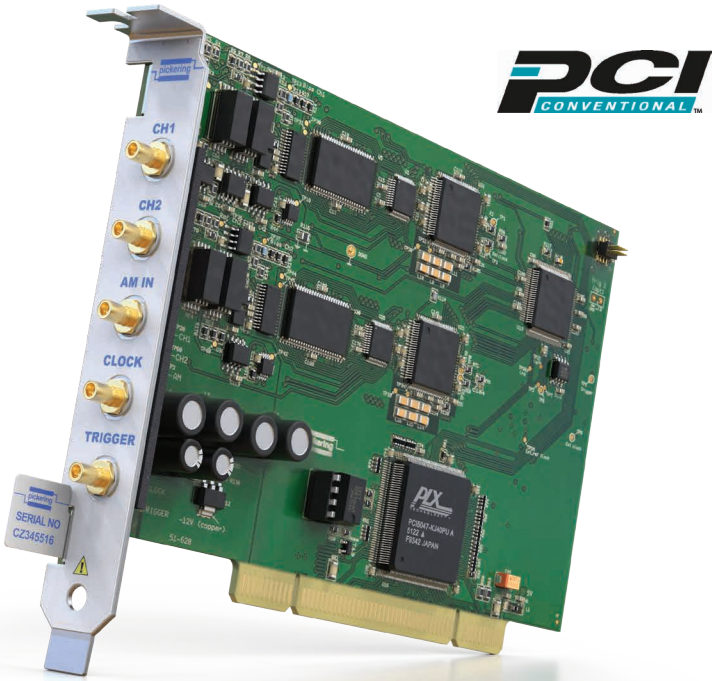
# PCI Function Generator Card



**© Copyright (2026) Pickering Interfaces.** All Rights Reserved.

No part of this publication may be reproduced, transmitted, transcribed, translated or stored in any form, or by any means without the written permission of Pickering Interfaces.

Technical details contained within this publication are subject to change without notice.

FM38792

# Technical Support

For Technical Support please contact Pickering Interfaces either by phone, the website or via e-mail.

## Warranty

All products manufactured by Pickering Interfaces are warranted against defective materials and workmanship for a period of three years, excluding programmable power supplies, from the date of delivery to the original purchaser. Any product found to be defective within this period will, at the discretion of Pickering Interfaces be repaired or replaced.

Products serviced and repaired outside of the warranty period are warranted for ninety days.

Extended warranty and service are available. Please contact Pickering Interfaces by phone, the website or via e-mail.

## Environmental Policy

Pickering Interfaces operates under an environmental management system similar to ISO 14001.

Pickering Interfaces strives to fulfill all relevant environmental laws and regulations and reduce wastes and releases to the environment. Pickering Interfaces aims to design and operate products in a way that protects the environment and the health and safety of its employees, customers and the public. Pickering Interfaces endeavors to develop and manufacture products that can be produced, distributed, used and recycled, or disposed of, in a safe and environmentally friendly manner.

## Worldwide Technical Support and Product Information

### Pickering Interfaces Headquarters

Stephenson Road, Clacton-on-Sea, CO15 4NL United Kingdom
Tel: +44 (0)1255-687900
e-mail: sales@pickeringtest.com

**USA**

Tel: +1 781 897 1710
e-mail: ussales@pickeringtest.com

**France**

Tel: +33 9 72 58 77 00
e-mail: frsales@pickeringtest.com

**Germany**

Tel: +49 89 125 953 160
e-mail: desales@pickeringtest.com

**Sweden**

Tel: +46 340-69 06 69
e-mail: ndsales@pickeringtest.com

**Czech Republic**

Tel: +420 558 987 613
e-mail: desales@pickeringtest.com

**China**

Tel: +86 4008 799 765
e-mail: chinasales@pickeringtest.com

**Malaysia**

Tel: +60 12 833 7980
e-mail: aseansales@pickeringtest.com

## Safety Symbols

The following safety symbols may be used on the product and throughout the product documentation.

| Meaning/Description | Symbol |
|---|---|
| **PROTECTIVE EARTH (GROUND)**<br>To identify any terminal which is intended for connection to an external conductor for protection against electric shock in case of a fault, or the terminal of a protective earth (ground) electrode. |  |
| **DANGEROUS VOLTAGE**<br>To indicate hazards arising from dangerous voltages. |  |
| **WARNING/CAUTION**<br>An appropriate safety instruction should be followed or caution to a potential hazard exists.<br>Refer to the relevant instructions detailed within the product manual. |  |
| **HEAVY**<br>If this product is heavy reference should be made to the safety instructions for provisions of lifting and moving. |  |
| **STATIC SENSITIVE**<br>To indicate that static sensitive devices are present and handling precautions should be followed. |  |

# Contents


Copyright Statement ..... (ii)
Technical Support and Warranty ..... (iii)
Product Safety ..... (iv)
Contents (this page) ..... (v)
Warnings and Cautions ..... (vi)

**Section 1**

Technical Specification ..... 1.1

**Section 2**

Technical Description ..... 2.1
- Functional Description ..... 2.1

**Section 3**

Installation ..... 3.1
- Pre Operation Checks ..... 3.1
- Hardware Installation ..... 3.2
- Software Installation ..... 3.2
- Testing Operation ..... 3.3

**Section 4**

Programming Guide ..... 4.1
- Programming Options for Pickering PCI Cards ..... 4.1
- Programming the Function Generator ..... 4.2
- Programming Examples ..... 4.5
- Function Tree ..... 4.7
- C Functions ..... 4.10
- Using Pickering Drivers In LabVIEW ..... 4.46

**Section 5**

Connector Information ..... 5.1

**Section 6**

Trouble Shooting ..... 6.1

**Section 7**

Maintenance Information ..... 7.1

# Warnings & Cautions

| WARNING - Hazardous Environments | Symbol |
|---|---|
| This product is not specifically designed for use in hazardous environments, for example in explosive atmospheres. If the product is to be used in hazardous environments we recommend that the user ensures suitable protective measures are taken. |  |

| WARNING - Danger of Electric Shock | Symbol |
|---|---|
| This module may contain hazardous voltages. Before removing the module from the rack remove all supplies and disconnect user I/O signals.<br>Unused slots in the PXI/PXIe/LXI chassis are populated with blanking plates to prevent access to user I/O signals that may be present. Blanking panels are available to order from Pickering in a variety of slot widths. If the product is not used in this manner for example by using an extender card then additional care must be taken to avoid contact with exposed signals.<br>Not to be used in safety critical circuits, refer to the Pickering Interfaces' terms & conditions of sale.<br>This module must not be used for the switching of Mains Circuits. For the switching of voltages up to the module's full specification, Secondary Circuit power supplies and the Device Under Test must be fully isolated from the Mains Circuit and the safety earth.<br>The host chassis must be correctly installed and be connected to mains or factory safety ground, please refer to your chassis installation manual. |  |

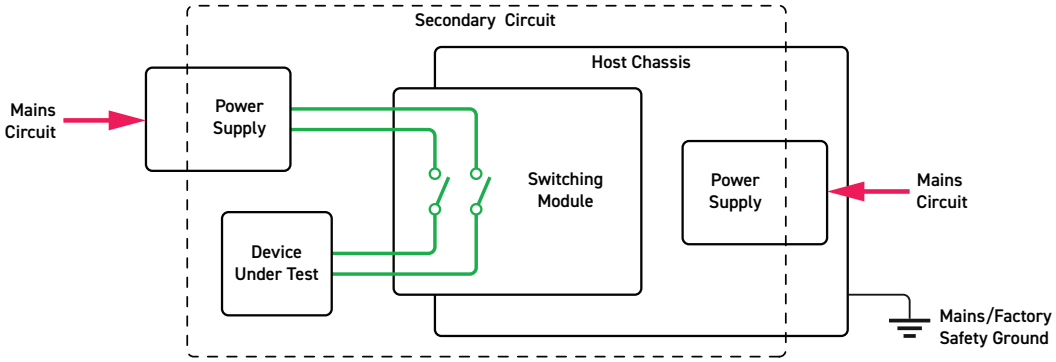



**Diagram Showing Separation Between The Mains Circuit and Secondary Circuit / DUT**

For the purposes of this document, a Mains Circuit is an installation classified as Overvoltage Category II up to 300 V. Separation between the Secondary Circuit and Mains Circuit is achieved by a transformer in which the primary windings are separated from the secondary windings by reinforced insulation, double insulation, or a screen connected to the protective conductor terminal.

| CAUTION - Handling of Electrostatic-Sensitive Devices | Symbol |
|---|---|
| Certain semiconductor devices used in this equipment are liable to damage due to static voltage. Observe the following precautions when handling these devices in their unterminated state, or sub-assemblies containing these devices:<br>• Persons removing sub-assemblies from equipment using these devices must be earthed by a wrist strap and a resistor at the point provided on the equipment.<br>• Soldering irons used during the repair operations must be low voltage types with earthed tips and isolated from the mains voltage by a double insulated transformer.<br>• Outer clothing worn must be unable to generate static charges.<br>• Printed Circuit Boards (PCBs) fitted with these devices must be stored and transported in anti-static bags. |  |

| CAUTION - Product Documentation | Symbol |
|---|---|
| Suitably qualified & trained users should ensure that the accompanying documentation is fully read and understood before attempting to install or operate the product. |  |

## Safety Instructions

| Safety Instructions |
|---|
| All cleaning and servicing requires the equipment to be isolated and disconnected from the power source and user I/O signals (refer to the Maintenance Section).<br>• Appropriate manual handling procedures should be followed as dictated by the weight of the individual module or the combined weight of the modules & chassis.<br>• Should a fault occur with the module or chassis, immediately isolate and disconnect the incoming power to the chassis and the user I/O signals.<br>• Ensure the equipment is installed, operated and maintained by trained and authorised personnel.<br>• For suitably equipped products in the event of an emergency press the red "emergency stop" button situated on the front of the unit. |

# Section 1 - Technical Specification

- DC to 10 MHz
- Two Channels in One PCI Slot
- 48-bit Frequency Resolution
- Simple Generation of Repetitive Arbitrary Waveforms
- DC Offset Capability
- Flexible Sweep Capability
- Amplitude Modulation Capability
- Uses 10 MHz Internal Clock or External Clock Reference
- External Trigger Support
- VISA Driver Supplied
- 3 Year Warranty




The 51-620 is a compact 2 channel function generator in short PCI card format. It is capable of generating sine waves to 10 MHz with 48-bit frequency resolution referenced to an onboard 10 MHz clock or to an external standard. It can generate arbitrary waveforms loaded into the internal 256 k memory, allowing the function generator to emulate many waveform types, including the typical waveforms of automotive and aerospace sensors.

The function generator provides a very simple method of providing variable output frequencies through the use of Direct Digital Synthesis (DDS), making it far easier to use than an ARB for repetitive waveform generation.

The 51-620 is capable of generating fast swept frequency signals, permitting the output to emulate the operation of variable speed devices. Sweeps can be single shot events or continuous up and down ramps. The DDS technology ensures the 51-620 settling time is extremely fast, being limited only by the update time over the PCI bus.

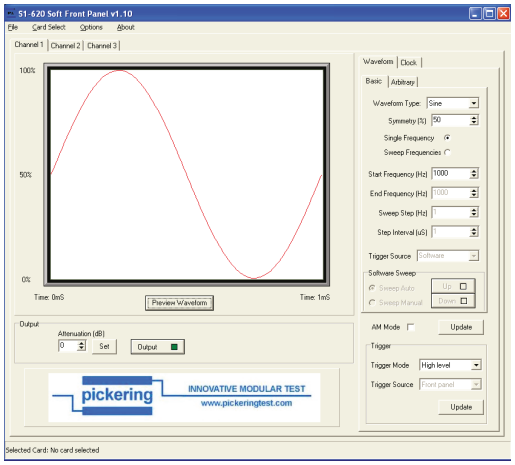
The channels of the 51-620 can be amplitude modulated from a single AM input connector allowing independent time varying output levels to be reproduced. One channel of the card can be externally connected to provide the time varying level control independently of the waveform being generated.

The waveforms of the function generators are each stored in independent 256 k memory blocks, permitting each channel to provide a different waveform shape.

Waveforms are easily created externally, using Excel or similar tools, and loaded to the card's memory via the backplane.

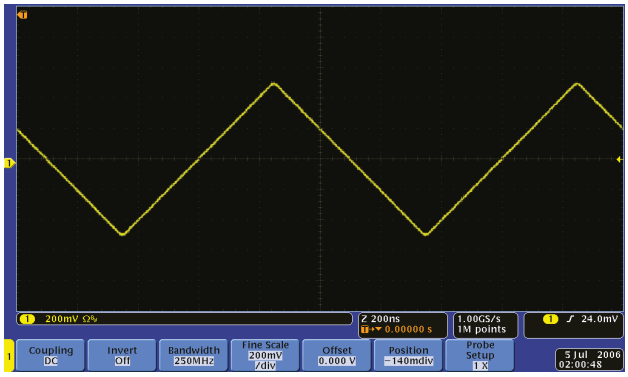
The 51-620 supports trigger functions to allow triggered events from other instruments to initiate waveform generation or sweeps.

The supplied soft front panel demonstrates the ability of the 51-620 to quickly generate common waveforms and the ability to import external waveforms. The soft front panel also supports swept modes of operation.




**Soft Front Panel for the 51-620 Function Generator**

## Typical Waveforms Generated by 51-620 With Output Attenuation Set to 10 dB and Loaded into 50 Ω



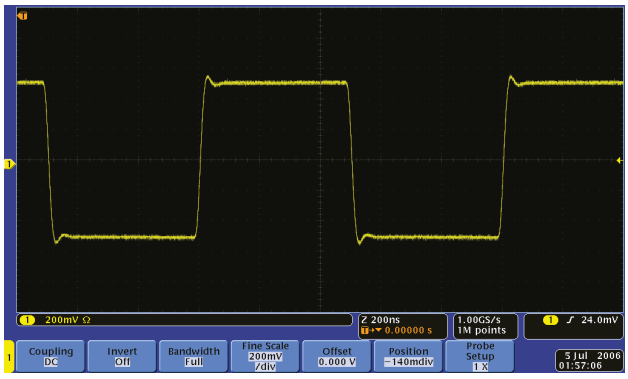

Triangular Waveform at Frequency of 1 MHz



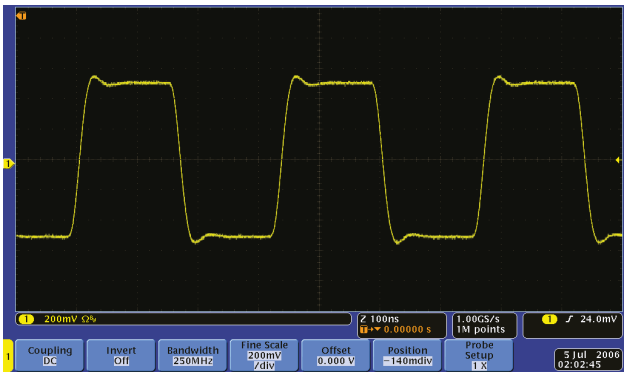

Triangular Waveform at Frequency of 3 MHz



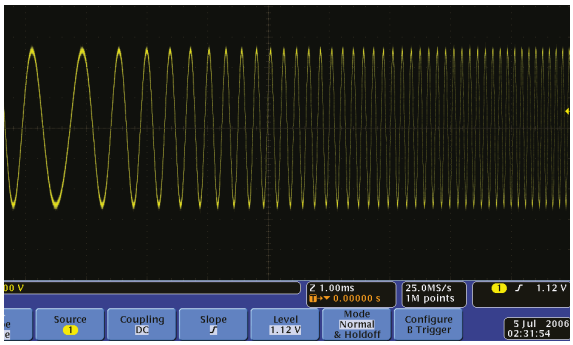

Triangular Waveform at Frequency of 5 MHz



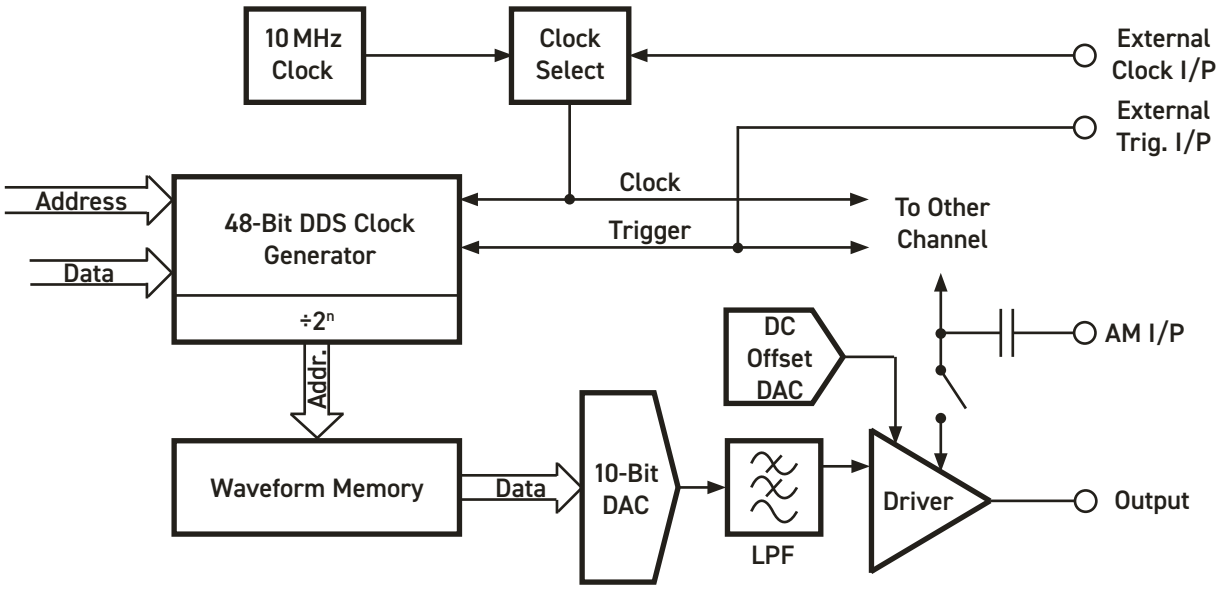

Square Wave at Frequency of 3 MHz




Square Wave at Frequency of 5 MHz




Swept Sine Wave from 1 kHz to 10 kHz in 10 Hz Steps at 10 µs Intervals




Function Generator Block Diagram

# Section 1 - Technical Specification

## Specification (each channel)

| | |
|---|---|
| **Frequency** | |
| Frequency Range: | DC to 10 MHz (sine wave). |
| Frequency Resolution: | 48-bit |
| Frequency Accuracy: | As onboard 10 MHz clock. |
| Frequency Sweep: | Single or continuous (ramp up and/or down) phase continuous through use of DDS sweep facility. |
| **Output** | |
| Maximum Output: | ±10 V, open circuit load. |
| Waveform Signal: | 10 V pk to pk, open circuit load |
| Out Offset Voltage: | Settable from -5 V to +5 V in 10 mV steps. |
| Output DAC Resolution: | 10-bit |
| Signal Level Control: | 0 to -40 dB, <0.1 dB steps. |
| Output Impedance: | 50 Ω |
| Output Loading: | 600 Ω with 10 V peak e.m.f. Typically capable of driving a 50 Ω load with 6 V peak e.m.f. output (DC and signal) |
| **AM Input** | |
| Functionality: | Single input can be used to modulate one, two or three output channels. AC coupled, modulation frequency range: 10 Hz to 20 kHz. |
| **Waveforms** | |
| Waveform Memory: | 256 k per channel fixed length. Permits any waveform to be loaded and replayed, including sine, square, ramp. Waveform fidelity limited by fixed 15 MHz low pass filter. |
| **Parametric Performance** (AM & DC offest to 0) | |
| SFDR: | >40 dB (DC to 1 MHz, 5 Vpp)<br>>35 dB (1 MHz to 10 MHz, 5 Vpp) |
| Amplitude Accuracy: | ±2 % DC to 1 MHz<br>±5 % 1 MHz to 10 MHz |
| **Trigger & Clock** | |
| Clock Source: | 10 MHz internal 50 ppm clock or external clock source. |
| Trigger: | Level or edge operation initiates single shot or continuous operation from external source. |

## Power Requirements

| +3.3 V | +5 V | +12 V | -12 V |
|---|---|---|---|
| 1.8 A | 110 mA | 75 mA | 150 mA |

## Mechanical Characteristics

Single slot short PCI format.
3D models for all versions in a variety of popular file formats are available on request.

## Connectors

Signals via SMB connectors, for pin outs please refer to the operating manual.

## Safety & CE Compliance

All cards are fully CE compliant and meet applicable EU directives:

Low-voltage safety EN61010-1:2010,
EMC Immunity EN61326-1:2013,
Emissions EN55011:2009+A1:2010.

## PCI Compliance

The 50-295 complies with the PCI Specification 2.0 (issued Feb 2004).

Signalling Environment: 33 MHz, 32-bit Universal (+3.3 V or +5 V).

