The CGT30 can be filled either with a buffer of 16 bit values or a buffer of 32 bit values (cast as int16 * in the call). In either case, the **length** parameter must be the number of 16 bit values in the buffer. If a buffer of 16 bit values is used, each two consecutive 16 bit values are treated as the low-high halves of a 32 bit word. For example, a counter pattern can be loaded as follows:

Value	Binary (32 bits)
0	00000000000000000000000000000000
1	00000000000000000000000000000001
2	00000000000000000000000000000010
3	00000000000000000000000000000011
4	00000000000000000000000000000100
5	00000000000000000000000000000101
6	00000000000000000000000000000110
7	00000000000000000000000000000111

In this case, the length parameter would be 16 (eight 32 bit values = sixteen 16 bit values).

Value	Binary (16 bits)
0	0000000000000000
0	0000000000000000
1	0000000000000001
2	0000000000000000
3	0000000000000010
4	0000000000000000
5	0000000000000011
6	0000000000000000
7	0000000000000100
8	0000000000000000
9	0000000000000101
10	0000000000000000
11	0000000000000110
12	0000000000000000
13	0000000000000111
14	0000000000000000

In this case, the buffer is filled with 16 bit values and the length is also 16.

Return Value

A 1 is returned upon successful completion of **ggen_load_vram_from_buffer**. A negative number is returned if an error occurred. The error code may be obtained by calling **ggen_get_error_code**.

See also

ggen_set_outer_loop_counter

Examples

C

```
ggen_load_vram_from_buffer (wave_length, (uInt16 *)wave_buffer, 1, 1);
```

Visual BASIC

```
ret = ggen_load_vram_from_buffer (SizeFile, buffer(0), 1, 1)
```

ggen_memory_test

Syntax

C
#include <ggen_drv.h>
int32 ggen_memory_test (uInt32 offset, uInt32 size);

Visual BASIC
Function ggen_memory_test (ByVal offset As Long, ByVal size As Long) As Long

Remarks

The routine **ggen_memory_test** is used to test the on-board memory of a CompuGen board. The routine will randomly create and load a pattern into the CompuGen's memory, then read it back and compare it to the original pattern.

The **offset** parameter is used to tell the routine where to begin testing. It is the offset in bytes from the beginning of the CompuGen's on-board memory. The **size** parameter is the number of bytes to be tested.

Return Value

If no errors occur, a -1 is returned. If there are errors (i.e. the pattern read back is not the same as the pattern loaded onto the CompuGen), the offset where the first error occurred is returned.

See also

ggen_load_vram_from_buffer

Examples

C
ret = ggen_memory_test (0, 32768);

Visual BASIC
ret = ggen_memory_test (0, 32768)

ggen_output_control

Syntax

C
#include <ggen_drv.h>
int16 ggen_output_control (int16 range, int16 filter);

Visual BASIC
Function ggen_output_control (ByVal range As Integer, ByVal filter As Integer) As Integer

Remarks

ggen_output_control sets the range and filter parameters of the analog output signal. The **range** parameter sets the gain of the generated signal. Output levels to ±5 volts (assuming a 50 ohm load) are possible. The **filter** parameter controls which analog filter is used. The analog filter is used to smooth out the signal above certain frequencies. Allowable values for the gain are:

Constant	Value
GGEN_TIMES_5	1
GGEN_TIMES_2	2
GGEN_TIMES_1	3
GGEN_TIMES_0_POINT_5	4
GGEN_TIMES_0_POINT_2	5
GGEN_TIMES_0_POINT_1	6

These ranges are for the CompuGen 1100. The CompuGen T30 only outputs a TTL signal. Note that the CompuGen 1100 assumes a 50 ohm load on the output. If there is none, the output will be double the expected range.

The valid values for the output filter are:

Constant	Value
GGEN_NO_FILTER	0
GGEN_FILTER_20MHZ	1
GGEN_FILTER_5MHZ	3

The filter values have no effect on a CompuGen T30. Valid values should still be used in the call to **ggen_output_control** if a CompuGen T30 is being used.

Return Value

A TRUE (non-zero) value is returned after a successful call. A FALSE (zero) value is returned if an error is encountered. The routine **ggen_get_error_code** can then be called to obtain the error code.

