User Manual

for

DA4300-12-4M-PCI

4 Channel, 300 MSPS, 12-Bit, PCI Arbitrary Waveform Generator Card

(Last Updated on 5/30/2012)

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

"da4300_manual.odt" was created on 6/9/04 and last modified on 5/30/2012

1 GENERAL INFORMATION

1.1 Introduction

The DA4300 is a (4) Channel, 12-bit, 300 MegaSample/second Arbitrary Waveform Generator on a single mid-sized PCI card. It comes standard with following general features:

- Programmable Clock Frequency from 300 MSPS, 150 MSPS, and 75 MSPS
- 12-bit Vertical Resolution
- TTL Output Marker
- Programmable Segment Size from 64 Words to full memory
- Programmable Number of Segments up to 32K
- External TTL clock and External TTL trigger input

The outputs consist of (4) 50 ohm SMA outputs. Each output has independant data but uses a common clock. Also, to provide maximum flexibilty and performance to the user, the outputs come unfiltered, although the output buffer BW is approximately 100 MHz at full scale. 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 DA4300 has TTL input triggering capability that allows a segment or segments of data to be output only after a trigger is present. Gating is also available which will start and stop the data from being output on high or low TTL levels respectively.

An external TTL clock input allows the use of precision clock sources such as the CG400 and also for synchronizing multiple cards to a master clock.

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 (800) 433-5177 (U.S.)
(503) 797-4207 (International)
FAX (503) 234-6762

1.3 Deliverables

1.3.1 Software

The DA4300 comes with DLL drivers for **Windows 98/ME/NT4/2000/XP/Vista/7**. Software can be downloaded off the web site at http://www.chase2000.com/da4300.shtml. Call Chase Scientific for for the latest information on drivers for other operating system platforms.

Windows software 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.

