User Manual

for

DA12000-12-4M-PCI
DA12000-12-16M-PCI

1 Channel, 2.0 GSPS, 12-Bit, PCI
Arbitrary Waveform Generator Card

Last Modified 5/23/2014

CHASE SCIENTIFIC COMPANY
P.O. Box 1487
Langley, WA 98260

Tel: 360-221-8455
Fax: 360-221-8457

Email: techsupport@chase2000.com
Web: http://www.chase2000.com

TABLE OF CONTENTS

ILLUSTRATIONS / TABLES

"da12000_manual.odt" was created on 7/11/10 and last modified on 5/23/2014

1 GENERAL INFORMATION

1.1 Introduction

The DA12000 is a (1) Channel, 12-bit, 2.0 GigaSample/second Arbitrary Waveform Generator on a single mid-sized PCI card. It comes standard with following general features:

- Standard Internal Fixed Clock at 2.0 GSPS (external at 2.0 GHz, 1.0 GHz, 500 MHz, 250 MHz)
- (1) Channel Analog Output, 12-bit Vertical Resolution
- (2) TTL Output Marker
- Programmable Segment Size from 64 Data Words to full memory
- Programmable Number of Segments up to 32K
- External AC clock and External TTL/ECL trigger input

The analog output consist of (1) 50 ohm SMA output. To provide maximum flexibilty and performance to the user, **the outputs come unfiltered. An appropriate low pass filter is generally added in-line for a particular application and can be bought from companies like Mini-Circuits or can be ordered and/or custom made directly from Chase Scientific.**

The DA12000 has TTL/ECL input triggering capability that allows a segment or segments of data to be output only after a trigger is present.

1.2 References

PCI Local Bus Specification, Rev. 2.1, June 1st, 1995. For more information on this document contact:
PCI Special Interest Group
P.O Box 14070
Portland, OR 97214

Phone: 503-619-0569
FAX: 503-644-6708
http://www.pcisig.com

1.3 Deliverables

1.3.1 Software

The DA12000 comes with 32-bit DLL drivers for **Windows 98/ME/NT4/2000/XP/Vista/7**. Software comes on a single CDR disk or USB thumb drive. Call Chase Scientific for for the latest information on drivers for other operating system platforms such as Linux and 64-bit versions.

Windows drivers are provided as a Dynamic Link Library (*.DLL) which is compatible with most 32-bit windows based development software including Microsoft C/C++, Borland C/C++, and Borland Delphi. This DLL uses the **"cdecl"** calling convention for maximum compatibility and was made using Borland C++ Builder. It automatically provides the interface to the system drivers **"Windrvr6.sys"** for Windows 98/ME/NT4/2000/XP.

Actual Listing of files on Mini-CD:

```
------------   D I R E C T O R I E S / F I L E S  ----------------
BASE_DIR
|
| readme.txt                 // This file.
|
| da12000_manual.pdf          // Manual for DA4300 in PDF format
| da12000_ref_drwg.pdf        // Reference Drawing (Connector Descriptions)
|
| Register_DA12000_Win2000_XP.bat     // Installs Kernel driver for Win2000/XP
| UnRegister_DA12000_Win2000_XP.bat   // Uninstalls Kernel driver for Win2000/XP
|
| Register_DA12000_Win98_ME_NT4.bat   // Installs Kernel driver for Win98/ME/NT4.0
| UnRegister_DA12000_Win98_ME_NT4.bat // Uninstalls Kernel driver for Win98/ME/NT4.0
|
| wdreg16.exe        // Called by Register_DA12000_Win98_ME_NT4.bat
| wdreg.exe          // Called by Register_DA12000_Win2000_XP.bat
| windrvr6.inf       // Setup information file automatically called by above exe(s).
|
| da12000_dll.dll          // DLL for 98/ME/NT4/2000/XP  ( extern "C" __declspec(dllimport) )
| da12000_dll_import.h     // Header file for DLL
| da12000_dll.lib          // Library file for DLL in Borland C++
|
| | Microsoft VC++ DLL Files
| | | da12000_dll.lib          // Include in MSVC Project to compile DLL above
| | | da12000_dll_import.h     // Header file for DLL
|
|
| da12000.exe         // Simple GUI to test DLL and Kernel drivers
|
| DA12000_PCI.inf   // Plug-And-Play file needed by 98/ME/NT4/2000/XP for automatic
|                     // hardware configuration.
|
| windrvr6.sys    // Windows 98/ME/NT4/2000/XP Driver - copy this virtual driver
|                 // to "c:\<windir>\system32\drivers\" if not automatically done
|                 // so after running batch file.
|
| | Sample Data Files
| | | 64bit_sqwv.txt      // 64 sample squarewave full scale
| | | 64K_Data.txt        // 64K sample of lorentzian pulses (disk drive)
| | | random_noise.txt    // Random noise
|
|

-------------   E N D   ---------------
```

1.3.2 Hardware

The DA12000 hardware consists of a single mid-sized PCI compliant card. The card is shipped with this manual which includes complete hardware and software descriptions.