See also

ggen_generate_mode and ggen_get_error_code

Examples

C

```
ret = ggen_output_control (board.output_range, board.output_filter);
```

Visual BASIC

```
ret  = ggen_output_control (board.output_range, board.output_filter)
```

ggen_pad_value

Note: This routine is for the CGT30 only.

Syntax

C
#include <ggen_drv.h>
void ggen_pad_value (int32 control, int32* value);

Visual BASIC
ggen_pad_value (ByVal control As Long, value As Long)

Remarks

This routine is used to pad the starting and ending values being generated on a CompuGen T30 in single shot mode to ensure that erroneous data in the pipeline does not enter into the data stream before the actual pattern is generated. If used, this routine should be called before the call to **ggen_load_vram_from_buffer** for it to have any effect. The **control** parameter is the action to take and the **value** parameter is either the value to use for padding (if the control parameter is one of the set constants), or the currently used padding value (if the control parameter is one of the get constants).

The available constants to use for the **control** parameter are:

Constant	Meaning
PAD_SET_START (0)	Pad the value in start position
PAD_GET_START (1)	Return the padding value for start position
PAD_SET_END (2)	Pad the value at the end position
PAD_GET_END (3)	Return the padding value for the end position

The available constants for the **value** parameter are:

Constant	Value	
PAD_VALUE_LOW	(0)	pad with 0
PAD_VALUE_HIGH	(1)	pad with 1
PAD_VALUE_ADJACENT	(2)	pad with the value of adjacent sample
PAD_VALUE_PREVIOUS	(3)	pad with the previous padding value

Return Value

None

See also

ggen_load_vram_from_buffer

Examples

C

```
ggen_pad_value (PAD_SET_START, PAD_VALUE_LOW);
```

Visual BASIC

```
ggen_pad_value (PAD_SET_START,  PAD_VALUE_LOW)
```

ggen_read_config_file

Syntax

C
#include <ggen_drv.h>
int16 ggen_read_config_file (LPSTR far_filename, uInt16 far *records);

Visual BASIC
Function ggen_read_config_file (ByVal filename As String, records As Integer) As Integer

Remarks

ggen_read_config_file reads the configuration file **GAGE_GEN.INC** and stores the data in an array of uInt16s. The parameter **filename** is a text string that tells the routine the path and name of the file that contains the board indexes and starting segment values for each of the installed CompuGen boards. This configuration file is named GAGE_GEN.INC. The SDK for DOS assumes the default location of this file is in the current directory. The SDKs for Windows assume that the default location is in the Windows directory. The full path to the file can be obtained by calling **ggen_get_config_filename**. Alternatively, the application program can provide its own path in the **filename** parameter.

Under DOS, the GAGE_GEN.INC file can be created with either the CGINST.EXE utility that comes with the SDK for DOS. Under Windows, use either CGWIN.EXE (on the CompuGen for Windows disk for CompuGen 1100) or CGT30.EXE (on the CGT30 Software disk for CompuGen T30). These utilities come with your CompuGen board.

The **records** parameter is assumed to be an uninitialized uInt16 array supplied by the application program. This array must be **GGEN_B_L_BUFFER_SIZE** (48) bytes long. The format of the array is as follows: the first **GGEN_B_L_STATUS_START** (32) uInt16s are for the board segment and index values, and each pair occupies **GGEN_B_L_ELEMENT_SIZE** (2) uInt16s, for each of the possible **GGEN_B_L_MAX_CARDS** (16) boards.

A status field is provided for each potential board location, which is **GGEN_B_L_STATUS_SIZE** (1) uInt16s in length. The values for the status field are constants that correspond to bit positions in the status field and must be masked to determine which errors occurred when initializing the board. The low nibble is for problems with the segment and index.

CompuGen **ggen_board_location** (defined in GGEN_DRV.H file) array "pseudo structure"

array index:	0	1	2	3	...	30	31	32	33	34	35	...	46	47
meaning:	S1	I1	S2	I2	...	S16	I16	E1	E2	E3	E4	...	E15	E16

where: Sx = segment for board x,
Ix = index for board x,
Ex = returned board type or error status for board x

The possible error codes for the status fields are:

GGEN_BAD_LSB_SEGMENT (0x0001) means that the low order byte of the segment was not equal to zero.

GGEN_BAD_MSB_SEGMENT (0x0002) is used when the segment is either less than A000 hex or greater than DF00 hex (the valid area in the memory map reserved for slot resources is 0xA0000 to 0xDFFFF).