Example Listing for Win98/XP files:

```
-----------   D I R E C T O R I E S / F I L E S  ---------------
BASE_DIR
|
| readme.txt                // This file.
|
| da4300_manual.pdf         // Manual for DA4300 in PDF format
| da4300_ref_drwg.pdf       // Connector Reference Drawing for DA4300
| da8150_ref_dwg.pdf        // Connector Reference Drawing for DA8150
|
| Register_DA4300_Win2000_XP.bat     // Installs Kernel driver for Win2000/XP
| UnRegister_DA4300_Win2000_XP.bat   // Uninstalls Kernel driver for Win2000/XP
|
| Register_DA4300_Win98_ME_NT4.bat   // Installs Kernel driver for Win98/ME/NT4.0
| UnRegister_DA4300_Win98_ME_NT4.bat // Uninstalls Kernel driver for Win98/ME/NT4.0
|
| wdreg16.exe        // Called by Register_DA4300_Win98_ME_NT4.bat
| wdreg.exe          // Called by Register_DA4300_Win2000_XP.bat
| windrvr6.inf       // Setup information file automatically called by above exe(s).
|
| da4300_dll.dll          // DLL for 98/ME/NT4/2000/XP  ( extern "C" __declspec(dllimport) )
| da4300_dll_import.h     // Header file for DLL
| da4300_dll.lib          // Library file for DLL in Borland C++
|
| | MS_LIB_File
| | | da4300_dll.lib        // Include in MSVC Project to compile DLL above
| | | da4300_dll_import.h   // Header file for DLL
| | MSVC6.0 [ EXAMPLE TEST CODE DIRECTORY ]
|
|
| da4300.exe         // Simple GUI to test DLL and Kernel drivers
|
| Chase_DA4300.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 Waveform 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 DA4300 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	DA4300-12-4M-PCI	300 MSPS, Arbitrary Waveform Generator, PCI card.
2	1	DA4300 Drivers	http://www.chase2000.com/da4300.shtml

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		(4) 50 ohm SMA outputs
Output Coupling		DC
Vertical Resolution		12 bits (1 part in 4096)
Amplitude	300 MS/s	2.0Vpp +/-3%, single ended into 50 ohms.
Programmable Attenuator		(Option 1)
Attenuation Range		31.5 dB
Number of Steps		64 steps
Attenuation per Step		0.5 dB (typical)
Insertion Loss		1.3 dB (typical at 100 MHz)
Rise Time (10% to 90%)	No Filters	2.5 nsec typical into 50 ohms
Fall Time (10% to 90%)	No Filters	2.5 nsec typical into 50 ohms
Clock Jitter	300 MS/sec	Less than 20 psec RMS at 300MHz
Trigger Delay	300 MS/sec	TBD
SFDR		
Fout < 50MHz	300 MS/sec	< -55 dB Typical
Fout = 50 – 100 MHz	300 MS/sec	< -50 dB Typical
Internal Clock Rate		
Frequency Range		300 MHz, 150 MHz, 75 MHz
Resolution		Multiples of 2
Stability	T=0ºC – 70ºC	20 ppm
Memory		
Waveform	Standard	1 MWords x 12-bits / Channel
# of User Segments		1 to 16K segments
Segment Size Range		64 Words up to total memory in 64 Word increments (minus pads)
Digital Outputs		
(1) TTL Markers		Fclk/4 resolution
Digital Inputs		

High Speed Clk input	75 MHz - 300 MHz	50 ohm SMA input
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	+5V DC +/- 10% => 500mA, 2.5 Watts (Typical using worst case waveform)
+3.3V	+3.3 VDC +/- 10% => 2.5 Amps, 8.4 Watts (Typical using worst case waveform)
+12V	+12 VDC +/- 10% => 216mA, 2.6 Watts (Typical using worst case waveform)
-12V	-12 VDC +/- 10% => 100mA, 1.2 Watts (Typical using worst case waveform)
Size	
Basic DA4300	(1) Mid-Sized PCI Card

1.5 Option Summary

OPTION SUMMARY

Option Name	Description
Option 1	Programmable Attenuator
Option 2	Custom Gain Setting
Option 3	CG400-PCI Clock Card

1.6 Technical Support / Software Updates

For technical support:

Phone:	360-221-8455
Fax:	360-221-8457
Email:	techsupport@chase2000.com
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 DA4300, 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 DA4300 hardware consists of the following major connections:

- (4) 300 MSPS, 12-bit analog outputs (SMA)
- (1) TTL clock inputs, 300 MHz, 150 MHz, 75 MHz
- (1) TTL Trigger input (SMA)
- (1) TTL Outputs Markers (SMA)

2.2 Block Diagram

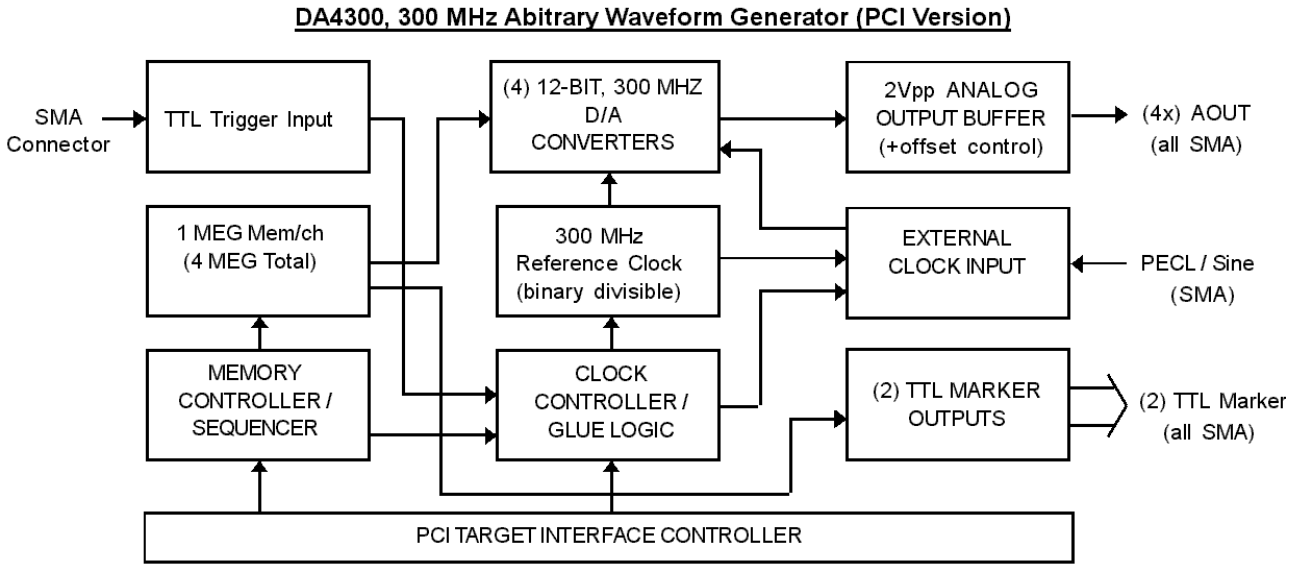

Figure 1 – Block Diagram

2.3 Board Drawing

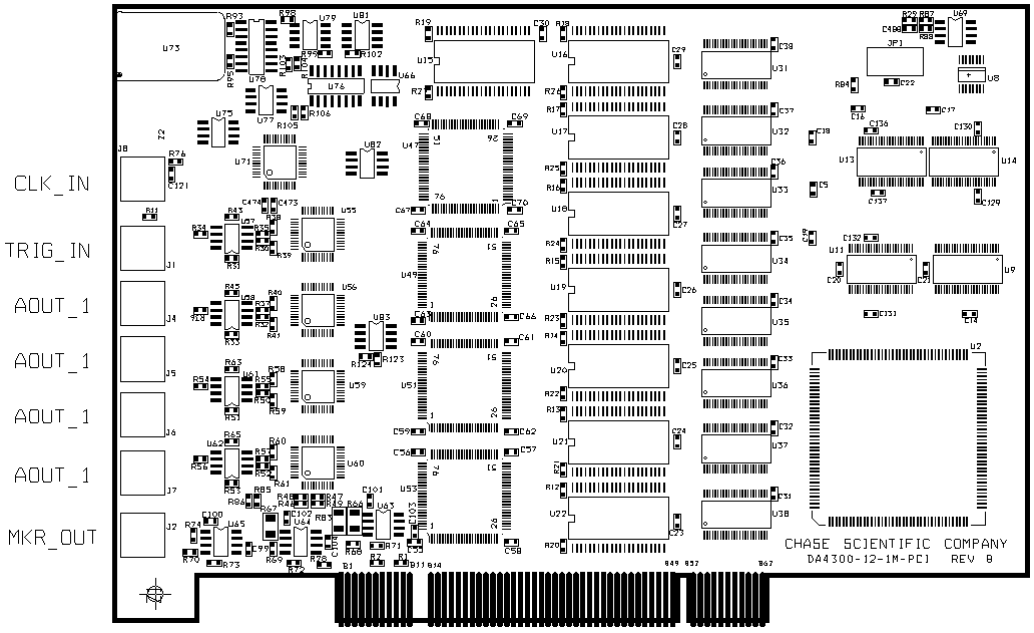

Figure 2 – Board Layout

2.3.1 PCI Memory Allocation