1.3.3 Checklist

Item #	Qty	Part Number	Description
1	1	DA12000-12-1M-PCI	2.0 GSPS, Arbitrary Waveform Generator, PCI card.
2	1	DA12000 Drivers	Mini-CDR with Dynamic Link Libraries for Windows 95/98/ME/NT4/2000/XP/Vista/7. Includes examples and manual.

1.4 Product Specification

(all specifications are at 25 ºC unless otherwise specified)

SPECIFICATIONS

Parameter	**Conditions**	**Typical Values unless otherwise indicated**
Analog Outputs		
Number of Outputs		(1) 50 ohm SMA outputs
Output Coupling		AC coupling through 0.1uF capacitor (50 ohm source impedance)
Vertical Resolution		12 bits (1 part in 4096)
Amplitude	2.0 GS/s	0.6Vpp +/-3%, single ended into 50 ohms.
Rise Time (20% to 80%)	No Filters	200 psec typical into 50 ohms
Fall Time (80% to 20%)	No Filters	200 psec typical into 50 ohms
Clock Jitter	2.0 GS/sec	Less than 10 psec RMS at 2 GHz
Trigger Delay	2.0 GS/sec	TBD
SFDR		
Fout < 660MHz	2.0 GS/sec	< -45 dB Typical
Fout = 660- 800 MHz	2.0 GS/sec	TBD
Internal Clock Rate		
Frequency Range		2.0 GHz Fixed
Resolution		N/A
Stability	T=0ºC – 70ºC	120 ppm
Memory		
Waveform Size	Standard	4 MWords / 16 MWords x 12-bits (-4M-PCI, -16M-PCI)
# of User Segments		1 to 16K segments
Segment Size Range		64 Words up to total memory in 64 Word increments
Digital Outputs		
(2) TTL Markers		Fclk/4 resolution
Digital Inputs		
External Clk input		50 ohm SMA input AC coupled. Can only use the following frequencies: 2.0 GHz, 1.0 GHz, 500 MHz, and 250 MHz. 0 dBm (0.633Vpp) < Input Amplitude < 12 dBm (2.53Vpp)
TTL Trigger input		Used to initiate any memory segment programmed for that purpose.

ENVIRONMENTAL

Parameter	**Typical Values unless otherwise stated**
Temperature	
Operating	0 to 70 degrees C standard
Non-Operating	-40 to +85 degrees C extended
Humidity	5 to 95% non-condensing
Operating	20% to 80%
Non-Operating	5% to 95%
Power	
+5V	TBD
+3.3V	TBD
+12V	TBD
-12V	N/A
Size	
DA12000	(1) Short PCI Card

1.5 Option Summary

OPTION SUMMARY

Option Name	Description
TBD	TBD

1.6 Technical Support / Software Updates

For technical support:

Email:	techsupport@chase2000.com
Phone:	360-221-8455
Fax:	360-221-8457
Mail:	Chase Scientific Company P.O. Box 1487 Langley, WA 98260

For software updates:

Email:	techsupport@chase2000.com
Web:	http://www.chase2000.com

1.7 Warranty

Chase Scientific Company (hereafter called Chase Scientific) warrants to the original purchaser that it's DA12000, and the component parts thereof, will be free from defects in workmanship and materials for a period of ONE YEAR from the data of purchase.

Chase Scientific will, without charge, repair or replace at its option, defective or component parts upon delivery to Chase Scientific's service department within the warranty period accompanied by proof of purchase date in the form of a sales receipt.

EXCLUSIONS: This warranty does not apply in the event of misuse or abuse of the product or as a result of unauthorized alterations or repairs. It is void if the serial number is altered, defaced or removed.

Chase Scientific shall not be liable for any consequential damages, including without limitation damages resulting from loss of use. Some states do not allow limitation or incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific rights. You may also have other rights that vary from state to state.

Chase Scientific warrants products sold only in the USA and Canada. In countries other than the USA, each distributor warrants the Chase Scientific products that it sells.

NOTICE: Chase Scientific reserves the right to make changes and/or improvements in the product(s) described in this manual at any time without notice.

2 HARDWARE DESCRIPTION

2.1 Introduction

The DA12000 hardware consists of the following major connections:

- (1) Normal, 1.0Gigasamples/second, 12-bit analog output (SMA)
- (1) PECL/Sinewave Clock Input, 250MHz, 500MHz, 1.0 GHz, 2.0 GHz ONLY (AC coupled)
- (1) TTL Trigger input (SMA)
- (2) TTL Outputs Markers (SMA)

