



Data Design Corporation
Gaithersburg, MD

DG101 CPCI MODULE 4 CHANNEL DIGITAL DELAY GENERATOR

May 2006

Data Design Corporation
7851-A Beechcraft Avenue
Gaithersburg, MD 20879
WWW.DATADESIGNCORP.NET
(301) 670-1157

CONTENTS

1.0 Introduction	3
1.1 Specifications Summary	4
1.2 Setup and Installation	5
1.3 Operational Background	5
1.4 Front Panel	6
2.0 Using DG101 Software.....	7
2.1 Configuring A DG101.....	8
2.2 Operating A DG101	8
2.3 Saving A Configuration	9
3.0 DG101 Software Source Code.....	10
3.1 DG101 Software Architecture	11
3.2 The DG101 API	13
3.2.1 Calibration Functions	15
3.2.2 Abstract Instrument Functions.....	16

1.0 Introduction

The DG101 Digital Delay Generator is a compact PCI (CPCI) instrument module designed to furnish four precisely timed outputs when triggered by an external input. Each output delay and terminal pulse width is independently programmable. Delays up to 167 ms may be programmed with one nanosecond resolution, while extremely low jitter makes each cycle precisely repeatable. Software controlled and automatic triggering features also make the DG101 operable as a precision pulse generator. Outputs are provided by high power pin driver devices with a programmable amplitude and include a reference output for applications which must compensate for insertion delay. A simple rising edge electrical trigger input with programmable threshold is provided. An external clock input is also provided to allow multiple modules to be synchronized to an external frequency reference.

The DG101 and included software provide a turnkey digital sequencing solution for the ubiquitous Windows operating system. The application is designed to take advantage of the modular instrument environment providing for systems with high channel count from multiple modules. The software provides a simple method to enter delay settings and other parameters and to control operation. System wide settings for an entire array of DG101 instruments can be saved and recalled. The application coexists seamlessly with other instrument system components and can often be used as the only interface to the delay generator.

Finally, the DG101 software source code is provided as open source. In a world of data acquisition now muddied by thousands of semi-standard and custom solutions it makes sense to provide a properly layered design accessible to the user for unique needs. The principles of the software design are defined in this manual and are intended to allow the advanced user to peel back the layers and snap in other front end software. Such efforts are always underway and popular additions will be published on the Data Design website as they become available.

1.1 Specifications Summary

Dataway Interface

Compact PCI at 32 Bits and 33 MHz
Compliant with PICMG 2.0 R3.0
Single Width 3U Compact PCI Card

Resolution

Delay: 1 nS
Width: 10 nS

Outputs

4 Channels, 0 to 5V amplitude
1 Reference, 5V amplitude
Sourced from 50 ohms

External Trigger Input

Single ended DC coupled positive edge
Programmable from 0V to 5V
470 Ω 10 pF

Ambient Temperature Range

0 To 70 C

Jitter

< 250pS RMS cycle to cycle
Measured from trigger to output

External Clock Input

10 MHz frequency
5V amplitude
1 K Ω 20 pF

1.2 Setup and Installation

The DG101 is a single width 3U CPCI module. To insert the DG101 in the CPCI crate, choose any convenient slot and slide the module into the crate with its top and bottom mated to the guide rails. Be sure that the module is properly aligned with the connector at the back of the crate. Push the module toward the back of the crate with gentle pressure. When the card pushes back, push the locking lever down and press the card into the crate with firm pressure on the top and bottom of the front panel. The locking lever will raise and lock into place. Fasten the retaining screws at the upper end of the front panel and at the bottom of the front panel below the lever. To remove the card, reverse the process by removing the retaining screws and pressing down firmly on the locking lever to disengage the card from the connector at the rear of the crate.

NOTE: The module should be inserted in the crate only after turning off the power to the crate. Otherwise, damage to the module or system is possible due to momentary misalignment of pins on the connector. CPCI hot swap operation is not supported.

It is recommended that the DG101 module and a controller be installed alone in the crate until the user is familiar with operation enough to integrate it with other modules in an instrumentation system. When the DG101 is installed in a slot where it has not been before, the CPCI system will detect it as a new device at which point it will need access to driver software. The software supplied with the DG101 is designed to operate on platforms with Microsoft Windows 2000 and newer operating systems. To install drivers on these systems, point the installation wizard to the location of the included CD. A CD drive may be located on the crate controller or it may be mounted over a network connection using the wizard's "specify a location" option. The files the installation wizard will need are located in the root directory of the CD. Follow the steps displayed by the installation wizard at the end of which the wizard should indicate that a "DG101 Delay Generator" has been installed.

To install the DG101 software, use the Windows explorer or similar tool to locate the software CD. Point to the directory on the CD identified as \DG101\Install. Run the program SETUP.EXE and follow the on screen instructions for the familiar software installation process. When any new software has just been installed, it is usually best to reboot the crate controller before attempting to use the software, even when this is not strictly necessary. At this point the installation is complete and the DG101 and turnkey software are ready to be used.

