User's Manual

for

DAx22000, DAx12000, DAx14000
Arbitrary WaveForm Generators

2-Channel, 2.5 GS/sec, 12-bit (DAx22000)
1-Channel, 2.5 GS/sec, 12-bit (DAx12000)
1-Channel, 4.0 GS/sec, 12-bit (DAx14000)

(2M, 8M Memory per channel - DAx22000)
(2M, 8M Memory per channel - DAx12000)
(4M, 16M Memory per channel - DAx14000)

WavePond®
P.O. Box 1487
Langley, WA 98260

Web: https://chase-scientific.com/wavepond.html

Original Document: dax22000_dax14000_manual.odt
(created 01/07/13, updated 07/17/21)

TABLE OF CONTENTS

ILLUSTRATIONS / TABLES

1 GENERAL INFORMATION

1.1 Introduction

The DAx22000 and DAx14000 are USB based Arbitrary Waveform Generators with maximum sampling rates of 2.5 GS/sec and 4.0 GS/sec respectively. These USB modules come in aluminum boxes by default, but can also fit into a PCI or PCIe bracket slot. They are designed to produce any waveform from DC up to and even past Nyquist frequency (1/2 * Sampling Rate). The on-board high resolution frequency synthesizer, as well as programmable segmentation, allow the user to seamlessly fit looping waveforms into memory, alleviating the need for more expensive and less reliable options.

The DAx22000 module has the following standard features:

- (2) Channel, 2.5 GS/sec, 12-bit D/A outputs (800 mVpp Typical) [SMA]
- DC Coupled outputs into 50 ohms
- 1ppm Internal Programmable Clock Synthesizer with < 5psec Jitter
- Internal Clock Synthesizer operates from 25 MHz to 2.5 GHz.
- SFDR less than -50 dB @ 825 MHz (typ)
- Full scale Trise/Tfall = 180 picoseconds (typical for 2.5 GS/sec)
- 2M, 8M Sample Memory per Channel
- 3.3V TTL Prog. Marker Out [SMA]
- 3.3V TTL TRIG_IN (Asynchronous Trigger Capability with 400 psec resolution) [SMA]
- Card comes in USB box or with optional bracket to fit into PCI, PCIe, or no-Slot
- All functions controlled through USB Mini-B connector

The DAx14000 module has the following standard features:

- (1) Channel, 4.0 GS/sec, 12-bit D/A outputs (635 mVpp Typical) [SMA]
- DC Coupled outputs into 50 ohms
- 1ppm Internal Programmable Clock Synthesizer with < 5psec Jitter
- Internal Clock Synthesizer operates from 25 MHz to 4.0 GHz.
- SFDR less than -40 dB @ 1500 MHz (typical for 4.0 GS/sec)
- Full scale Trise/Tfall = 180 picoseconds (typ)
- 4M, 16M Sample Memory per Channel
- 3.3V TTL Prog. Marker Out [SMA]
- 3.3V TTL TRIG_IN (Asynchronous Trigger Capability with 250 psec resolution) [SMA]
- Card comes in USB box or with optional bracket to fit into PCI, PCIe, or no-Slot
- All functions controlled through USB Mini-B connector

1.2 References

See USB 2.0 specifications.

1.3 Deliverables

1.3.1 Software

The DAx22000 and DAx14000 modules come with USB drivers (usually automatically downloaded by OS), GUI, and programmers API software for **WinXP-32, Win7-32/64, Win8-32/64**. All software can be downloaded from "www.wavepond.com". Check with WavePond for the latest information on drivers for other operating systems.

The GUI program can perform many tasks including loading waveforms from a file, generating sine, square, triangle, and sawtooth waves, changing clock rates, triggering etc. The GUI program executable can also be used to execute command line API calls when parameters are detected. If any parameters are detected on the command line then the GUI itself is not displayed and the appropriate calls are made to the hardware. After each command line call the program removes itself from memory and returns control to the calling program. This API method was added to eliminate any possible compatibility issues.

There is also the standard DLL API. It offers the fastest access to the board's functions but your software compiler must be able to perform a static compile or dynamic run-time load using only the DLL. It is up to the user to work out any interface requirements that are required for their particular compiler/application. WavePond has tested the DLL against MinGW/GCC g++ version 3.4.2 and Lazarus 1.2.6 (Object Pascal) without any issues. WavePond uses these compilers internally for testing because they are open source. WavePond uses 100% open source tools for software and hardware development.

The GUI and DLL softwares themselves are written using the Lazarus IDE for readability and to comply with WavePond's internal policy of 100% open source development tools. However, even though Lazarus' compiler is open source, the user's software license strategies are not restricted in any way.

See section on software drivers for description of each function along with an example.

1.3.2 Hardware

By default the DAx22000/14000 hardware consists of an aluminum box (example shown below) which houses the arbitrary waveform generator card, 3ft USB cable (A Male to USB 5-Pin Mini B Male), and 5V power supply module. Optionally, a PCI/PCIe bracket can be added to the card instead of the box.

1.3.3 DAx22000 Checklist

Item #	Qty	Part Number	Description
1	1	DAx22000-2M [8M]	2.5 GSPS Abitrary Waveform Generator System housed in USB controlled aluminum box with 2M [8M] memory per channel.
2	1	USB Cable	3 or 6 foot USB cable (A Male to USB 5-Pin Mini B Male).
3	1	AC/DC Power Adatper	AC/DC Desktop Adapter, 5V, 18W (not available for bracket version below)
4	1*	PCI / PCIe Bracket	*Optional PCI Bracket version INSTEAD of BOX. USB controlled.

1.3.4 DAx14000 Checklist

Item #	Qty	Part Number	Description
1	1	DAx14000-4M [16M]	4.0 GSPS Abitrary Waveform Generator System housed in USB controlled aluminum box with 4M [16M] memory per channel.
2	1	USB Cable	3 or 6 foot USB cable (A Male to USB 5-Pin Mini B Male).
3	1	AC/DC Power Adatper	AC/DC Desktop Adapter, 5V, 18W (not available for bracket version below)
4	1*	PCI / PCIe Bracket	*Optional PCI Bracket version INSTEAD of BOX. USB controlled.