2.2 Block Diagram

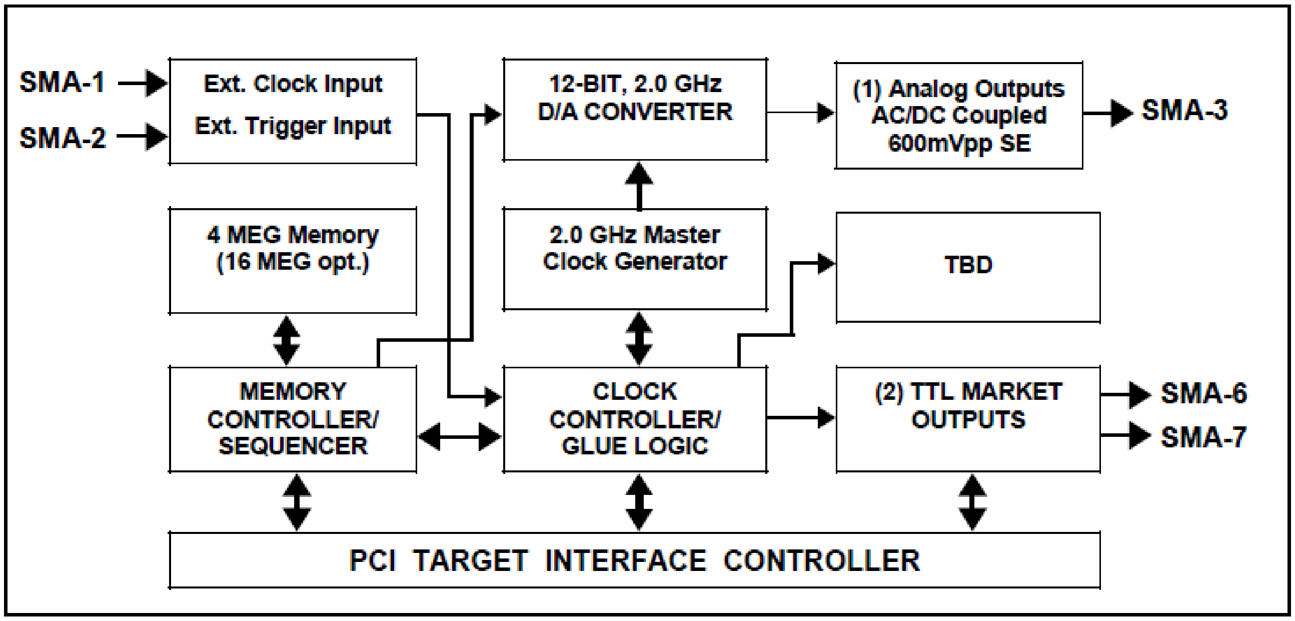
SMA-1
SMA-2
Ext. Clock Input
Ext. Trigger Input
12-BIT, 2.0 GHz D/A CONVERTER
(1) Analog Outputs AC/DC Coupled 600mVpp SE
SMA-3
4 MEG Memory (16 MEG opt.)
2.0 GHz Master Clock Generator
TBD
MEMORY CONTROLLER/ SEQUENCER
CLOCK CONTROLLER/ GLUE LOGIC
(2) TTL MARKET OUTPUTS
SMA-6
SMA-7
PCI TARGET INTERFACE CONTROLLER