GGEN_BAD_LSB_INDEX (0x0004) is set when the least significant bit of the index is not zero.

GGEN_BAD_MSB_INDEX (0x0008) is used when the high order byte of the index is either equal to 00 hex or greater than 03 hex (the valid area in the I/O map reserved for slot resources is 0100 hex to 03ff hex).

Return Value

The return value represents the number of records initialized if the return value is greater than 0. If the number is less than zero then the number corresponds to the error encountered. The errors are:

-1, file does not exist
-2, file cannot be opened
-3, file size cannot be determined
-4, file size modulo four is not zero
-5, file size indicates that more boards than the DLL supports are present
-6, file cannot be read successfully
-7, file cannot be closed
0, reserved for future use

See also

ggen_driver_initialize and ggen_get_config_filename

Examples

C

```
expected = ggen_read_config_file ((LPSTR)("GAGE_GEN.INC"), (uInt16 far*)(&ggen_board_location));
```

Visual BASIC

```
i = ggen_get_config_filename (board_loc_file)
expected = ggen_read_config_file (board_loc_file, ggen_board_location (0))
```

ggen_select_board

Syntax

C
```
#include <ggen_drv.h>
int16 ggen_select_board (int16 board);
```

Visual BASIC
```
Function ggen_select_board (ByVal board As Integer) As Integer
```

Remarks

ggen_select_board is used to select a CompuGen board and make it the current board. Most CompuGen driver functions are performed on the current board.

Return Value

The integer returned equals the value passed to the function as **board**. If an error occurs or the value passed to the function exceeds the number of boards installed in the system, then the return value does not equal **board** and **ggen_get_error_code** may be called to obtain the error code.

See also

ggen_driver_initialize

Examples

C
```
if (ggen_select_board (1) != 1)
        sprintf(str, "Error while selecting board!\n", );
```

Visual BASIC
```
ret = ggen_select_board (1)
If ret <> 1 Then
        MsgBox "Error while selecting board # 1", 48, "Error"
        Exit Sub
End If
```

ggen_set_clock_level

Note: This routine is for CompuGen boards with the External Clock option only.

Syntax

C
```
#include <ggen_drv.h>
void ggen_set_clock_level  (uInt16 level);
```

Visual BASIC
```
Sub ggen_set_clock_level (ByVal As Integer)
```

Remarks

This routine can be used to change the external clock threshold level on a CompuGen board. The threshold level is the level at which data will be clocked out of the CompuGen board. The level is a uInt16 value ranging from 0 to 255, where 0 represents -2.5 volts and 255 represents 2.5 volts. Note that the differential between the actual clock signal and the threshold value should not be too large for proper operation of the CompuGen board.

The default external clock threshold value is 1.2 volts.

Return Value

None.

See also

ggen_generate_mode

Examples

C
```
ggen_set_clock_level (192);
```

Visual BASIC
```
ggen_set_clock_level (192)
```

ggen_set_filter_on

Note: This routine is for CG1100 only.

Syntax

C
#include <ggen_drv.h>
int16 ggen_set_filter_on (int16 filter);

This routine is not available in Visual BASIC.

Remarks

This routine is used to set the analog filter for the CompuGen 1100. The analog filter is used to smooth out the generated signal above certain frequencies. There are no filters available for CGT30. It is recommended that the filter be set by calling **ggen_output_control,** rather than by calling **ggen_set_filter_on**.

The available filter values are:

Constant	Value
GGEN_NO_FILTER	0
GGEN_FILTER_20MHZ	1
GGEN_FILTER_5MHZ	3

Return Value

A 0 is returned if an error occurred while trying to set the filter. **ggen_get_error_code** can then be called to determine the error. If 1 is returned, if the routine completed successfully.

See also

ggen_output_control

Examples

C
ret = ggen_set_filter_on (GGEN_FILTER_5MHZ);

ggen_set_independent_operation

Syntax

C
#include <ggen_drv.h>
void ggen_set_independent_operation (int16 on);

Visual BASIC
Sub ggen_set_independent_operation (ByVal on As Integer)

Remarks

This routine sets all CompuGen boards found in the system to work in Multiple/Independent mode. Multiple/Independent mode means that each CompuGen board can have a different sample rate and does not have to generate data at the same time. By default, the driver will assume that multiple boards are in Master/Slave mode. In Master/Slave mode, multiple CompuGen boards share the same sample rate clock and trigger. In this mode, all boards in the system will begin generation when the Master board (board 1) is triggered.