## Operating/Storage Conditions

| | |
|---|---|
| Operating Temperature: | 0 °C to +55 °C |
| Humidity: | Up to 95 % non-condensing |
| Altitude: | 5000 m |
| Storage Temperature: | -20 °C to +75 °C |
| Humidity: | Up to 95 % non-condensing |
| Altitude: | 15000 m |

# Section 1 - Technical Specification

## Product Order Codes

| | |
|---|---|
| Dual Channel PCI Function Generator | 51-620-002 |

## Accessories

| | |
|---|---|
| Calibration/Verification Cable (2 required): | 40-978-501-1M |

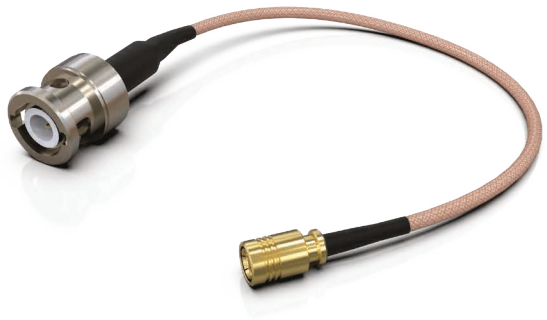
## Product Customization

Pickering PCI cards are designed and manufactured on our own flexible manufacturing lines, giving complete product control and enabling simple customization to meet very specific requirements.

All customized products are given a unique part number, fully documented and may be ordered at any time in the future. Please contact your local sales office to discuss.

## Mating Connectors & Cabling

For connection accessories for the 51-620 card please refer to the 90-011D RF Cable Assemblies data sheet where a complete list and documentation can be found for accessories.



**Pickering can supply mating RF connectors and cable assemblies to enable easy integration of the 51-620 series of function generator cards**

## Functional Description

The function generator can generate all commonly required waveforms and can be loaded with arbitrary waveforms generated by the user and imported into the function generator memory. The use of a DDS (Direct Digital Synthesizer) clock for the function generator allows very fine frequency control and a great deal of flexibility in the use of the module.

### CW Frequency Generation

The frequency reference for the module is typically a 10 MHz source from either the front panel or the from the onboard clock. This reference frequency is used for both channels. The reference does not have to be 10 MHz but the following description assumes that is.

Each function generator uses the same design. The 10 MHz signal is passed to a DDS that internally multiplies the clock frequency by a factor of between 4 and 20 (m). For most applications the multiplication factor should be set to 20 in order to have the maximum DDS clock frequency of 200 MHz and eliminate the risk of alias signals.

The DDS is then programmed to generate a typical frequency of between 40 MHz and 80 MHz using the Frequency Tuning Word (FTW).

The DDS is used to clock a function generator that uses an address counter and a memory block. The waveform to be generated is stored in a 256 k by 16 bit memory block, this waveform can be of any shape (including sine wave) but has to be stored with a fixed length (i.e. all 256 k points must be stored). The address counter is an 18 bit binary counter that can be programmed to increment through the waveform memory with a different step size (n) of between 0 and 15.

> **Example:** For a function generator output frequency of 10 MHz the DDS is set to produce 80 MHz. Full scale plus one LSB on the FTW is 200 MHz. The address counter (n) is set to increment in steps of 215 (n=15). Since the counter size is 18 bits it takes 23 cycles for the counter to cycle through the memory waveform. To reduce the output frequency to 5 MHz n can be reduced to 14, or the DDS frequency can be reduced to 40 MHz.

For most applications the DDS output frequency should be between 40 and 80 MHz and the frequency is lowered by reducing the value of n, each reduction in n covering a new octave of frequency cover. If very low output frequencies are required then n can be set to 0 and the frequency further lowered by entering FTW that produces DDS frequencies below 40 MHz. It is possible to generate waveforms that take days to complete using this method.

The general formula for the output frequency is as follows:

$$f_{signal} = (2^{(n-18)} \cdot f_{ref} \cdot m \cdot FTW) / 2^{48}$$

Where

$f_{signal}$ is the output frequency.

**n** is the address generator step size from 0 to 15.

**m** is the DDS reference clock multiplier, normally set to 20 for a clock frequency of 10 MHz.

$f_{ref}$ is the reference clock frequency, normally 10 MHz.

**FTW** is the frequency tuning word to the DDS.

It should be noted that the DDS clock frequency must always be at least twice the frequency at the output of the DDS.

### Swept Frequency Operation

For swept frequency operation the DDS frequency is swept to provide a variable output frequency. The highest frequency in the sweep should be used to determine the setting for n. The value of n is kept fixed which will mean that lower frequencies are generated with a DDS output frequency of less than 40 MHz.

> **Example:** If the output is to be swept from 1 MHz to 10 MHz then m is set to 20 and n is set to 15. With the FTW set to produce 80 MHz at the DDS output the output frequency will be 10 MHz (same example as before). With the FTW set to produce 8 MHz at the DDS the function generator output frequency is 1 MHz.

The DDS frequency is swept by entering a start frequency using the FTW and a Delta Frequency Word (DFW) to determine the step size. The DFW can be positive or negative, if it positive the frequency sweeps up and if it is negative the frequency sweeps down.

> **Example:** In case above the start frequency should be set with a FTW producing 8 MHz, producing a function generator output frequency of 1 MHz. If the frequency increment is required to be 10 kHz then the DFW should be set to 80 kHz (the factor of 8 reflects the division ratio resulting from setting and increment size of n=15 in the 18 bit binary counter).

The frequency sweep can be stopped at any time.

### FSK Mode

The output frequency can be set to change between two frequencies.

### Waveform Definition

Each function generator channel can provide an output that corresponds to a waveform loaded in the memory.

The memory size is 256k words. In many cases the full waveform is not used. In the example where a 10 MHz signal is being generated from a 80 MHz DDS frequency, just 8 of the samples are sampled, as the output frequency is lowered, using the factor n, more samples are used. The user can elect to load just a few samples of the waveform when this is the case, but needs to be aware that if the waveform is then "played" at a lower frequency with different values of n then the missing samples will appear as errors.

If a waveform is recorded that has sharp changes of level between waveform points, but this signal is only sampled at a few places these changes may be missed. This is in keeping with the intended way the function generator operates since this only occurs for high output frequencies where the analog BW of the system would in any case restrict the rate of change of output voltage.

### Alternative Modes of Use for Frequency Generation

There is a considerable degree of hardware flexibility in the design of the 51-620 and it can be used it a number of different ways. The External standard frequency need not be 10 MHz, it could for example be some binary referenced frequency standard. The value of the DDS clock multiplier (m) can be set to a frequency other than 20 (e.g. to 16) to retain this binary relationship. The DDS clock frequency must be set less that 200 MHz. The DDS itself can be set to frequency ranges other than 40 to 80 MHz (a facility used on the sweep). However the DDS output includes an anti-alias filter designed to reduce alias frequencies that cause clock timing dither. In most cases this dither is negligible, but users can set the various parts of the block diagram to conditions where unusual interpretations of the waveform can occur.

**Output Level Processing.**

Having generated a variable output frequency with a defined shape the digital signal is then converted to an analog signal using a 10 bit DAC, the output of which is filtered by a fixed low pass filter. The output level can be changed in fine step sizes by adjustment of the Reference DAC that supplies a reference to the 10 bit DAC. The Reference DAC can be used to vary the output level over a 20 dB range. Further reduction in output level is obtained using a 20 dB switched attenuator.

A DC offset can be applied to the output amplifier using the DC Offset DAC.

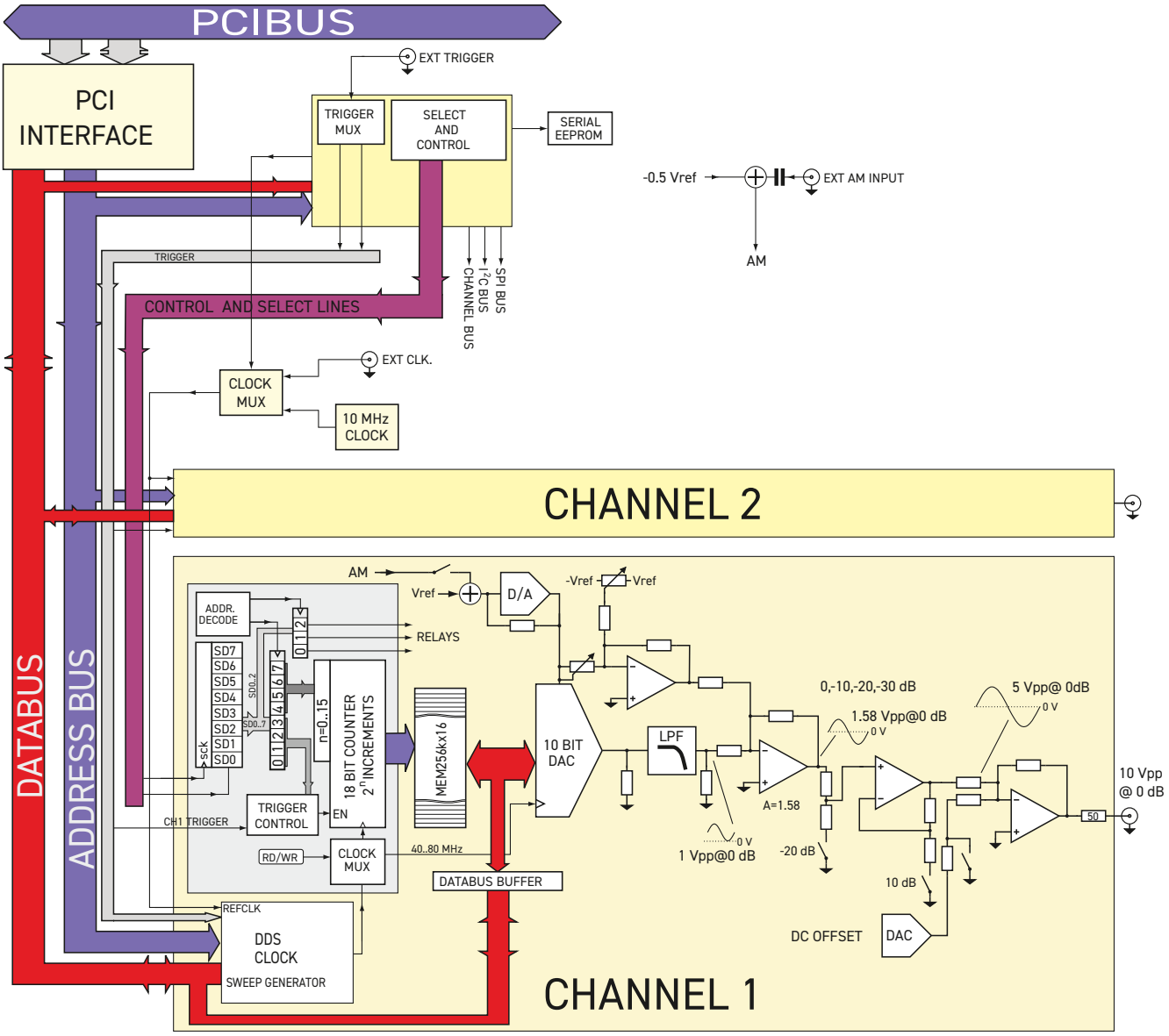



**Figure 2.1 - 51-620: Functional Block Diagram**

The module has two independent programmable function generator channels. Each channel consists of a DDS clock generator and a user programmable function generator. The DDS clock generator provides the function generator with a high resolution programmable clock, variable between 40 and 80 MHz. The clock generator uses a reference clock (10 MHz on-board clock or an external clock source) shared by both channels.

Each function generator channel consists of:

- An **address counter** the increments of which can be programmed in 2n steps. It can operate in continuous or single shot mode.
- **Trigger hardware** that starts the address counter, it is positive/negative and edge/level sensitive programmable.
- A **signal memory** for a user programmable waveform of 256k points, 16 bit wide.
- **A 10 bit D/A converter** (signal DAC) that generates the programmed waveform.
- **Variable gain control** for setting signal levels from 10 Vpp down to 100 mVpp, programmable in < 0.1dB steps.
- **DC offset DAC** that adds a programmable DC offset to the output signal.
- **AM function**. The output can be amplitude modulated by an external AC signal. The signal is shared by both channels and varies the reference voltage of the signal DAC the modulation frequency range is 10 Hz to 10 kHz.

### SMB CH1 & CH2 Output Connector & AM Connector Ground

To mitigate noise issues when using PC power supply grounds as a reference the 51-620 uses a separate analog ground which is connected by a 2.2 Ω resistor to the PC power supply ground. The outer of the SMB connectors is connected to the analog ground and insulated from the 51-620 front panel, in doing so referencing the 51-620 analog circuits to the remote user ground. Users should not use the coaxial shield as a means of grounding a test system to avoid ground currents causing noise or damage. Users should ensure that the test system has a local ground that the 51-620 uses as its ground reference.

# Section 3 - Installation

## Refer to the Warnings and Cautions at the front of this manual

Modular products require installation in a suitable chassis. The card is designed for indoor use only.

## Pre-operation Checks (Unpacking)

1. Check the card for transport damage and report any damage immediately to Pickering Interfaces.
   Do not attempt to install the product if any damage is evident.
2. Position the chassis relative to any other equipment the card(s) will connect with. Ensure the chassis is not connected to the electrical supply.
3. Ensure that the designated area for the chassis containing the card(s) is of flat and solid construction to withstand and support the combined weight of the PC chassis and cards.



**Figure 3.1 - Inserting a PCI Card into a PC Chassis**

## Hardware Installation Electrical Connection (Chassis)

The PCI card must be electrically installed in accordance with local regional electrical codes by a qualified engineer.

The PC chassis must be grounded.

Electrostatic discharge can damage the components on the card and in the chassis. To avoid such damage in handling the board

refer to page vii.

Ensure that there is adequate ventilation.

### The card should be installed in accordance with the following procedure:

1. Ensure that the system is turned OFF but still connected to mains so that it remains grounded.
2. Choose an appropriate free PCI slot in the host system.
3. Remove the blanking plate for the chosen slot and retain the fixing screw.
4. Carefully remove the card from its anti-static bag and insert the card's edge connector into the free slot in the host system (refer to Figure 3.1).
5. Ensure the card is fully inserted into the edge connector and the front panel is fully in contact with the back of card aperture in the host chassis.
6. Use the fixing screw to secure the front panel of the card to the host chassis' mounting rail.
7. Replace the chassis cover(s).
8. Power-up the host system.

## Software Installation

First install the appropriate Pickering PCI switch card drivers by running the installer program Setup.exe (provided in a compressed zip file), either from the CD-ROM supplied, or by downloading the latest version from our website: pickeringtest.com - the recommended method. Setup is accompanied by a ReadMe file containing additional installation information. A single installation covers all Pickering switching and simulation PCI, PXI and PXIe products.

When installation completes, the installed drivers' ReadMe file is offered for display. It can also be displayed later using a shortcut on the Programs>>Pickering menu.

If you are not a LabVIEW user you should choose the "full" version, and once that has been installed run the LabVIEW Runtime Engine installer via the shortcut on the Programs>>Pickering menu. In the absence of LabVIEW the Runtime Engine is required to support the Pickering Test Panels application.

## Testing Operation

After installation of the hardware and Pickering PXI Driver software package start the General Soft Front Panel (GSFP) using the desktop icon or via the Start menu (***Start Menu-> Pickering Interfaces Ltd -> General Soft Front Panel***).




**Figure 3.2 - General Soft Front Panel Icon**

A selector panel will appear, listing all installed Pickering PCI, PXI or LXI switch cards and resistor cards. Click on the card you wish to control, and a graphical control panel is presented allowing operation of the card. Panels can be opened simultaneously for all the installed cards.

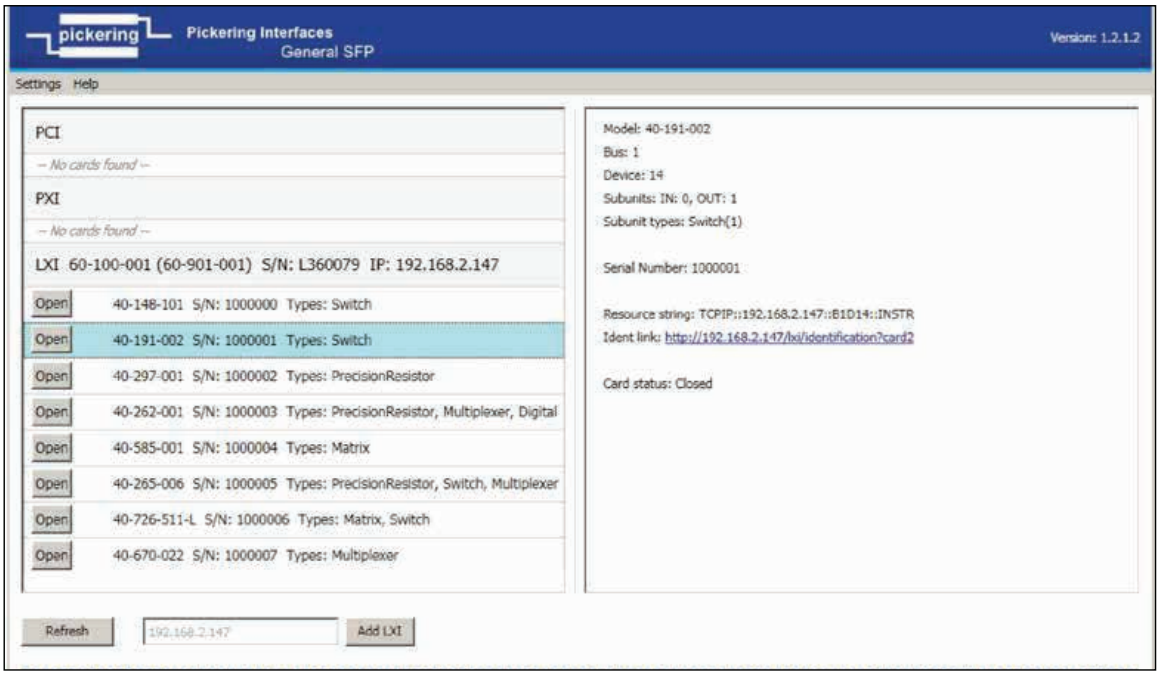



**Figure 3.3 - General Soft Front Panel Selector**

More details how to use the GSFP can be found in the manual, via menu ***Help\Manual***.

# Section 4 - Programming

## Programming options for Pickering Interfaces PCI Cards

For information on the installation and use of drivers and the programming of Pickering's products in various software environments, please refer to the Software User Manual.

This is available as a download from:
pickeringtest.com/support/software-drivers-and-downloads

Also, it can be accessed from the DVD supplied with the product:
*<DVD_DRIVE>\downloads\Documentation\Software-Manual.pdf*

## Programming the Function Generator

### Initialization

Initialization part consists of locating card and opening it (retrieving it's handle) so that other operations can be done with card. To find all cards compatible with 41-620, there is function pi620_FindInstruments. This function looks up all devices on PCI/PXI bus and returns bus/device pairs for all 41-620 cards.

When location of card is found, call pi620_init function. First parameter would be resource descriptor (PXI<bus>::<device>::INSTR), second is IDQuery which  if we are really trying to open 41-620 compliant card, third is flag that resets card on initialization. The last parameter is the handle of card that init function returns.

### Channel Configuration

Even though the card itself can be configured (for example its ID number). The main object to configure is one of card's channels. Card's driver was designed such, that before configuration block, channel is selected and after that all configuration commands will apply on currently selected channel. This is done using function pi620_SetActiveChannel first parameter is instrument handle from init, second is number of channel to configure/operate.

When configuring channel, first of all we need to set it to configurable state. In this state signal generation of channel is stopped. The shape of signal that channel generates can't be changed "on fly". Frequency of signal can be changed but this change has its restrictions. To stop channel from generating and put it into configuration mode, use function pi620_ OutputOff. When finished with configuring card, use function pi620_OutputOn. This will start generation of signal again. Note that functions pi620_GenerateSignal and pi620_GenerateSweep will start generation upon exiting them. Even though pi620_OutputOn won't hurt system, but it doesn't have to be used. Also pi620_GenerateSignalEx has a parameter generate which starts generation of signal upon leaving function if set to VI_TRUE.

Configuration of channel itself can be broken into single steps, each configuring different aspect of waveform generated: DC offset, modulation, shape of signal, frequency and also triggering.

### DC Offset Setting

Setting DC offset is very easy. pi620_SetOutputOffsetVoltage sets output offset voltage to wanted value (between -5 and 5 volts). In case, DC offset is not needed, use PI620_DO_DISCONNECT constant (or simply VI_FALSE) as third parameter. This will completely ignore DC offset mechanism on card.

### Background on Waveforms

41-620 cards have capability to make AM modulated signal. Base signal is generated by card, while modulation signal is taken from AM input on front panel. As of software only thing needed is to turn on AM modulation capability. This is done by pi620_SetAMMode function. Second parameter is 0 if AM modulation is not wanted, any other value will enable AM modulation mechanism.

Waveform itself can be described by several parameters: shape, amplitude and frequency. There are very simple ways to change these and also more complicated ways which give better control on these aspects.

First of all we need to understand how is signal generated from card. Each channel has its own dedicated waveform memory of size 256 KB. When card is set into generate mode, pointer goes through this memory and  DAC transforms number in memory place into output voltage. Depending on values in memory we can create almost any shape we want.

If we want to alter amplitude of signal, we can make numbers in memory lower (or nearer to center 50 value). The easier way is to use attenuation system. Attenuation can be set anywhere between 0 and 40 dB. 0 dB attenuation means the amplitude of the signal will be maximum possible – 10 Vpp.

Any other attenuation uses this formula to count output voltage: U = 10(-att/20) * 10

**Important:** before trying to write data into memory of channel, be sure to disable channel's triggering. The same pointer that is used when generating signal from memory is used when writing memory. If triggering is enabled, it has influence over this pointer and it won't let it move through memory. Please refer to article about triggering to see how to disable triggers.