Figure 1 – Block Diagram

2.3 Board Drawing

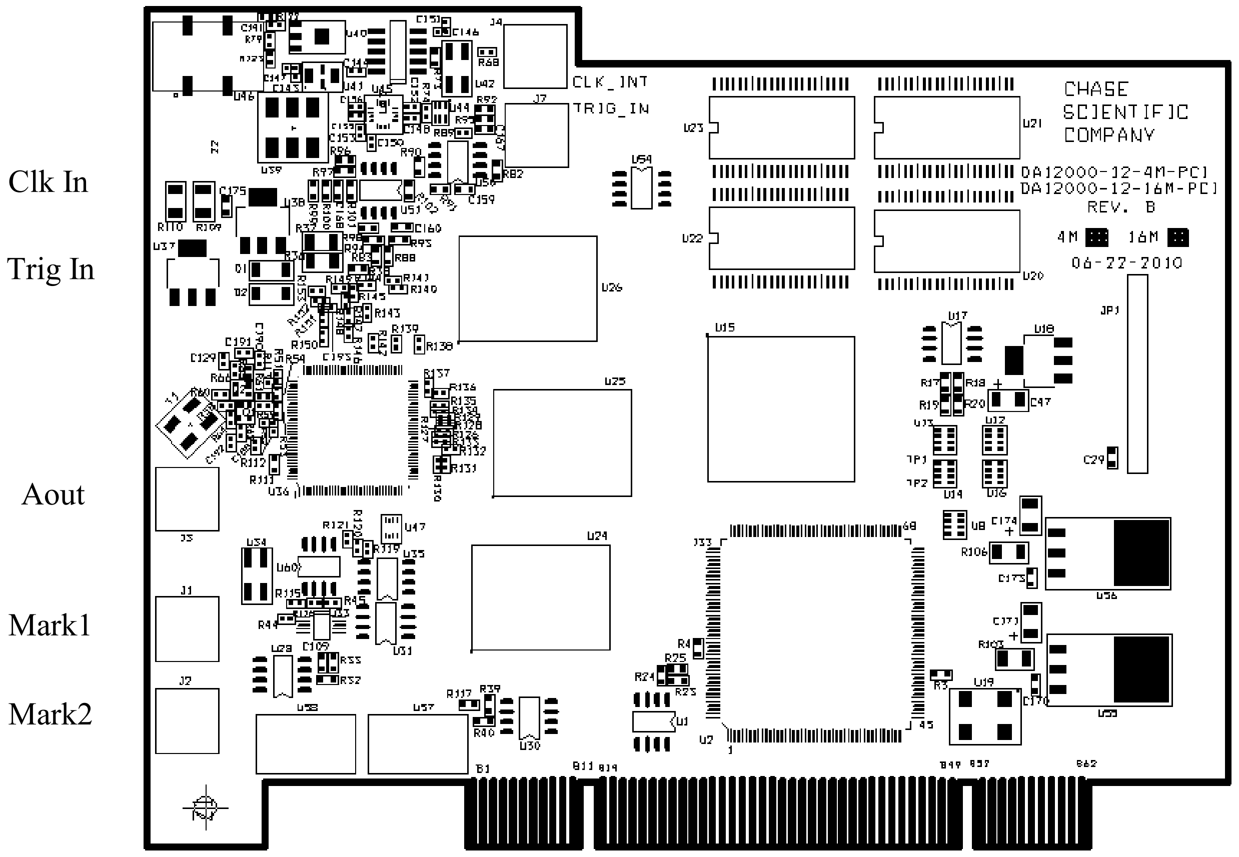

Figure 2 – Board Layout

2.4 External Clock Jumper Configurations

U42 (ext/int clock select)

Orientation for shunt is top-to-bottom. Left position uses internal clock while right position uses external clock.

U34 (phase adjust)
Orientation for shunt is top-to-bottom. Default position is one jumper in left position. This selection is done at factory and should not be changed by user without contacting Chase.

2.5 PCI Memory Allocation

DA12000 on-board memory is mapped automatically when a PCI 2.1 (or newer) motherboard powers up. If the DA12000 has 4 MegaSamples of memory, then the motherboard will allocate 8 Megabytes of memory. Once installed, the DA12000 software drivers will find the board or boards without the user changing any jumpers or worrying about addressing.

3 THEORY OF OPERATION

3.1 Introduction

Although the DA12000 is primarily comprised of a **Segment Sequencer** (or memory manager) and a 4:1 **High Speed Multiplexor**, it's how the software interacts with the hardware that makes it work. The following sections should provide enough operational theory for better understanding when using the software drivers.

3.2 Downloading and Outputting User Data to the DA12000

The DA12000 RAM memory IC's not only contain the user's waveform data, but it also contains special command codes that run the Segment Sequencer. These codes are placed into the upper nibble (4 bits) of selected individual sample points (16 bit words), leaving the lower 12 bits for user data. The Segment Sequencer reads these codes to determine where and when to jump to another segment, how many times to loop, when to wait for a trigger, and when to shut down. This is the heart of the DA12000 memory management.

Downloading a Single User Waveform (single segment) into memory is performed by simply calling *da12000_CreateSingleSegment(DWORD CardNum, DWORD NumPoints, DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn).* The user must be sure to pass the size of the waveform (*NumPoints)*, the number of times to repeat the waveform (*NumLoops),* a pointer variable pointing to the user array containing the data *(UserArrayPtr),* and finally, whether the segment will be self triggered or triggered by an external signal (*TrigEn).*

Downloading Multiple Linked Waveform Segments is performed by calling *da12000_CreateSegments(DWORD CardNum, DWORD NumSegments, PVOID PtrToSegmentsList)*. This function call requires the user to create a structure containing all the critical information on the segments that the user wants to download. The actual structure for each segment looks like the following:

```
typedef struct
{
       DWORD    SegmentNum;     // Current Segment Number
       PVOID    SegmentPtr;     // Pointer to current user segment
                                // ==> elements of one dimensional array must
                                //     be of type DWORD
       DWORD    NumPoints;      // Number of points in segment (must be multiple of 64)
       DWORD    NumLoops;       // Number of times to repeat segment (applies
                                // to next segment)
       DWORD    BeginPadVal;    // Reserved for future use.
       DWORD    EndingPadVal;   // Reserved for future use.
       DWORD    TrigEn;         // If > 0 then wait for trigger before going to
                                // next segment.
       DWORD    NextSegNum;     // Next segment to jump to after completion
                                // of current segment activities
} SegmentStruct;
```

The user must create an array of these segments and pass the pointer (*PtrToSegmentsList) to the function call.*

After the appropriate waveform data has been downloaded to the DA12000, da12000_SetTriggerMode() is enabled and the output begins.

4 SOFTWARE DRIVERS

4.1 Introduction

Our primary objective in designing software drivers is to get the user up and running as quickly as possible. While the details on individual function calls are listed in sections 4.3.x, the programming examples in section 4.4.x will show you how to include them into your programs. Please note that function calls are the same whether you are calling them under Windows 98, ME, NT4, 2000, or XP.