1.4 Product Specification

DAx22000 *(all specifications are at 25C unless otherwise specified)*

I/O SPECIFICATIONS

Analog Outputs (SMA)	
Number of D/A Outputs	2
Vertical Resolution	12-bits
Output Impedance	50 ohms
Amplitude	750 mVpp typical into 50 ohms
T(rise) / T(fall)	180 psec typical
Memory Size	2M, 8M per channel
Maximum # of Segments	60
Segment Size Range	48 samples up to total memory; Modulo 16 only.
Segment Resolution	16 samples
Maximum Segment Loops	65,534
LVTTL Outputs (SMA)	(2) Marker Outputs (3.3V TTL). One at beginning of waveform and one user programmable.

LVTTL Inputs (SMA)	(1) Trigger input (3.3V). Input is DC coupled to 0.9V into 50 ohms. Actual trigger level is 1.0V. Trigger delay = (53*(1/SR)) + 14ns. See software function description for functionality. Maximum re-trigger rate is 2 MHz. (1) 10 MHz Clock Reference. AC coupled into 50 ohms. 3Vpp max. Software selectable internal/external.
Master Clock (internal)	
Frequency	25 MHz to 2.5 GHz standard
Phase Noise	-100dBc/Hz @ 1KHz Offset Typical @ Fc = 2.5 GHz
Jitter	< 5 picoseconds

GENERAL

Power Supply (Vcc)	+ 5.0V +/- 10%, 3A typical. [uses PC internal supplies for ATA/SATA drives or USB power brick]
Operating Temperature	15 to 30 degrees C standard
Operating Humidity	5 to 95% non-condensing
Size	4.3" x 3.9" x 0.5" (fits into PCI, PCIe, no-slot, USB box.)
Data Bus	USB 2.0

DAx14000 *(all specifications are at 25C unless otherwise specified)*

I/O SPECIFICATIONS

Analog Outputs (SMA)	
Vertical Resolution	12-bits
Output Impedance	50 ohms
Amplitude	635mVpp typical into 50 ohms
T(rise) / T(fall)	180 psec typical
Memory Size	4M, 16M per channel
Maximum # of Segments	60
Segment Size Range	128 samples up to total memory; Modulo 32 only.
Segment Resolution	32 samples
Maximum Segment Loops	65,534
LVTTL Outputs (SMA)	(2) Marker Outputs (3.3V TTL). One at beginning of waveform and one user programmable.
LVTTL Inputs (SMA)	(1) Trigger input (3.3V). Input is DC coupled to 0.9V into 50 ohms. Actual trigger level is 1.0V. Trigger delay = (53*(1/SR)) + 14ns. See software function description for functionality. Maximum re-trigger rate is 2 MHz. (1) 10 MHz Clock Reference. AC coupled into 50 ohms. 3Vpp max. Software selectable internal/external.
Master Clock (internal)	
Frequency	25 MHz to 4.0 GHz standard
Phase Noise	-90dBc/Hz @ 1KHz Offset Typical @ Fc = 4.0 GHz
Jitter	< 5 picoseconds

GENERAL

Power Supply (Vcc)	+ 5.0V +/- 10%, 2.4A typical. [uses PC internal supplies for ATA/SATA drives or USB power brick]
Operating Temperature	15 to 30 degrees C standard
Operating Humidity	5 to 95% non-condensing
Size	4.3" x 3.9" x 0.5" (fits into PCI, PCIe, no-slot, USB box.)
Data Bus	USB 2.0

1.5 Technical Support / Software Updates

For technical support:

Email	techsupport@chase-scientific.com
Mail	WavePond P.O. Box 1487 Langley, WA 98260
Web	https://chase-scientific.com

For software updates:

Email	techsupport@chase-scientific.com
Web	https://chase-scientific.com/wavepond.html

1.6 FCC Warning

This equipment is intended for use in a laboratory test environment only. The equipment generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case, the user, at their own expense, is required to take whatever measures may be required to correct this interference.

1.7 Warranty

WavePond warrants to the original purchaser that its DAx22000, DAx14000, or DAx12000, 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.

WavePond will, without charge, repair or replace at its option, defective or component parts upon delivery to WavePond'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.

WavePond 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.

WavePond warrants products "directly" sold anywhere in the world. If a WavePond product is purchased through an authorized distributor then warranty details are resolved through them.

NOTICE: WavePond 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 DAx22000 hardware consists of the following I/O functions:

- (2) 12-bit 2.5 GSPS D/A Outputs
- (2) 3.3V Marker Outputs
- 25 to 2.5 GHz Clock Synthesizer
- Memory Controller
- USB 2.0 Interface

The DAx14000 hardware consists of the following I/O functions:

- (1) 12-bit 4.0 GSPS D/A Outputs
- (2) 3.3V Marker Outputs
- 25 to 4.0 GHz Clock Synthesizer
- Memory Controller
- USB 2.0 Interface

2.2 Hardware Installation

The standard configuration is an aluminum box which is tethered to the PC with USB 2.0. An additional desktop power brick is provided for 5V power. Optionally, the DAx22000 can be configured at the factory to install in any PCI, PCIe, or empty slot as long as there is location to attach the bracket. A molex connector (with SATA power adapter) is the primary power for this configuration and cables should be standard inside any PC. See board drawing for location of connectors.