1.3 Operational Background

The principal functionality of the DG101 is to produce a precisely timed delay sequence from a time reference point defined by a trigger event. In the idle state all outputs are at zero volts. A rising edge on a trigger starts the sequence. After an insertion delay, a rising edge will appear at the reference output. This insertion delay will be consistently a few tens of nanoseconds for a given instrument, but may vary slightly from one DG101 to another. Following the reference rising edge event, rising edges will appear on each channel after a specified *delay* time. Each channel will remain at full output voltage for the specified *width* time

and then return to zero volts. When the last channel returns to zero volts output, the instrument will have returned to the idle state. This sequence of idle – trigger – delay – width – idle is referred to as a *cycle*. The cycle is complete when the last channel has returned to the idle state.

To respond to a trigger event, the DG101 must be *armed*. When a cycle is in process the instrument will not respond to a subsequent trigger. After a cycle is complete, the instrument will not respond to a trigger until it is *rearmed*. The DG101 can be instructed to automatically rearm when the cycle is complete.

Finally, the DG101 can be programmed to automatically insert a new trigger event when the cycle is complete. In this mode the channel outputs appear to operate as a pulse generator with pulses of the programmed width appearing at each channel. The period, or specifically the time between rising edges, will be the sum of the cycle time (the longest combined delay plus width of the four channels) and the time it takes the DG101 to internally prepare the module for another trigger. The first portion of the period is therefore directly programmed and the second portion is a fixed time of a little over a microsecond. This does limit the pulse frequency to somewhat less than one megahertz. Though software could be written to make a fairly usable pulse generator from the DG101, this is not really intended to be the strong point of the module. However, the auto trigger mode can be quite useful for system setup tasks such as quickly verifying programmed pulse widths and sequences.

1.4 Front Panel

The DG101 front panel provides seven SMB snap fit coaxial connection points to the instrument electronics. These are labeled for the four delayed signal output channels, a reference output channel, a trigger input, and an input for an optional external frequency standard. There is also a serial number label on the card ejector lever. This number will be presented by the DG101 software to indicate which DG101 is being addressed, which may be important when there are multiple such instruments in the system.

While the instrument has some immunity from typical connection mistakes, care should be used when connecting external devices. Always observe the maximum ratings indicated in the specifications for each input and output. Information on input tolerance, output levels, and input and output impedances will have a significant impact on instrument system design.

The front panel trigger input is a 0 to 5 volt signal typical of the input drive to TTL logic devices and is terminated in 470 ohms. However, the trigger is taken as an analog signal and the threshold is programmable, but is always taken on the rising edge. The programmable threshold allows other logic standard amplitudes to be used. When the module is armed, this signal or an equivalent signal provided by software will be required to start the cycle.

The amplitude of the signal from the channel outputs are programmable from 0 to 5 volts, but as a group and not individually. The reference output signal is always 5 volts in amplitude. All outputs are set to open circuit levels and are sourced from 50 ohms. The signal on each channel will remain at full amplitude for the programmed output width. The reference output will be at full amplitude for 40 ns after the insertion delay has elapsed. The reference clock can be an internal source or can be externally applied. When an external 10 MHz clock source is applied to the front panel connector, the DG101 automatically locks to that clock instead of the internal source.

2.0 Using DG101 Software

The DG101 is provided with a turnkey software package which allows the user to quickly take advantage of all instrument features in a standalone configuration. This software is designed to coexist with other instruments as well as to allow multiple DG101 instruments to be installed and operated in the same CPCI crate. Installation of the software is discussed in section 1.2 above.

Upon starting the DG101 software, a splash screen will be displayed while the software searches for and configures all DG101 instruments in the system. The software will locate each DG101 and complain if none can be found. The software will configure each unit found to default settings and the control panel screen shown in Figure 2.1 will appear. This interface allows the user to control all DG101 instruments in the system from one instance of the control panel. The DG101 being addressed is specified by serial number at the top of the window. This serial number corresponds to that found on the card ejector lever of the instrument. The status and settings of each DG101 can be recalled and adjusted by clicking on this selection bar to address other units in the system. No more than one application should attempt to control DG101 instruments in the system at the same time.