4.2 Driver Installation

4.2.1 Windows 98 / ME / NT4

1) Do not install DA12000 card at this time.

2) UnZip all files into directory "C:\temp\da12000\" (create directories if needed) You can move and/or copy the files later to a directory of your choice.

3) Run da12000_Register_Win98_ME_NT4.bat. This will copy the Kernel driver windrvr6.sys to "c:\<windir>\system32\drivers\" directory and will register the Kernel driver in the Windows Registry so that it starts up each time the computer is rebooted.

4) Power off computer. Insert DA12000 card. Power up computer.

5) When OS asks for Driver File point to "da12000_PCI.inf". If OS does not ask for file, then check hardware configuration and update if not listed properly under "Jungo" in Device Manager (see below).

To check to see which driver is installed, do the following:

```
=> Control Panel
  => System
    => Hardware
      => Device Manager
        => Jungo
           DA12000_4Meg    (Both this and WinDriver below should be present)
           WinDriver
```

If you see another driver in place of "DA12000_4Meg", then right click the first device under Jungo and click properties. Update the driver by pointing to "DA12000_4Meg". You may have to go through a series of menus.

4.2.2 Windows 2000 / XP

1) Do not install DA12000 card at this time.

2) UnZip all files into directory "C:\temp\da12000\" (create directories if needed) You can move and/or copy the files later to a directory of your choice.

3) Run da12000_Register_Win2000_XP.bat. This will copy the Kernel driver windrvr6.sys to "c:\<windir>\system32\drivers\" directory and will register the Kernel driver in the Windows Registry so that it starts up each time the computer is rebooted.

4) Power off computer. Insert DA12000 card. Power up computer.

5) When OS asks for Driver File point to "da12000_PCI.inf". If OS does not ask for file, then check hardware configuration and update if not listed properly under "Jungo" in Device Manager (see below).

To check to see which driver is installed, do the following:

```
=> Control Panel
  => System
    => Hardware
      => Device Manager
        => Jungo
           DA12000_4Meg    (Both this and WinDriver below should be present)
           WinDriver
```

If you see another driver in place of "DA12000_4Meg", then right click the first device under Jungo and click properties. Update the driver by pointing to "DA12000_4Meg". You may have to go through a series of menus.

4.2.3 Windows Vista / Windows 7

TBD

4.3 Function Calls

4.3.1 C Header File for DLL

```
//---------------------------------------------------------------------------
//  USER ROUTINES
//---------------------------------------------------------------------------


#define IMPORT extern "C" __declspec(dllimport)

IMPORT DWORD da12000_CountCards(void);
IMPORT DWORD da12000_Open(DWORD CardNum);
IMPORT DWORD da12000_Close(DWORD CardNum);

IMPORT void da12000_SetClock(DWORD CardNum, DWORD Frequency);

IMPORT void da12000_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
IMPORT void da12000_SetSoftTrigger(DWORD CardNum);
IMPORT void da12000_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
IMPORT void da12000_SetOffset(DWORD CardNum, int Mode, int Offset);

IMPORT void da12000_CreateSingleSegment(DWORD CardNum, DWORD ChanNum, DWORD NumPoints,
                                       DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn);
IMPORT void da12000_CreateSegments(DWORD CardNum, DWORD NumSegments, PVOID UserSegmentsPtr);
IMPORT void da12000_UpdateSegmentCmds(DWORD CardNum, DWORD NumSegments, PVOID PtrToSegmentsList);
```

4.3.2 Function Call Descriptions / Usage

4.3.2.1 da12000_CountCards()

Description
Returns number of DA12000 cards present on computer.

Declaration
```
DWORD da12000_CountCards(void);
```

Parameters
none

Return Value
Returns with an encoded value which represents the number of DA12000.

Return Values:

1-4:	Number of DA12000 boards detected.
0:	Indicates that no boards were found but that drivers are working properly.
13:	Software drivers are not installed properly. working correctly. "13"

Example
```
DWORD Num_da12000_Boards = da12000_Open() & 0x3;
```

4.3.2.2 da12000_Open()

Description
Loads the DA12000 software drivers and sets the DA12000 board to its default state.

Declaration
```
DWORD da12000_Open(DWORD CardNum);
```

Parameters
CardNum: 1 <= CardNum <= 4

Return Value
Returns with error code. A "0" means everything is fine. See below for details for other values.

Return Values:

0:	Opened Windriver Successfully and DA12000 Card Found Successfully
1:	Opened Windriver Successfully, but NO DA12000 CARDS FOUND
2:	Opened Windriver Successfully, Card found, but unable to open.
3:	Opened Windriver Successfully, Board already open.
6:	Card number exceeds number of cards.
13:	FAILED TO OPEN Windriver Kernel Driver

Example
```
DWORD OpenErrorCode = da12000_Open(1);  // Opens Board Number 1 and stores value.
```

4.3.2.3 da12000_Close()

Description
Closes DA12000 drivers. Should be called after finishing using the driver. However, if no other software uses the "windrv.xxx" (usual situation), then there is no need to close it until user is ready to completely exit from using their main software program which calls "windrv.xxx". If the user is loading the "windrv.xxx" dynamically (during run time), then they should close before unloading the driver.