The **ggen_set_independent_operation** routine should be called after the driver has been initialized with **ggen_driver_initialize**. If it is called from elsewhere in the program, all parameters should be reset by calling the appropriate functions. If the parameter **on** is non-zero, the driver will set multiple boards to Multiple/Independent mode. A 0 value will set the driver to Master/Slave mode. Note that the routine assumes that you have the proper hardware (either Master/Slave or Multiple/Independent) for the mode you have selected. The routine has no effect if there is only one CompuGen board in the system.

Return Value

None.

See also

ggen_driver_initialize

Examples

C
```
int16 on ;
on = 1;
ggen_driver_initialize ((uInt16 far *)ggen_board_location, GGEN_MEMORY_SIZE_TEST);
ggen_set_independent_operation(on);
```

Visual BASIC
```
ggen_driver_initialize (ggen_board_location(0), GGEN_MEMORY_SIZE_TEST)
ggen_set_independent_operation (1)
```

ggen_set_outer_loop_counter

Syntax

C
#include <ggen_drv.h>
void ggen_set_outer_loop_counter (uInt16 count);

Visual BASIC
Sub ggen_set_outer_loop_counter (ByVal count As Integer)

Remarks

The **ggen_set_outer_loop_counter** routine is used to tell the CompuGen board how many times to generate the pattern loaded into its internal memory. The **count** parameter is the number of times to generate the pattern. A value 0 will generate continuously. If the parameter is non-zero, the hardware will generate the pattern **count** number of times before stopping. The maximum number of times to generate in non-continuous mode is 65535.

This return is preferable to the **ggen_single_shot** routine, as it is more flexible. The **ggen_single_shot** routine allows the user to choose between one-shot mode or continuous mode.

Return Value

None

See also

ggen_load_vram_from_buffer

Examples

C
ggen_set_outer_loop_counter (100);

Visual BASIC
ggen_set_outer_loop_counter (0)

ggen_set_records

Syntax

C

```
#include <ggen_drv.h>
int16 ggen_set_records (uInt16 far *records, int16 record, uInt16 segment, uInt16 index, uInt16 status);
```

Visual BASIC

```
Function ggen_set_records (records As Integer, ByVal record As Integer, ByVal segment As
                           Integer, ByVal index As Integer, ByVal status As Integer) As Integer
```

Remarks

ggen_set_records initializes the structure that is used for the initialization of the DLL without requiring the configuration file that contains the board locations. **records** is an array of words with **GGEN_B_L_BUFFER_SIZE** elements. **record** is the record in the array records to be updated. **segment**, **index** and **status** are the three values to be used to initialize the records array. This routine is often used to try and initialize a CompuGen board using default values if no GAGE_GEN.INC file is found.

The **records** parameter is assumed to be an uninitialized uInt16 array supplied by the application program. This array must be **GGEN_B_L_BUFFER_SIZE** (48) bytes long. The format of the array is as follows: the first **GGEN_B_L_STATUS_START** (32) uInt16s are for the board segment and index values, and each pair occupies **GGEN_B_L_ELEMENT_SIZE** (2) uInt16s, for each of the possible **GGEN_B_L_MAX_CARDS** (16) boards.

A status field is provided for each potential board location which is **GGEN_B_L_STATUS_SIZE** (1) uInt16s in length. The values for the status field are constants that correspond to bit positions in the status field and must be masked to determine which errors occurred when initializing the board. The low nibble is for problems with the segment and index.

CompuGen **ggen_board_location** (defined in GGEN_DRV.H file) array "pseudo structure"

array index:	0	1	2	3	...	30	31	32	33	34	35	...	46	47
meaning:	S1	I1	S2	I2	...	S16	I16	E1	E2	E3	E4	...	E15	E16

where: Sx = segment for board x,
Ix = index for board x,
Ex = returned board type or error status for board x,

The possible error codes for the status fields are:

GGEN_BAD_LSB_SEGMENT (0x0001) means that the low order byte of the segment was not equal to zero.

GGEN_BAD_MSB_SEGMENT (0x0002) is used when the segment is either less than A000 hex or greater than DF00 hex (the valid area in the memory map reserved for slot resources is 0xA0000 to 0xDFFFF.

GGEN_BAD_LSB_INDEX (0x0004) is set when the least significant bit of the index is not zero.