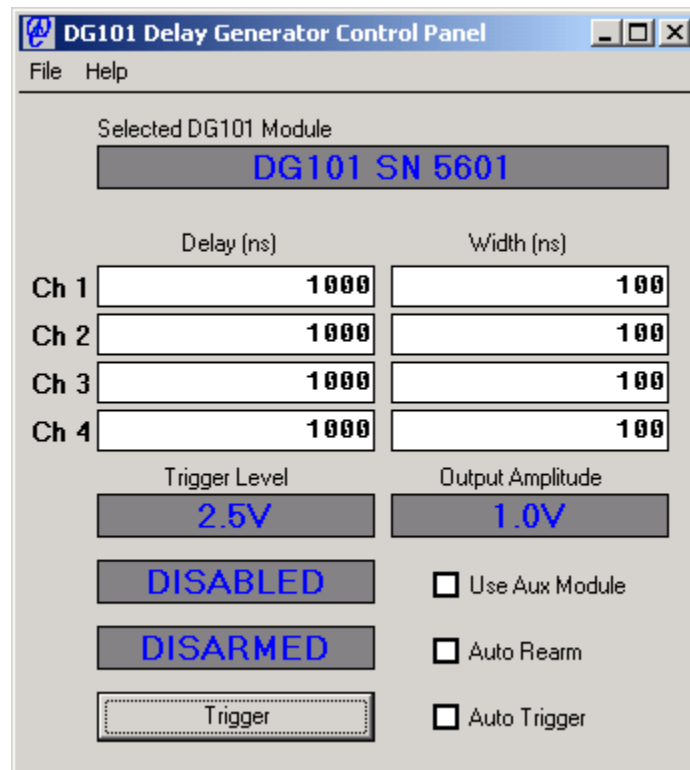


Figure 2.1 DG101 Control Panel

2.1 Configuring A DG101

The DG101 is generally configured before any cycles are started, even though this is not specifically required. The configuration will pass directly to the instrument being addressed as specified in the *Selected DG101 Module* control. The delay and width associated with each channel can be entered directly. As new values are entered the module is updated. The *Trigger Threshold* level and the *Output Amplitude* can be adjusted by clicking on these controls.

The blue on gray controls in the DG101 software can be adjusted by clicking on them with the mouse buttons. For most controls the left mouse button will increase the value of the control and the right mouse button will return its value.

2.2 Operating A DG101

The controls *Enabled* and *Armed* establish the state of the instrument. Clicking on these controls changes their state. Arming the unit prepares it to receive a trigger and run a cycle. The enable control is an output enable. That is, even as cycles are run the outputs will stay at zero volts unless the unit is enabled. The controls will indicate the true status of the DG101 on a regular update basis, so if a status change request is not accepted or if the status changes through the normal course of operation, these controls will reflect the current status.

The DG101 is designed to accommodate an auxiliary output module in 6U form factor. Such modules provide special output features such as high voltage amplitude or fiber optic interfaces. The control *Use Aux Module* enables outputs from this auxiliary module when present but only when the DG101 itself is enabled. In units which do not have the auxiliary module attached this control has no effect. Contact the factory for available auxiliary module configurations or custom output requirements.

In the armed state a trigger event will start a cycle at the outputs. When a trigger event is received, the programmed output cycle is executed and the DG101 returns to the disarmed state. If the *Auto Rearm* control is checked, the DG101 will then rearm at the end of each cycle and wait for the next trigger event without user intervention. The control panel will continue to show the DG101 as armed.

The trigger event can come from the front panel input or from software. Click the *Trigger* button to issue a trigger event. If the DG101 is armed, this will cause a cycle to be executed. If the *Auto Trigger* control is checked, a trigger event will also be issued and then reissued internally after each cycle is complete. For this to cause repeated cycles in a pulse generator fashion, the module must be armed and the *Auto Rearm* control must also be checked. If the module is disarmed, a new trigger must be issued to start the pulses again.

For a system of multiple DG101 instruments there is still only one unit being addressed by the software at a time as identified by serial number. The typical procedure will be to address and configure each unit in the system. Then each unit in the system can be addressed again to be enabled and armed. At that point all such units can be operated for single or multiple cycles as configured without software intervention.

2.3 Saving A Configuration

The DG101 software monitors the configuration and status of each unit in the CPCI system. The configuration information for all units in a system can be stored to disk so that it can be restored when the system is next operated in that configuration. The *Save Settings* option in the *File* menu allows the user to save the configuration of all DG101 units in the system to a user specified file.

The *Open Settings* control in the *File* menu allows the user to restore a specified configuration to a system of DG101 modules. It is important to note that this configuration information is specific to a particular system with units of a known serial number installed in a known CPCI slot. If anything about the hardware configuration has changed, the configuration restore process will fail and a new configuration will have to be constructed manually. Constructing a configuration is not very time consuming of course, but when several units are in use, the *Save Settings* and *Open Settings* functions can be convenient.

3.0 DG101 Software Source Code

Instruments such as the DG101 are only as useful as the software which runs them. While the software provided with the DG101 makes the instrument very functional in a wide variety of applications, it is impossible to predict the full range of user needs or the software environment in which the instrument may be installed. Data Design has decided that the best way to meet as many user needs as possible is to provide a window into the workings of the instrument. To accomplish this, the source code for the software is published with the software distribution.

This open source approach provides the user with nearly infinite flexibility. It also provides the opportunity for many bad headaches. The inner workings of the software are not for the novice user. While the DG101 software in particular is not a very demanding application, complex tricks and traps exist with any advanced instrument design. It can be a challenge for even the most skilled programmer. The good news is that the design is carefully structured and this section of the manual provides a roadmap to the advanced user.

IMPORTANT NOTICE Information on source code and the source code itself is provided AS IS and without warranty or support. Data Design is not able to provide technical assistance to those attempting to modify the source code. It simply would not be possible to provide the instrument at reasonable cost if such support were included in the price. Please review the source code license agreement at the end of this manual before using the source code. That said however, Data Design does offer design services for hire. Information can be found on the website at the front of this manual. Also, from time to time, additional applications and source code will be posted on the website for free download.

It should be noted that this section of the manual covers the workings of a particular code base for a particular instrument. Any such software works with a hardware-software interface defined in part by the hardware itself. At the simplest level this interface amounts to writing various command and configuration registers and reading various data and status registers. Still, understanding many of the complexities found internal to the code base might require a working knowledge of the hardware. The hardware itself is not discussed in this manual and is considered somewhat proprietary to Data Design. While it will become evident in this discussion that the hardware could also be modified to meet additional unique and complex requirements, this is substantially beyond the scope of most users' needs and abilities. As such, to maximize the usability of the open source environment, the interface to hardware is left somewhat abstract in order to simplify the discussion. This understanding should be kept in mind when reviewing the source code and deciding what should or should not be modified.