2.3 Block Diagrams

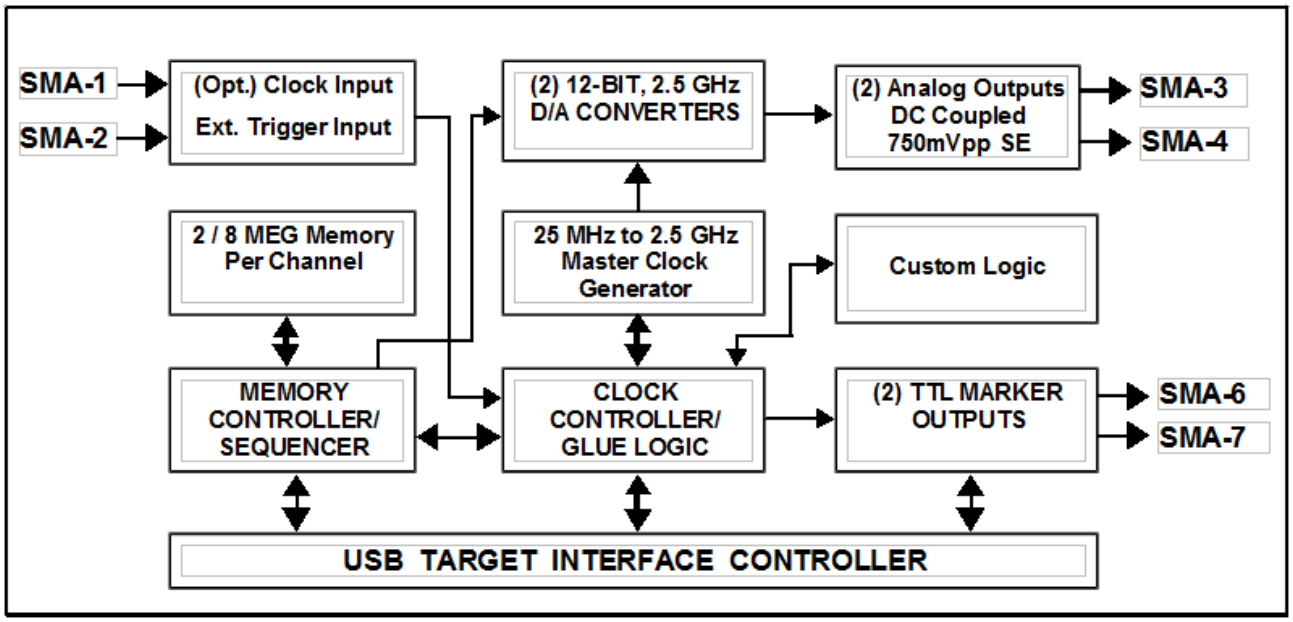


Figure 1 – DAx22000 Block Diagram

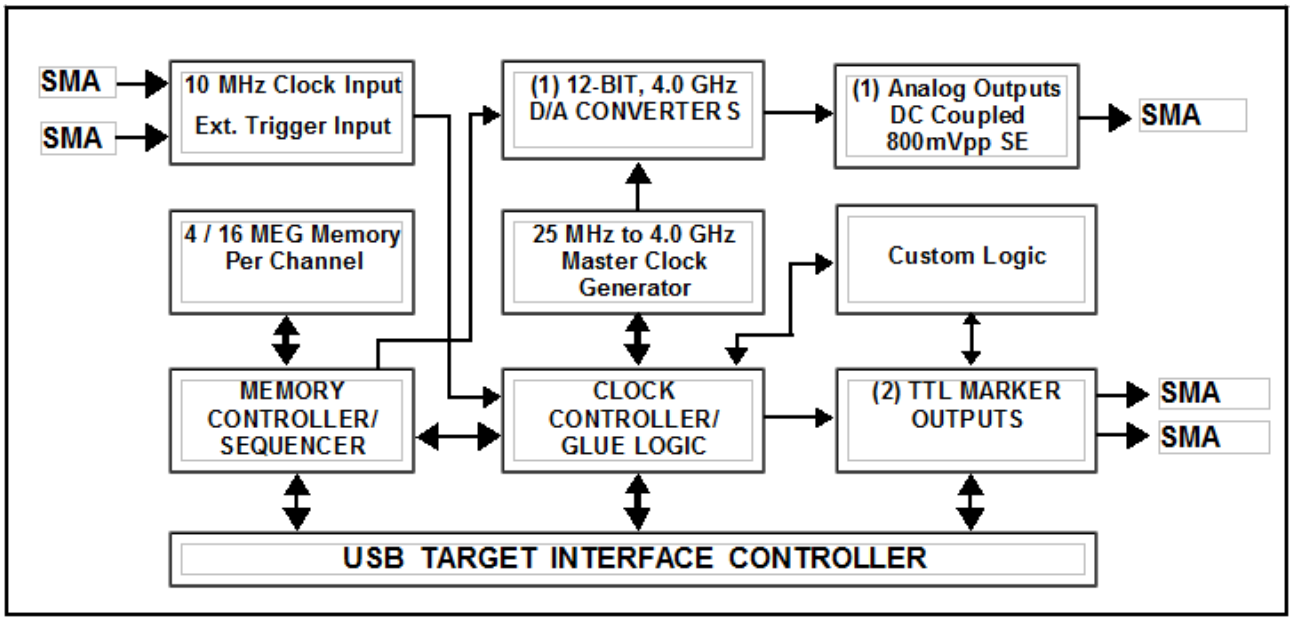


Figure 2 – DAx14000 Block Diagram

2.4 I/O Connections

Figure 3 – I/O for DAx22000

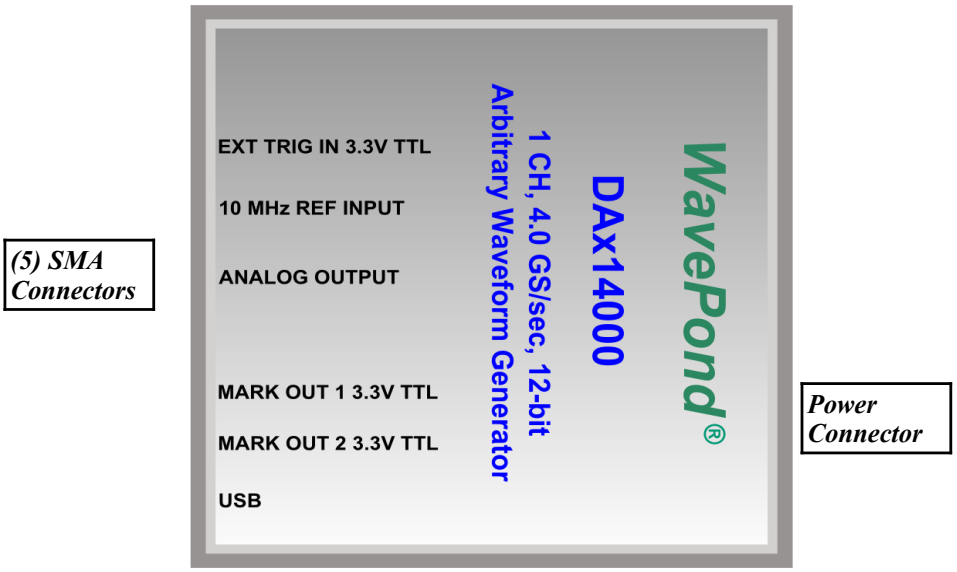


Figure 4 – I/O for DAx14000

3 THEORY OF OPERATION

3.1 Introduction

The DAx22000 and DAx14000 modules (hereafter refered to as DAx modules) primarily consist of a **Segment Sequencer** (i.e. memory sequencer), **SRAM**, **Frequency Synthesizer**, **FPGA**, **D/A Converters,** and the **USB 2.0 peripheral controller** (USB interface). The most relevant parts will be described briefly below.