GGEN_BAD_MSB_INDEX (0x0008) is used when the high order byte of the index is either equal to 00 hex or greater than 03 hex (the valid area in the I/O map reserved for slot resources is 0100 hex to 03ff hex).

Return Value

The return value is true (non-zero) if the **record** specified is put into the array **records** and is false (zero) otherwise. Possible causes for a return of 0 are that the specified record is less than 0 or greater than **GAGE_B_L_BUFFER_SIZE** (48).

See also

ggen_driver_initialize and ggen_read_config_file

Examples

C

```
void    SetDefaultBoardLocation (uInt16 seg, uInt16 ind)
{
   int16     i;
   ggen_set_records ((uInt16 far *)ggen_board_location, 0, seg, ind, 0);
   for (i = 1 ; i < GGEN_B_L_MAX_CARDS ; i++)
      ggen_set_records ((uInt16 far *)ggen_board_location, i, 0, 0, 0);
}
```

Visual BASIC

```
Sub SetDefaultBoardLocation (ByVal seg As Integer, ByVal ind As Integer)
   Dim i As Integer, dummy As Integer
   dummy = ggen_set_records (ggen_board_location (0), 0, set, ind, 0)
   For i = 1 To GGEN_B_L_MAX_CARDS - 1
      dummy = ggen_set_records (ggen_board_location (0), i, 0, 0, 0)
   Next i
End Sub
```

ggen_set_sync_out

Note: This routine is for CG1100 only.

Syntax

C
#include <ggen_drv.h>
int16 ggen_set_sync_out (int16 sync_out);

Visual BASIC
Function ggen_set_sync_out (ByVal sync_out As Integer) As Integer

Remarks

This routine is used to set which analog sync output is being used for the CG1100. The available parameters are:

Constant	Value
CG1100_SYNC_OUT_1	1
CG1100_SYNC_OUT_2	2

CG1100_SYNC_OUT_1 can be used to set sync bit 1 and CG1100_SYNC_OUT_2 can be used to set synch bit 2. By default, both sync bits are turned on and either can be used. The synch bits can also be used by turning them on and off from within the pattern itself. Turning off bits 14 and 15 in the pattern will turn on both the sync bits at that position. Turning the bits on will turn off the sync bits. For example:

```
/* Turn off the synch output at position 1000 by making bits 14 and 15 high */
buffer[1000] = buffer[1000] | 0xC000;

/* Turn on the synch output at position 2000 by making bits 14 and 15 low */
buffer[2000] = buffer[1000] & 0x3FFF;
```

Return Value

If the routine is successful a non-zero value is returned. If an error occurs, a zero value is returned and **ggen_get_error_code** can be called to determine the error.

Examples

C
ret = ggen_set_sync_out (CG1100_SYNC_OUT_1);

Visual BASIC
ret = ggen_set_sync_out (CG1100_SYNC_OUT_1)

ggen_single_shot

Syntax

C
#include <ggen_drv.h>
void ggen_single_shot (int16 oneshot);

Visual BASIC
Sub ggen_single_shot (ByVal oneshot As Integer)

Remarks

ggen_single_shot controls whether only one complete data generation cycle is allowed. A call to this function with a TRUE (non-zero) value parameter allows only one data generation cycle after the CompuGen hardware receives a trigger event. A FALSE (zero) value will cause the data to be generated continuously. It is recommended that application programs use **ggen_set_outer_loop_counter** rather than this routine, as it allows for greater flexibility.

Return Value

None.

See also

ggen_set_outer_loop_counter

Examples

C
```
ggen_single_shot (board.generate_once);   /*  board.generate_once can be TRUE of FALSE.*/
```

Visual BASIC
```
ggen_single_shot board.generate_once            ' Either TRUE (non-zero) or FALSE (zero).
```

ggen_software_clock_pulse

Syntax

C
```
#include <ggen_drv.h>
void ggen_software_clock_pulse (void);
```

Visual BASIC
```
Sub ggen_software_clock_pulse ()
```

Remarks

ggen_software_clock_pulse will generate data conversions under software control. This routine is usually used to generate a very slow sample conversion rate or when a special function is desired and the regular clocked sample rate will not do. Another use for this routine is to single-step data generation on a CompuGen T30. A clock pulse is sent out each time **ggen_software_clock_pulse** is called.