Declaration
```
DWORD da12000_Close(DWORD CardNum);
```

Parameters

CardNum:	1 <= CardNum <= 4

Return Value
Returns with error code. A "0" means everything is fine. See below for details for other values.

Return Values:

0:	Closed Windriver Successfully for DA12000 card requested.
5:	DA12000 Card Already Closed for card requested.
13:	FAILED TO ACCESS Windriver Kernel Driver

Example
```
DWORD CloseErrorCode = da12000_Close(1);
```

4.3.2.4 da12000_SetClock() [Not Used]

Description
Sets the Digital to Analog converter clock rate. This function does nothing (placeholder) at this time on the DA12000. The card is fixed at 2.0 GSPS. There are jumpers on the PCB to allow for external clock features (see section 2.4).

Declaration
```
void da12000_SetClock(DWORD CardNum, DWORD Frequency);
```

Parameters

CardNum:	1 <= CardNum <= 4
Frequency:	1000000000

Return Value
none

Example
```
da12000_SetClock(300000000); // Sets clock rate to 300 MHz.
```

4.3.2.5 da12000_SetTriggerMode()

Description
Sets triggerring modes. This command should be called (using mode=0) just after the driver is opened to initialize internal hardware registers before calling any other routines. This function also selects whether board is in triggered mode or not and polarity of external TTL triggered signal.

Declaration

```
void da12000_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
```

Parameters

CardNum: 1 <= CardNum <= 4

Mode:

0: Shuts down all output operations. Asychronously resets RAM address counter and repeat counters to zero.

1: Used for starting single segment operation for segment created with "da12000_CreateSingleSegment()". Repeats indefinitely until mode set back to 0. External or "soft" trigger has no effect in this mode. Also works for "da12000_CreateSegments()", but any segments specified as triggered will immediately jump to next segment (no trigger required).

2: Sets up first segment for external or "soft" trigger mode. Individual segment(s) created as triggered will wait until external or soft trigger has occurred. If segment was created not to be triggered, then segment will follow previous segment in a continous fashion (no trigger needed). See da12000_CreateSegments for more information on multi-segment functioning.

ExtPol:

Not Used at time manual was written/updated.

Return Value

none

Example

```
da12000_SetTriggerMode(2,0);      //  First segment will wait for trigger before
                                  //  running.
```

4.3.2.6 da12000_SetSoftTrigger() [TBD]

Description

Emulates external triggering in software. Since this fuction actually toggles polarity of external input signal, it is "ORed" with external signal.

Declaration

```
void da12000_SetSoftTrigger(DWORD CardNum);
```

Parameters

none

Return Value

none

Example

```
da12000 SetSoftTrigger(1);     //  Initiates software trigger on Card Number 1
```

4.3.2.7 da12000_SetMarkers() [See notes below]

Description

This function is not used. Add 1 to waveform data BIT12 for Marker #1, BIT13 for Marker # 2 (modulo 4 only).

4.3.2.8 da12000_CreateSingleSegment()

Description
Creates a single segment in memory. The user determines the size of the array and whether the segment is started automatically or waits for an external input trigger. After creating a single segment waveform, the user must call SetTriggerMode() to turn on/off output waveforms.

See "**da12000_CreateSegments()**" for generating multipled segments.

Declaration
```
void da12000_CreateSingleSegment(DWORD CardNum,
                                 DWORD ChanNum,
                                 DWORD NumPoints,
                                 DWORD NumLoops,
                                 PVOID UserArrayPtr,
                                 DWORD TrigEn);
```

Parameters
CardNum:	1 <= CardNum <= 4	
ChanNum:	0x01, 0x02, 0x04, 0x08 for channels 1,2,3, and 4 [DA12000]	
NumPoints:	0 <= NumPoints <= (MaxMem-128)	`[ *** MUST BE MULTIPLE OF 64 *** ]`
NumLoops:	Set to 0 (other values not available)	[0 = Continuous]
UserArrayPtr:	Pointer to user array of WORD for WinXP/7-32 and DWORD for Win7-64.	
TrigEn:	High enables external trigger (must also set da12000_SetTriggerMode to triggered)	

Return Value
None.

Example
```
da12000_CreateSingleSegment(1,                        // Card Number 1
                            1,                        // Channel 1
                            128,                      // 128 Words contained
                            0,                        // Loops continuously
                            UserArrayPointer,         // Pointer to user data (DWORD)
                            0);                       // Externa trigger not enabled
```

4.3.2.9 da12000_ CreateSegments()

Description
Creates any number of segments up to the size of memory. Each segment can be programmed for repeat counts up to 16K and can jump to any other segment. See below for data structures for creating user segments. User must provide the correct array structures and pass a pointer to it along with how many sequencial segments are desired to be used.