DA4300 on-board memory is mapped automatically when a PCI 2.1 (or newer) motherboard powers up. If the DA4300 has 4 MegaSamples of memory, then the motherboard will allocate 8 Megabytes of memory. Once installed, the DA4300 software drivers will find the board or boards without the user changing any jumpers or worrying about addressing. Unless the user manages to use up the entire memory available to the PCI motherboard (usually 256 Megabytes or more), then how the memory is allocated and where it is located in the system is completely transparent.

3 THEORY OF OPERATION

3.1 Introduction

Although the DA4300 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 DA4300

The DA4300 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 DA4300 memory management.

Downloading a Single User Waveform (single segment) into memory is performed by simply calling *da4300_CreateSingleSegment(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 *da4300_CreateSegments(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 diminsional array must
                                 //     be of type WORD
        DWORD    NumPoints;      // Number of points in segment (must be even multiple
                                 // of 64)
        DWORD    NumLoops;       // Number of times to repeat segment (applies
                                 // to next segment)
        DWORD    BeginPadVal;    // Pad value for beginning of triggered segment
        DWORD    EndingPadVal;   // Pad value for ending of triggered segment
        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 DA4300, da4300_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 4.2.xx, the programming examples in section 4.3.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 95, 98, or NT.

4.2 Driver Installation

4.2.1 Windows 98 / ME / NT4

1) Do not install DA4300 card at this time.

2) UnZip all files into directory "C:\temp\DA4300\" (create directories if needed) You can move and/or copy the files later to a directory of your choice.

3) Run da4300_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 DA4300 card. Power up computer.

5) When OS asks for Driver File point to "Chase_DA4300.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
Chase_DA4300 (Both this and WinDriver below should be present)
WinDriver

If you see another driver in place of "Chase_DA4300", then right click the first device under Jungo and click properties. Update the driver by pointing to "Chase_DA4300". You may have to go through a series of menus.

4.2.2 Windows 2000 / XP

1) Do not install DA4300 card at this time.

2) UnZip all files into directory "C:\temp\DA4300\" (create directories if needed) You can move and/or copy the files later to a directory of your choice.

3) Run da4300_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 DA4300 card. Power up computer.

5) When OS asks for Driver File point to "Chase_DA4300.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
Chase_DA4300 (Both this and WinDriver below should be present)
WinDriver

If you see another driver in place of "Chase_DA4300", then right click the first device under Jungo and click properties. Update the driver by pointing to "Chase_DA4300". You may have to go through a series of menus.

4.3 Function Calls

4.3.1 C Header File for DLL

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


#define IMPORT extern "C" __declspec(dllimport)

IMPORT DWORD da4300_CountCards(void);
IMPORT DWORD da4300_Open(DWORD CardNum);
IMPORT DWORD da4300_Close(DWORD CardNum);

IMPORT void da4300_SetClock(DWORD CardNum, DWORD Frequency);

IMPORT void da4300_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
IMPORT void da4300_SetSoftTrigger(DWORD CardNum);
IMPORT void da4300_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
IMPORT void da4300_SetOffset(DWORD CardNum, int Mode, int Offset);

IMPORT void da4300_CreateSingleSegment(DWORD CardNum, DWORD ChanNum, DWORD NumPoints,
                                       DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn);
IMPORT void da4300_CreateSegments(DWORD CardNum, DWORD NumSegments, PVOID UserSegmentsPtr);
IMPORT void da4300_UpdateSegmentCmds(DWORD CardNum, DWORD NumSegments, PVOID PtrToSegmentsList);
```