Several things need to happen before you can output the waveform. First and foremost you have to decide on Clock Rate, Trigger source/type, where the waveform data is coming from (software or file), and whether you're using the internal or external reference clock. Then you have to download a waveform. During the download process special segment sequencer instructions are automatically (invisibly) added to the download process. After this, you can RUN the AWG module using either the "RUN" button on the GUI or via the API function call "DAx22000_Run".

By default the main tab of the GUI creates and uses a single segment while the Multi-Segment tab can handle up to 7 segments. The software API can create up to 60 linked segments and each segment has controls such as wait for trigger, loop X times, and leading/trailing DAC levels.

The DAC's of both modules are 12-bits and their output range is defined as 0 to 4095. The output of both modules is set to 0 V for a DAC value of 2047. A DAC value of 0 would be the most negative value (e.g. -400 mV) while a DAC value of 4095 would be the most positive value (e.g. +400mV).

Multi-Segment operation is sequencial in nature where segment 2 comes after segment 1, and segment 3 comes after segment 2. You cannot jump from segment 5 to segment 10, but you can jump from the last segment, say segment 100, back to segment 0 if you're going to repeat the sequence.

3.2 Downloading and Outputting User Data

The DAx module's SRAM contain the user's waveform data while the special command codes that run the Segment Sequencer are stored in separate SRAM. 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 DAx memory management.

Downloading a Single User Waveform (single segment) into memory is performed by simply calling

```
DAx22000_CreateSingleSegment(
   DWORD CardNum,
   DWORD ChanNum,
   DWORD NumPoints,
   DWORD NumLoops,
   DWORD PAD_Val_Beg,
   DWORD PAD_Val_End,
   PVOID pUserArrayWORD,
   DWORD Triggered
   );
```

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 (*Triggered).*

Downloading Multiple Linked Waveform Segments is performed by calling

```
DAx22000_CreateSegments(
   DWORD CardNum,
   DWORD ChanNum,
   DWORD NumSegments,
   DWORD PAD_Val_Beg,
   DWORD PAD_Val_End,
   PVOID pSegmentsList,
   bool Triggered
   );
```