While there is some discussion of hardware details within the source code itself, the discussion in this manual is aimed at common reasons why an advanced user would require access to the source code. These include such tasks as interfacing to a custom top level application, connecting the instrument to an existing software environment, or using the instrument in another operating system. As such, the discussion of the top level application presented in section 2 is also limited, though the source code for that application is also provided.

3.1 DG101 Software Architecture

The hardware and software design of the DG101 is rooted in well accepted flexible, layered design concepts. The functional stack of all elements in the instrument design and data flow are shown in Figure 3.1 below.

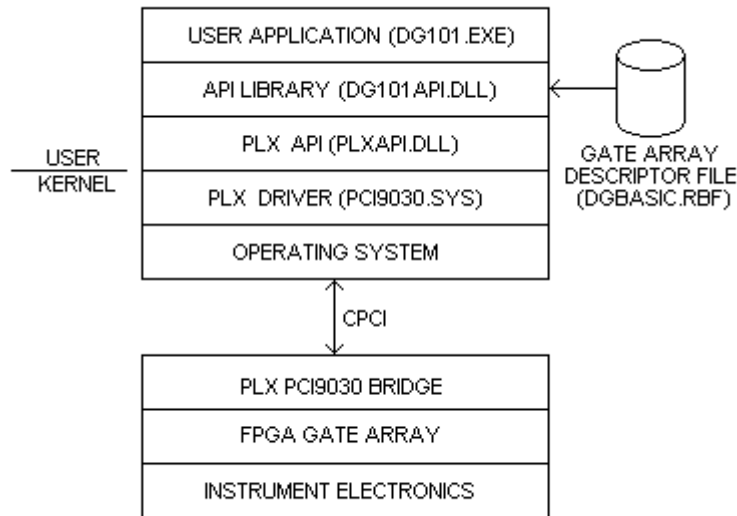


Figure 3.1 DG101 Software Architecture And Data Flow

The instrument hardware is mostly defined by a design which is software loadable into a field programmable gate array (FPGA) on the main board. The binary image of this hardware design is saved on disk in the host computer and loaded to the instrument when the software is started, or more specifically when the instrument application programming interface (API) library is loaded in memory. The FPGA hardware design contains functional blocks to interface with the instrument electronics and the PCI interface. Where the software addresses specific registers on the instrument hardware, these registers are located within the FPGA hardware.

The PCI interface itself contains a popular PCI bridge from PLX Technologies. The advantage of using this device is that the gate array can be soft loaded and even potentially reloaded after the host system is running. This provides flexibility and the ability to apply future updates and enhancements to the hardware without the need to return the instrument to the factory. The PCI9030 also has support available for a number of operating systems, including native kernel support in some popular operating systems such as Linux. Under Windows a comprehensive set of drivers and API features takes a lot of the work out of commonly needed operations.

The DG101 software distribution includes an installation information file which directs the installation wizard to install the Windows WDM kernel level driver and the associated PLX API for the benefit of a newly installed DG101 instrument. These driver and API files contain the code necessary to work with the operating system and the bridge chip on the other side of the PCI bus. They are provided by PLX Technologies for distribution with systems that use these bridge chips. When working with the software at this level under any operating system it may be helpful to acquire a PLX Technologies software development kit. Generally, because the

instrument API is developed well within the user layer, it will not be necessary to use any kernel level driver development kits. However, some operating systems may draw this line at other points.

The DG101 API library contains code which performs the most difficult functions required to operate the instrument and exposes to the layer above a comprehensible set of calls which perform abstract functions related to DG101 hardware. The API is provided as a dynamic link library (DLL) written in the C programming language. This is where the bulk of the DG101 software is located and is the primary subject of this section of the manual. The API is designed to relieve the top level application from even the most rudimentary house keeping duties. When the library has been successfully loaded by a top level user application, all DG101 units in the system will be configured and ready for use.

The top level user application is left with very little substantial work to do. This is the layer which provides all the fancy features – the buttons, lights, dialogs, and message boxes. In the DG101 turnkey software package, the top layer is written in Microsoft Visual Basic 6 (VB). It could have as well been written in National Instruments LabVIEW, Borland Delphi, Microsoft C, or any other environment which can access a dynamic link library as an API. The source code for the VB application is included in the distribution but is not discussed in this manual. It may be instructive to review the source code in the VB environment. The program begins by loading the main form, aptly called Form1, which will automatically load the API library.

3.2 The DG101 API

The DG101 application programming interface (API) is provided as a Windows dynamic link library (DLL) which is installed in the Windows system directory (\WinNT\System32 or equivalent). Also stored at this location is the binary image file for the gate array in the instrument. When the API DLL (DG101API.DLL) is loaded by the application, or by the operating system on behalf of the application, initialization code is executed which locates all modules in the system, loads the gate arrays on those modules, and sets default configuration.

The API exports functions which provide an easy path to use features of the instrument. Exported functions, data structures, and definitions which will be needed by an application calling the API are defined in the DG101API.H header file. All exported functions are defined with a name prefix of DG101_ and will always return an unsigned 32-bit number which will be set to 1 if the function performs as expected and 0 if a difficulty is encountered. Any substantive data returned by a function is returned by reference in the arguments.