Note: prior to calling **ggen_software_clock_pulse**, the multiplier parameter must be set to **GGEN_SOFTWARE_CLOCK** with a call to **ggen_generate_mode**. The rate parameter in the call to **ggen_generate_mode** has no effect on this call.

Return Value

None.

See also

ggen_generate_mode

Examples

C
```
ggen_generate_mode (GGEN_DUAL_MODE, GGEN_SOFTWARE_CLOCK,
                                    GGEN_SOFTWARE_CLOCK);
/* other set up calls, i.e. ggen_output_control, etc. */
ggen_software_trigger ();
ggen_dump_data ();
ggen_software_clock_pulse ();   /* Send out a clock pulse. */
```

Visual BASIC
```
temp = ggen_generate_mode (GGEN_DUAL_MODE, GGEN_SOFTWARE_CLOCK,
                                    GGEN_SOFTWARE_CLOCK)
/* Other set up calls, i.e. ggen_output_control, etc. */
Call ggen_software_trigger
Call ggen_dump_data
Call ggen_software_clock_pulse/* Send out a clock pulse. */
```

ggen_software_trigger

Syntax

C
```
#include <ggen_drv.h>
void ggen_software_trigger (void);
```

Visual BASIC
```
Sub ggen_software_trigger ()
```

Remarks

ggen_software_trigger allows the CompuGen to trigger from the result of a software event (immediate output) rather than waiting for a hardware trigger event to occur. The trigger source must be previously set to **GGEN_SOFTWARE** when calling to **ggen_trigger_control**.

Return Value

None.

See also

ggen_trigger_control

Examples

C
```
if (board.t_source == GGEN_SOFTWARE){
        ggen_software_trigger ();
}
```

Visual BASIC
```
If  board.t_source = GGEN_SOFTWARE Then
        ggen_software_trigger
End If
```

ggen_trigger_control

Syntax

C

```
#include <ggen_drv.h>
int16 ggen_trigger_control (int16 source, int16 slope, int16 range, int16 level);
```

Visual BASIC

```
Function ggen_trigger_control (ByVal source As Integer, ByVal slope As Integer,
                               ByVal range As Integer, ByVal level As Integer) As Integer
```

Remarks

ggen_trigger_control is used to set up the trigger parameters of the CompuGen. The **source** parameter sets the trigger source, which can be either external or software. If the source is set to software, the CompuGen board will trigger as soon as the routine **ggen_software_trigger** is called. If the trigger source is set to external, the board(s) will not generate data until an external trigger is received. The following constants can be used:

Constant	Value
GGEN_EXTERNAL	0
GGEN_SOFTWARE	1

The trigger **slope** can be either positive or negative, using the constants **GGEN_POSITIVE** and **GGEN_NEGATIVE**. This determines if the board(s) will be triggered on the rising or falling edge of an external trigger. The slope parameter has no effect if the trigger **source** is software. The following constants are available:

Constant	Value
GGEN_POSITIVE	0
GGEN_NEGATIVE	1

The trigger **range** is used to determine the range of the external trigger. The only available ragne is ±5 volts The **range** parameter has no effect if the trigger **source** is set to software. The following constant should be used:

Constant	Value
GGEN_TIMES_5	1

The trigger **level** can be any value between 0 and 255, with 0 equal to the lowest voltage in the current trigger range and 255 equal to the highest voltage in the current trigger range. The **level** parameter has no effect if the trigger **source** is set to software. The CompuGen T30 has only a TTL range for external trigger.

Valid values should be used for all parameters in the call, regardless of whether or not they are being used.

Return Value

A TRUE (non-zero) value is returned if the routine was successful, otherwise a FALSE (zero) value is returned and a call to **ggen_get_error_code** can be used to obtain the error code. Most errors are due to the use of an invalid parameter.

See also

ggen_generate_mode and ggen_output_control

Examples

C

```
ret = ggen_trigger_control (GGEN_EXTERNAL, GGEN_POSITIVE, GGEN_TIMES_5, 192);
```

Visual BASIC

```
ret = ggen_trigger_control (GGEN_EXTERNAL, GGEN_POSITIVE, GGEN_TIMES_5, 192)
```

Quick Reference:
Sample Programs Included with the CompuGen SDKs

Program name	Files included	Description
CGDEMO In C for Win 95/98 and Win NT	**C files** cgdemo.c support.c sigfile.c **Header files** cgdemo.h sigfile.h whichgen.h cg1100.h ggen_drv.h **Lib file** cgwindll.lib **Resource files** cgdemo.ico cgdemo.rc **Inc file** gage_gen.inc	CGDEMO.EXE is a demo program for CompuGen 1100 and CompuGen T30 boards. It supports up to 8 boards working together in either Master/Slave or Multiple/ Independent mode. The main features of this program include: 1. A greater range of signals to choose from, including sine wave, square wave, triangle wave, linear, and Gage signal file (*.sig). 2. For CG1100 boards, the user can select parameters for trigger control, sample frequency, loop number, output range and filter. 3. For CGT30 boards, the user can select parameters for trigger control, clock source, and timer frequency. 4. Full range memory test.