Shape and frequency can be set very simply using function pi620_GenerateSignal. This function will automatically generate and write waveforms into channel's memory. It will also set frequency. Constants for signal shape are these: PI620_SS_SINE, PI620_SS_TRIANGLE and PI620_SS_SQUARE. Symmetry parameter changes shape of triangle and square waveforms. For triangle, it will change position of triangle top (0 – beginning, 50 – middle, 100 – end). For square it defines duty cycle. Note: when leaving pi620_GenerateSignal function, channel will be automatically switched into generate mode. There is Ex version of this function, which prevents this. It is described in triggering section.

### Sweep Generator

Very similar is function pi620_GenerateSweep. The signal definition is the same, but frequency is altered during generation. You select start and end frequency as well as frequency change and time interval between changes. When function executes, it generates signal and changes its frequency in time as defined. There are 2 types of sweep. Triangle (mode 0) starts at Start Frequency and sweeps to End Frequency with Frequency step size with Frequency step time duration at each step. When End Frequency is reached it will sweep back to Start Frequency. This will repeat till another mode is selected. Single (mode 1) sweeps Start Frequency to End Frequency with FSK (Frequency Shift Keying) pin high or from End Frequency to Start Frequency with FSK pin low. FSK pin can be changed using function pi620_SetFSKPin.

If we only want to fill channels memory with data, there are 2 functions available. pi620_SetSignal is similar to pi620_ GenerateSignal. It also lets change signal amplitude in % ("lower" values are written to memory) and start phase of signal (in degrees).

### Arbitrary Waveforms

41-620 function generator is also able to load any other waveform. Waveform is in form of array of doubles. There are a few limitations to this array. Its size has to be minimum 8 and maximum 256K. Size has to be power of 2 (e. g. 8, 16, 32, 64, 1024, 16384 etc.). The number in array must be between 0 and 100. Use function pi620_LoadArbitraryWaveform to load this array into channel's memory. pi620_LoadArbitraryWaveformEx can also set repetition rate of the waveform (e. g. number of periods of waveform generated per second). To read data from file easily (the format is every number on a single line), you can use pi620_ReadWaveformFromFile function. Note that waveform parameter is allocated and destroyed by caller. Function returns waveform length parameter, which can be fed directly into pi620_LoadArbitraryWaveform.

### Frequency Setting

Frequency can be set manually as well. In this card we don't set frequency of signal, but speed at which pointer moves from memory place to memory place. There are 3 factors in here: main base frequency, counter step and fine tuning of speed of pointer movement. pi620_SetClockSource is function that sets information about base frequency. Here we can set if internal counter is going to be used or frequency of signal in ext clock on front panel will be used. Frequency of internal clock is PI620_PXICLOCK (10 MHz). Clock multiplier is number between 4 and 20. pi620_SetCounterStep is function to set how big will be gap between each pointer's steps. There is 256KB memory in card. Card is generated such, that pointer is incremented by counter step value each moment and value in memory is transformed to output voltage. If we fill only part of memory, counter step needs to change. Counter step equals to Max memory (256KB) / size of waveform. Note: as size of waveform has to be power of 2 number, counter step will also be power of 2 number.

### Clock Mode

The last function to manipulate frequency of signal is pi620_SetClockMode. The main point of importance is start frequency parameter here. This parameter comes from equation: frequency counter = waveform length*desired frequency (in kHz). There is limit to this frequency though. Frequency given to SetClockMode mustn't be higher than half of base clocking frequency multiplied by SetClockSource multiplier.

### Triggering

To setup triggering system for channel, there is function pi620_SetTriggerMode. On PXI card, there are 10 hardware trigger lines. On PCI there is only front panel trigger line. When setting trigger, select source – hardware trigger line and mode. There are 6 modes + one mode that ignores trigger. High/low level trigger let's channel generate when trigger line is in high/low state. Once trigger line changes state, channel stops generating signal again. Negative/positive edge triggers on right impulse of trigger line. It can also be configured as single shot (memory will be generated only once). Trigger mode 6 means continuous trigger or simply said no trigger at all, signal is generated regardless trigger line. Channel can be triggered from more sources at the same time. In that case every source must meet triggering condition for channel to generate.

Software trigger can be configured using pi620_SetTriggerMode function along with writing into certain register of card, but there are a few functions, that do this more comfortable and understandable way. Software trigger lines are mapped over first 2 hardware lines. This means for PXI its external and PXI0 line, in PCI it's external line and line 1 which is software only. pi620_PrepareChannelForSoftTrigger is function, that tells driver source is going to be used as software trigger instead of hardware. Using this function on source will disable hardware line and only software commands will set this trigger line conditions. pi620_LaunchSoftTrigger is function that will set trigger source in trigger state – driver will consider this trigger source as if it meet trigger conditions. pi620_ClearSoftTrigger is opposite to pi620_PrepareChannelForSoftTrigger. It enables hardware trigger on source again. To manually fiddle with software trigger settings, there are functions pi620_Set/GetSoftTriggerStatus. Enable parameter means disable hardware line and enable software trigger on source. Status parameter indicates state of trigger line. If it's true, trigger line is considered in HIGH state, false means LOW state.

Usually it's good to setup trigger by the end of configuration chain. Trigger won't react anyhow if channel is in configuration state (e. g. it's not generating anything). After setting channel to generate mode, trigger will become armed and wait for trigger condition to happen and after that channel starts generating signal.

It's vital to disable all triggers when writing data into channel's memory. This is done by setting all used trigger sources into continuous mode. This mode is implicit, so it doesn't have to be changed for unused trigger lines.

In case you want to use GenerateSignal function, you have to use Ex version of it. First of all set channel into configuration state (OutputOff), after that disable triggers. Call pi620_GenerateSignalEx but set generate parameter to VI_FALSE and thus don't start signal generation when leaving function. Setup triggers as you like and set channel into generate mode (OutputOn). After this channel will be ready to generate and wait for triggers to meet their conditions.

### Closing Card

The last command which should be at the end of code that operates card releases handle of card.

### Programming Examples

Examples are available in the instruments installation folder (CVI / LabVIEW)

For easy reading the below examples are displayed with LabVIEW VI icons.

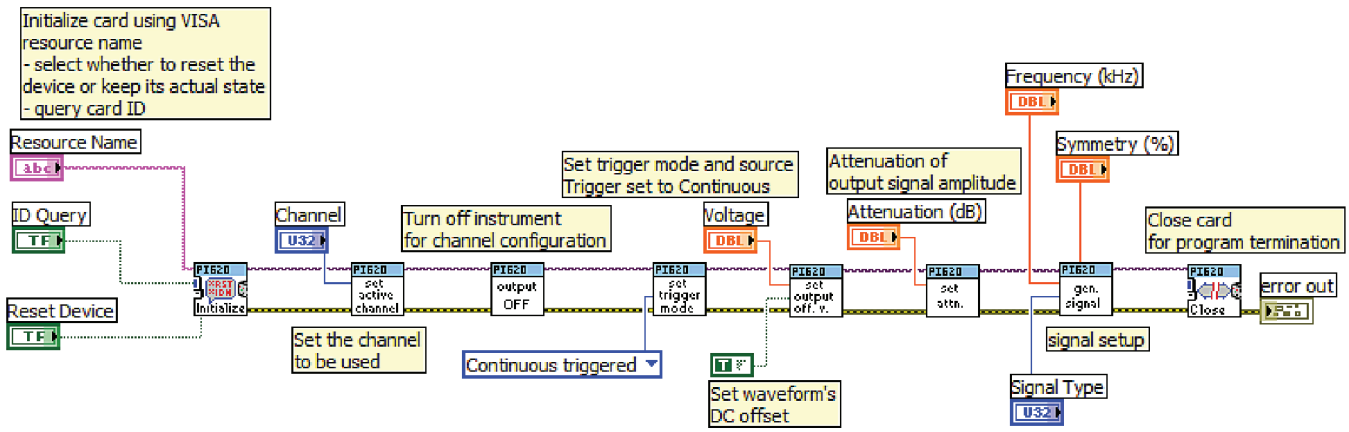



**Figure 6.1 - Programming Example - Simple Waveform Generation**

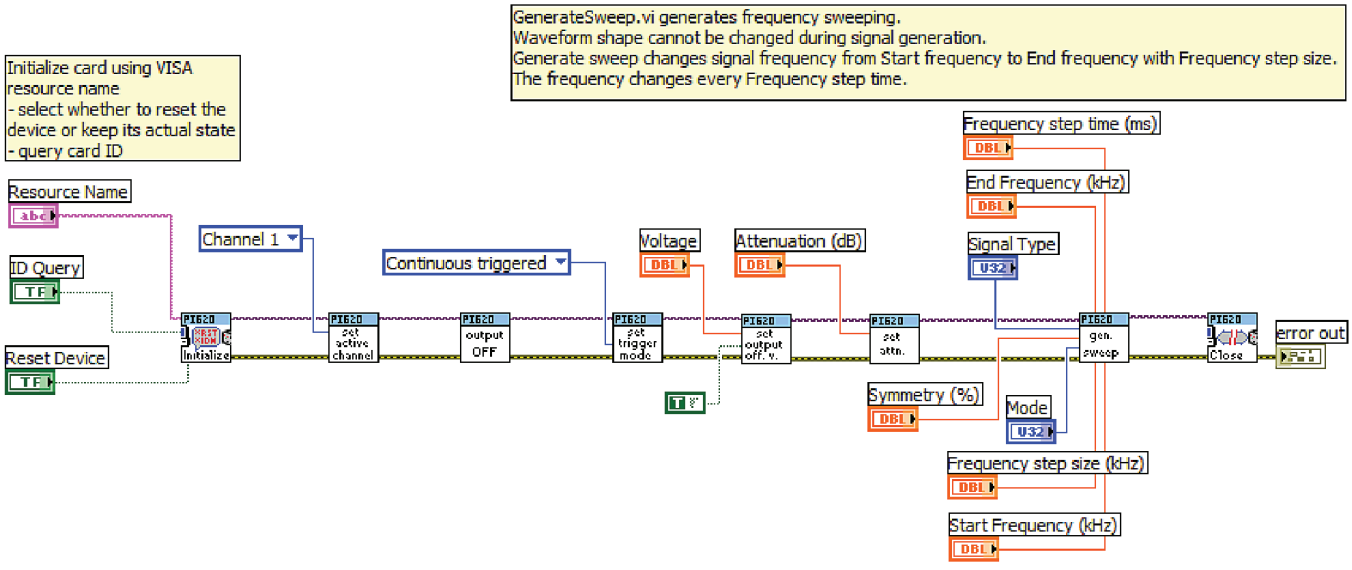



**Figure 6.2 - Programming Example - Sweep Waveform Generation**

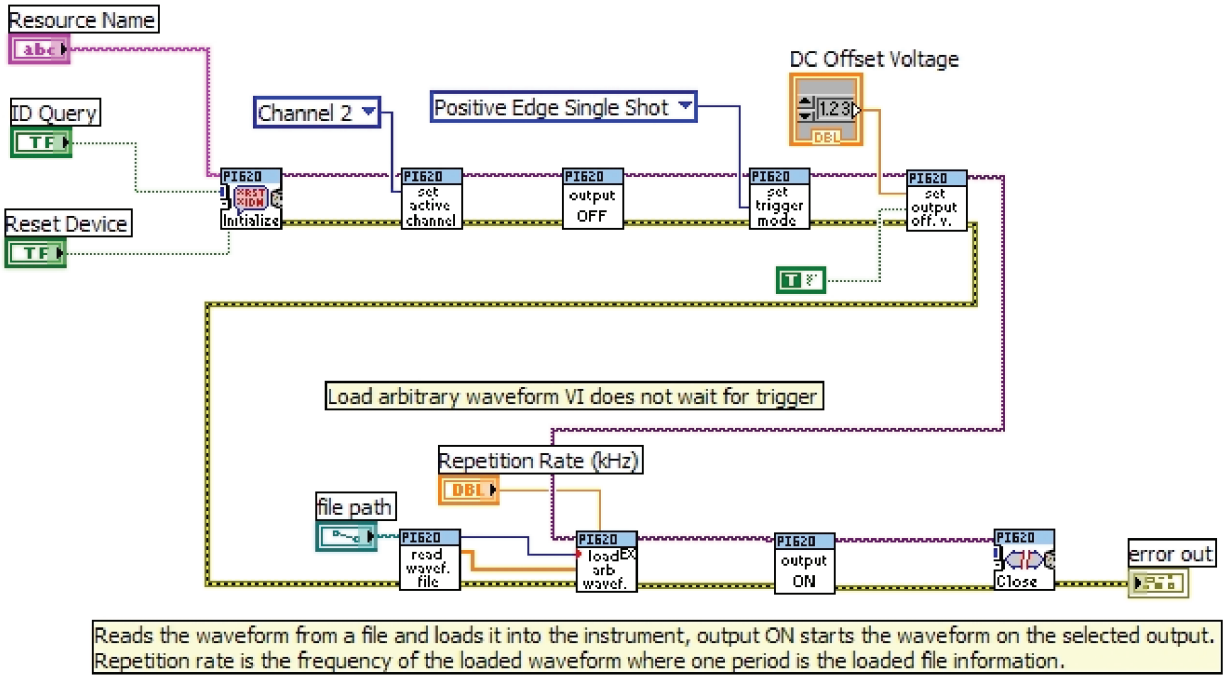
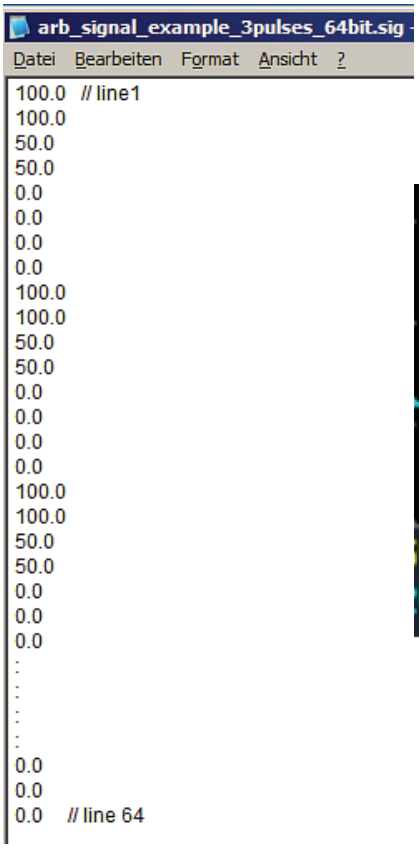



**Figure 6.3 - Programming Example - Arbitrary Waveform Generation**

Wavefrom file numbers define the percentage of the output signal voltage. Number of bits has always to be a power of 2, means 2, 4, 8, 16, 32, 64...

For example, the shown waveform file with 64 bits generates a signal with 3 pulses in a period of 100 kHz (Repetition Rate set to 100 kHz).
Each pulse changes its amplitude from 100% to 50% to 0%.
Each bit is 1/100 kHz/64 = 156 ns wide, one pulse is 4 bit wide = 625 ns



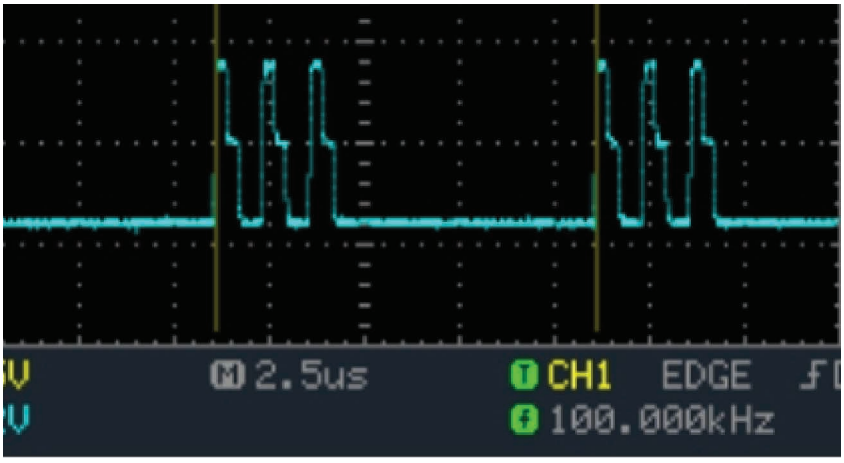




**Figure 6.4 - Waveform Generation Example**

### 41-620 3 Channel Function Generator Function Tree

| Class Name | Function Name |
|---|---|
| Initialize | pi620_init |
| Configuration | |
| Signal Generation | |
| Manual | |
| Load Arbitrary Waveform | pi620_LoadArbitraryWaveform |
| Load Arbitrary Waveform Ex | pi620_LoadArbitraryWaveformEx |
| Read Waveform From File | pi620_ReadWaveformFromFile |
| Set Clock Mode | pi620_SetClockMode |
| Set Clock Source | pi620_SetClockSource |
| Set Counter Step | pi620_SetCounterStep |
| Set Instrument Mode | pi620_SetInstrumentMode |

| | |
|---|---|
| Set Signal | pi620_SetSignal |
| Generate Signal | pi620_GenerateSignal |
| Generate Signal Ex | pi620_GenerateSignalEx |
| Generate Sweep | pi620_GenerateSweep |
| Output Off | pi620_OutputOff |
| Output On | pi620_OutputOn |
| Clear Soft Trigger | pi620_ClearSoftTrigger |
| Get Soft Trigger Status | pi620_GetSoftTriggerStatus |
| Launch Soft Trigger | pi620_LaunchSoftTrigger |
| Prepare Channel For SoftTrigger | pi620_PrepareChannelForSoftTrigger |
| Set Active Channel | pi620_SetActiveChannel |
| Set AM Mode | pi620_SetAMMode |
| Set Attenuation | pi620_SetAttenuation |
| Set FSK Pin | pi620_SetFSKPin |
| Set Soft Trigger Status | pi620_SetSoftTriggerStatus |
| Set FSK Source | pi620_SetFSKSource |
| Set Output Offset Voltage | pi620_SetOutputOffsetVoltage |
| Set Trigger Mode | pi620_SetTriggerMode |
| Calibration | |
| Get Calibration Date | pi620_GetCalibrationDate |
| Get Offset Calibration Code | pi620_GetOffsetCalCode |
| Get Output Offset Cal. Voltages | pi620_GetOutputOffsetCalVoltages |
| Get Range Limit Voltages | pi620_GetRangeLimitVoltages |
| Set Offset Calibration Code | pi620_SetOffsetCalCode |
| Set Output Offset Cal. Voltages | pi620_SetOutputOffsetCalVoltages |
| Set Range Limit Voltages | pi620_SetRangeLimitVoltages |
| Store Calibration Data | pi620_StoreCalibrationData |
| Write Calibration Date | pi620_WriteCalibrationDate |

- Utility
  - Debug
    - Memory Test — pi620_MemoryTest
    - Read Eeprom — pi620_ReadEeprom
    - Read Instrument Memory — pi620_ReadInstrumentMemory
    - Read Instrument Memory Array — pi620_ReadInstrumentMemoryArray
    - Read Register — pi620_ReadRegister
    - Reset Address Counter — pi620_ResetAddressCounter
    - Set Lock Mode — pi620_SetLockMode
    - Set Main Dac Code — pi620_SetMainDacCode
    - Set Output Offset Dac Code — pi620_SetOutputOffsetDacCode
    - Set Range Dac Code — pi620_SetRangeDacCode
    - Write Eeprom — pi620_WriteEeprom
    - Write Instrument Memory — pi620_WriteInstrumentMemory
    - Write Instrument Memory Array — pi620_WriteInstrumentMemoryArray
    - Write Register — pi620_WriteRegister
  - Error Message — pi620_error_message
  - Error Query — pi620_error_query
  - Find Instruments — pi620_FindInstruments
  - Get Card ID — pi620_GetCardId
  - Reset — pi620_reset
  - Revision Query — pi620_revision_query
  - Self Test — pi620_self_test
  - Set Output Voltage — pi620_SetOutputVoltage
  - Write Card ID — pi620_WriteCardId
- Close — pi620_close

The following C functions are in alphabetical order. All functions return following status codes:

**Return Value**

Returns the status code of this operation. The status code either indicates success or describes an error or warning condition. You examine the status code from each call to an instrument driver function to determine if an error occurred. To obtain a text description of the status code, call the pi620_error_message function.

| Value | Meaning |
|---|---|
| 0 | Success |
| Positive Values | Warnings |
| Negative Values | Errors |

| Status | Description |
|---|---|
| ERRORS: | BFFC0901 Memory test error |
| WARNINGS: | 3FFC0901 Eeprom verify warning |
| | 3FFC0902 Calibration parameter out of range |

This instrument driver also returns errors and warnings defined by other sources. The following table defines the ranges of additional status codes that this driver can return. The table lists the different include files that contain the defined constants for the particular status codes:

| Numeric Range (in Hex) | Status Code Types | |
|---|---|---|
| 3FFF0000 to 3FFFFFFF | VISA | Warnings |
| BFFF0000 to BFFFFFFF | VISA | Errors |

## pi620_ClearSoftTrigger

ViStatus pi620_ClearSoftTrigger (ViSession instrumentHandle, ViUInt32 triggerSource);

**Purpose**

Releases specified trigger line. Removes masking bit, that masks triggerline hardware input.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

triggerSource

Variable Type ViUInt32

Select the trigger source:

1 - Launch software trigger on external trigger line
2 - Launch software trigger on PXI0 trigger line
3 - Launch software trigger on both trigger lines

Note: for 51-620 card is source parameter ignored

## pi620_close

ViStatus pi620_close (ViSession instrumentHandle);

**Purpose**

Closes the instrument I/O session and deallocates all memory resources used by the driver.
Once this function is called the session handle that was returned from the pi620_init function is no longer valid.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

## pi620_error_message

ViStatus pi620_error_message (ViSession instrumentHandle, ViStatus statusCode, ViChar _VI_FAR errorMessage[]);

**Purpose**

Converts a status code returned by any instrument driver function into a user-readable string.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

statusCode

Variable Type ViStatus

Error code returned by one of the instrument driver functions.

errorMessage

Variable Type ViChar[]

A text description of the error code.