Several steps are taken to improve compatibility with a wide variety of programming environments. An export definition file (DG101API.DEF) is part of the source code so that exported names are explicitly defined for the benefit of environments which may not be aware of Microsoft name mangling standards. All arguments of exported functions are atomic types and pointers to atomic types as opposed to complex data structures which may be complicated to define in some environments. The source code is written in the C programming language with some Microsoft conventions geared towards the Windows programming model for the Intel architecture. The following are some useful relationships between data types for arguments in several software environments.

API Argument Type	Description	VB Type	LabVIEW Type
BYTE	Unsigned 8-bit number	ByVal Byte	u8
DWORD	Unsigned 32-bit number	ByVal Long	u32
long	Signed 32-bit number	ByVal Long	s32
char*	Pointer To Text String	ByVal String	s8*

There are several points to consider regarding range of arguments. Any type may also be presented in an argument as a pointer to data of that type, while the char (strictly a signed 8-bit number) always appears as a pointer in the API and refers to the location of a null terminated text string. Generally, it is an error for null pointers to be passed in arguments. Some environments require more work than others to avoid passing null pointers to strings. The BYTE data type is sometimes used as a number, but is often used as a proxy for a boolean type where 0 is false and anything else is true. The API avoids the use of 16-bit words of any kind because some environments do not support them or do not support an unsigned version. The use of the long integer type implies that the API is expecting a signed number. The DWORD data type implies that the API is expecting an unsigned number. However, in the later case the range will in most cases be less than 31 bits so that environments such as Visual Basic (VB) which do not have an unsigned 32-bit number can use a signed 32-bit number in its place.

Most advanced programming environments provide an interface to DLL's. However, all have peculiarities which must be understood for successful use. For example, in the VB environment an argument passed to the API is considered a pointer to a data type unless it is noted to be passed 'ByVal'. The VB type 'String' is a pointer, so the argument would be a pointer to a pointer unless the ByVal qualifier is applied to make it a pointer to a character array. The definitions of the API functions as required by VB can be found in the source code for the DG101 top level application.

The exported function list contains several classes of functions. These include abstract functions which contain identifiable hardware and user features of a the instrument – such as issuing a trigger. Another set of functions deals with calibration of the instrument or other house keeping tasks.

In addition to exported items and others enumerated in the header file, the source code contains a number of private functions and data variables. For the most part the private data are unique to a particular process. A process is an instance of the top level application, such as the user application for the DG101. The DG101 software is designed with the intent that one application will handle all DG101 instruments installed in the system. Therefore, only one instance of such an application should be running at any one time.

Even most advanced users will use this source code as a reference to build additional higher level applications. It would be unusual to actually modify this source code. However, the source code can be a valuable reference in understanding how each of the functions behave. There is extensive documentation in the source code itself. This manual will provide a roadmap of exported functions, an overview of their intended use, and how they fit into the overall software design.

3.2.1 Calibration Functions

There are no calibration points in the DG101 hardware. However, as a matter of consistency items which are set by the factory are established as calibration functions. For the DG101 the serial number is the only such permanently configurable data point. The function declaration is as follows.

DCAPI DG101_CAL_SerialNumber(DWORD *SerNum, BYTE OverWrite);

The API itself makes use of calibration functions to load associated variables and uses these variables throughout the API in normal operation. Because the turnkey application is available, it is hard to imagine a situation where a programmer would need to use current or future calibration functions in an alternative environment. Probably such things should in fact be avoided. The exception is the function DG101_CAL_SerialNumber which will return by reference the serial number written into nonvolatile memory on the module currently being addressed by the API. Be sure to set the *OverWrite* argument to 0 (false). If the serial number has not been corrupted by misuse of this function, the value returned will be the number on the front panel ejector handle of the module.

Note: A serial number setting control exists in the DG101 application and can be exposed by running the application with the *-C* argument. Generally, the user should not set the serial number, but in the tradition of open source nothing is held back for better or worse. However, the serial number from the point of view of the factory will always be that originally applied to the unit labeling.

3.2.2 Abstract Instrument Functions

The abstract instrument functions provide a means to perform a set of operations which are recognizable and expected for the delay generator. Presented below are the exported functions with a brief description of their place in the design. They may be ordered differently than in the source code for clarity of discussion.

The API opens, configures, and examines each DG101 instrument in the system when the library is first loaded. The configuration and other specifics about each instrument found are stored in a linked list in the API. This information is important for two reasons in understanding the software architecture. First, the API can only be accessed by one process (top level application) which controls all DG101 units in the system as only the first such process will be able to open any instruments. This is a typical configuration and is used by the DG101 turnkey software. This is also convenient for single process interpretive environments such as LabVIEW. Finally, the exported concept of opening an instrument actually just addresses an instrument in the linked list. Configuration values for each instrument are typically stored separately in the API and the instrument while status is read directly from the instrument.

Control functions for the DG101 mostly come in dual *Set* and *Get* forms. The arguments of both are passed by reference for consistency. The *Get* form allows an application to poll for the current state of items such as the armed status which are expected to change. With a given instrument addressed, the *Get* functions will return configuration values for the addressed unit as set by the API on library load or those later set by the user. The status values returned will be those provided by the hardware of the addressed instrument.