4.3.2 Function Call Descriptions / Usage

4.3.2.1 da4300_CountCards()

Description
Returns number of DA4300 cards present on computer.

Declaration
```
DWORD da4300_CountCards(void);
```

Parameters
none

Return Value
Returns with an encoded value which represents the number of DA4300.

Return Values:
- 1-4: Number of DA4300 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_da4300_Boards = da4300_Open() & 0x3;
```

4.3.2.2 da4300_Open()

Description
Loads the DA4300 software drivers and sets the DA4300 board to its default state.

Declaration
```
DWORD da4300_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 DA4300 Card Found Successfully
- 1: Opened Windriver Successfully, but NO DA4300 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 = da4300_Open(1);  // Opens Board Number 1 and stores value.
```

4.3.2.3 da4300_Close()

Description
Closes DA4300 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 da4300_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 DA4300 card requested.
- 5: DA4300 Card Already Closed for card requested.
- 13: FAILED TO ACCESS Windriver Kernel Driver

Example
```
DWORD CloseErrorCode = da4300_Close(1);
```

4.3.2.4 da4300_SetClock()

Description
Sets the Digital to Analog converter clock rate. This function call is also used to select the external clock, if the external clock option is present. To select an external clock, the user must a value greater than than 300000000.

Declaration
```
void da4300_SetClock(DWORD CardNum, DWORD Frequency);
```

Parameters
CardNum: 1 <= CardNum <= 4
Frequency: 300000000, 150000000, 75000000
Note: If (Frequency > 300000000), then external clock is selected.

Return Value
none

Example
```
da4300_SetClock(300000000); // Sets clock rate to 300 MHz.
```

4.3.2.5 da4300_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 da4300_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 "da4300_CreateSingleSegment()". Repeats indefinitely until mode set back to 0. External or "soft" trigger has no effect in this mode. Also works for "da4300_CreateSegments()", but any segments specified as triggered will immediately jump to next segment (no trigger required) and beginning and ending pads will be present in output for these segments.

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 da4300_CreateSegments for more information on multi-segment functioning.

ExtPol:

0: Trigger is inititiated on RISING edge of TTL waveform.

1: Trigger is inititiated on FALLING edge of TTL waveform.

Return Value

none

Example