### pi620_error_query

ViStatus pi620_error_query (ViSession instrumentHandle, ViPInt32 errorCode, ViChar _VI_FAR errorMessage[]);

**Purpose**

Error Query is not supported by this instrument.
This function only exists for compliance with the VXI Plug and Play specifications.

It returns a defined value VI_WARN_NSUP_ERROR_QUERY.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

errorCode

Variable Type ViInt32 (passed by reference)

Error code returned from the instrument.

errorMessage

Variable Type ViChar[]

Text description of the error code.

### pi620_FindInstruments

ViStatus pi620_FindInstruments (ViUInt32 _VI_FAR buslist[], ViUInt32 _VI_FAR devicelist[], ViPUInt32 count);

**Purpose**

Returns the number of available instruments and a list with the bus and device number for each instrument.
Buslist and devicelist should be arrays of at least 100 places.

**Parameter List**

buslist

Variable Type ViUInt32[]

Buslist for the instrument(s).

devicelist

Variable Type ViUInt32[]

Device list of the instrument(s).

count

Variable Type ViUInt32 (passed by reference)

Number of available instruments.

### pi620_GenerateSignal

ViStatus pi620_GenerateSignal (ViSession instrumentHandle, ViReal64 frequency_kHz,
ViUInt32 signalType, ViReal64 symmetry_);

**Purpose**

Generate a standard signal with the desired Frequency on the output of the active channel.
This function will automatically fill the memory and adjust the update clock and counter step to get the desired Signal Frequency.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

frequency_kHz

Variable Type ViReal64

Frequency of the signal.

signalType

Variable Type ViUInt32

Shape of the signal:

0 = Sine

1 = Triangle
2 = Square

symmetry_

Variable Type ViReal64

Symmetry of the signal type if Triangle or Square. A symmetry of 50 percent will result in a square with an duty cycle of 50 or a triangle with the top in the middle (equal up-going and down-going ramp).

### pi620_GenerateSignalEx

ViStatus pi620_GenerateSignalEx (ViSession instrumentHandle, ViReal64 frequency_kHz, ViUInt32 signalType, ViReal64 symmetry_, ViReal64 startPhase, ViBoolean generate);

**Purpose**

Generate a standard signal with the desired Frequency on the output of the active channel. This function will automatically fill the memory and adjust the update clock and counter step to get the desired Signal Frequency.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

frequency_kHz

Variable Type ViReal64

Frequency of the signal.

signalType

Variable Type ViUInt32

Shape of the signal:

0 = Sine
1 = Triangle
2 = Square

symmetry_

Variable Type ViReal64

Symmetry of the signal type if Triangle or Square. A symmetry of 50 percent will result in a square with an duty cycle of 50 or a triangle with the top in the middle (equal up-going and down-going ramp).

startPhase

Variable Type ViReal64

Start phase of the signal in memory.

generate

Variable Type ViBoolean

Detects if signal should be generated upon leaving the function (true) or instrument is left in configuration mode (false)

### pi620_GenerateSweep

ViStatus pi620_GenerateSweep (ViSession instrumentHandle, ViUInt32 signalType, ViReal64 symmetry_, ViUInt32 mode, ViReal64 startFrequency_kHz, ViReal64 endFrequency_kHz, ViReal64 frequencyStepSize_kHz, ViReal64 frequencyStepTime_ms);

**Purpose**

Generate a signal with a sweep on the output of the active channel of the instrument. This function will automaticly fill the memory and adjust the update clock and counter step to get the desired Frequencies.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.
Default Value: None

signalType

Variable Type ViUInt32

Shape of the signal:
0 = Sine
1 = Triangle
2 = Square

symmetry_

Variable Type ViReal64

Symmetry of the signal type if Triangle or Square. A symmetry of 50 percent will result in a square with an duty cycle of 50 or a triangle with the top in the middle (equal up-going and down-going ramp).

mode

Variable Type ViUInt32

Sweep mode:
0 = Signal starts at Start Frequency and sweeps to End Frequency with Frequency step size with Frequency step time duration at each step.
When End Frequency is reached it will sweep back to Start Frequency. This will repeat till another mode is selected.
1 = Signal sweeps Start Frequency to End Frequency with FSK pin high or from End Frequency to Start Frequency with FSK pin low.

startFrequency_kHz

Variable Type ViReal64

Lower frequency of the signal.

endFrequency_kHz

Variable Type ViReal64

Upper frequency of the signal.

frequencyStepSize_kHz

Variable Type ViReal64

Increment/decrement value of signal frequency.

frequencyStepTime_ms

Variable Type ViReal64

The time spent at each intermediate frequency.

### pi620_GetCalibrationDate

ViStatus pi620_GetCalibrationDate (ViSession instrumentHandle, ViPUInt32 year, ViPUInt32 month, ViPUInt32 day);

**Purpose**

Read the last calibration date. The calibration date is stored in on-board eeprom.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

year

Variable Type ViUInt32 (passed by reference)

Year of last calibration.

month

Variable Type ViUInt32 (passed by reference)

Month of last calibration.

day

Variable Type ViUInt32 (passed by reference)

Day of last calibration.

### pi620_GetCardId

ViStatus pi620_GetCardId (ViSession instrumentHandle, ViPUInt32 id);

**Purpose**

Returns the card id. The card id is stored in on board eeprom.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

id

Variable Type ViUInt32 (passed by reference)

The card id.

### pi620_GetOffsetCalCode

ViStatus pi620_GetOffsetCalCode (ViSession instrumentHandle, ViPUInt32 offsetCode, ViUInt32 offset);

**Purpose**

Returns the current offset calibration code. This instrument does have 2 offset calibration codes, called the "static" offset and the "dynamical" offset. The static offset code can vary for each of the 4 ranges (0, 10, 20 and 30 dB attenuation). The Offset calibration codes are calibrated with the function pi620_SetOffsetCalCode.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.

The handle identifies a particular instrument session.

Default Value: None

offsetCode

Variable Type ViUInt32 (passed by reference)

Offset code.

offset

Variable Type ViUInt32

Return the static or dynamical offset calibration code:

0 = Dynamical offset code
1 = Static offset code

### pi620_GetOutputOffsetCalVoltages

ViStatus pi620_GetOutputOffsetCalVoltages (ViSession instrumentHandle, ViPReal64 maxVoltage, ViPReal64 minVoltage);

**Purpose**

Returns the output offset calibration voltages, calibrated with the function pi620_SetOutputOffsetCalVoltages.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

maxVoltage

Variable Type ViReal64 (passed by reference)

Maximum output offet calibration voltage.

minVoltage

Variable Type ViReal64 (passed by reference)

Minimum output offset calibration voltage.

### pi620_GetRangeLimitVoltages

ViStatus pi620_GetRangeLimitVoltages (ViSession instrumentHandle, ViPReal64 maxVoltage, ViPReal64 minVoltage);

**Purpose**

Returns the range (calibration) limit voltages, calibrated with the function pi620_SetRangeLimitVoltages. There are four ranges with each a set of range limit voltages:

Attenuation:

0 - 10 dB
10 - 20 dB
20 - 30 dB
30 - 40 dB

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

maxVoltage

Variable Type ViReal64 (passed by reference)

Maximum range limit voltage.

minVoltage

Variable Type ViReal64 (passed by reference)

Minimum range limit voltage.

### pi620_GetSoftTriggerStatus

ViStatus pi620_GetSoftTriggerStatus (ViSession instrumentHandle, ViUInt32 triggerSource,
ViPBoolean STEnabled,ViPBoolean STStatus);

**Purpose**

Returns mask and software trigger status for any trigger line.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

triggerSource

Variable Type ViUInt32

Select the trigger source:

1 - Launch software trigger on external trigger line
2 - Launch software trigger on PXI0 trigger line
3 - Launch software trigger on both trigger lines

**Note:** only “ST Status” parameter is relevant in this function when operating 51-620 card

STEnabled

Variable Type ViBoolean (passed by reference)

true if hardware input of the trigger line is being masked otherwise false

Note: only "ST Status" parameter is relevant in this function when operating 51-620 card

STStatus

Variable Type ViBoolean (passed by reference)

true if software trigger level is high, if level is low, this is false

### pi620_init

ViStatus pi620_init (ViRsrc resourceName, ViBoolean IDQuery, ViBoolean resetDevice, ViPSession instrumentHandle);

**Purpose**

This function opens a session to the instrument, specified by the rsrcName parameter, and returns the handle to that session. It must be called prior to using any other functions in this driver. It will return a unique session handle each time it is called, however it is not recommended to open more than one handle at a time.

If the id_query parameter is set to VI_TRUE, this function will verify that the instrument is supported by this driver.

Reset isn't supported by this device.

**Parameter List**

resourceName

Variable Type ViRsrc

Pass the resource name of the device to initialize.

Refer to the following table below for the exact grammar to use for this parameter.
Optional fields are shown in square brackets ([]).

Syntax

---

PXI[bus]::<device number>::INSTR

If you do not specify a value for an optional field, the following values are used:

Optional Field - Value

---

bus - 0

The following table contains example valid values for this parameter.

"Valid Value" - Description

---

"PXI1::16::INSTR" - PXI bus 1, device 16

Default Value: ""

IDQuery

Variable Type ViBoolean

Specify whether you want the instrument driver to perform an ID Query.

Valid Range:
VI_TRUE (1) - Perform ID Query (Default Value)
VI_FALSE (0) - Skip ID Query

When you set this parameter to VI_TRUE, the driver verifies that the instrument you initialize is a 41-620

Circumstances can arise where it is undesirable to perform an ID Query. When you set this parameter to VI_FALSE, the function initializes the instrument without performing an ID Query.

resetDevice

Variable Type ViBoolean

initialization procedure.

Valid Range:
VI_TRUE (1) - Reset Device
VI_FALSE (0) - Don't Reset (Default Value)

instrumentHandle

Variable Type ViSession (passed by reference)

Returns a ViSession handle that you use to identify the instrument in all subsequent instrument driver function calls.

Notes:

(1) This function creates a new session each time you invoke it. This is useful if you have multiple physical instances of the same type of instrument.

(2) Avoid creating multiple concurrent sessions to the same physical instrument. Although you can create more than one session for the same resource, it is best not to do so.

### pi620_LaunchSoftTrigger

ViStatus pi620_LaunchSoftTrigger (ViSession instrumentHandle, ViUInt32 triggerSource);

**Purpose**

Executes specified software trigger making small spike on specified trigger line.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

triggerSource

Variable Type ViUInt32

Select the trigger source:

1 - Launch software trigger on external trigger line
2 - Launch software trigger on PXI0 trigger line
3 - Launch software trigger on both trigger lines

Note: for 51-620 card is source parameter ignored

### pi620_LoadArbitraryWaveform

ViStatus pi620_LoadArbitraryWaveform (ViSession instrumentHandle, ViUInt32 waveformLength,
ViReal64 _VI_FAR waveform[]);

**Purpose**

Load the instrument memory with an arbitrary waveform.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None waveformLength

Variable Type ViUInt32

Size of the waveform. Should be a power of 2. Maximum size is determined by the current value of counterstep: 256k/
counterstep.

waveform

Variable Type ViReal64[]

Arbitrary waveform with normalize values between 0 and 100% (100% = maximum dac code, 0% = minimum dac code).

### pi620_LoadArbitraryWaveformEx

ViStatus pi620_LoadArbitraryWaveformEx (ViSession instrumentHandle, ViUInt32 waveformLength,
ViReal64 _VI_FAR waveform[], ViReal64 repetitionRate);

**Purpose**

Load the instrument memory with an arbitrary waveform.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

waveformLength

Variable Type ViUInt32

Size of the waveform. Should be a power of 2. Maximum size is determined by the current value of counterstep: 256k/ counterstep.

waveform

Variable Type ViReal64[]

Arbitrary waveform with normalize values between 0 and 100% (100% = maximum dac code, 0% = minimum dac code).

repetitionRate

Variable Type ViReal64

Repetition rate of the waveform. This value in kHz defines how fast will be the whole waveform generated on output.

### pi620_MemoryTest

ViStatus pi620_MemoryTest (ViSession instrumentHandle, ViPUInt32 errorAddress, ViPUInt32 errorData, ViPUInt32 expectedData);

**Purpose**

Perform instrument memory test of active channel.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

errorAddress

Variable Type ViUInt32 (passed by reference)

Returns address at which the error occurs, if there is any error.

errorData

Variable Type ViUInt32 (passed by reference)

Returns the read data of Error Address.

expectedData

Variable Type ViUInt32 (passed by reference)

Returns the expected data for Error Address.

### pi620_OutputOff

ViStatus pi620_OutputOff (ViSession instrumentHandle);

**Purpose**

Turns the selected channel output off. Stops signal generating and disconnects output offset voltage.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

### pi620_OutputOn

ViStatus pi620_OutputOn (ViSession instrumentHandle);

**Purpose**

Turns the selected channel output on. Starts signal generating and connects output offset voltage.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

### pi620_PrepareChannelForSoftTrigger

ViStatus pi620_PrepareChannelForSoftTrigger (ViSession instrumentHandle, ViUInt32 triggerSource);

**Purpose**

Prepares channel for using software trigger. It sets trigger mode of channel in proper state (POS_EDGE, correct source), also enables software trigger on the specified trigger line

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

triggerSource

Variable Type ViUInt32

Select the trigger source:

0 - Stop triggering (trigger mode is put into continuous)
1 - Use external trigger line as software trigger
2 - Use PXI0 trigger line as software trigger

Note: for 51-620 card is source parameter ignored

### pi620_ReadEeprom

ViStatus pi620_ReadEeprom (ViSession instrumentHandle, ViUInt32 eepromAddress, ViPUInt32 data);

**Purpose**

Read from on board eeprom.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

eepromAddress

Variable Type ViUInt32

Eeprom address.

data

Variable Type ViUInt32 (passed by reference)

Read data.

### pi620_ReadInstrumentMemory

ViStatus pi620_ReadInstrumentMemory (ViSession instrumentHandle, ViPUInt32 data);

**Purpose**

Read from Instrument memory. Set address counter at defined postion with the function pi620_ ResetAddressCounter.

This function will increment the address counter with 1 step.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

data

Variable Type ViUInt32 (passed by reference)

Read data from instrument memory

### pi620_ReadInstrumentMemoryArray

ViStatus pi620_ReadInstrumentMemoryArray (ViSession instrumentHandle, ViUInt32 length, ViUInt32 _VI_FAR buf32[]);

**Purpose**

Read an array of data from instrument memory. Init address counter with function pi620_ResetAddresCounter. Maximum of available memory places are 256k/counterstep.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

length

Variable Type ViUInt32

Length of buffer. Maximum buffer size is 256k/counterstep.

buf32

Variable Type ViUInt32[]

Reference to buffer for read data elements.

### pi620_ReadRegister

ViStatus pi620_ReadRegister (ViSession instrumentHandle, ViUInt32 address, ViPUInt32 data);

**Purpose**

Reads the instrument register at the specified offset and returns the value.
Refer to the instrument user's manual for register definitions.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

address

Variable Type ViUInt32

The offset address of the register to read.

data

Variable Type ViUInt32 (passed by reference)

Returns the value read from the specified offset.

### pi620_ReadWaveformFromFile

ViStatus pi620_ReadWaveformFromFile (ViSession instrumentHandle, ViChar _VI_FAR fileName[],
int waveformFieldSize, ViReal64 _VI_FAR waveform[], int *waveformLength);

**Purpose**

Reads arbitrary waveform from file

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

fileName

Variable Type ViChar[]

string containing path to file

waveformFieldSize

Variable Type int

size of the waveform field for safety reasons

waveform

Variable Type ViReal64[]

Arbitrary waveform with normalize values between 0 and 100% (100% = maximum dac code, 0% = minimum dac code).

waveformLength

Variable Type int (passed by reference)

length of the waveformpi620_reset

ViStatus pi620_reset (ViSession instrumentHandle);

**Purpose**

A reset is not supported by this instrument.
It returns a defined value VI_WARN_NSUP_RESET.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

### pi620_ResetAddressCounter

ViStatus pi620_ResetAddressCounter (ViSession instrumentHandle);

**Purpose**

Set the memory address counter to zero.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

### pi620_revision_query

ViStatus pi620_revision_query (ViSession instrumentHandle, ViChar _VI_FAR driverRevision[],
ViChar _VI_FAR instrumentRevision[]);

**Purpose**

This function returns the revision numbers of the instrument driver.

This instrument does not contain firmware so the instr_rev parameter is not valid. It only exists for compliance with the VXI Plug and Play specification.

This function always returns the defined value VI_WARN_NSUP_REV_QUERY.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

driverRevision

Variable Type ViChar[]

Returns the instrument driver revision.

instrumentRevision

Variable Type ViChar[]

Returns the instrument firmware revision.

This instrument does not contain firmware so this parameter will always be a NULL string.

### pi620_self_test

ViStatus pi620_self_test (ViSession instrumentHandle, ViPUInt16 selfTestResult, ViChar _VI_FAR seftTestMessage[]);

**Purpose**

Self Test is not supported by this instrument.
It returns a defined value VI_WARN_NSUP_SELF_TEST.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

selfTestResult

Variable Type ViUInt16 (passed by reference)

Numeric result of the self test

0 = no error (test passed)

seftTestMessage

Variable Type ViChar[]

The status message from the self test.

### pi620_SetActiveChannel

ViStatus pi620_SetActiveChannel (ViSession instrumentHandle, ViUInt32 channel);

**Purpose**

Activate a channel. The active channel can be adjusted with the other driver functions.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

channel

Variable Type ViUInt32

Select the channel:

1 = Channel A
2 = Channel B
3 = Channel C

### spi620_SetAMMode

ViStatus pi620_SetAMMode (ViSession instrumentHandle, ViUInt32 AMMode);

**Purpose**

Enable or disable the am-mode switch. When am-mode is enabled the generator signal frequency can be modulated with the signal supplied at the AM-input.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

AMMode

Variable Type ViUInt32

Enable or disable AM-Mode:

0 = disabled
1 = enabled

### pi620_SetAttenuation

ViStatus pi620_SetAttenuation (ViSession instrumentHandle, ViReal64 attenuation_dB);

**Purpose**

Select the attenuation of the output signal. The output can be attenuated in a range from 0 till 40 dB.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

attenuation_dB

Variable Type ViReal64

Attenuator value (dB).

### pi620_SetClockMode

ViStatus pi620_SetClockMode (ViSession instrumentHandle, ViUInt32 mode, ViReal64 startFrequency_kHz, ViReal64 endFrequency_kHz, ViReal64 frequencyStepSize_kHz, ViReal64 frequencyStepTime_ms);

**Purpose**

Configure the update rate clock. Be sure the clock source (and clock multiplier) is configured by pi620_SetClockSource first.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function. The handle identifies a particular instrument session.

Default Value: None

mode

Variable Type ViUInt32

Select (update) clock mode:

0 = Continuous: Update frequency has constant frequency of Start Frequency.

1 = Triangle sweep: Update frequency sweeps from Start Frequency to End Frequency (highest frequency) and back.

2 = Single sweep: Update frequency sweeps from Start Frequency to End Frequency (highest frequency) with FSK pin high and from End Frequency to Start Frequency with FSK pin low.

3 = FSK: Update frequency is Start Frequency with FSK pin low and End Frequency with FSK pin high.

startFrequency_kHz

Variable Type ViReal64

Lower update frequency , see description of Mode.

endFrequency_kHz

Variable Type ViReal64

Upper update frequency, see description of Mode.

frequencyStepSize_kHz

Variable Type ViReal64

Increment/decrement value of update frequency in sweep mode.

frequencyStepTime_ms

Variable Type ViReal64

The time spent at each intermediate frequency.

### pi620_SetClockSource

ViStatus pi620_SetClockSource (ViSession instrumentHandle, ViUInt32 clockSource,
ViReal64 externalClockFrequency_MHz, ViUInt32 clockMultiplier);

**Purpose**

Select the clock source and configure the external clock frequency. The clock source is fed to all 3 channels. Each channel can multiply the clock frequency with the clock multiplier value. The internal clock frequency (clock frequency * Clock Muliplier) may not exceed 300 MHz. An external clock frequency below 5 MHz is not recommended.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

clockSource

Variable Type ViUInt32

Select the clock source:

0 = 10 MHz PXI Clock
1 = External clock source. Fill in the external clock source value.

externalClockFrequency_MHz

Variable Type ViReal64

The signal frequency of the external clock source (if used).

clockMultiplier

Variable Type ViUInt32

Each channel can multiply the (common) clock source with the Clock Multiplier.
The resulting internal clock is use to generate the update clock.

### pi620_SetCounterStep

ViStatus pi620_SetCounterStep (ViSession instrumentHandle, ViUInt32 counterStep);

**Purpose**

Set the step size for the counter. This will change the number of used samples of the module memory and so influence the signal frequency.
Number of effective samples: 256k/counterstep.

**Parameter List**

instrumentHandles

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

counterStep

Variable Type ViUInt32

The value for the Counter Step. Counter Step should be a power of 2.