After creating a complete waveform, the user must call SetTriggerMode() to turn on/off output waveforms. We also highly recommend using 64 sample pad (with value=2048) at beginning and end of waveform to give the cleanest start/stop waveforms.

Declaration

```
void da12000_CreateSegments(DWORD CardNum,
                            DWORD ChanNum,
                            DWORD NumSegments,
                            PVOID PtrToSegmentsList);
```

Parameters

```
CardNum:      1  <=  CardNum  <=  4
ChanNum:      0x01  (only valid number)

NumSegments:        Number of segment structures (see below) which user has
                    defined and wants to use.
PtrToSegmentsList:  Pointer to user array with each element with structure
                    defined as shown below.

typedef struct
{
        DWORD    SegmentNum;     // Current Segment Number
        PVOID    SegmentPtr;     // Pointer to current user segment
                                 // ==> elements of one dimensional array must
                                 //     be of type DWORD
        DWORD    NumPoints;      // Number of points in segment
                                    [ *** MUST BE MULTIPLE OF 64 *** ]

        DWORD    NumLoops;       // Number of extra times to repeat current segment
        DWORD    BeginPadVal;    // [Reserved for future use]
        DWORD    EndingPadVal;   // [Reserved for future use]
        DWORD    TrigEn;         // If > 0 then wait for trigger before going to
                                 // next segment.
        DWORD    NextSegNum;     // Next segment to jump to after completion
                                 // of current segment activities
} SegmentStruct;


**** Note that a segment is determined to be a triggered segment by the previous segment.
So setting Segment 5 as triggered will stop the sequence after Segment 5 has executed and
will wait for trigger event before "NextSegNum" is started.

The first segment is a special case and is determined by default as a triggered type if
SetTriggerMode() is set to "mode=2". The user in this case may use an external trigger or
a "soft" trigger to initiate the output process.
```

Return Value:

none.

Example

```
//  Create Array for SegmentList and Segments
SegmentStruct SegmentsList[2];

DWORD Segment0_Data[64];
DWORD Segment1_Data[64];

// Create Segment #1
for (i=0; i < (64); i++) {
    Segment0_Data[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(32) ) );
}
SegmentsList[0].SegmentNum     = 0;
SegmentsList[0].SegmentPtr     = Segment0_Data;
SegmentsList[0].NumPoints      = 64;
SegmentsList[0].NumLoops       = 0;
SegmentsList[0].BeginPadVal    = 2047;
SegmentsList[0].EndingPadVal   = 2047;
```

```
SegmentsList[0].TrigEn         = 0;
SegmentsList[0].NextSegNum     = 1;

// Create Segment #2
for (i=0; i < (64); i++) {
    Segment1_Data[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(8) ) );
}
SegmentsList[1].SegmentNum     = 1;
SegmentsList[1].SegmentPtr     = Segment1_Data;
SegmentsList[1].NumPoints      = 64;
SegmentsList[1].NumLoops       = 0;
SegmentsList[1].BeginPadVal    = 1000;
SegmentsList[1].EndingPadVal   = 1000;
SegmentsList[1].TrigEn         = 1;
SegmentsList[1].NextSegNum     = 0;    // Loops back to 1

da12000_CreateSegments(1,1,2,SegmentsList);
```

4.3.2.10 da12000_Set_Atten()   [TBD]

Description
This function call sets the amount of attenuation of the selected channel. The step size is 0.5dB. Typical insertion loss is 1.3dB. Only the first 6 bits of the "Atten_Value" are used, making the maximum amount of attenuation of 31.5dB (+ insertion loss).

Declaration
```
void da12000_Set_Atten(DWORD CardNum, DWORD ChanNum, DWORD Atten_Value)
```

Parameters
```
CardNum:     1  <=  CardNum  <=  4
ChanNum:     0x01, 0x02,  for channels 1,2  [DA12000]
Atten_Value: 0 <= 63;
```

Return Value:
none.

Example

```
da12000_Set_Atten(1,3,30);  // Sets Channel 1, Card 1, to 15dB attenuation.
```

4.3.2.11 da12000_UpdateSegmentCmds ()   [TBD]

Description
This function call works that same as "`da12000_CreateSegments()`" except that it does not download the data from system memory to card memory. Only the sequence commands are are downloaded to the card's memory. This saves time when the user wants to change the order of the segments because the segment data does not have to be updated. (The micro-commands tell the memory sequencer how many times to loop, when to jump, etc.)