The DG101 addressed can be identified by serial number. Only one unit is addressed at a time. The typical procedure will be to address and configure each unit in the system. As each unit is addressed the application will retrieve configuration and status information for that unit. Each unit in the system can be addressed again to be enabled and armed. At that point all such units can be operated for single or multiple cycles as configured without software intervention.

DCAPI DG101_OpenNext(void);

When the library is loaded, the API will automatically open a software path to all DG101 instruments in the system. The first such unit will be addressed. Calling this function will cause the API to address the next DG101 in the list. All API functions will affect only the unit so addressed. If there is only one DG101 in the system, this function has no affect. Generally, when a new DG101 is addressed in this way, the software will need to retrieve its serial number to identify the new unit along with configuration and status information for that unit.

DCAPI DG101_SetEnabledStatus(BYTE *Enabled, BYTE *AuxEnabled);

DCAPI DG101_GetEnabledStatus(BYTE *Enabled, BYTE *AuxEnabled);

These functions directly control the enables of the currently addressed DG101 instrument. The arguments are 1 for enabled and 0 for disabled. The *Set* function updates the hardware of the currently addressed instrument with the specified values. The *Get* function returns in the arguments values read from the hardware. In general, an application will be aware that *AuxEnabled* really has no physical effect unless *Enabled* is also true.

DCAPI DG101_SetArmedStatus(BYTE *Armed, BYTE *AutoRearm);
DCAPI DG101_GetArmedStatus(BYTE *Armed, BYTE *AutoRearm);

These functions directly control the arming of the currently addressed DG101 instrument. The arguments are 1 for armed and 0 for disarmed. The *Set* function updates the hardware of the currently addressed instrument with the specified values. The *Get* function returns in the arguments values read from the hardware.

DCAPI DG101_SetDelay(BYTE Channel, DWORD *Delay);
DCAPI DG101_GetDelay(BYTE Channel, DWORD *Delay);

These functions directly control the delay values written to the currently addressed DG101 instrument. The delay is specified in nanoseconds and is stored to the linked list in this form. The *Get* function simply returns this value from the linked list. The *Set* function contains code to write a multipart hardware register which has a more complex format than nanoseconds alone and is not specified further.

DCAPI DG101_SetWidth(BYTE Channel, DWORD *Width);
DCAPI DG101_GetWidth(BYTE Channel, DWORD *Width);

These functions directly control the width values written to the currently addressed DG101 instrument. The width is specified in nanoseconds and is stored to the linked list in this form. However, because the width is only actually adjustable in 10 nanosecond steps, the value stored will be truncated to an even 10 nanoseconds. The *Get* function simply returns this value from the linked list. The *Set* function writes the final tens of nanoseconds value to a hardware register with little or no modification.

DCAPI DG101_SetOutputAmplitude(BYTE *Amplitude);
DCAPI DG101_GetOutputAmplitude(BYTE *Amplitude);
DCAPI DG101_SetTriggerLevel(BYTE *Level);
DCAPI DG101_GetTriggerLevel(BYTE *Level);
DCAPI DG101_LevelToVoltsString(BYTE Level, char *Value);

The output amplitude and the threshold for the front panel analog trigger input are both set as an 8-bit unsigned number representing 256 steps from 0 to 5 volts. The *Get* functions return a value from the linked list. The *Set* functions write a value to a hardware register. In some programming languages it can be challenging to translate the discrete stepped value to a human readable value in volts. The *LevelToVoltsString* function will accept a specified level from 0 to 255 and return a human readable character string corresponding to the level from 0 to 5 volts. While the resolution is 256 steps, most applications will not require this level of resolution. The string value presented is rounded to the nearest tenth of a volt.

DCAPI DG101_Trigger(BYTE *Auto);
DCAPI DG101_GetAutoTriggerStatus(BYTE *Auto);

The trigger function causes a trigger event in the hardware. If the module is armed, the trigger event will cause a cycle to start. The start of cycle event can be seen on the reference output. If the auto flag is true (not zero) then the completion of a cycle will cause a subsequent trigger event. This will continue until the auto flag is cleared. The *Get* function returns the current setting of the auto flag from the linked list.

DCAPI DG101_DefaultSettings(void);

When an application is started it will not be aware of the current status of connected instruments. This function is used by the API while the linked list is being created to install initial values for all delays, widths, and other settings. These initial values are also installed in the instrument hardware. The specific values clearly are somewhat arbitrary and could change. However, the function will serve the purpose of synchronizing the linked list with the hardware when the application is loaded. Because the API does this, there is really no need for an application to call this function unless the default values are of particular interest. An application displaying configuration values should use the *Get* functions to find the current settings when the application starts or after this function is called.

DCAPI DG101_SaveSettings(char *SettingsFileName);**DCAPI DG101_LoadSettings(char *SettingsFileName);**

The settings of a full system of DG101 instruments can be saved to disk. A settings file essentially contains the contents of the linked list. A unique filename including path information should be specified for the save or load target file. The usual error checking is undertaken, but it is the responsibility of the application to prevent unwanted overwrites of existing files. The load function requires that the settings file specified not only contains valid information but also specifies the same configuration of instruments currently existing in the system. This check comes into play as the instruments are reconfigured, making it possible that a system could be partially reconfigured before an inconsistency is detected and an error message displayed. This consequence is taken as an unusual event seen only when a system is changed and causes no problems other than a minor annoyance.