### pi620_SetFSKPin

ViStatus pi620_SetFSKPin (ViSession instrumentHandle, ViUInt32 FSKPinState);

**Purpose**

Set FSK pin of internal DDS generator high (= 1) or low (= 0). The FSK pin state is used in single sweep and FSK mode. This function will only alter the FSK pin if the FSK source is set to Software (see pi620_SetFSKSource)

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

FSKPinState

Variable Type ViUInt32

FSK pin state:

0 = low
1 = high

### pi620_SetFSKSource

ViStatus pi620_SetFSKSource (ViSession instrumentHandle, ViUInt32 FSKPinState);

**Purpose**

Select the source for the FSK pin. The FSK pin state is used in single sweep and FSK mode.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

FSKPinState

Variable Type ViUInt32

FSK pin source:

0 = Software
1 = Front
2 = PXI 0
3 = PXI 1
4 = PXI 2
5 = PXI 3
6 = PXI 4
7 = PXI 5
8 = PXI 6
9 = PXI 7
10 = PXI Star

### pi620_SetInstrumentMode

ViStatus pi620_SetInstrumentMode (ViSession instrumentHandle, ViUInt32 mode);

**Purpose**

The active channel of the module should be in Signal Genereration mode to generate a signal on the output. The module memory of the active channel and main dac cannot be accessed if the channel of the module is in Signal Generation Mode. This function calls pi620_SetLockMode(..) and programs the Main dac half scale (8000 hex) if Mode is set at Signal Configuration Mode.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

mode

Variable Type ViUInt32

Set module in Signal Configuration or Signal Generation Mode:

1 = Signal Generation mode
0 = Signal Configuration mode

### pi620_SetLockMode

ViUInt32 pi620_SetLockMode (ViSession instrumentHandle, ViUInt32 mode);

**Purpose**

The module should be locked to generate a signal on the output.
The module memory and main dac cannot be accessed if the module is locked.
This function is called by pi620_SetInstrumentMode

. Parameter List

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

mode

Variable Type ViUInt32

Lock or unlock the module:

0 = Unlock
1 = Lock

### pi620_SetMainDacCode

ViStatus pi620_SetMainDacCode (ViSession instrumentHandle, ViUInt32 code);

**Purpose**

Program the main dac with a specified code. Dac code is left justified.
If instrument contains a 12 bit main dac, the lower 4 bits are unused.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

code

Variable Type ViUInt32

Code for main dac (left justified).

### pi620_SetOffsetCalCode

ViStatus pi620_SetOffsetCalCode (ViSession instrumentHandle, ViUInt32 offsetCode, ViUInt32 offset);

**Purpose**

Set the offset calibration code. There are two different offset calibration values: the dynamical offset and the static offset. The dynamical offset is independent of the range. The static offset should be calibrated for the ranges 0, 10, 20 and 30 dB.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

offsetCode

Variable Type ViUInt32

The offset code.

offset

Variable Type ViUInt32

Set the static or dynamical offset code:

0 = dynamical offset code.
1 = static offset code.

### pi620_SetOutputOffsetCalVoltages

ViStatus pi620_SetOutputOffsetCalVoltages (ViSession instrumentHandle,
ViReal64 maxVoltage,
ViReal64 minVoltage);

**Purpose**

Set calibration voltages for output offset dac.
These voltages are necessary for a calibrated output offset voltage.
Before calibrating the output offset voltages, the static and dynamical offset should be calibrated first.
Procedure to determine limit voltages:
- Select Channel to calibrate ( SetActiveChannel(..) )
- Be sure channel is not running (SetInstrumentMode (0) )
- Program Main dac at half scale (SetMainDacCode(0x8000) )
- Set attenuation at 30 dB (SetAttenuation(30) )
- Program Output offset dac at maximum and connect offset voltage (SetOutputOffsetDacCode(vi, 0x3FF,1) )
- Measure voltage at output with accurate voltage meter
- Program output offset dac at minimum ( SetOutputOffsetDacCode(vi, 0x0,1) )
- Measure voltage at output with accurate voltage meter
- Call this routine with the measured voltages as arguments
- Repeat steps for every channel
- Call StoreCalibration for storing data in eeprom

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

maxVoltage

Variable Type ViReal64

Maximum measured ouput offset voltage.

minVoltage

Variable Type ViReal64

Minimum measured ouput offset voltage.

### pi620_SetOutputOffsetDacCode

ViStatus pi620_SetOutputOffsetDacCode (ViSession instrumentHandle, ViUInt32 code, ViUInt32 connect);

**Purpose**

Program the ouput offset dac with a specified (10-bit) code. Use this function for calibration or debug purposes only.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

code

Variable Type ViUInt32

Code for output offset dac.

connect

Variable Type ViUInt32

Connect or disconnect the output offset dac:
0 = disconnect
1 = connect

### pi620_SetOutputOffsetVoltage

ViStatus pi620_SetOutputOffsetVoltage (ViSession instrumentHandle, ViReal64 voltage, ViUInt32 connect);

**Purpose**

Adjust the output offset voltage and connect or disconnect the output offset voltage. The output offset voltage is a DC

voltage added to the output signal.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

voltage

Variable Type ViReal64

The output offset voltage.

connect

Variable Type ViUInt32

Connect or disconnect the output offset voltage.

### pi620_SetOutputVoltage

ViStatus pi620_SetOutputVoltage (ViSession instrumentHandle, ViReal64 voltage, ViUInt32 method);

**Purpose**

Program an (dc) output offset voltage with the attenator and main dac.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function. The handle identifies a particular instrument session.

Default Value: None

voltage

Variable Type ViReal64

The output offset voltage.

method

Variable Type ViUInt32

Program the output voltage by adjusting the main dac code (with attenuator at 0 dB) or by adjusting the Attenuator (with main dac code at maximum for positive voltage or at minimum for negative voltage).

### pi620_SetRangeDacCode

ViStatus pi620_SetRangeDacCode (ViSession instrumentHandle, ViUInt32 code);

**Purpose**

Program the range dac with a specified (10-bit) code. Use this function for calibration or debug purposes only.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function. The handle identifies a particular instrument session.

Default Value: None

code

Variable Type ViUInt32

Code for range dac.

### pi620_SetRangeLimitVoltages

ViStatus pi620_SetRangeLimitVoltages (ViSession instrumentHandle, ViReal64 maxVoltage, ViReal64 minVoltage);

**Purpose**

Set limit voltages of range dac. These voltages are necessary for a calibrated range voltage.
Be sure offset is calibrated first.
Procedure to determine limit voltages:
- Select Channel to calibrate ( SetActiveChannel(..) )
- Be sure channel is not running (SetInstrumentMode (0) )
- Disconnect output offset (SetOutputOffsetVoltage (0.0, 0) )
- Set Dac code at maximum (SetDacCode(0xFFFF) ) - Set attenuator at 0 dB (SetAttenuator (0.0) )
- Program range dac at maximum ( SetRangeDacCode(ci, 0x3FF) )
- Measure output voltage with accurate voltage meter
- Program range dac at minimum ( SetRangeDacCode(ci, 0x0) )
- Measure output voltage with accurate voltage meter
- Call this routine with measured voltages
- Repeat this at 10 dB, 20 dB and 30 dB attenuation
- Repeat steps with next channel
- Call StoreCalibration for storing data in eeprom

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

maxVoltage

Variable Type ViReal64

Maximum measured voltage.

minVoltage

Variable Type ViReal64

Minimum measured voltage.

## pi620_SetSignal

ViStatus pi620_SetSignal (ViSession instrumentHandle, ViUInt32 signalType, ViReal64 normalizedAmplitude_,
ViReal64 startPhase, ViReal64 symmetry_);

### Purpose

Fill instrument memory with standard signal.
The number of effective memory samples are influency by the counter step value: 256k/counterstep.

### Parameter List

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

signalType

Variable Type ViUInt32

Select type of signal:

0 = Sine
1 = Triangle
2 = Square

normalizedAmplitude_

Variable Type ViReal64

The normalized amplitude influences the number of used main dac codes. 100% uses all dac codes. The amplitude can also be changed by adjusting Attenuator during signal generation (pi620_SetAttenuation).

startPhase

Variable Type ViReal64

Start phase of the signal in memory.

symmetry_

Variable Type ViReal64

symmetry of the signal if type is Triangle or Square. A symmetry of 50% will result in a square with an duty cycle of 50 or a triangle with the top in the middle (equal up-going and down-going ramp).

### pi620_SetSoftTriggerStatus

ViStatus pi620_SetSoftTriggerStatus (ViSession instrumentHandle, ViUInt32 triggerSource,
ViBoolean STEnabled, int STStatus);

**Purpose**

Directly sets mask and software trigger status for any trigger line.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

triggerSource

Variable Type ViUInt32

Select the trigger source:

1 - Launch software trigger on external trigger line
2 - Launch software trigger on PXI0 trigger line
3 - Launch software trigger on both trigger lines

Note: only "ST Status" parameter is relevant in this function when operating 51-620 card

STEnabled

Variable Type ViBoolean

set whether to mask hardware external input or not
true - mask, false - don't mask

Note: only "ST Status" parameter is relevant in this function
when operating 51-620 card

STStatus

Variable Type int

set level of signal on software trigger input
true - high level, false - low level

### pi620_SetTriggerMode

ViStatus pi620_SetTriggerMode (ViSession instrumentHandle, ViUInt32 source, ViUInt32 triggerMode);

**Purpose**

Select the trigger source and trigger mode.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

source

Variable Type ViUInt32

The trigger source:

0 = Front
1 = PXI 0
2 = PXI 1
3 = PXI 2
4 = PXI 3
5 = PXI 4
6 = PXI 5
7 = PXI 6
8 = PXI 7
9 = PXI Star

triggerMode

Variable Type ViUInt32

The trigger mode:

0 = High level
1 = Low level
2 = Positive edge
3 = Negative edge
4 = Postive edge single shot
5 = Negative edge single shot
6 = Continuous triggerd

### pi620_StoreCalibrationData

ViStatus pi620_StoreCalibrationData (ViSession instrumentHandle);

**Purpose**

Store calibration data to eeprom.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

### pi620_WriteCalibrationDate

ViStatus pi620_WriteCalibrationDate (ViSession instrumentHandle, ViUInt32 year, ViUInt32 month, ViUInt32 day);

**Purpose**

Set the last calibration date. Date will be stored in on board eeprom.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

year

Variable Type ViUInt32

Last year of calibration.

month

Variable Type ViUInt32

Last month of calibration.

day

Variable Type ViUInt32

Last day of calibration.

### pi620_WriteCardId

ViStatus pi620_WriteCardId (ViSession instrumentHandle, ViUInt32 ID);

**Purpose**

Write an id (32-bit) to the onboard serial eeprom. The id will be placed in the on board eeprom.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

ID

Variable Type ViUInt32

Id for the instrument (4 bytes).s

### pi620_WriteEeprom

ViStatus pi620_WriteEeprom (ViSession instrumentHandle, ViUInt32 eepromAddress, ViUInt32 data);

**Purpose**

Write an value directly to the onboard serial eeprom. This function will not check if you overwrite a calibration value in
the eeprom! Use this function for debug purposes only!

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

eepromAddress

Variable Type ViUInt32

Eeprom address.

data

Variable Type ViUInt32

8-bit data to write.

### pi620_WriteInstrumentMemory

ViStatus pi620_WriteInstrumentMemory (ViSession instrumentHandle, ViUInt32 data);

**Purpose**

Write data word to instrument memory. Initialize address counter with pi620_ResetAddressCounter.
This function will increment the address counter with 1.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

data

Variable Type ViUInt32

Data word to write. Memory is 16 bit wide.

### pi620_WriteInstrumentMemoryArray

ViStatus pi620_WriteInstrumentMemoryArray (ViSession instrumentHandle, ViUInt32 length, ViUInt32 _VI_FAR buf32[]);

**Purpose**

Write buffer with data to the instrument memory.Init address counter with function pi620_ResetAddresCounter.
Maximum of available memory places are 256k/counterstep.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

length

Variable Type ViUInt32

Lengh of the buffer.

buf32

Variable Type ViUInt32[]

Buffer with data for the instrument memory.

### pi620_WriteRegister

ViStatus pi620_WriteRegister (ViSession instrumentHandle, ViUInt32 address, ViUInt32 data);

**Purpose**

Writes data to the instrument register at the specified offset.
Refer to the instrument user's manual for register definitions.

**Parameter List**

instrumentHandle

Variable Type ViSession

The ViSession handle that you obtain from the pi620_init function.
The handle identifies a particular instrument session.

Default Value: None

address

Variable Type ViUInt32

The offset address of the register to write.

data

Variable Type ViUInt32

The data to write to the specified offset.

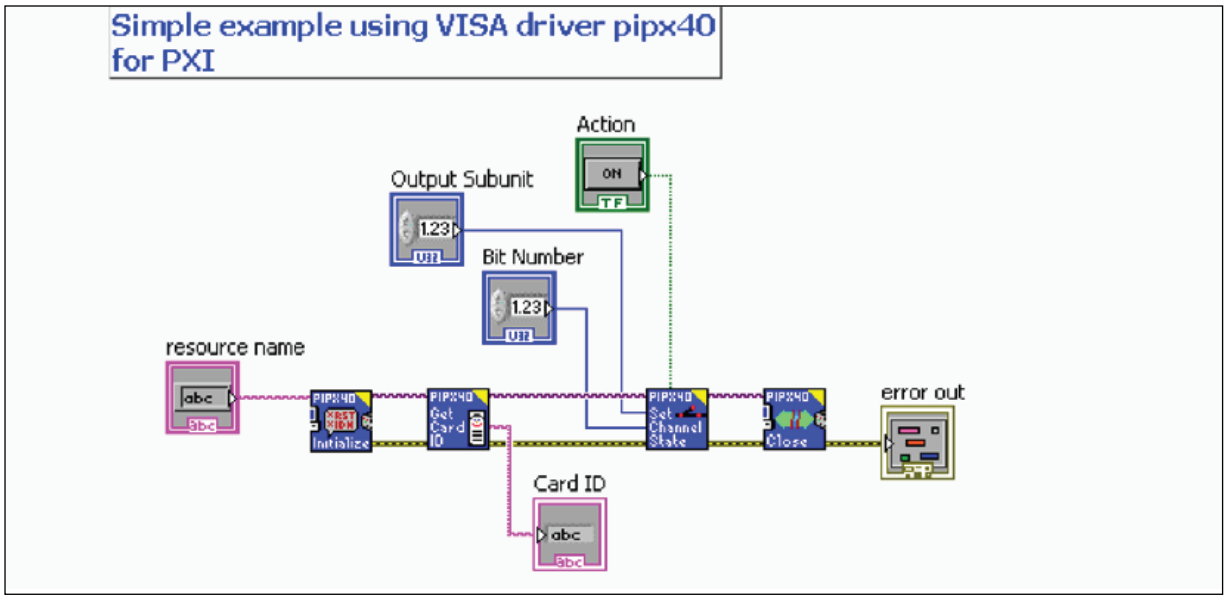
## Using Pickering Drivers In LabVIEW

Most Pickering drivers include a LabVIEW wrapper to permit full operation of the Pickering product from the LabVIEW environment. These wrappers are normally installed to the current LabVIEW folder system during installation of the Pickering driver.

The process of controlling a Pickering product normally consists of the following steps:

1. Open a control session on the product.
2. Use functions in the library to operate the product.
3. Close the control session.

Following is a simple example of the use of the most common Pickering drivers:




**Example Use of Pickering Drivers in LabVIEW**

To download LabVIEW examples, please go to:

Using Pickering Drivers in LabVIEW

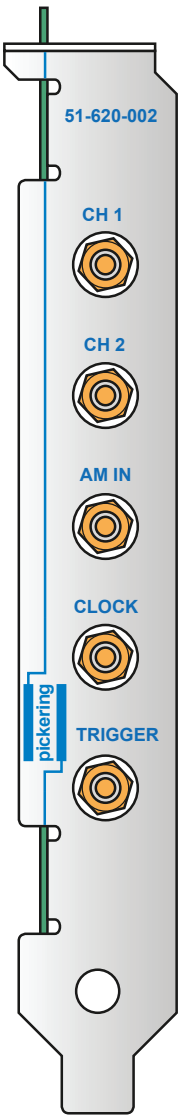



**Figure 5.1 - 51-620-002 PCI Function Generator Front Panel: Connector Positions**

# Section 6 - Troubleshooting

**Refer to the Warnings and Cautions at the front of this manual.** 

## Installation Problems

The Plug & Play functionality of Pickering switch cards generally ensures trouble-free installation.

If you do experience any installation problems you should first ensure that all cards are properly seated in their slots. Improperly mated cards may go undetected by the operating system, or may be detected as a card of an unknown type. They can also cause the computer to freeze at various stages in the boot sequence.

## Diagnostic Utility

The Pickering Diagnostic Utility (accessible through the Programs>>Pickering>>PXI Utilities menu) generates a diagnostic report of the system's PCI configuration, highlighting any potential configuration problems. Specific details of all installed Pickering switch cards are included. All the installed Pickering switch cards should be listed in the "Pilpxi information" section - if one or more cards is missing it may be possible to determine the reason by referring to the PCI configuration dump contained in the report, but interpretation of this information is far from straightforward, and the best course is to contact Pickering support: support@pickeringtest.com, if possible including a copy of the diagnostic report.

In the "VISA information" section, if VISA is not installed its absence will be reported. This does not affect operation using the Direct I/O driver, and is not a problem unless you wish to use VISA. VISA is a component of National Instruments LabWindows/CVI and LabVIEW, or is available as a standalone environment.

If VISA is present and is of a sufficiently recent version, the section "Pipx40 information" should present a listing similar to "Pilpxi information".

Please note that the Diagnostic Utility cannot access cards if they are currently opened by some other application, such as the Test Panels or Terminal Monitor.

# Section 7 - Maintenance Information

**Refer to the Warnings and Cautions at the front of this manual.** 

## Periodic Maintenance

The PCI card does not require any periodic maintenance. Please consult the chassis supplier regarding any periodic maintenance required for the chassis.

## General Cleaning

- Isolate the electrical power from the chassis and ensure that no user I/O signals are being applied.
- Wipe the product & chassis surfaces with a clean dry anti-static cloth only.

## Software Update

For PCI cards operating in a PCI system, no module software updates are required. For the latest version of the driver please refer to our web site pickeringtest.com where links to our Software Download page will provide the latest version of the driver software for the various programming environments encountered.

## Calibration

For applications where regular module level calibration is required then we suggest using a one year calibration interval.

### Calibration Procedure

The calibration procedure is based on using the supplied soft front panel to control the module. Each channel of the module should be checked.

### Calibration/Verification Cables

Cables for the calibration/verification procedure are available from Pickering.
Part number: 40-978-501-1M (2 required).

### Frequency

Use a frequency counter to measure the output frequency of the module for each channel in turn.

Connect the channel output to a frequency counter (set for 1 Hz resolution) and connect the frequency standard output (10 MHz) from the counter to the module's frequency standard input (Clock). Set the module to accept an external frequency standard.

**Note:** Using an external standard ensures that module and the counter use the same frequency reference.

Set the 51-620 to generate a sine wave with an output voltage 1 V peak to peak. Set the output frequency to the values listed in Table 7.1a and check that the output frequency is within 1 Hz of the requested value.

Disconnect the frequency standard connection between the counter and the module and set the module to internal frequency standard. The output frequency will change, the accuracy being dependent upon the frequency standard accuracy of the PXI chassis which is typically within 1 in 20,000 (chassis specification dependent). The output signal should continue to be generated.

### Output Level Accuracy

Connect the module output to a DMM with a high input impedance (1 MΩ or greater).

Set the DMM to measure AC Voltage

Set the module output frequency to 1 kHz sine wave.

Set the output level to the values listed in Table 7.2 and record the DMM results measured as an RMS value, check that the results are within the limits indicated.

### DC Offset

Connect the module to a DMM measuring DC Voltage.

Set module output frequency to 1 MHz and 0.1 V peak to peak (minimum output).

Set the DC Offset Voltages to the values shown in Table 7.3, record the DMM results and check the results are within limits indicated.

**Note:** The results are a functional test only.

### Frequency Response

Connect the module to a scope with at least 100 MHz bandwidth and set the input impedance of the scope to 50 Ω.

Set the module output level to 1 V peak to peak

Set the output frequency to 1 kHz sine wave

Set the waveform on the scope to full scale display, visually check the waveform is smooth with no obvious discontinuities.

Set the frequency to the values shown in Table 7.4 and measure the change in peak to peak signal amplitude relative to 1 kHz and check that it remains within limits.

### Spectral Purity

Connect the module to a spectrum analyzer and set the module output frequency to 100 kHz. Set the output level to 5 V peak to peak.

Use the spectrum analyzer to measure the maximum spurious signal level and record the result at the frequencies in Table 7.5.