Program name	Files included	Description
CG_OUT In C for Win 95/98 and Win NT	**C files** cg_out.c error_msg.c generate.c ofiles.c **Header files** error_msg.h generate.h ofiles.h ggen_drv.h whichgen.h **Lib file** cgwindll.lib **Text file** Default.cgi **GageScope signal file** Default.sig	This program will read a CGI file with all the required parameters and generate a signal. **Note:** Format of the command line to run this program: drive:\path *.exe *.cgi e.g.: c:\cg_out.exe default.cgi
CG_OUT In Visual BASIC for Win 95/98 and Win NT	**Form** main.frm frmMsg.frm frmHelp.frm frmAbout.frm ComDia.frm **Modules** Global.bas module1.bas module2.bas **Text file** Default.cgi **GageScope signal file** Default.sig	This program will read a CGI file with all the required parameters and generate a signal. **Note:** With this version of the program there is no command line, but in order to read the CGI file, you must click on the **OPEN CGI FILE** button.

Glossary

Digital to Analog Conversion

Digital to Analog (D/A) conversion is the process by which an analog signal is generated based on an n-bit digital word or series of words. In other words, D/A conversion converts a discrete digital signal into a continuous analog signal.

Vertical Resolution

The number of bits with which a D/A system can specify the amplitude of the analog signal is its vertical resolution.

D/A Conversion Rate

In any practical D/A system, conversion is controlled by a clock. The frequency of this clock is called the Conversion Rate and is measured in MegaSamples per Second (MS/s).

Arbitrary Waveform Generation

One common application of D/A conversion is Arbitrary Waveform Generation. An Arbitrary Waveform Generator (ARB) is an instrument which allows the user to generate complex aperiodic waveforms by specifying a mathematical equation or a digital pattern.

Digital Pattern Generation

One interesting by-product of Arbitrary Waveform Generators is a Digital Pattern Generator. A Pattern Generator can output an n-bit-wide digital pattern which the user can program using simple keyboard commands.

Conversion Rate vs. Output Frequency

An analog signal is generated by an ARB by specifying a series of digital words which are converted to analog at the conversion rate. If the analog signal must be cyclical, e.g. a sine wave, the digital pattern must be repeated over and over again to create the cyclical analog output. In general, there must be 8 to 10 digital words (points) in a pattern to create a good-quality analog signal.

Memory Depth

The maximum number of digital points that an ARB can use to generate analog signals is called the Memory Depth. Memory depth is measured in kilosamples or Megasamples.

Memory Looping

One way of increasing or optimizing the memory depth in an ARB is to allow it to loop on a specific pattern for a given number of times. While most ARBs do not have this capability, the higher-end models such as the CompuGen 1100 do. With memory looping, it is possible to generate 1000 cycles of a 1 MHz sine wave at a conversion rate of 80 MS/s, using only 80 sample points. Without memory looping, the same output would have required 80,000 sample points!

Filtering

The output of an ideal D/A converter is a step function which contains very high frequency components, usually not found desirable for testing analog circuits. As such, the output of the ARB may need to be filtered in order to smooth the signal. Additional filters may be applied to reduce noise and provide band-limited signals.

Output Bandwidth

The output bandwidth of an ARB is related to the D/A conversion rate as well as the analog bandwidth of the output amplifier. For example, the bandwidth of the CompuGen 1100 is specified as 20 MHz, even though the conversion rate is 80 MS/s. As discussed earlier, the bandwidth has to be four to ten times less than the conversion rate.

Glitch Energy

One of the key characteristics of any ARB is the Glitch Energy of the D/A. Glitch Energy is defined as the area under the curve around zero-crossing, when the signal is changing its polarity. In simpler terms, Glitch Energy measures the error signal generated by switching the MSB from, say low to high, while all other bits are switching from high to low (and vice-versa). This error is caused by a small skew in the internal switching of the signals.