Declaration
```
void da12000_UpdateSegmentCmds(DWORD CardNum,
                               DWORD ChanNum,
                               DWORD NumSegments,
                               PVOID PtrToSegmentsList);
```

Parameters

```
CardNum:      1  <=  CardNum  <=  4
ChanNum:      1  <=  CardNum  <=  4

NumSegments:        Number of segment structures (see below) which user has
                    defined and wants to use.
PtrToSegmentsList:  Pointer to user array with each element with structure
                    defined as shown below.

typedef struct
{
        DWORD    SegmentNum;     // Current Segment Number
        PVOID    SegmentPtr;     // Pointer to current user segment
                                 // ==> elements of one dimensional array must
                                 //     be of type DWORD
        DWORD    NumPoints;      // Number of points in segment
        DWORD    NumLoops;       // Number of times to repeat segment (applies
                                 // to next segment)
        DWORD    BeginPadVal;    // Reserved for future use.
        DWORD    EndingPadVal;   // Reserved for future use.
        DWORD    TrigEn;         // If > 0 then wait for trigger before going to
                                 // next segment.
        DWORD    NextSegNum;     // Next segment to jump to after completion
                                 // of current segment activities
} SegmentStruct;
```

`**** Note that a segment is determined to be a triggered segment by the previous segment. So setting Segment 5 as triggered will stop the sequence after Segment 5 has executed and will wait for trigger event before “NextSegNum” is started.`

`The first segment is a special case and is determined by default as a triggered type if SetTriggerMode() is set to “mode=2”. The user in this case may use an external trigger or a “soft” trigger to initiate the output process.`

Return Value:

none.

Example

See `da12000_CreateSegments() above for example.`

4.4 Programming Examples

4.4.1 Using Windows 95/98/NT DLL

Example Program

```
//
// DA12000 DLL C Example Test File
// ======================================
//
// 32-bit Borland C++ 5.0
//
// Web site: http://www.chase2000.cm
// Email:    support@chase2000.com
//
```

```
// (C) Chase Scientific 1999
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "da12000_dll_import.h"
#pragma link "da12000_dll.lib"


int main(int argc, char **argv)
{
    DWORD TempArray[1048575];
    int NumCards = 0;
    DWORD MemoryDepth = 1048576;
    double pi = 3.14159265358979;

// Check to see if card available
   NumCards = da12000_CountCards();            // Counts number of DA12000 cards

// OPEN DRIVER
    if (NumCards > 0) then DWORD OpenErrorCode = da12000_Open(1);   // Opens card # 1
    else exit(0);                                                  // Else exits

// INITIALIZE BOARD
    da12000_SetTriggerMode(1,0,0);     // VERY IMPORTANT !!!

// PUT WAVEFORM INTO ARRAY
    for (DWORD i=0; i < (MemoryDepth); i++) {
        TempArray[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(64) ) );
    }

// CREATE SINGLE SEGMENT WITH INFINITE LOOP
    da12000_CreateSingleSegment(1,1,MemoryDepth, 0, TempArray, 0);

// OUTPUT DATA
    da12000_SetTriggerMode(1,1,0);   // Enables out of data on brd# 1

// SHUT DOWN OUTPUT
//    da12000_SetTriggerMode(1,0,0);  // Use this to shut down output on brd# 1


// CLOSE DRIVER
    if (NumCards > 0) da12000_Close(1);  // Closes brd# 1.

}
```

Header File (for Reference)

```
//---------------------------------------------------------------------------
#ifndef da12000_dllH
#define da12000_dllH
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
//  USER ROUTINES
//---------------------------------------------------------------------------


#define IMPORT extern "C" __declspec(dllimport)

IMPORT DWORD da12000_CountCards(void);
IMPORT DWORD da12000_Open(DWORD CardNum);
IMPORT DWORD da12000_Close(DWORD CardNum);

IMPORT void da12000_SetClock(DWORD CardNum, DWORD Frequency);

IMPORT void da12000_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
IMPORT void da12000_SetSoftTrigger(DWORD CardNum);
IMPORT void da12000_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
IMPORT void da12000_SetOffset(DWORD CardNum, DWORD ChanNum, int Mode, int Offset);

IMPORT void da12000_CreateSingleSegment(DWORD CardNum, DWORD ChanNum, DWORD NumPoints,
DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn);
IMPORT void da12000_CreateSegments(DWORD CardNum, DWORD ChanNum, DWORD NumSegments, PVOID
PtrToSegmentsList);
IMPORT void da12000_UpdateSegmentCmds(DWORD CardNum, DWORD ChanNum, DWORD NumSegments,
PVOID PtrToSegmentsList);
IMPORT void da12000_Set_Atten(DWORD CardNum, DWORD ChanNum, DWORD Atten_Value);


#endif
```

5 MISCELLANEOUS

5.1 Calibration

The DA12000 has no user feature to calibrate. The gains and offsets are calibrated at the factory to be within specifications at 25ºC and nominal voltages.

5.2 Maintanence

No maintanence is required. However, a yearly calibration is recommended if the user desires to maintain the DA12000 specified accuracy. Call factory for maintainance and/or extended warranty information.

5.3 Changes/Corrections to this manual

Date	Description
07-11-2010	First release.
08-10-2010	Updated various references to UserArrayPtr such that Pointer points to user array of DWORD. Previously was WORD.

.