This function call requires the user to create a structure (SegmentsList) 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
{
   PVOID    SegmentPtr;     // Pointer to User Data (Type = Array of WORD)
   DWORD    NumPoints;      // Number of points in segment (min=48, then mod 16)
   DWORD    NumLoops;       // Number of extra times to repeat current segment
   DWORD    TrigEn;         // If > 0 then wait for trigger before next segment.
} SegmentStruct;
```

*** Please note that the first segment in a mult-segment operation is special with regards to triggering. The boolean parameter Triggerd applies to the first segment while "TrigEn" in the SegmentStruct directly effect the other segments.

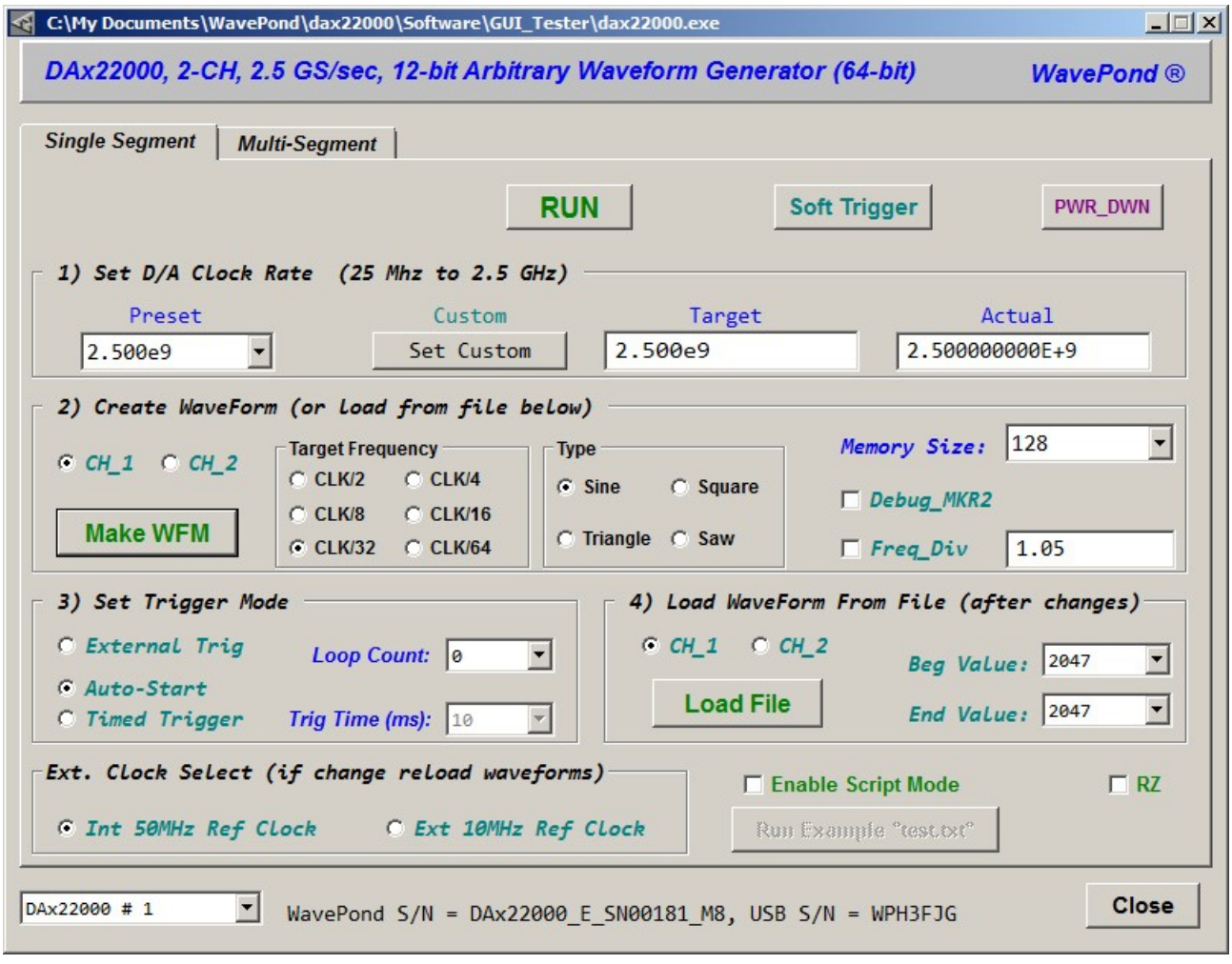
4 SOFTWARE GUI APPLICATION

4.1 Introduction

The GUI application is provided to check whether the DAx board is detected and operating properly, to run custom waveforms, to run some built-in sine/square/triangle/sawtooth waveforms, and to load Script API (see section 6).

4.2 Operation

Below is a screen shot of the "dax22000_GUI_64.exe". In general, the on-screen controls are self explanatory, but a few comments here will help you get started.



Simply run “dax22000_GUI_32.exe” or “dax22000_GUI_64.exe” to display the DAx22000 exerciser program. Please note that these programs require "ftd2xx.exe" and "ftdxx64 respectively to be in same directory.

Before making any changes be sure the the green RUN button is visible (and does not show red STOP). If the button shows red STOP then click it to return it to the green RUN display. There are exceptions to this rule, but we'll keep it simple for now.

1) Set D/A Clock Rate:
You can use the drop-down menu to select some preset values (e.g. 1.000e9, 2.000e9, 2.500e9) or leave it as is. You can also set a custom value by entering a number in the "Target" box and clicking "Set Custom" button. The "Target" box accepts floating point numbers and positive integers.

2) Create Waveform:
Clicking the "Make WFM" button and then the "RUN" button is the minimum required to output a waveform. You may make additional selections in this group box if desired. For example, entering "4" into the Freq_Div box will scale the divide-by frequency by 4, thus enabling one to create a sine wave with a period of 256 instead of 64.

3) Set Trigger Mode:
Auto-Start provides a one-time trigger to allow the waveform to start immediately and loop a number of times according to "Loop Count". If "Loop Count" is 0 then it will loop continuously. "Timed Trigger" will provide a periodic trigger where "Loop Count" is a number greater than 1 (zero is not permitted). Each time the "Timed Trigger" occurs the waveform will repeat "Loop Count" times. If the "External Trig" radio button is selected then it waits for external trigger events (TTL) to initiate the "Loop Count" selected.

4) Load WaveForm From Text File:
This groupbox loads a single waveform from a user file and will behave according to #1 and #3 above. Each line in the file represents a single point in an increasing time sequence and must be between 0 and 4095 (no minus signs permitted). The time between data points (i.e. lines) is 1/(Clock Rate). The minimum number of points permissible for the DAx22000 is 64, and for the DAx14000 is 128. The minimum granuality of increasing waveform sizes is 16 for the DAx22000 and 32 for the DAx14000. The "Beg Value" and "End Value" is only relevant if the waveform is running off an external trigger or Timed Trigger and will determine what the desired waveform value is leading up to a trigger event and the value after the waveform has completed.

5) Ext. Clock Select:
This groupbox only applies if the DAx22000 is Rev. D or greater, or any version of DAx14000.

6) Enable Script Mode
If this checkbox is selected then the GUI will behave the same as if the Script API was activated (see Section 6 for more details). Normally this mode is actived by "dax22000_GUI_64.exe load_api" on computer boot and the GUI is not visible.

5 Programmers Interface #1 (dll API)

5.1 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 below, we have also included some programming examples as well. Please note that function calls are the same whether you are calling them under Win2000, WinXP, or Win7/8.

*** Very important ==> The DAx14000 uses the same software as the DAx22000. The only difference is that there is no channel 2 on the DAx14000.

The details on installing the USB drivers are listed in 5.2. The listing of function calls and their parameter definitions are listed in section 5.4.xx and the programming examples in section 5.5.x will show you how to include them into your programs.

The drivers are designed to work under Windows 32/64-bit systems [check for availability on Mac OS X, Linux, and Android].

5.2 USB Driver Installation

Windows XP, 7, and 10 should install drivers automatically when card is plugged in "if" auto-update is enabled. If not, see USB driver installation guide for the appropriate operating system instructions.

5.3 API Installation

There is No installation necessary for the programmers API. Just include the following files into the same directory as your application software:

dax22000_lib_DLL32.dll, dax22000_lib_DLL32.h, ftd2xx.dll **(for 32-bit)**
dax22000_lib_DLL64.dll, dax22000_lib_DLL64.h, ftd2xx64.dll **(for 64-bit)**

5.4 Function Calls

5.4.1 Function List

C Header File (reference)

```
//----------------------------------------------------------------------------
#ifndef dax22000_lib_DLL64H
#define dax22000_lib_DLL64H
//----------------------------------------------------------------------------

#define DWORD unsigned long int    // Use any of these if not defined.
#define WORD unsigned short int    //
#define BYTE unsigned char         //
#define PVOID void *               //

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

#define IMPORT extern "C" __declspec(dllimport)

IMPORT int DAx22000_GetNumCards(void);
IMPORT int DAx22000_Open(int CardNum);
IMPORT int DAx22000_Close(int CardNum);

IMPORT int DAx22000_Initialize(int CardNum);

IMPORT double DAx22000_SetClkRate(int CardNum, double User_Freq);
IMPORT int DAx22000_SelExtTrig(int CardNum, bool ExtTrig);

IMPORT int DAx22000_Run(int CardNum, bool TriggerNow);
IMPORT int DAx22000_Stop(int CardNum);

IMPORT int DAx22000_SoftTrigger(int CardNum);
IMPORT int DAx22000_Place_MRK2(int CardNum, int Mod16_CNT);

IMPORT int DAx22000_CreateSingleSegment(
   DWORD CardNum,
   DWORD ChanNum,
   DWORD NumPoints,
```

```
   DWORD NumLoops,
   DWORD PAD_Val_Beg,
   DWORD PAD_Val_End,
   PVOID pUserArrayWORD,
   DWORD Triggered
   );

IMPORT int DAx22000_CreateSegments(
   DWORD CardNum,
   DWORD ChanNum,
   DWORD NumSegments,
   DWORD PAD_Val_Beg,
   DWORD PAD_Val_End,
   PVOID pSegmentsList,
   bool Loop
   );