Trigger Control

CGWin allows the user to take advantage of the flexible triggering capabilities of the CompuGen cards. Both internal and external triggering are supported.

Technical Support

Gage Applied Technologies, Inc. offers free technical support for all its drivers.

Technical support is available by phone at:

(800) 567-4243 (within North America)
(514) 633-7447 (all other locations)

from 9:00 A.M. to 5:00 P.M. Eastern Standard Time, Monday to Friday.

Support is also available by fax at

(800) 780-8411 (within North America)
(514) 633-0770 (all other locations)

or by e-mail at

prodinfo@gage-applied.com

Updated drivers are available at Gage's Web site:

http://www.gage-applied.com

When calling for support we ask that you have the following information available:

1. Version and type of your CompuGen SDK.
   (The version number can be obtained at the top of any of the driver source files or on the distribution diskette.)

2. Type, version and memory depth of your CompuScope card.

3. Type and version of your operating system.

4. Type and speed of your computer and bus.

5. Contents of your CONFIG.SYS and AUTOEXEC.BAT files.

6. Any extra hardware peripherals (i.e. CD-ROM, joystick, network card, etc.)

7. Were you able to reproduce the problem with CGWIN.EXE or CGT30.EXE.?

If the problem is with an application program you are writing, the simplest approach is often to send us some of the code you are having problems with, along with other details such as sample rate, trigger source, input signal, etc., either by fax or e-mail. This way, we can try to reproduce the problem.

Gage Products

For ordering information, see Gage's product catalog, or visit our web site at http://www.gage-applied.com

CompactPCI Bus Products	CompuScope 85GC	8 bit, 5 GS/s Analog Input Card
	CompuScope 82GC	8 bit, 2 GS/s Analog Input Card
	CompuScope 14100C	14 bit, 100 MS/s Analog Input Card
	CompuScope 1610C	16 bit, 10 MS/s Analog Input Card
	CompuScope 3200C	32 bit, 100 MHz Digital Input for CompactPCI Bus
PCI Bus Products	CompuScope 1610	16 bit, 10 MS/s Analog Input Card
	CompuScope 1602	16 bit, 2.5 MS/s Analog Input Card
	CompuScope 14200	14 bit, 200 MS/s Analog Input Card
	CompuScope 14105	14 bit, 105 MS/s Analog Input Card
	CompuScope 14100	14 bit, 100 MS/s Analog Input Card
	CompuScope 1450	14 bit, 50 MS/s Analog Input Card
	CompuScope 12100	12 bit, 100 MS/s Analog Input Card
	CompuScope 1250	12 bit, 50 MS/s Analog Input Card
	CompuScope 1220	12 bit, 20 MS/s Analog Input Card
	CompuScope 85G	8 bit, 5 GS/s Analog Input Card
	CompuScope 82G	8 bit, 2 GS/s Analog Input Card
	CompuScope 8500	8 bit, 500 MS/s Analog Input Card
	CompuScope 3200	32 bit, 100 MHz Digital Input for PCI Bus
CompuGen	CompuGen 1100	12 bit, 80 MS/s Analog Output Card
	CompuGen 3250	32 bit, 50 MHz Digital Output Card
Application Software	GageScope	World's Most Powerful Oscilloscope Software
	GageBit	Digital Input/Output Software
	CompuGen for Windows	Arbitrary Waveform Generator Software for Windows
Software Development Kits	CompuScope SDK for C/C++ for Windows	
	CompuScope LabVIEW SDK for Windows	
	CompuScope MATLAB SDK for Windows	
	CompuScope LabWindows/CVI SDK (for CompactPCI/PXI bus CompuScope cards)	
	CompuGen Analog SDK for C/C++ for Windows	
	CompuGen Digital SDK for C/C++ for Windows	
	CompuGen Analog LabVIEW SDK for Windows	
	CompuGen Digital LabVIEW SDK for Windows	
	CompuGen Analog MATLAB SDK for Windows	
	CompuGen Digital MATLAB SDK for Windows	
Instrument Mainframes	Instrument Mainframe 7000 Instrument Mainframe 2000	Instrument Mainframes for Housing CompuScope and CompuGen Products.
	Instrument Mainframe 8000C	Instrument Mainframes for Housing CompactPCI/PXI CompuScope Products.