```
da4300_SetTriggerMode(2,0);       //  First segment will wait for trigger before
                                  //  running.
```

4.3.2.6 da4300_SetSoftTrigger()

Description

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

Declaration

```
void da4300_SetSoftTrigger(DWORD CardNum);
```

Parameters

none

Return Value

none

Example

```
da4300 SetSoftTrigger(1);      //  Initiates software trigger on Card Number 1
```

4.3.2.7 da4300_SetMarkers()

Description
Sets up TTL output marker locations relative to waveform memory. It is up to the user to place the markers correctly. There is always a startup 64 sample leading pad when a waveform first outputs. Please note that all segments have a 64 sample leading pad and a 64 sample trailing pad, regardless of whether they repeat or not.

Resolution of the markers is 1/4 of the clock rate. Also, please note that this function call must be called after creating any segments since da4300_ CreateSegments() and da4300_CreateSingleSegment() will overwrite the markers with zeros if done in the reverse order.

Declaration
```
void da4300_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
```

Parameters
CardNum: 1 <= CardNum <= 4

PointAddr: RAM address location. Minimum resolution is 4 clock samples.

Nib1: 0 <= Nib1 <= 0xF [see board layout for connector information]
Nib2: 0 <= Nib2 <= 0xF

Return Value
None.

Example
```
da4300_SetMarkers(1, 64, 0xF, 0xF);      // Place marker on all bits at
                                          // beginning of 1st data segement of
                                          // board number 1.
```

4.3.2.8 da4300_SetOffset() [Not Available at this time]

Description
Sets output voltage offsets on Normal and Complementary outputs on primary DA4300 board.

Declaration
```
void da4300_SetOffset(DWORD CardNum, int Mode, int Offset);
```

Parameters
CardNum: 1 <= CardNum <= 4

Mode:
0 = Adjust offset on Complimentary Output
1 = Adjust offset on Normal Output
2 = Adjust both at same time

Offset: -250 <= Offset <= 250 [Resolution = 1; Units are in millivolts DC]

Return Value
none

Example
```
da4300_ SetOffset(1,2,22);    // Sets both output DC offset to 22mV for card 1.
```

4.3.2.9 da4300_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.

In triggered mode there is a 64 samples of pad at the beginning and end of the segment with a level set at 2047. In non-triggered mode the only pad that is visible is the beginning pad when the output is started, then repeats data portion indefinitely until reset. All segments, regardless of whether it's triggered or not, have 64 sample pads at the beginning and end of the segments in actual memory, but may not be visible depending on whether the segment is triggered or not. See "**da4300_CreateSegments()**" for generating multipled segments.

Declaration
```
void da4300_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 [DA4300]

NumPoints: 0 <= NumPoints <= (MaxMem-16) [Must be in multiples of 16]

NumLoops: Set to 0 (other values not available) [0 = Continuous]

UserArrayPtr: Pointer to user array of WORD

TrigEn: High enables external trigger (must also set da4300_SetTriggerMode to triggered)

Return Value
None.

Example
```
da4300_CreateSingleSegment(1,                       // Card Number 1
                           2,                       // Channel 2
                           128,                     // 128 Words contained
                           0,                       // Loops continuously
                           UserArrayPointer,        // Pointer to user data
                           0);                      // Externa trigger not enabled
```

4.3.2.10 da4300_ CreateSegments()

Description
Creates any number of segments up to the size of memory. All segments have 64 samples of beginning pad and 64 samples of trailing pad which the user cannot access except to determine the default levels. However, when repeating or jumping in non-triggered mode, the user will not see the pad fields. 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.