### TEST RESULTS

The following page can be printed out (one for each channel) and used to record the results of the calibration procedure.

| | |
|---|---|
| Part Number: | Channel Number: |
| Serial Number: | Date of Calibration: |
| Operator Name: | |
| Test Equipment Used: | |

Table 7.1a Frequency Test - External Frequency Standard

| Set frequency | Measured Frequency | Limit | Pass/Fail |
|---|---|---|---|
| 1000 Hz | | ±1 Hz | |
| 10 kHz | | ±1 Hz | |
| 100 kHz | | ±1 Hz | |
| 1 MHz | | ±1 Hz | |
| 10 MHz | | ±1 Hz | |

Table 7.1b Frequency Test - PXI Frequency Standard

| Set frequency | Measured Frequency | Limit | Pass/Fail |
|---|---|---|---|
| 1000 Hz | | ±2 Hz | |

Table 7.2 Output Level Accuracy

| Set Ouptut | RMS Value | Measured Value | Limits | | Pass/Fail |
|---|---|---|---|---|---|
| | | | Min | Max | |
| 10 | 3.53607 | | 3.46535 | 3.60679 | |
| 5 | 1.76803 | | 1.73267 | 1.80339 | |
| 2.5 | 0.88402 | | 0.86634 | 0.90170 | |
| 1.25 | 0.44201 | | 0.43317 | 0.45085 | |
| 1 | 0.35361 | | 0.34653 | 0.36068 | |
| 0.5 | 0.17680 | | 0.17327 | 0.18034 | |
| 0.25 | 0.08840 | | 0.08663 | 0.09017 | |
| 0.1 | 0.03536 | | 0.03465 | 0.03607 | |

Table 7.3 DC Offset (functional Test)

| Set Ouptut | Measured Value | Nominal Limits | |
|---|---|---|---|
| | | Min | Max |
| 5 | | 4.88000 | 5.12000 |
| 4 | | 3.90000 | 4.10000 |
| 3 | | 2.92000 | 3.08000 |
| 2 | | 1.94000 | 2.06000 |
| 1 | | 0.96000 | 1.04000 |
| 0 | | -0.02000 | 0.02000 |
| -1 | | -1.04000 | -0.96000 |
| -2 | | -2.06000 | -1.94000 |
| -3 | | -3.08000 | -2.92000 |
| -4 | | -4.10000 | -3.90000 |
| -5 | | -5.12000 | -4.88000 |

Table 7.4 Output Frequency Response at 1V peak to peak

| Frequency | Measured Change % | Change Limit | Pass/Fail |
|---|---|---|---|
| 1 kHz | REF | | |
| 1 MHz | | ±2% | |
| 5 MHz | | +2% to -15% | |
| 10 MHz | | +0% to -15% | |

Table 7.5 Spectral Purity

| Frequency | Spectrum Analyzer span | Max spurious level | Limit | Pass/Fail |
|---|---|---|---|---|
| 100 kHz | 10 kHz to 500 kHz | | -40 dBc | |
| 500 kHz | 100 kHz to 3 MHz | | -40 dBc | |
| 1 MHz | 100 kHz to 10 MHz | | -35 dBc | |
| 10 MHz | 1 MHz to 50 MHz | | -35 dBc | |

### Adjustment

In the unlikely event that the module needs adjustment the procedures described as follows can be used to make an adjustment. In normal use no adjustment is needed and we strongly recommend that modules are only adjusted if a repair has been performed on the module.

### Adjustment Procedure

Adjustment of each module channel is carried out using the following steps:

Make sure that the module channels are not running, and in bus mode.

## Dynamical Offset

This procedure minimizes the offset voltage change at maximum Signal DAC reference voltage change, with the Signal DAC set to half scale (0 Volts) One Calibration is valid for all ranges.

For programming the dynamical offset RDAC, please refer to "Offset correction and calibration RDAC" in Appendix A of this document.

- Connect a volt meter to the channel output, AC voltage range.
- Write 8000 hex to the Signal DAC to set it to half scale.
- Apply a signal of 10 Vpp to the AM input.
- Set the channel to AM mode.
- Minimize the AC output voltage by adjusting the dynamical offset potentiometer.
- Deselect AM mode.

## Channel Offset

This procedure minimizes the offset voltage of each channel, and should be done each attenuation range (0 dB..-9.9 dB, -10 dB..-19.9 dB, -20..-29,9 dB and –30..-40 dB). For programming the channel offset RDAC, please refer to "Offset correction and calibration RDAC" in Appendix A of this document.

- Connect a voltmeter to the channel output, DC voltage range.
- Select the desired attenuation range by setting the relays.
- Write 8000 hex to the Signal DAC to set it to half scale.
- Make sure the DC Offset DAC is disconnected.
- Minimize the DC output voltage by adjusting the digital offset potentiometer.
- Resolution steps: 9 mV for 0 dB attenuation range.
  3 mV for –10 dB attenuation range.
  0.9 mV for –20 dB attenuation range.
  0.3 mV for –30 dB attenuation range.

### Channel Gain

This procedure records the upper an lower limits of the output voltage of the signal DAC for each attenuation range (0 dB..-9.9 dB, -10 dB..-19.9 dB, -20..-29,9 dB and –30..-40 dB). These steps should be repeated for each attenuation range setting. For programming the Reference DAC, please refer to “DC offset and Reference DAC” in Appendix A of this document.

- Connect a voltmeter on the channel output, DC voltage range.
- Select the desired attenuation range by setting the relays.
- Write FFFF hex to the Signal DAC to set it to full scale.
- Make sure the DC Offset DAC is disconnected.
- Write 3FFh to the Reference DAC.
- Measure the channel DC output voltage and store this voltage using the pi620_SetRangeLimitVoltages driver-function.
- Write 0h to the Reference DAC.
- Measure the channel DC output voltage and store this voltage using the pi620_SetRangeLimitVoltages driver-function.

### DC Offset DAC

This procedure records the upper an lower output voltage limits of the Offset DAC. For programming the Offset DAC, please refer to “DC Offset and Reference DAC” in Appendix A of this document.

- Connect a voltmeter to the channel output, DC voltage range.
- Select the –30 dB range to minimize the possible channel offset.
- Write 8000 hex to the Signal DAC to set it to half scale.
- Make sure the DC Offset DAC is connected.
- Write 3FFh to the Offset DAC.
- Measure the channel DC output voltage and store this voltage using the pi620_SetOutputOffsetCalVoltages driver function.
- Write 0h to the Offset DAC.
- Measure the channel DC output voltage and store this voltage using the pi620_SetOutputOffsetCalVoltages driver function.

# Appendix A

### Register Space bit definitions

All addresses of the module are selected by means of 9 address lines: A0..A8

The two highest address lines select the channel.

Registers that are shared by the two channels can be found in each channel register space. For instance, the I2C bus, can be found in two places in the register space. In the table, shared addresses are indicated with "**".

Each channel has a DDS clock generator, each having two areas in the channels register space.

The first is a DDS control register, controlling two external control signals of the DDS chip.

The second is the register space of the DDS chip itself, this register space is selected with a logic high on address line A6. The DDS chip register space is described in the DDS clock generator section.

| ADDRESS A8 | ADDRESS A7 | ADDRESS A6.. A0 | READ OR WRITE | DATA BIT | DESCRIPTION |
|---|---|---|---|---|---|
| 0 | 0 | xxxxxxx | R/W | D0..D15 | Channel One Register Space |
| **Signal Memory** | | | | | |
| 0 | 0 | 000 0000 (offset 0) | R/W | D0..D15 | Read or write to signal memory. Every read or write operation automatically increments the memory address counter (note 1). |
| **Main DAC/reset address counter (power-on reset: xxxx xxxx xxxx xxxx) - note 3** | | | | | |
| 0 | 0 | 000 0001 (offset 1) | W | D6..D15 | Write directly to main DAC, This also resets the address counter (note 1, 2 and 4). |
| **Strobe Channel Serial Trigger control register and increment value** | | | | | |
| 0 | 0 | 000 0010 (offset 2) | W | Uses generator serial bus. | Write or read operation latches channel serial bus register contents into the trigger control and increment value registers.<br>D0..D4 = Trigger control.<br>D4..D7 = register increment value. |
| **Strobe Channel Serial Relay register** | | | | | |
| 0 | 0 | 000 0011 (offset 3) | W | Uses generator serial bus. | Write or read latches channel serial bus register contents into the relays register.<br>(D0..D2) |
| **Trigger source (power-on reset: xxxx xxxx xxxx xxx0)** | | | | | |
| 0 | 0 | 000 0100 (offset 4) | W | D0..D4 = 0 | External trigger (leave set to zero as no other trigger source is available for this module). |

# Appendix A

| ADDRESS A8 | ADDRESS A7 | ADDRESS A6.. A0 | READ OR WRITE | DATA BIT | DESCRIPTION |
|---|---|---|---|---|---|
| **Serial bus for offset DAC and reference DAC (power-on reset state: xxxx xxxx xxx0 0x00)** | | | | | |
| 0 | 0 | 000 0101 (offset 5) | W | | The serial DACs are 10 bit.<br>The REF DAC scales the Signal DAC output value between Vref and -Vref, 0000 hex corresponds with -Vref, 3FF corresponds with Vref (note3).<br>The offset DAC sets the signal offset voltage between –5 V (code0) and +5 V (code 3FF). |
| | | | W | D0 | Serial data output SDO. |
| | | | W | D1 | Serial DAC SCLK. |
| | | | W | D2 | Not used. |
| | | | W | D3 | CS 10-bit Offset DAC. |
| | | | W | D4 | CS 10-bit REF DAC. |
| **Lock mode register (power-on reset state: xxxx xxxx xxxx xxxx)** | | | | | |
| 0 | 1 | 000 0110 (offset 6) | W | D0 = 0 | Sets module Bus access mode:<br>All registers and memory are accessible from the bus. |
| | | | W | D0 = 1 | Set Module in Measure mode:<br>The DAC and the Signal memory are not accessible from the bus. |
| **DDS control register** | | | | | |
| 0 | 0 | 000 0111 (offset 7) | W | | This register controls the status of update clock and the FSK/HOLD pins of the DDS chip. |
| | | | W | D0 | Controls Status of DDS Update clock input. |
| | | | W | D1 | DDS FSK/ HOLD software control. |
| | | | W | D2..D5 = 0 | FSK/HOLD controlled by D1 (software). |
| | | | W | D2..D5 = 1 | FSK/HOLD = EXT trigger level. |
| **** Reference Clock source for both channels (power-on reset state: xxxx xxxx xxxx xxx0)** | | | | | |
| 0 | 0 | 000 0111 (offset 8) | W | D0 = 0 | Select onboard 10 MHz clock as source. |
| | | | W | D0 = 1 | Select external clock as source. |

# Appendix A

| ADDRESS A8 | ADDRESS A7 | ADDRESS A6.. A0 | READ OR WRITE | DATA BIT | DESCRIPTION |
|---|---|---|---|---|---|
| **** Generator Serial bus (power-on reset state: xxxx xxxx xxxx x000)** | | | | | |
| 0 | 0 | 000 1001 (offset 9) | W | | This serial bus is shared by the signal generators of the two channels. On each channel the data is shifted into an 8 bit serial register. By writing to register address 2 or 3 of each channel, the shifted data is stored into the channel's control register or relay register. |
| | | | W | D0 | Serial data output SDO. |
| | | | W | D1 | Serial register CLK. |
| **** I2C bus (reset state: xxxx xxxx xxxx xx00)** | | | | | |
| 0 | 0 | 000 1010 (offset A) | R/W | | Reads xxxx xxxx xxxx xxxx<br>A 2 wire I2C bus that controls the RDACS for the offset calibration.<br>Also the output offset can be enabled or disabled writing a control bit in the RDAC chip.<br>Channel 1 RDACS are on I2C address 0.<br>Channel 2 RDACS are on I2C address 1. |
| | | | R/W | D0 | SDA is an open drain serial data output when read on the 9th SCL pulse, the acknowledge state of the RDAC can be read. |
| | | | | D1 | SCL clock output. |
| | | | | D2 | Output enable for SDA (active high), set low to read back the status of SDA (D0). |
| **** Module serial EEPROM 256x16 control (reset state: xxxx xxxx xxxx x000)** | | | | | |
| 0 | 0 | 000 1111 (offset F) | R/W | D0 | EEPROM Data input/output. |
| | | | W | D1 | EEPROM SCLK. |
| | | | W | D2 | EEPROM CS. |
| **DDS chip register space** | | | | | |
| 0 | 0 | 1xx xxxx (40..67) | R/W | D0..D7 | DDS chip for generating the Clock signal. |
| | | DDS chip register space is described in DDS clock generator section. | | | |

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | xxxxxxx | R/W | D0..D15 | Channel 2 Register Space |
| 0 | 1 | Address 80..FF | Register space identical to channel 1. | | |

**Note 1:** Although Data bits D0 to D5 can be read from and written to the signal memory, they are not connected to the main DAC. The LSB of the 10 bit signal DAC is connected to data bit D6 and the MSB of the main DAC is connected to bit D15.

**Note2:** The main DAC register is physically in the DAC, not in the PLD. The power up state of the main DAC register is not defined. The DAC has no reset pin, so at reset the DAC output will not change. Only a positive edge on the DAC clock can change the DAC output voltage. Proper initialisation of the DAC register should be performed by software.

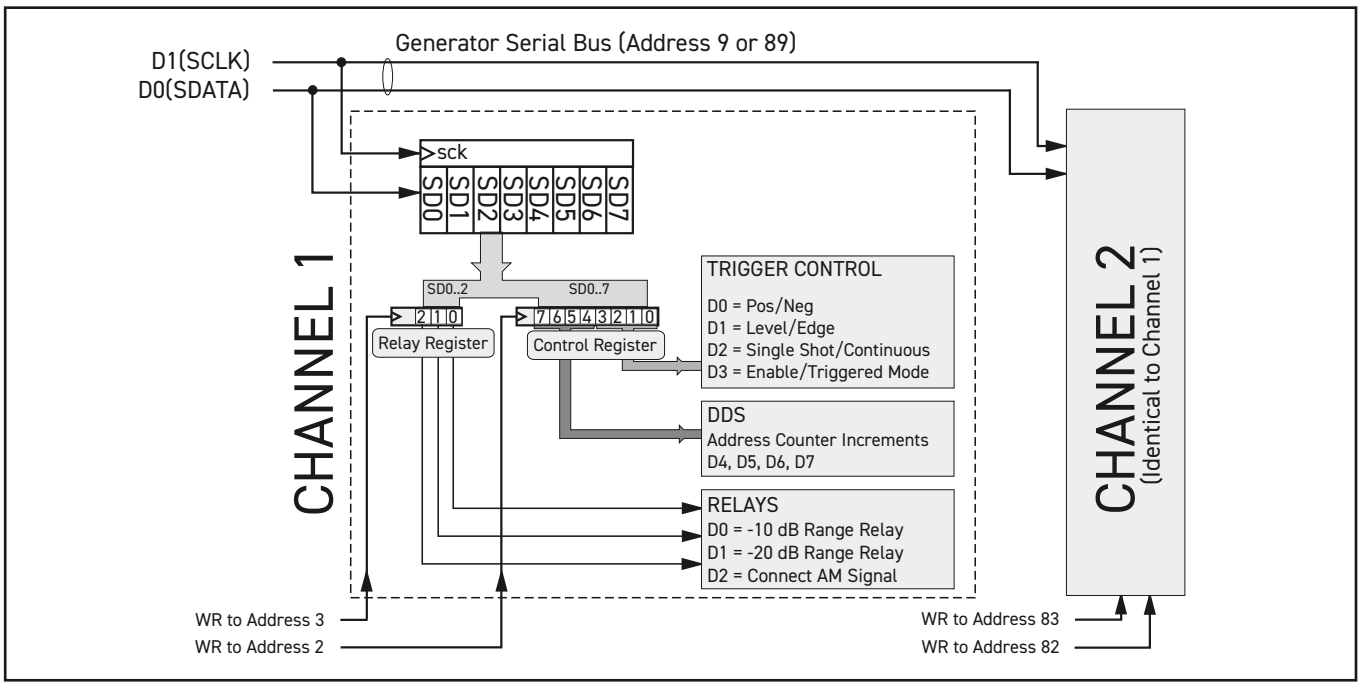
**Note3:** The unused registers are not implemented in the PLD. No register means, no reset state and when a read operation is done, the status of the data bus is read, which will be xxxx xxxx xxxx xxxx.

**Note 4:** To write a value to the main DAC directly, the write operation should be performed two times, successively. The DAC latches the data in on the rising edge of the write. The data DAC is updated on the following falling edge of the write line. The write signal is an active low signal, so the latching will occur during the first write operation, the DAC is updated at the start of the next write operation. This note should only be considered when the DAC is written directly (lockmode is inactive).

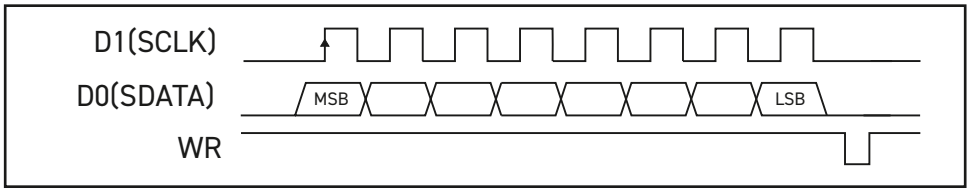
## Function Generator

### Generator Serial Bus

This serial bus is shared by the function generators of the three channels. The data is shifted into three identical 8 bit serial registers, so the contents of the shift registers are the same for each channel. By writing to register address with offset 2 or 3 in the channel's specific register space, the shifted data is stored into the channel's control register or relay register. The data written to these offset addresses is "don't care" since the shifted data is stored into the registers.




The diagram below shows the timing of the generator serial bus:



### Generator Relay Register

The bits D0 and D1 in the relay register control the status of the fixed range relays. D2 switches the AM signal to the signaldacs reference voltage.

| D2 | D1 | D0 | Amplitude Modulation | Output Signal Range (refdac programmed to 0 dB attenuation) |
|---|---|---|---|---|
| 0 | 0 | 0 | Off | 0 dB (10 Vpp) |
| 0 | 0 | 1 | | -10 dB (3.162 Vpp) |
| 0 | 1 | 0 | | -20 dB (1 Vpp) |
| 0 | 1 | 1 | | -30 dB (0.3162 Vpp) |
| 1 | 0 | 0 | On | 0 dB (10 Vpp) |
| 1 | 0 | 1 | | -10 dB (3.162 Vpp) |
| 1 | 1 | 0 | | -20 dB (1 Vpp) |
| 1 | 1 | 1 | | -30 dB (0.3162 Vpp) |

### Generator Control Register

This register is divided in two parts.

The upper 4 bits determine the increment value of the address counter.

The signal frequency from the signal generator is dependent of the clock frequency applied by the DDS clock generator and the address counter increment step size. The increment size is 2n, where n ranges from 0 to 15. The value for n is programmed by writing the upper 4 bites of the control register.

Assuming that there is one complete signal period programmed in the memory, the signal frequency can be calculated:

$$f_{signal} = f_{clock}\, 2^{(n-18)}$$

Where:

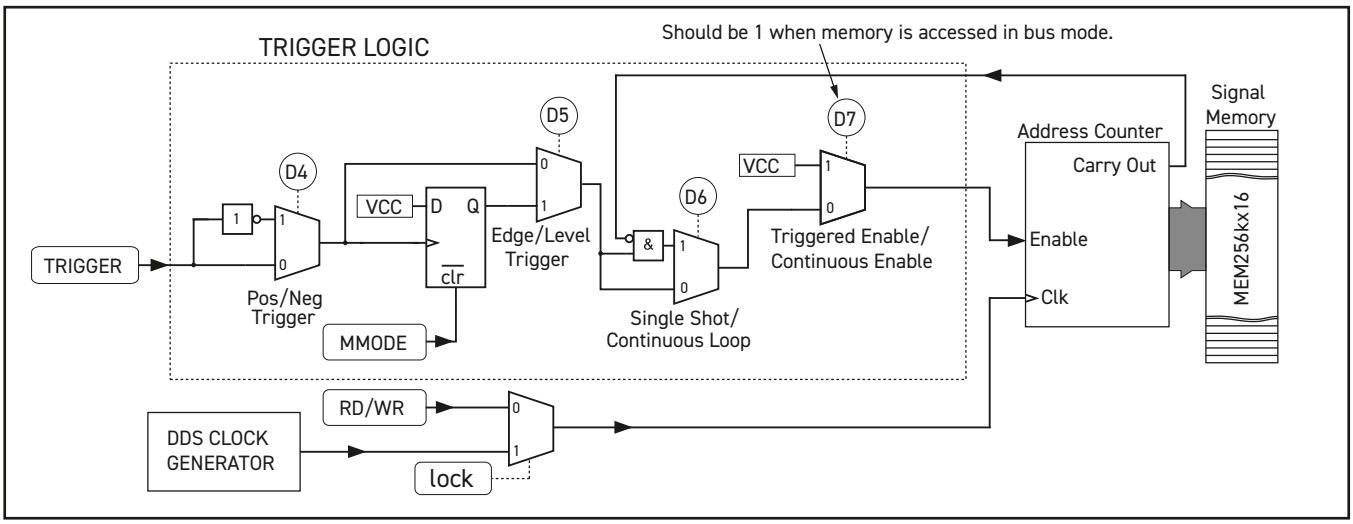
$f_{clock}$ is the Clock Frequency Generated By the DDS Clock Generator.

**n** is the value programmed by D0..D3 in the control register.

The lower 4 bits control the trigger setup of the channel:

| Data Bit | | Description |
|---|---|---|
| D0 | 0 | High level or rising edge sensitive trigger |
| | 1 | Low level or falling edge sensitive trigger |
| D1 | 0 | Level triggering |
| | 1 | Edge triggering |
| D2 | 0 | Continuous mode |
| | 1 | Single shot mode, re-triggerable if edge triggering is enabled |
| D3 | 0 | Enable counter continuously (i.e. for writing to memory) |
| | 1 | Enable counter on trigger |

The diagram below shows how the trigger and clock circuit for the address counter are implemented:




Please note that the Channel trigger ***source*** is selected on offset address 4. To reset the address counter, perform a read or write operation to the signaldac (register address with offset 1).