IMPORT int DAx22000_Debug(int CardNum, int ModeNum);

IMPORT int DAx22000_Ext10MHz(int CardNum, int Enable);

#endif
```

5.4.2 Function Descriptions / Usage

5.4.2.1 DAx22000_GetNumCards

Description
Returns number of DAx22000 systems detected. Please note that all DAx22000 systems must be powered up for this to occur. The DAx22000 is not powered over the USB bus.

Declaration
```
int DAx22000_GetNumCards(void);
```

Parameters
n/a

Return Value
Number of DAx22000 systems detected on USB bus.

Example
```
int NumCards = DAx22000_GetNumCards();
```

5.4.2.2 DAx22000_Open

Description
Opens API driver so that other commands may be used. The return value must be "0" before proceeding.

Declaration
```
int DAx22000_Open(int CardNum);
```

Parameters
CardNum: 1 <= CardNum <= 4

Return Value
0 if successful.

Example
```
DAx22000_Open(1);     // Open card number 1.
```

5.4.2.3 DAx22000_Close

Description
Closes API driver. This is usually called before user application is terminated to prevent memory leakage.

Declaration
```
int DAx22000_Close(int CardNum);
```

Parameters
CardNum: 1 <= CardNum <= 4

Return Value
0 if successful.

Example
```
DAx22000_Close(1);
```

5.4.2.4 DAx22000_Initialize

Description
This call initializes the USB peripheral controller and all the internal registers on the DAx22000. It must be called after DAx22000_Open().

Declaration
```
int DAx22000_Initialize(int CardNum);
```

Parameters
CardNum: 1 <= CardNum <= 4

Return Value
0 if successful.

Example
```
DAx22000_Initialize(1);     // Initializes card number 1.
```

5.4.2.5 DAx22000_SetClkRate

Description
This call sets the sampling rate of the D/A's. The user can choose anything between 25 Mhz and 2.5 GHz as a target value and returns the actual value it was able to set. The DAx22000 uses a Fractional-N Synthesizer so the results are usually within 1ppm.

Declaration
```
double DAx22000_SetClkRate(int CardNum, double User_Freq);
```

Parameters
CardNum: 1 <= CardNum <= 4
User_Freq: 25e6 <= User_Freq <= 2.5e9 (target value for DAx22000)
User_Freq: 25e6 <= User_Freq <= 4.0e9 (target value for DAx14000)

Return Value
Actual value the synthesizer was able to set ... usually within 1ppm

Example
```
DAx22000_SetClkRate(1, 2.205e9);   // Sets sampling rate to 2.205 GHz
```

5.4.2.6 DAx22000_SelExtTrig

Description
When the ExtTrig value is set to "true", the DAx22000 uses the external trigger input for all triggered modes except for soft triggers or starting trigger which the user can set at any time.

Declaration
```
int DAx22000_SelExtTrig(int CardNum, bool ExtTrig);
```

Parameters
CardNum: 1 <= CardNum <= 4
ExtTrig: true = external trigger enabled; false = external input ignored

Return Value
0 if successful.

Example
```
DAx22000_SelExtTrig(1, true);     // Selects external trigger
```

5.4.2.7 DAx22000_Run

Description
This call is performed after all other setup calls have been done. It tells the card to start outputting waveforms. If TriggerNow is true, then the card creates a soft trigger once to get things going.

Declaration
```
int DAx22000_Run(int CardNum, bool TriggerNow);
```

Parameters
CardNum: 1 <= CardNum <= 4
TriggerNow: true => add software trigger to start things off. Otherwise the trigger must be supplied somewhere else like software trigger or external trigger.

Return Value
0 if successful.

Example
```
DAx22000_Run(1, true);     // Starts outputting waveforms from board #1. Provides
                              initial trigger to start things off.
```

5.4.2.8 DAx22000_Stop

Description
This is the opposite of DAx22000_Run. It basically shuts off all output signals.

Declaration
```
int DAx22000_Stop(int CardNum);
```

Parameters
CardNum: 1 <= CardNum <= 4

Return Value
0 if successful.

Example
```
DAx22000_Stop(1);
```

5.4.2.9 DAx22000_SoftTrigger

Description
This is a software trigger and works whether or not the external trigger is enabled.

Declaration
```
int DAx22000_SoftTrigger(int CardNum);
```

Parameters
CardNum: 1 <= CardNum <= 4

Return Value
0 if successful.

Example
```
DAx22000_SoftTrigger(1);   // Creates artificial trigger for card #1.
```

5.4.2.10 DAx22000_Place_MRK2

Description
This fuction call allows the user to pick one point during the waveform play time to output a positive pulse on TTL output Marker #2. The user must calculate the value by picking a point in the waveform (a time) and dividing that by 16. 16 is the resolution of the comparator. The width of the pulse is 1/16 of the sample rate (e.g. 8ns at 2.0 GS/sec).

Declaration
```
int DAx22000_Place_MRK2(int CardNum, int Mod16_CNT);
```

Parameters
CardNum: 1 <= CardNum <= 4
Mod16_CNT: 0 <= Mod16_CNT <= (max memory)/16

Return Value
0 if successful.

Example
```
DAx22000_Place_MRK2(1, 2);        // Produces a 3.3V TTL output on Marker #2 at
                                     sample location (16*2)= 32.
```

5.4.2.11 DAx22000_CreateSingleSegment

Description
This call provides the simplest way to create a continuous waveform looping on itself (NumLoops = 0), or a single triggered waveform.

Declaration
```
int DAx22000_CreateSingleSegment(
   DWORD CardNum,
   DWORD ChanNum,
   DWORD NumPoints,
   DWORD NumLoops,
   DWORD PAD_Val_Beg,
   DWORD PAD_Val_End,
   PVOID pUserArrayWORD,
   DWORD Triggered
);
```

Parameters
```
CardNum:           1  <= x <=  4
ChanNum:           1  <= x <= 2
NumPoints:         48 <= x <= (Max Memory - 1024)  [must be modulo 16]
NumLoops:          1  <= x <= 65,534  (0 = continuous loop)
PAD_Val_Beg:       0  <= x <= 4095
PAD_Val_End:       0  <= x <= 4095
pUserArrayWORD:    Pointer to UserArray of type WORD (unsigned short).
Triggered:         0 => Waveform starts up first time but cannot be triggerd later
                        without shutting down first.
                   1 => Allows waveform to be triggered more than once while running.
```

Return Value
0 if successful.

Example
```
DAx22000_CreateSingleSegment(
   CardNum,
   ChanNum,
   MemoryDepth,
   NumLoops,
   2047,
   2047,
   pTempArrayWord,
   1);
```

5.4.2.12 DAx22000_CreateSegments

Description
This function allows the programmer to create multple segments and define the characteristics of each segment.

Declaration
```
int DAx22000_CreateSegments(
   DWORD CardNum,
   DWORD ChanNum,
```