DG101 and *DG101 Software* are products of Data Design Corporation. No portion of this manual or any accompanying software may be reproduced by any means for any purpose other than for user archives and application of the *DG101* products without the express written consent of the copyright owner.

Copyright © 2006 Data Design Corporation
All rights reserved.

PRODUCT WARRANTY:

1. Product. The "Product" referred to herein is the instrumentation product *DG101* and the software product *DG101 Software*. The product as defined does not include the *DG101 Software Source Code* or documentation of such source code.

2. Limited Warranty. Data Design Corporation warrants that the Product will perform substantially in accordance with the accompanying documentation for a period of one year from the date of receipt. Any implied warranties on the Product are limited to one year. SOME STATES DO NOT ALLOW LIMITATIONS ON DURATION OF AN IMPLIED WARRANTY, SO THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

3. Remedies. The entire liability of Data Design Corporations and/or its suppliers and your exclusive remedy shall be, at the option of Data Design Corporation, either (a) return of the purchase price you paid or (b) repair or replacement of the Product that does not meet the Limited Warranty and that is returned to the place of purchase with a copy of your receipt. The Limited Warranty is void if failure of the Product has resulted from accident, abuse, or misapplication. Any replacement Product will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. THE FOREGOING CONSTITUTES YOUR SOLE AND EXCLUSIVE REMEDY UNDER THIS WARRANTY. EXCEPT FOR THE WARRANTIES SET FORTH ABOVE, THE PRODUCT AND DOCUMENTATION ARE PROVIDED "AS IS," AND DATA DESIGN CORPORATION AND/OR ITS SUPPLIERS DISCLAIM ANY AND ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

4. Limitations of Liability. The cumulative liability of Data Design Corporation and/or its suppliers to you or any other party for any loss or damages resulting from any claims, demands, or actions arising out of or relating to this warranty shall not exceed the purchase price you paid for the Product and documentation. In no event shall Data Design Corporation and/or its suppliers be liable for any indirect, incidental, consequential, special, or exemplary damages, loss of business profits, business interruption, or loss of business information, even if advised of the possibility of such damages. SOME STATES DO NOT ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

TRADEMARKS:

Commercial names and trademarks are used for identification purposes only and are recognized as the property of the respective trademark owners.

REVISION LEVEL:

Firmware Date: May 1, 2006
Software Version: 2.20
Manual Release Date: May 1, 2006

TERMS AND CONDITIONS OF LICENSE REGARDING *DG101 SOFTWARE SOURCE CODE*:

PLEASE REVIEW THE FOLLOWING TERMS AND CONDITIONS CAREFULLY BEFORE DOWNLOADING OR USING ANY SOURCE CODE OR SOURCE CODE DOCUMENTATION RELATED TO THE DG101 PRODUCT (the "SOURCE CODE"). BY USING THE SOURCE CODE, YOU INDICATE YOUR ACCEPTANCE OF THESE TERMS AND CONDITIONS, WHICH CONSTITUTE A LICENSE AGREEMENT (the "AGREEMENT") BETWEEN YOU AND DATA DESIGN CORPORATION ("DATA DESIGN"). IN THE EVENT THAT YOU DO NOT AGREE WITH ANY OF THESE TERMS AND CONDITIONS, DO NOT DOWNLOAD, COPY, OR USE THE SOURCE CODE. THE SOURCE CODE IS GOVERNED SOLELY BY THIS AGREEMENT AND NOT BY ANY OTHER AGREEMENT OR LICENSE THAT YOU MAY HAVE WITH DATA DESIGN, UNLESS PROVIDED IN WRITING. BY DOWNLOADING OR USING THE SOURCE CODE, LICENSEE ACKNOWLEDGES THAT HE/SHE HAS READ THIS AGREEMENT, UNDERSTANDS IT, AND AGREES TO BE BOUND BY ITS TERMS AND CONDITIONS

Under this Agreement, Data Design is providing source code for software related to the DG101 product as delivered and as may be amended or augmented via various mediums from time to time. LICENSEE MAY NOT USE, COPY, MODIFY, DISTRIBUTE, SUBLICENSE OR TRANSFER THE SOURCE CODE, OR MERGED OR COMBINED PORTION THEREOF, IN WHOLE OR IN PART, EXCEPT AS EXPRESSLY PROVIDED FOR IN THIS AGREEMENT. THE SOURCE CODE MAY NOT BE USED FOR ANY PURPOSE OTHER THAN THE DEVELOPMENT OF SOFTWARE APPLICATIONS FOR THE DG101 PRODUCT.

"Source Code" means one or more hardware or software design files in source format.

"Licensee" means YOU, the user of the Source Code, under this Agreement.

License Subject to the terms and conditions of this Agreement, Data Design grants to Licensee, a non-transferable, non-exclusive, perpetual license to use the Source Code. The Source Code, and the algorithms, concepts, techniques, methods and processes embodied therein, are proprietary to Data Design. Data Design retains all rights with respect to the Source Code, including any copyright, patent, and other proprietary rights, not expressly granted herein.