In edge triggered mode, the pattern starts on a rising or falling edge of the trigger signal. The pattern stops when Lockmode (MMODE) is reset (bit D0, offset address 6).

In single-shot, edge triggered mode, the pattern can be re-triggered, without resetting lockmode (MMODE) bit.

### DDS clock generator

The DDS clock generator can be programmed by means of 8 bits parallel data and a 5 bit parallel address bus. The DDS registers can be found at address:

**40h..67h** for channel one; **C0h..E7Fh** for channel two.

The DDS chip has some features that have not be used in this design. This document will only describe the features necessary for the DDS clock generator function.

## Chip Register Layout of the DDS

| Address offset | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | Default value |
|---|---|---|---|---|---|---|---|---|---|
| 00 | Phase1 adjust register [13:8] bits 15 and 14 are don't care | | | | | | | | 00h |
| 01 | Phase1 adjust register [7:0] | | | | | | | | 00h |
| 02 | Phase2 adjust register [13:8] bits 15 and 14 are don't care | | | | | | | | 00h |
| 03 | Phase2 adjust register [7:0] | | | | | | | | 00h |
| 04 | Frequency tuning word 1 [47:40] | | | | | | | | 00h |
| 05 | Frequency tuning word 1 [39:32] | | | | | | | | 00h |
| 06 | Frequency tuning word 1 [31:24] | | | | | | | | 00h |
| 07 | Frequency tuning word 1 [23:16] | | | | | | | | 00h |
| 08 | Frequency tuning word 1 [15:8] | | | | | | | | 00h |
| 09 | Frequency tuning word 1 [7:0] | | | | | | | | 00h |
| 0A | Frequency tuning word 2 [47:40] | | | | | | | | 00h |
| 0B | Frequency tuning word 2 [39:32] | | | | | | | | 00h |
| 0C | Frequency tuning word 2 [31:24] | | | | | | | | 00h |
| 0D | Frequency tuning word 2 [23:16] | | | | | | | | 00h |
| 0E | Frequency tuning word 2 [15:8] | | | | | | | | 00h |
| 0F | Frequency tuning word 2 [7:0] | | | | | | | | 00h |
| 10 | Delta Frequency word [47:40] | | | | | | | | 00h |
| 11 | Delta Frequency word [39:32] | | | | | | | | 00h |
| 12 | Delta Frequency word [31:24] | | | | | | | | 00h |
| 13 | Delta Frequency word [23:16] | | | | | | | | 00h |
| 14 | Delta Frequency word [15:8] | | | | | | | | 00h |
| 15 | Delta Frequency word [7:0] | | | | | | | | 00h |
| 16 | Update clock [31:24] | | | | | | | | 00h |
| 17 | Update clock [23:16] | | | | | | | | 00h |
| 18 | Update clock [15:8] | | | | | | | | 00h |
| 19 | Update clock [7:0] | | | | | | | | 40h |
| 1A | Ramp rate clock [19:16] bits 13, 22, 21 and 20 are don't care | | | | | | | | 00h |
| 1B | Ramp rate clock [15:8] | | | | | | | | 00h |
| 1C | Ramp rate clock [7:0] | | | | | | | | 00h |
| 1D | | | | Comparator powerdown | Reserved, always 0 | ctrl. DAC powerdown | DAC powerdown | Digital powerdown | 10h |
| 1E | | PLL range | Bypass PLL | refmultiply bit4 | refmultiply bit3 | refmultiply bit2 | refmultiply bit1 | refmultiply bit0 | 64h |
| 1F | ClrACC1 | ClrACC2 | Triangle | | Mode 2 | Mode 1 | Mode 0 | INT/EXT update clk | 01h |
| 20 | | Bypass inv. sinc | OSK EN | OSK INT | | | LSB First | SDO active | 20h |
| 21 | Output Shape multiplier [11:8] | | | | | | | | 00h |
| 22 | Output Shape multiplier [7:0] | | | | | | | | 00h |
| 23 | Don't care | | | | | | | | 00h |
| 24 | Don't care | | | | | | | | 00h |
| 25 | Output shape key ramp rate [7:0] | | | | | | | | 80h |
| 26 | Control DAC register [11:18] (Not used) | | | | | | | | 00h |
| 27 | Control DAC register [0:7] (Not used) | | | | | | | | 00h |

Grey cells are don't care or not used. On the next pages a short description of each register is listed.

## DDS Chip Register Description

**Offset address 04h..09h Frequency Tuning Word 1 (FTW1)**

FTW1 programs the signal frequency in single frequency operation.

**Offset address 0Ah..0Fh Frequency Tuning Word 2 (FTW2)**

FTW2 is used in FSK mode operation. In this mode the output frequency of the DDS is a function of the values loaded in FTW1 and FTW2. In ramped FSK mode, FTW1 contains the lowest frequency, FTW2 the highest frequency.

**Offset address 10h..15h Delta frequency word (DFW)**

DFW is used in ramped FSK and Chirp mode. In FSK mode it should be programmed as a positive two's complement value (MSB=logic 0). In Chirp mode, the value may be either positive or negative. The DFW affects the number of intermediate frequencies between F1 and F2 when the frequency ramps.

**Offset address 1Ah..1Ch Ramp rate clock (RRC)**

RRC is a 20 bit register an is used in ramped FSK and Chirp mode. It affects the time spent at each intermediate frequency. The RRC is a count down counter that outputs a single pulse whenever the count reaches zero. On a pulse of the RRC frequency accumulator increments and ramps the frequency.

**Offset address 1Dh Chip Control Register 1**

**bit 0, 1, 2, 4:** Several individual stages may be powered down. In this design the control DAC is not used, this can be powered down (write 1). For normal operation all other power down bits should be programmed low. **Write bit 0, 1 and 4 logic 0, and bit 2 logic 1.**

**bit 3:** Reserved a logic 1 causes the DDS to stop working. **Write logic 0.**

**bit 5, 6, 7:** Don't care. **Write logic 0.**

**Offset address 1Eh Chip Control Register 2**

**bit 0, 1, 2, 3, 4:** Reference clock multiplier value. Valid values: 4h..14h (4..20dec)

**bit 5:** Bypass PLL, to allow direct clocking of the DDS chip.
Power on state is logic 1.

**bit 6:** Controls the PLL VCO gain. For operation from 200 MHz to 300 MHz this bit should be set to logic 1.

**bit 7:** Don't care. **Write logic 0.**

**Offset address 1Fh Chip Control Register 3**

**bit 0:** Selects internal or external update clock. After writing to the registers described in this section, the update clock transfers the contents of the registers into the internal works of the DDS chip for processing. This can be done internally (as function of the internal system clock) or external. When set to external, (logic 0), this bit is controlled by bit D0 of the DDS control register, at channel offset address 7. **Write logic 0.**

**bit 1, 2, 3:** DDS operational mode select bits mode0..mode3.

| mode2 | mode1 | mode0 | Description |
|---|---|---|---|
| 0 | 0 | 0 | Single tone mode |
| 0 | 0 | 1 | Unramped FSK mode |
| 0 | 1 | 0 | Ramped FSK |
| 0 | 1 | 1 | Chirp mode |
| 1 | 0 | 0 | BPSK mode |

BPSK mode is not used

**bit 4:** Don't care. **Write logic 0.**

**bit 5:** The Triangle bit is used in FSK mode. When set to logic 1, the frequency ramps up from FTW1 to FTW2 and back.

**bit 6:** When set, both Phase Accumulator (ACC2) and the frequency accumulator are cleared. Output is DC when set back to logic 0, the output signal starts at phase 0 and the frequency is set by FTW1.

**bit 7:** When set, the Frequency Accumulator (ACC1) is cleared. When in FSK or Chirp mode, the output frequency is set back immediately to the frequency set by F1, on each rising edge of "update clock".

**Offset address 20h Chip Control Register 4**

**bit 0:** Serial port SDO active bit, not used in parallel mode. **Write logic 0.**

**bit 1:** Serial port MSB/LSB first, not used in parallel mode **Write logic 0.**

**bit 4:** Int/Ext Output Shaped Keying. OSK function is not used. **Write logic 0.**

**bit 5:** Output Shaped Keying enable bit. OSK is not used. **Write to logic 0, after reset, this bit is set.**

**bit 6:** Bypass inverse sinc filter. When this bit is set, data from the DDS block goes directly to the 12 bit cosine DAC. When logic 0, the inverse sinc filter performs sinx/x amplitude correction. **Write logic 0.**

**bit 2, 7:** Don't care. **Write logic 0.**

**Offset address 21h..22h**

Output Shape Multiplier. This function applies to OSK and is not used.

**Offset address 25h**

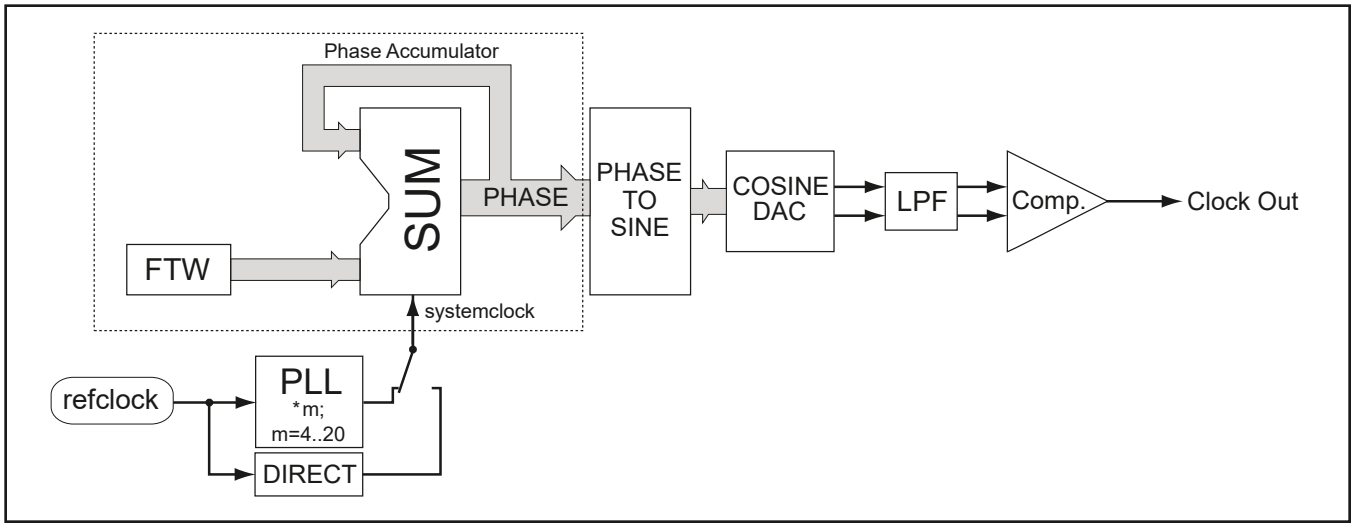
Output Shape Keying ramp rate. This function applies to OSK and is not used.

**Offset address 26h..27h**

Control DAC register. the Control DAC is not used.

### Single Frequency Operation (mode 000bin)

In this operational mode, the output clock frequency is constant. In the diagram, the single frequency operation of the DDS is drawn. A phase accumulator increments the phase, in steps determined by the FTW (Frequency Tuning Word) on each system clock. A small value in the FTW register results in small phase steps and a lower output frequency. A sine wave is generated from the incrementing phase number. The sine wave is then transformed into a block signal by means of a differential comparator.




The output frequency is affected by three factors:

- Reference clock frequency (ƒref)
- Reference clock Multiplier value (m), integer from 4 to 20 or, if bypassed, m = 1
- Frequency Tuning word (FTW), an 48 bit integer value ranging from 20 to 248

The output clock signal can then be determined:

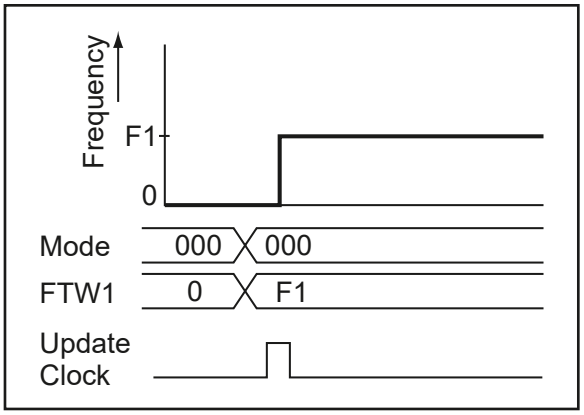
$$f_{clock} = \frac{FTW}{2^{48}} \cdot f_{ref} \cdot m$$

**Note:** to Keep Clock Under Nyquist**:**

$$\frac{FTW}{2^{48}} \leq 0.5 \Rightarrow FTW \leq 2^{47}$$

The reference clock is shared by the two channels and the source can be the onboard reference clock (10 MHz) or an external (user) clock. The DDS chip has a PLL based reference clock multiplier circuit that allows the user to select an integer clock multiplying value over the range of 4x to 20x. Use of this function allows users to input a low frequency refclock (i.e. 10 MHz) to produce an 200 MHz internal system clock. The refclock multiplier function can also be bypassed to allow direct clocking of the DDS chip.

The figure below shows the transition from the default condition to a user defined frequency F1. First the mode and FTW1 register are written to. The DDS changes applies these settings on the rising edge of update clock (D0 of the DDS control register at channel offset address 7).




### FSK/Hold Pin Control

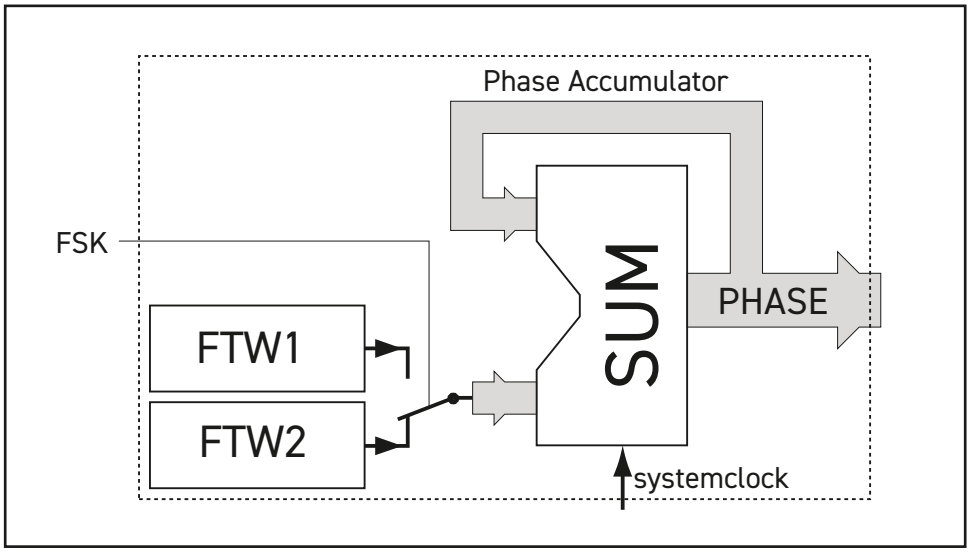
In the operation modes described below, the DDS output frequency can be controlled by a DDS pin, called FSK/HOLD. In the unramped FSK mode for instance the status of this pin switches the output frequency between FTW1 and FTW2. The logical state of the FSK/hold pin can be controlled by firmware by setting or resetting bit 1 at channel offset address 7. Alternatively the status of this pin can be controlled by the External Trigger input.

| D5 | D4 | D3 | D2 | FSK/HOLD status controlled by: |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | D1 (software) |
| 0 | 0 | 0 | 1 | EXT trigger level |

### Unramped FSK Operation (mode 001bin)

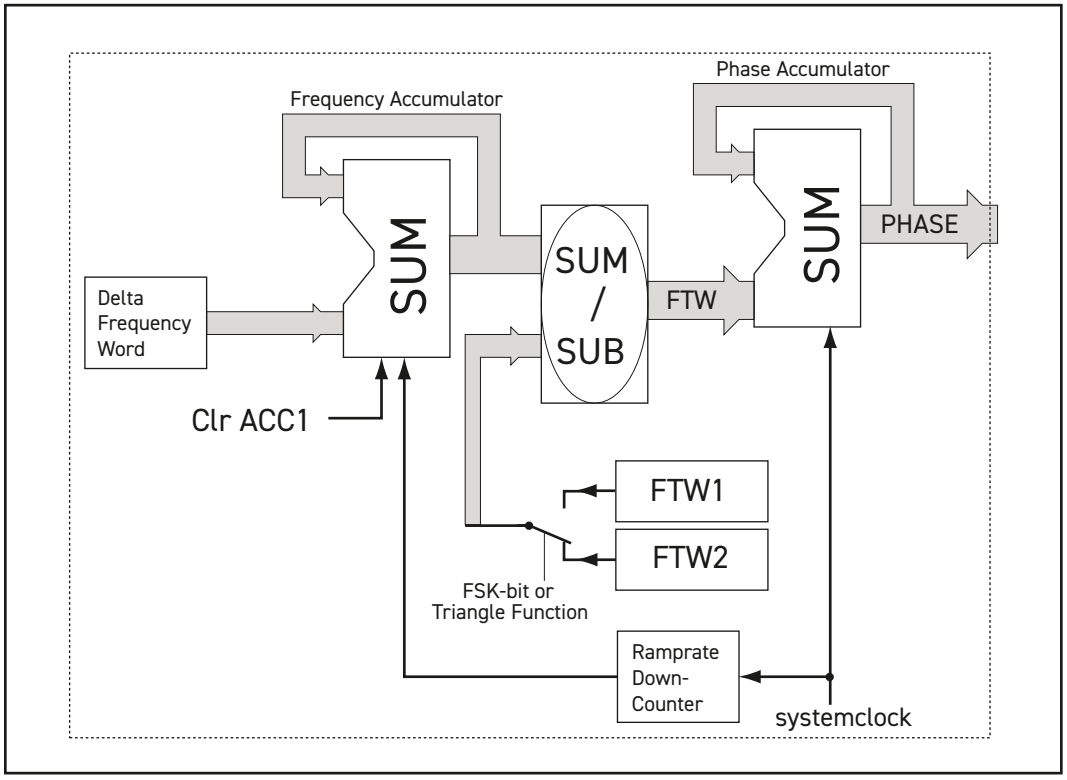
This operational mode resembles that of the single frequency operation.

Now, the output frequency is a function of the values loaded in Frequency Tuning Word 1 and Frequency Tuning Word 2 and the logic level of the FSK bit. The FSK bit is controlled by bit D1 of the DDS control register, or by the external trigger input or one of the PXI triggers. The control of the FSK pin is determined by bit D1..D5 at channel offset address 7. A logic 0 level chooses FTW1, a logic 1 chooses FTW2 as input for the phase accumulator.



### Ramped FSK Operation (mode 010bin)

This is a method of FSK whereby changes from F1 to F2 are not instantaneous, but are accomplished in a frequency sweep. The frequency sweep made is linear and is automatically accomplished. The ramp speed may be changed by changing the ramp rate and the ramp step size. The lowest frequency (F1) must be programmed by FTW1.




The delta frequency word (DFW) determines the frequency step and is required to be programmed as a positive two's complement value.

The time spent at each intermediate frequency can be programmed in the 20-bit Ramp Rate Counter (RRC), a down counter that clocks the Frequency Accumulator.

The RRC time can be derived from:

$$\text{time} = (N + 1) \cdot \text{Syslockperiod} = \frac{(N + 1)}{f_{ref} \cdot m}$$

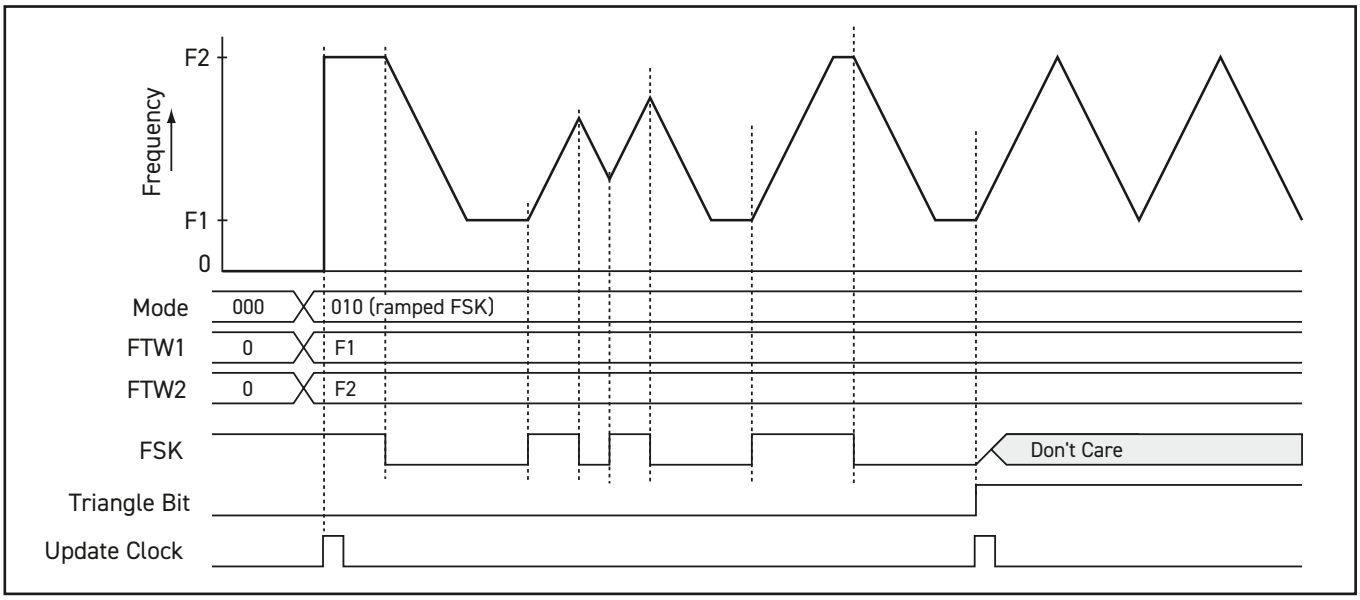
Where:

| | |
|---|---|
| **m** | is the Reference Clock Multiplier Value. |
| $\mathbf{f_{ref}}$ | is the Reference Clock Frequency. |
| **N** | is the 20 Bit Ramp Rate Clock Value. |

The RRC stops automatically when the destination frequency has been reached.