==> VERY IMPORTANT: NumPoints must be an even multiple of 64 (e.g. 64, 128, 192, etc).

Declaration

```
void da4300_CreateSegments(DWORD CardNum,
                           DWORD ChanNum,
                           DWORD NumSegments,
                           PVOID PtrToSegmentsList);
```

Parameters

```
CardNum:     1  <=  CardNum  <=  4
ChanNum:     0x01, 0x02, 0x04, 0x08  for channels 1,2,3, and 4  [DA4300]

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 diminsional array must
                                 //     be of type WORD
        DWORD    NumPoints;      // Number of points in segment (must be even multiple
                                 // of 64)
        DWORD    NumLoops;       // Number of times to repeat segment (applies
                                 // to next segment)
        DWORD    BeginPadVal;    // Pad value for beginning of triggered segment
        DWORD    EndingPadVal;   // Pad value for ending of triggered segment
        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];

WORD Segment0_Data[64];
WORD 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

da4300_CreateSegments(1,1,2,SegmentsList);
```

4.3.2.11 da4300_Set_Atten() [only works for DA4300 with option 1]

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 da4300_Set_Atten(DWORD CardNum, DWORD ChanNum, DWORD Atten_Value)
```

Parameters
```
CardNum:     1  <=  CardNum  <=  4
ChanNum:     0x01, 0x02, 0x04, 0x08  for channels 1,2,3, and 4  [DA4300]
Atten_Value: 0 <= 63;
```

Return Value:
none.

Example

```
da4300_Set_Atten(1,3,30);  // Sets Channel 1, Card 1, to 15dB attenuation.
```

4.3.2.12 da4300_UpdateSegmentCmds ()

Description
This function call works that same as "`da4300_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.)

==> VERY IMPORTANT: NumPoints must be an even multiple of 64 (e.g. 64, 128, 192, etc).

Declaration

```
void da4300_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 diminsional array must
                                  //     be of type WORD
        DWORD    NumPoints;       // Number of points in segment (must be even multiple
                                  // of 64)
        DWORD    NumLoops;        // Number of times to repeat segment (applies
                                  // to next segment)
        DWORD    BeginPadVal;     // Pad value for beginning of triggered segment
        DWORD    EndingPadVal;    // Pad value for ending of triggered segment
        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 `da4300_CreateSegments()` above for example.