```
   DWORD NumSegments,
   DWORD PAD_Val_Beg,
   DWORD PAD_Val_End,
   PVOID pSegmentsList,
   bool Loop
);
```

Parameters

```
CardNum:            1  <= x <= 4
ChanNum:            1  <= x <= 2
NumSegments:        1  <= x <= 60
PAD_Val_Beg:        0  <= x <= 4095
PAD_Val_End:        0  <= x <= 4095
pSegmentsList:      Pointer to SegmentsList which is array of SegmentStruct (see below).
Loop:               1 allows it to loop entire string of segments.
```

```
typedef struct
{
   PVOID    SegmentPtr;     // Pointer to User Data (Type = Array of WORD)
   DWORD    NumPoints;      // Number of points in segment (min=48, then mod 16)
   DWORD    NumLoops;       // Number of extra times to repeat current segment
   DWORD    TrigEn;         // If > 0 then wait for trigger before next segment.
} SegmentStruct;
```

Return Value

0 if successful.

Example

```
DAx22000_CreateSegments(
   CardNum,
   ChanNum,
   NumSegments,
   PAD_Val_Beg,
   PAD_Val_End,
   pSegmentsList,
   Loop
);
```

5.4.2.13 DAx22000_PWR_DWN

Description

This call powers down the DAx14000 for serial numbers > 231 and the DAx22000 for serial numbers > 178. "DAx22000_Initialize" will start it up again. Please note that after initialization the user must reload all settings since they are lost during power down.

Declaration

```
int DAx22000_PWR_DWN(int CardNum);
```

Parameters

```
CardNum:            1  <= x <= 4
```

Return Value

0 if successful.

Example

```
DAx22000_PWR_DWN(1);    // Shuts down DAx14000 module #1.
```

5.4.2.14 DAx_RUN_STATUS *(optional hardware after 2017.10.29)*

Description
This function call returns "77h" when a waveform is outputing, "33h" when idle. If the generator is waiting for a trigger, then it is considered idle.

Declaration
```
DWORD DAx_RUN_STATUS(DWORD CardNum);
```

Parameters
```
CardNum:            1  <= x <= 4
```

Return Value
0x77 when outputing, 0x33 when idle.

Example
```
if (DAx_RUN_STATUS(1) = $33) then else;  // Pascal
```

5.4.2.15 DAx_Set_TRIG_REP_RATE *(optional hardware after 2017.10.29)*

Description
This function call sets an internal asynchronous trigger repetition rate between 20 KHz and 200 KHz. It does not enable counter. See DAx_EN_TRIG_REP to enable/disable this internal trigger.

Declaration
```
DWORD DAx_Set_TRIG_REP_RATE(DWORD CardNum, float REP_RATE);
```

Parameters
```
CardNum:            1  <= x <= 4
REP_RATE:            20e3 < REP_RATE < 200e3
```

Return Value
Actual Value with format DWORD.

Example
```
DAx_Set_TRIG_REP_RATE(1,20e3);    // Sets up trigger counter for 20KHz clock rate.
```

5.4.2.16 DAx_EN_TRIG_REP *(optional hardware after 2017.10.29)*

Description
This function call enables/disables the internal trigger repetition rate circuitry. See "DAx_Set_TRIG_REP_RATE" to set the actual trigger rate.

Declaration
```
DWORD DAx_EN_TRIG_REP(DWORD CardNum, DWORD Enable);
```

Parameters
```
CardNum:            1  <= x <= 4
Enable:             Enabled for anything greater than 0.
```

Return Value
0 if successful.

Example
```
DAx_EN_TRIG_REP(1,1);    // Enables internal clock rate generator.
```

5.4.2.17 DAx22000_Ext10MHz

Description
This function call must be called AFTER an *optional* call to "DAx22000_Initialize". When (Enable = 1) the DAx22000 expects a 10 MHz external clock reference signal. Maximum amplitude = 3Vpp; min = 0.7Vpp. For low jitter the clock should be squarewave not sinewave. When (Enable = 0) then the DAx22000 uses the internal 50 MHz clock.

Declaration
```
int DAx22000_Ext10MHz(int CardNum, int Enable);
```

Parameters
```
CardNum:            1  <= x <= 4
Enable:             Enabled for anything greater than 0.
```

Return Value
0 if successful.

Example
```
DAx22000_Ext10MHz(1,0);    // Sets up DAx22000 for 50 MHz clock reference (int/ext).
```

5.5 Programming Examples

5.5.1 C/C++ Example File (included with drivers)

The following C++ code was compiled using G++ wich is part of the GNU Compiler Collection (GCC). Since WavePond only uses 100% open source development tools, it does not directly support proprietary compilers. Internally, WavePond and Chase Scientific Company use Lazarus/FreePascal (i.e. Object Pascal) for readability, consistancy, reliability, and security.