The 48 bit delta frequency word is added to the accumulator's output every time it receives a pulse from the RRC. This value is then added to or subtracted from the contents of FTW1 or FTW2, which is then fed to the input of the 48 bit Phase accumulator that again forms the numerical phase steps for the sine output. In this fashion, the output frequency is ramped up and down, according to the logic state of FSK/HOLD pin. The status of this pin is dependent of the status of D1..D5 at channel offset address 7.

The state of the FSK pin is copied into the internal works of the DDS chip at the edge of update clock, and then affects the start frequency (F1 when logic 0, or F2 when logic 1). In the figure below, FSK is logic 1 at the edge of Update clock, so, the start frequency is F2.



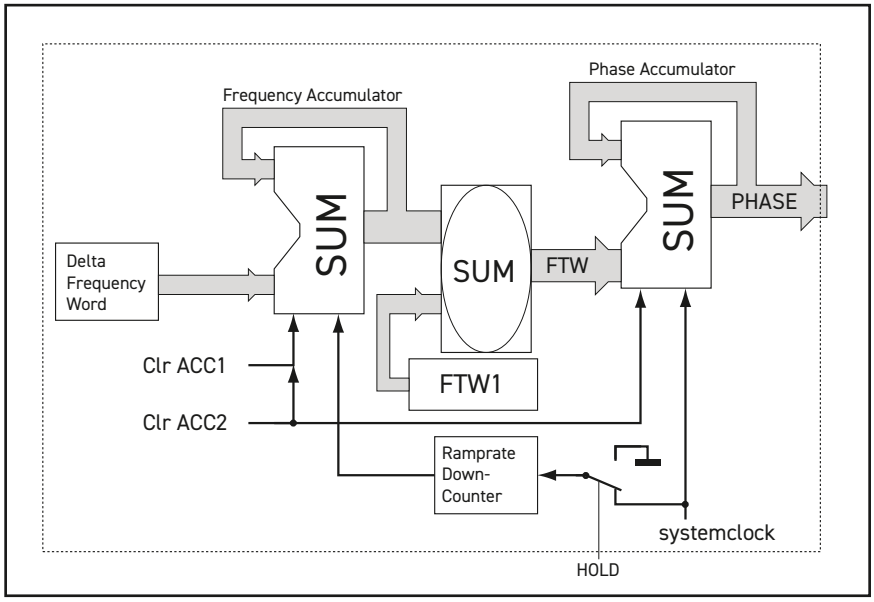

Chip Control Register 3 (DDS offset address 1F) contains a Triangle bit (bit5). Setting this bit in the ramped FSK operational mode, causes an automatic ramp-up and ramp-down, without toggling the FSK bit (triangle mode). In fact the FSK bit has no effect once the triangle bit is set.

Prior to operation, the "Clr ACC1" bit in Chip Control Register 3 (bit 7) should be toggled to assure that the frequency accumulator starts from an "all zeros" output condition.

**Note:** in Triangle mode, when one of the frequency limits is reached, the DDS may take more than one RRC-time to switch the ramp slope from positive to negative or to negative or opposite.

### Chirp Operation (mode 011bin)

In chirp operation, the Delta frequency word defines the frequency step and direction of the ***frequency*** ramp. If the DFW is negative (MSB logic 1), then the incremental frequency changes will be in negative direction. Frequency tuning word 1 (FTW1) is only the staring point of the FM chirp.

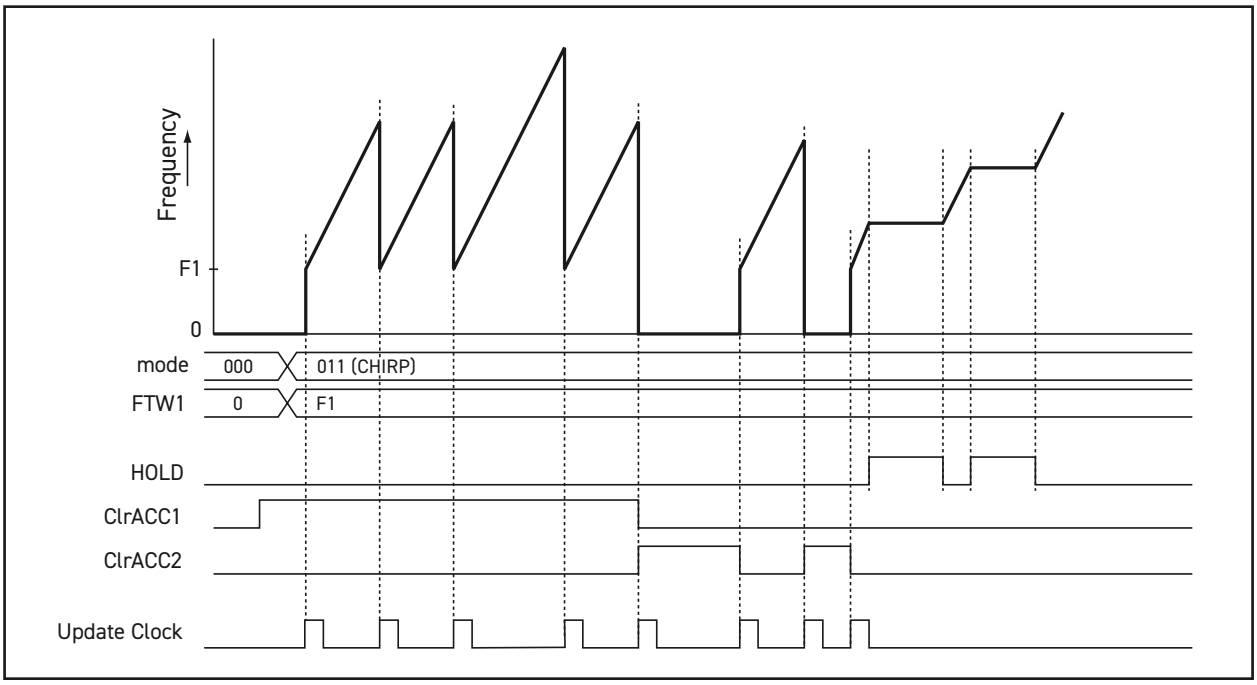



Once the FM chirp has begun its free move (from DC to half the system clock frequency). Instant return to FTW1 can be achieved by clearing the frequency accumulator, by setting **Clr ACC1** (bit 7 of Chip Control Register 3 at DDS offset address 1Fh). When **Clr ACC1** is logic 1, the rising edge of update clock causes the frequency accumulator to reset, setting the output frequency back to F1.

When Clr ACC2 is set high, both the frequency and phase accumulator are reset. The output of the phase accumulator will result in 0 Hz as long as **Clr ACC2** is set high. To return back to previous operation, **Clr ACC2** should be set to logic 0.

Another function that is available is **Hold (FSK/hold pin as described earlier sections).** The effect is to halt the chirp at the existing frequency just before HOLD was set logic 1. The status of this pin is, again, dependent of the status of D1..D5 at channel offset address 7.

The figure shows the effect of Clr ACC1, Clr ACC2 and Hold.



### DC offset and Reference DAC

The Offset and Reference DACs are serially programmed through a serial peripheral interface. This interface is at offset address 5 of each channel register space definition. The 3-wire serial port accepts 16-bit serial words. The three leading bits contain the programming command, (C-bits) followed by 10 data bits, the remaining three sub-bits (S-bits) should be programmed as zero. See the table below for detailed information about the function of the C bits and the S bits.

| 16 Bit Serial Word | | | | | Function |
|---|---|---|---|---|---|
| C2 | C1 | C0 | D9..D0 | S2..S0 | |
| X | 0 | 0 | 10 bits data | 000 | Load input register; DAC register immediately updated (also exit shutdown) |
| X | 0 | 1 | 10 bits data | 000 | Load input register; DAC register unchanged |
| X | 1 | 0 | xxxxxxxxxx | xxx | Update DAC register from input register (also exit shutdown; recall previous state) |
| 1 | 1 | 1 | xxxxxxxxxx | xxx | Shutdown |
| 0 | 1 | 1 | xxxxxxxxxx | xxx | No operation |

x = don't care

Data is loaded MSB first. Data is shifted into the D0(DIN) input on the rising edge of CLK. CS must be taken high before transferring data. After transferring data, CS must be taken low so that the DAC can act on the data.

*For the Offset DAC (Address 3 in the DACBUFFER register space):*

Code 0x0000 results in –5 Volts offset and code 0xFFFF results in +5 V offset measured on the buffer outputs. The 3-wire interface is defined as: D0=SDO, D1=SCK; D3=CS

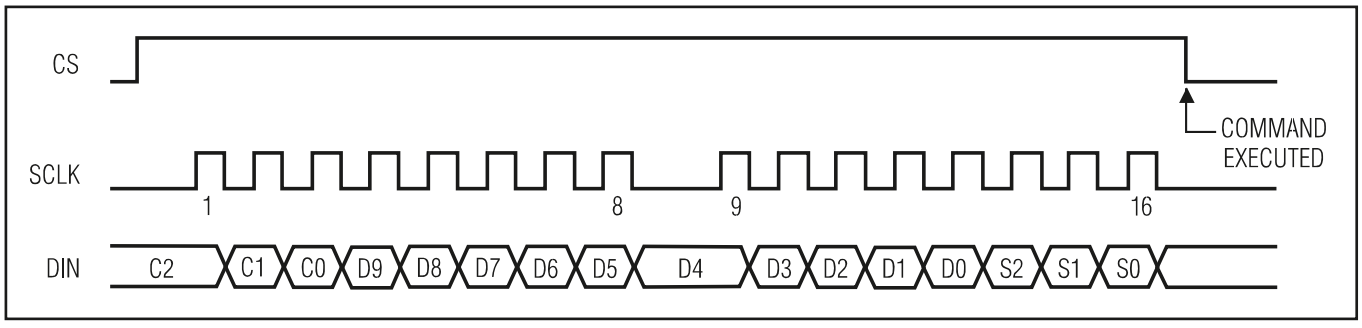
Please note that the offset voltage is connected to the output by programming logic output O1 of the offset calibration RDAC, see the section ***"Programming the Offset correction and calibration RDAC"***.

*For the Reference DAC (Address B in the DAC register space):*

The Variable Gain control is obtained With the Reference DAC.

Code 0x0000 results in -13 dB and code 0xFFFF results in +1.5 dB relative to the full scale signal DAC output voltage swing. The 3-wire interface is defined as: D0=SDO, D1=SCK; D4=CS

For timing data see below:



## Offset correction and calibration RDAC

The RDAC (AD5242) in the gain calibration circuit is controlled via an I2C compatible 2-wire serial bus. The RDAC is connected to this bus as slave device.

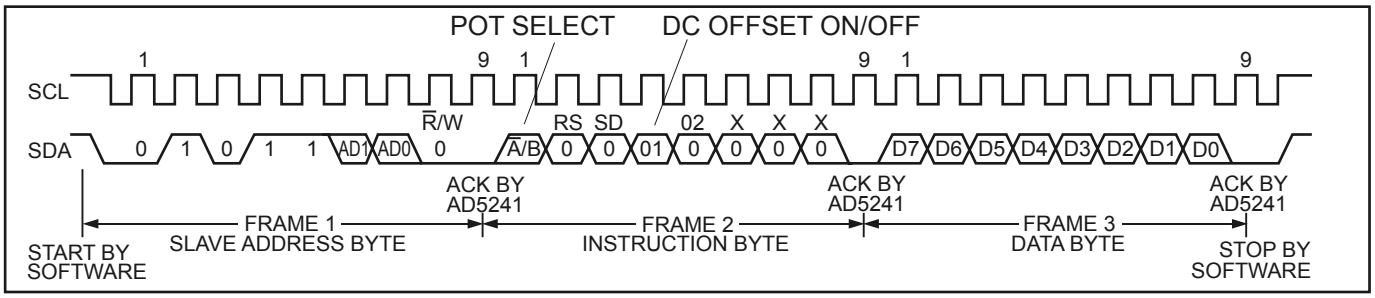
This bus uses the signal SDA (D0 of DAC register address E) and SCL (D1 of DAC register address E) Both signals are open-drain outputs. When writing to SDA, the output of SDA should be enabled (D2 of DAC register address E written high). To read from the status SDA line, D2 of the DAC register should be written low.

The status of SDA can be read back from the same address to read out the "acknowledge" state of the RDAC. Data is loaded MSB first.

Referring to the figure below, the first byte of AD5241 is a Slave Address Byte. It has a 7-bit slave address: the 7 MSBs should be 0101100 where the last two bits are determined by the state of the AD0 and AD1 Pins of the device.

| Channel | AD1 | AD0 |
|---|---|---|
| 1 | 0 | 0 |
| 2 | 0 | 1 |

The 7 bits address is followed by an R/W Bit that should be logic low, because the RDAC will always be written.




The maximum SCL frequency is 100 kHz. The low time of SCL should be at least 4.7 µs, the High time of SCI should be at least 4 µs. A 300 ns minimum data setup time should be taken in account (minimum time between a status change on SCL and SDA and vice versa).

### The 2-Wire I2C Serial Bus Protocol

**1.** The software initiates data transfer by establishing a START condition, which is when a high-to-low transition on the SDA (D0) line occurs while SCL (D1) is high. The following byte is the Slave Address Byte (Frame 1), which consists of the 7-bit slave address followed by a logic "low" (this bit determines that data will be written to the slave device).

On the ninth clock in the first frame, the slave responds by pulling the SDA line low during the ninth clock pulse (this is termed the Acknowledge Bit). The clock pulse remains in control by the software. The software can check the acknowledge bit by reading data line SDA (D0) from address E. To read back the status of SDA, the PLD-SDA pin should be set to input, writing logic low to D2 (SDA Output enable).

Now, the device waits for data to be written to its serial register.

**2.** The Write operation contains an Instruction Byte (Frame2) followed by a data byte (Frame3). The MSB of the Instruction Byte labelled A/B is the RDAC sub address select.

***Set A/B to logic 0 to select the dynamical offset correction potentiometer.***

***Set A/B to logic 1 to select the offset calibration potentiometer.***

The second MSB, RS, is the mid-scale reset. A logic high on this bit moves the wiper of the RDAC to the centre tap. The third MSB, SD, is a shutdown bit. This bit serves the same function as the SHDN Pin except that SHDN Pin reacts to active low. The SD bit should be written low.

The following two bits are O2 and O1. They are extra programmable logic outputs.

***O1 is used for switching on the output offset.*** O2 can always be written low. The three LSBs are Don't Care and can be written low as well. The slave responds by pulling the SDA line low during the ninth clock pulse as acknowledge.

**3.** After acknowledging the Instruction Byte, the last byte in Write Mode is the Data Byte, (Frame 3). Data is transmitted over the serial bus in sequences of nine clock pulses (eight data bits followed by an Acknowledge Bit). The transitions on the SDA line must occur during the low period of SCL and remain stable during the high period of SCL. The slave responds by pulling the SDA line low during the ninth clock pulse as acknowledge.

**4.** When all Data Bits have been written, a STOP condition is established by the software. ***A STOP condition is defined as a low-to-high transition on the SDA line while SCL is high.*** So, the Software will pull the SDA line high during the tenth clock pulse to establish a STOP condition.

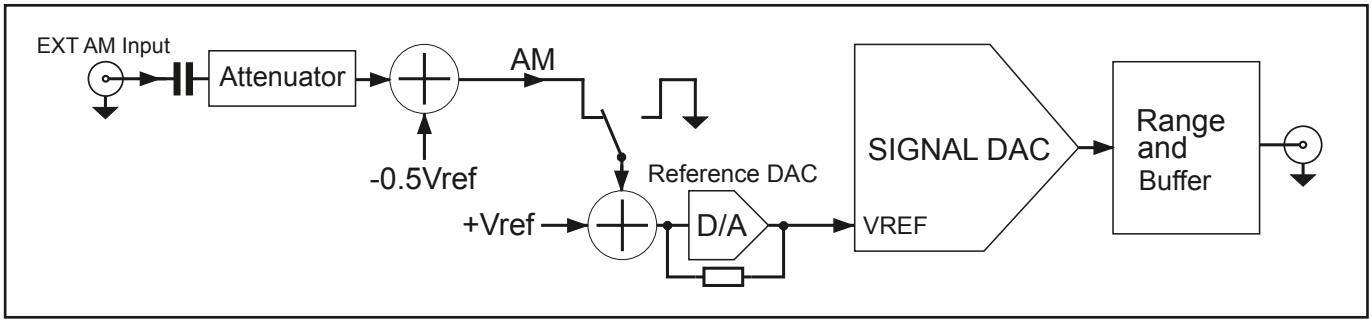
A repeated write function gives the user flexibility to update the RDAC output a number of times after addressing and instructing the part only once. During the Write cycle, each Data Byte will update the RDAC output. For example, after the RDAC has acknowledged its Slave Address and Instruction Bytes, the RDAC output will be updated. If another byte is written to the RDAC while it is still addressed to a specific slave device with the same instruction, this byte will update the output of the selected slave device. If different instructions are needed, the Write Mode has to start a whole new sequence with a new Slave Address, Instruction, and Data Bytes transferred again.

## Amplitude modulation

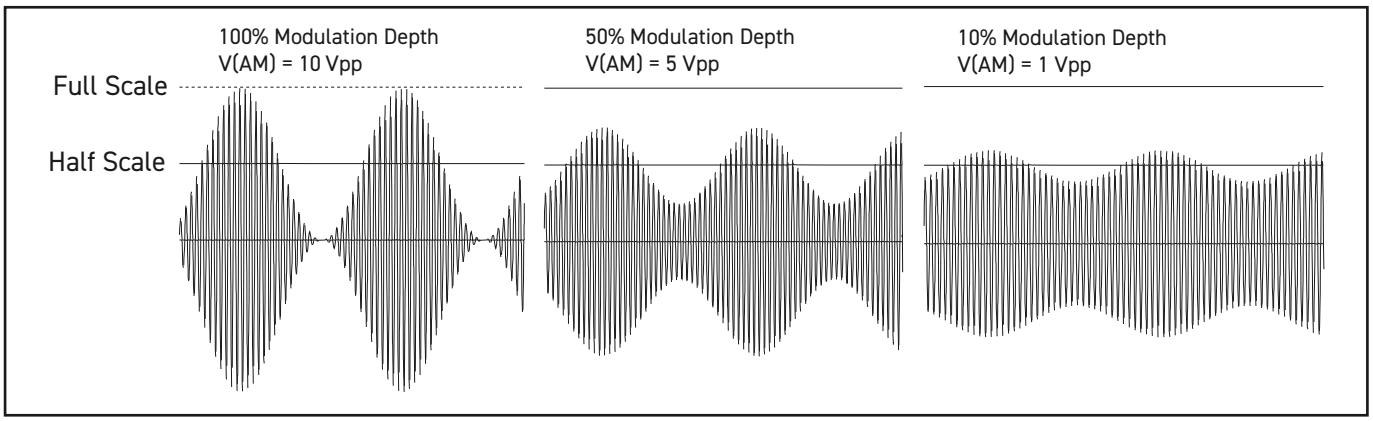
The output signals of the two channels can be amplitude modulated, using the AM input.

The AM input is AC coupled and, after attenuation, the AM signal is summed with minus half Vref. This AM signal is shared by both channels, each channel can be set in AM mode independently.

Then, when AM mode is switched on, the AM signal is summed with +Vref and applied to the signal DAC. The AM input frequency range is 10 Hz..20 kHz.



Assuming that the programmed signal is full scale (The signal memory holds codes from 0000hex to 3FFhex), full amplitude modulation (100%) is achieved with an input voltage of 10 Vpp. When no AM signal is applied, the output signal (carrier) has an amplitude of half full scale.




Like in normal operation, the full scale output voltage is dependent of the chosen range. The range is determined by the range relay bits in the generator relay register and the setting of the reference DAC (for variable Gain control).

AM mode is switched on by setting bit D2 logic 1 in the generator relay register. For details on programming this bit, refer to the "Function Generator" section in this document.

### Formulas

To derive a frequency tuning word:

$$FTW = \frac{f_{Signal} \cdot 2^{48}}{2^{(n-18)} \cdot f_{ref} \cdot m}$$

Where:

**n** Is the Integer Determining the Generator Stimulus Memory Step Size , N=0..15

**m** Is the DDS Refclock Multiplier.

$\mathbf{f_{Ref}}$ Is the Reference Clock Frequency.

$\mathbf{f_{Signal}}$ is the Desired Signal Frequency.

**Note:** to keep DDS clock frequency under nyquist:

$$\frac{FTW}{2^{48}} \leq 0.5 \Rightarrow FTW \leq 2^{47}$$

For best performance FTW should not exceed code 66 66 66 66 66 66hex.
(at 10 MHz refclock this results in a 80 MHz clock frequency)