4.4 Programming Examples

4.4.1 Using Windows 95/98/NT/2000/XP DLL

Example Program in MSVC 6.0

```
//
// DA4300 DLL C Example Test File
```

```
// ====================================
//
// 32-bit MSVC 6.0
//
// Web site: http://www.chase2000.cm
// Email:    support@chase2000.com
//
// (C) Chase Scientific 2006
//

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

#include "da4300_dll_import.h"

int main(void)
{
//------------------------------------------------------------
//  INITIALIZE VARIABLES
//------------------------------------------------------------
   int NumCards = 0;
   int CardNum = 1;
   DWORD MemoryDepth = 65536;
   double pi = 3.14159265358979;

   typedef WORD *Array;   // One Dimensional Array, 16-bit unsigned
   Array DataArray;

   if ((DataArray = (Array) calloc(MemoryDepth,sizeof(WORD *))) == NULL)
   {
      printf( "Not Enough Memory\n" ) ;
      return(98) ;
   }


//------------------------------------------------------------
//  COUNT NUMBER OF DA4300 CARDS
//------------------------------------------------------------

// Check to see if card available
    NumCards = da4300_CountCards();           // Counts number of DA4300 cards


//------------------------------------------------------------
//  OPEN DA4300 DRIVERS IF CARD(S) AVAILABLE
//------------------------------------------------------------

// OPEN DRIVER
    if (NumCards > 0) DWORD OpenErrorCode = da4300_Open(CardNum);   // Opens card # 1
    else return(99);                                  // Else exits


//------------------------------------------------------------
//  INITIALIZE HARDWARE TRIGGERING & CLOCKRATE
//------------------------------------------------------------

    da4300_SetTriggerMode(CardNum,0,0);     // VERY IMPORTANT !!!

    da4300_SetClock(1, 300000000);


//------------------------------------------------------------
//  CREATE SINEWAVE AND WRITE TO "DataArray"
//------------------------------------------------------------
```

```
// PUT WAVEFORM INTO ARRAY
    for (DWORD i=0; i < (MemoryDepth); i++) {
        DataArray[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(32) ) );
    }


//---------------------------------------------------------
//  UPLOAD "DataArray" TO DA4300 MEMORY
//---------------------------------------------------------

// CREATE SINGLE SEGMENT WITH INFINITE LOOP
    da4300_CreateSingleSegment(CardNum,1,MemoryDepth, 0, DataArray, 0);

// SET TTL MARKER FOR SCOPE TRIGGER AT BEGINNING OF WAVEFORM
      da4300_SetMarkers(CardNum, 64, 0xFF, 0xFF);


//---------------------------------------------------------
//  UPLOAD "DataArray" TO DA4300 MEMORY
//---------------------------------------------------------

// OUTPUT DATA
    da4300_SetTriggerMode(CardNum,1,0);   // Enables output of data on brd# 1

// SHUT DOWN OUTPUT
//    da4300_SetTriggerMode(CardNum,0,0);  // Use this to shut down output on brd# 1


//---------------------------------------------------------
//  CLOSE DA4300 DRIVER
//---------------------------------------------------------

    if (NumCards > 0) da4300_Close(CardNum);  // Closes brd# 1.


//---------------------------------------------------------
//  FREE "DataArray" ARRAY, RETURN "NumCards"
//---------------------------------------------------------

    free(DataArray);

      return(NumCards);

}
```

Header File (for Reference)

```
//---------------------------------------------------------------------------
#ifndef da4300_dllH
#define da4300_dllH
//---------------------------------------------------------------------------


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

//#define DWORD unsigned long     // USE THESE IF NOT DEFINED
//#define WORD unsigned short     // USE THESE IF NOT DEFINED
//#define BYTE unsigned char      // USE THESE IF NOT DEFINED
//typedef void *PVOID;            // USE THESE IF NOT DEFINED


#define IMPORT extern "C" __declspec(dllimport)

IMPORT DWORD da4300_CountCards(void);
```

```
IMPORT DWORD da4300_Open(DWORD CardNum);
IMPORT DWORD da4300_Close(DWORD CardNum);

IMPORT void da4300_SetClock(DWORD CardNum, DWORD Frequency);

IMPORT void da4300_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
IMPORT void da4300_SetSoftTrigger(DWORD CardNum);
IMPORT void da4300_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
IMPORT void da4300_SetOffset(DWORD CardNum, DWORD ChanNum, int Mode, int Offset);

IMPORT void da4300_CreateSingleSegment(DWORD CardNum, DWORD ChanNum, DWORD NumPoints,
DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn);
IMPORT void da4300_CreateSegments(DWORD CardNum, DWORD ChanNum, DWORD NumSegments, PVOID
PtrToSegmentsList);
IMPORT void da4300_UpdateSegmentCmds(DWORD CardNum, DWORD ChanNum, DWORD NumSegments,
PVOID PtrToSegmentsList);
IMPORT void da4300_Set_Atten(DWORD CardNum, DWORD ChanNum, DWORD Atten_Value);

#endif
```

5 MISCELLANEOUS

5.1 Calibration

The DA4300 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 DA4300 specified accuracy. Call factory for maintainance and/or extended warranty information.

5.3 Changes/Corrections to this manual

Date	Description
06/09/04	First release
11/02/04	Added programmable attenuator function call
10/31/06	Added MSVC 6.0 Sample Code
5/30/12	Changed to 4M Version Only

.

Trademarks:
MS-DOS, Windows 3.1, Windows 95, and Windows NT are registered trademarks of Microsoft Corporation.