```
--------------------------------------------------------------------------------
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
//#include <string.h>
#include <math.h>
//#include <windows.h>


#include "dax22000_lib_DLL64.h"

//using namespace std; // better to use std::cout, std::cin

/* run this program using the console pauser or add your own getch, system("pause") or input loop
*/

int main(int argc, char** argv)
{
   DWORD NumCards = 0;
   DWORD CardNum = 1;
   DWORD Chan = 1;
   int x;
   double Actual_Frequency;

   WORD TempArray[8000];
   DWORD MemoryDepth = 512;
   double pi = 3.14159265358979;

//--------------------------------------------------------------------------
// CREATE SAMPLE SINEWAVE TO UPLOAD
//--------------------------------------------------------------------------
   for (x=0; x < (MemoryDepth); x++) {
      TempArray[x] = (unsigned int) ( ceil( 2047.5 + 2047.5*sin( 2.0*pi* x/(32) ) ) );
   }

   std::cout<<"TempArray loaded with Sinewave.\nHit key to continue ...\n";
   std::cin.get();

//--------------------------------------------------------------------------
// CHECK IF CARD DETECTED; IF NOT THEN EXIT PROGRAM
//--------------------------------------------------------------------------
   NumCards = DAx22000_GetNumCards();

   std::cout << "Number of Cards Detected = " << NumCards << "\nHit key to continue ...\n";
   std::cin.get();

   if (NumCards != 1) exit(0);

//--------------------------------------------------------------------------
// OPEN DRIVER, INITIALIZE, SET CLOCK RATE
//--------------------------------------------------------------------------

   x = DAx22000_Open(1);

   std::cout << "DAx22000_Open = " << x << "\nHit key to continue ...\n";
```

```
   std::cin.get();

   x = DAx22000_Initialize(1);

   std::cout << "DAx22000_Initialize = " << x << "\nHit key to continue ...\n";
   std::cin.get();

   Actual_Frequency = DAx22000_SetClkRate(1, 2.0e9);

   std::cout << "DAx22000_SetClkRate = " << Actual_Frequency << "\nHit key to continue ...\n";
   std::cin.get();


//---------------------------------------------------------------------------
// UPLOAD USER WAVEFORM
//---------------------------------------------------------------------------

   x = DAx22000_CreateSingleSegment(
      1,           // DWORD CardNum
      1,           // DWORD ChanNum
      64,          // DWORD NumPoints,
      0,           // DWORD NumLoops,        // 0 = Continuous Loop
      2047,        // DWORD PAD_Val_Beg,
      2047,        // DWORD PAD_Val_End,
      TempArray,   // PVOID pUserArrayWORD,
      1            // DWORD Triggered        // 1 = User can initiate "NumLoops" above by
   );                                        //     triggering externally or SoftTrigger.

   std::cout << "DAx22000_CreateSingleSegment = " << x << "\nHit key to continue ...\n";
   std::cin.get();


//---------------------------------------------------------------------------
// OUTPUT DATA
//---------------------------------------------------------------------------

   DAx22000_Run(1, true);

   std::cout << "Outputing Data.\nHit key to close driver and shut off card.\n";
   std::cin.get();

//---------------------------------------------------------------------------
// STOP OUTPUT AND CLOSE DRIVER
//---------------------------------------------------------------------------

   DAx22000_Stop(1);

   DAx22000_Close(1);

   return 0;
}
```

5.5.2 (TBD)

6 Programmers Interface #2 (Script Method)

6.1 Introduction

The Script method consists of a persistent API, loaded at boot time, and a simple text file with your instructions. The simplest instructions to output a waveform looks like this (assume file name is test.txt):

```
Load sweep.txt
run
```

Then to execute these commands you would simply copy "test.txt" over to "dax_cmd.txt". The API (loaded at boot) will sense that "dax_cmd.txt" file exists, execute it, erase it, and then wait for the file (your commands) to show up again.

That's it. If you want something more automated, you can use batch files or your favorite programming tool (C/C++, Pascal, Python, Java, etc.). The key here is that compiler compatibilities are no longer an issue (i.e. no DLL's, weird error messages, etc.).

6.2 USB Driver Installation

WinXP, Win7/8/10 should install drivers automatically when card is plugged in "if" auto-update is enabled. If not, see USB driver installation guide for the appropriate operating system instructions.

6.3 Script API Installation

Create a batch file with the following command line:

```
start "" "dax22000_GUI_64.exe" load_api
```

Use "dax22000_GUI_32.exe" for 32-bit Windows. Then use the "Task Scheduler" on Windows 7/8/10 (or autoexec.bat for older versions) to execute this at computer startup/login.

To unload the RAM resident Script API code you can double click on the Chase Logo in the System Tray or add this at the end of your Script file: `unload_api`

However, this is not necessary. Your computer will remove it when the computer is rebooted or powered down.

6.4 Script API Calls

The API calls are the same as the calls in the DLL section **BUT they have no brackets, commas, or the leading "DAx22000_"** with the exception of the following functions:

```
Load (text file)    // Assumes module number 1, pre/post 2047 value, loop infinite.

CreateSingleSegment 1 1 128 0 2047 2047 sq_wave.txt 0
CreateSegments 1 1 3 2047 2047 test_segs.txt true
```

The first one was added to create the simplest call possible. Standard calls are slightly more complicated as shown in the 2nd two.

For CreateSingleSegment please note that instead of a pointer you include a filename where the data is found. The file format is the same as that required for the GUI. Each line is a value between 0 and 4095.

And for CreateSegments the "test_segs.txt" file includes a list of instructions in the following example format:

```
sweep.txt 128 1 0
64K_Data.txt 512 1 0
sq_wave.txt 128 1 0
```

Where the parameters are as follows:

SegmentFile => File of User Data
NumPoints => Number of points in segment
NumLoops => Number of extra times to repeat current segment
TrigEn => If > 0 then wait for trigger before going to next segment.

The number of points can be less than those in the file, but cannot be more.

6.5 Script API Examples

Below is an example of a more realistic script that is representative of the DLL calls.

```
// Simple Script

Stop 1                  // Make sure generator is stopped. Must be stopped before
                        // any waveform is created (i.e. downloaded) or clock is
                        // changed.

SetClkRate 1 1e9        // Changed D/A sampling rate one module 1 to 1.0 GHz.

CreateSingleSegment 1 1 128 0 2047 2047 sq_wave.txt 0
                // Creates a single segment of 128 samples using "sq_wave.txt"
                // on channel #1 looping continuously.

Run 1 true      // Turns on the output using pre-programmed waveform data above
                // and provides a software trigger to get it started.
```

==> Please refer to DLL section for the individual parameter descriptions.

7 (TBD - placeholder)

7.1 (TBD)

8 MISCELLANEOUS

8.1 Calibration

The DAx14000, DAx22000, and DAx12000 have no user features to perform calibration. DC offsets on the analog outputs can be adjusted by the factory to place the center value (2047 out of 4095) above, below, or at zero volts.

8.2 Maintenance

No maintenance is required.

Trademarks:

WavePond is a registered trademark of Chase Scientific Company. Windows is registered trademarks of Microsoft Corporation. MAC OS X is a registered trademark of Apple Computer. Android is a registered trademark of Google.