Licensee may implement the Source Code in a larger work, modify the Source Code, and create derivative works thereof. However, no copy or derivative work may be presented for sale in any venue without the express written consent of Data Design. All such works must be applicable only to use of the DG101 product. Any copy or portion of the Source Code, including any derivative works thereof, including any portion merged into a design and any design or product that incorporates any portion of the Source Code, will continue to be subject to the terms and conditions of this Agreement.

Any direct copies of the Source Code made by Licensee under this Agreement shall include all intellectual property notices, including copyright and proprietary rights notices, appearing on the Source Code as provided by Data Design.

No License To Trademarks Data Design reserves exclusive use of any and all of its trademarks.

Term This Agreement is effective until terminated. Licensee may terminate this agreement at any time by destroying the Source Code together with all copies, derivative works and portions thereof in any form (including any portions merged into a design). This Agreement will also terminate immediately upon Licensee's material breach or if Licensee fails to comply with any term or condition of this Agreement. Upon any termination of this Agreement, the license and rights of Licensee under this Agreement shall terminate, and Licensee shall destroy the Source Code, including all copies, derivative works and portions thereof in any form (including any portions thereof merged into a design).

Payment The Source Code is being provided to Licensee at no cost, except to the extent that the Source Code may only be used in conjunction with a product which Data Design offers for sale. Data Design shall be compensated for any customization of the Source Code performed by Data Design as agreed upon in writing by the parties.

No Warranties, Support, or Maintenance THE SOURCE CODE IS PROVIDED TO LICENSEE "AS-IS". LICENSEE ALSO AGREES THAT DATA DESIGN DOES NOT PROVIDE MAINTENANCE OR SUPPORT FOR THE SOURCE CODE. NO WARRANTIES, REPRESENTATIONS, OR GUARANTIES, EITHER EXPRESS OR IMPLIED, ARE MADE WITH RESPECT TO THE SOURCE CODE, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT. LICENSEE ASSUMES THE ENTIRE RISK AS TO THE QUALITY, COMPLETENESS AND PERFORMANCE OF THE SOURCE CODE AND ANY DESIGN OR PRODUCT IN WHICH THE SOURCE CODE MAY BE USED. SHOULD THE SOURCE CODE PROVE DEFECTIVE, DATA DESIGN ASSUMES NO LIABILITY FOR ANY COST OF ANY NECESSARY SERVICING, REPAIR, OR CORRECTION. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you in full, but shall be interpreted to apply to the maximum extent permissible under applicable law. DATA DESIGN DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE SOURCE CODE WILL MEET ANY REQUIREMENTS, OR THAT THE OPERATION OF THE SOURCE CODE WILL BE UNINTERRUPTED OR ERROR-FREE. LICENSEE ALSO ASSUMES RESPONSIBILITY FOR THE SELECTION OF THE SOURCE CODE TO ACHIEVE INTENDED RESULTS AND FOR THE INSTALLATION, USE, AND RESULTS OBTAINED FROM THE SOURCE CODE.

Limitations of Liability In no event shall the aggregate liability of Data Design relating to this Agreement or the subject matter hereof under any legal theory (whether in tort, contract, or otherwise), including any liability for any loss or damages directly or indirectly suffered by Licensee relating to the Source Code, exceed One Dollar (\$1.00). IN NO EVENT SHALL DATA DESIGN BE LIABLE UNDER ANY LEGAL THEORY, WHETHER IN TORT, CONTRACT OR OTHERWISE (a) FOR ANY LOST PROFITS, LOST REVENUE OR LOST BUSINESS, (b) FOR ANY LOSS OF OR DAMAGES TO OTHER SOFTWARE OR DATA, OR (c) FOR ANY INCIDENTAL, INDIRECT, CONSEQUENTIAL, PUNITIVE OR SPECIAL DAMAGES RELATING TO THIS AGREEMENT OR THE SUBJECT MATTER HEREOF, INCLUDING BUT NOT LIMITED TO THE DELIVERY, USE, SUPPORT, OPERATION OR FAILURE OF THE SOURCE CODE, EVEN IF DATA DESIGN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LIABILITY AND NOTWITHSTANDING ANY FAILURE OF THE ESSENTIAL PURPOSE OF ANY LIMITED REMEDY STATED HEREIN. LICENSEE ACKNOWLEDGES THAT DATA DESIGN HAS NO RESPONSIBILITY OR DUTY TO DEFEND, INDEMNIFY OR HOLD LICENSEE HARMLESS FROM AND AGAINST ANY CLAIMS, SUITS, PROCEEDINGS, DAMAGES, LOSSES, COSTS AND EXPENSES, BASED ON PATENT OR OTHER INTELLECTUAL PROPERTY CLAIMS.

U.S. Government Restricted Rights If Licensee is an agency or instrumentality of the United States Government, the Source Code and related documentation are "commercial computer software" and "commercial computer software documentation", and pursuant to FAR 12.212 or DFARS 227.7202, and their successors, as applicable, use, reproduction and disclosure of the Source Code and related documentation are governed by the terms of this Agreement.

Severability If any provision of this agreement is held by a court of competent jurisdiction to be legally ineffective or unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable and the validity of the remaining provisions shall not be affected.