DynamicSignals, LLC

P635 User's Manual

PXI

P635 User's Manual
Release 1.2.0

Table Of Contents

Chapter 1: Introduction

Description

The P635 is a single-width, 3U, PXI module with eight frequency counter channels. This counter module monitors a variety of frequency sources. Moreover, its unique circuitry allows the monitoring of a wide frequency range without changing any module settings. TTL inputs are provided as well as differential input circuits, with filtering and hysteresis to provide high noise immunity. The threshold at which a signal is recognized is programmable. AC or DC coupling of the differential inputs is programmable on a per-channel basis.

P635 Specifications

Item	Specifications
Inputs	
Number of input channels	8
Type	Differential and TTL
Differential inputs:	
Coupling	AC or DC, per-channel programmable
Input impedance (switchable)	1 MΩ/100Ω(DC coupling) >10 MΩ/100Ω(AC coupling)
Input range (per channel programmable)	P635-AA21: ±20, 40, 100, 200 mV to ±20 V (all ranges) (Max further limited to ±5 V with the 100Ω termination active) P635-AB21: ±100, 200, 500, 1000 mV to ±20 V (all ranges) (Max further limited to ±5 V with the 100Ω termination active)
Switching threshold*	30% of input range minimums as shown above (typical)
Hysteresis*	After a positive-going signal passes the positive threshold, the signal must pass the negative-going threshold to cause switching.
Input protection	47kΩ series resistors followed by ±10 V diode clamps
Common-mode input voltage	±10 V MAX (operating)
Maximum safe input voltage	±50 V, continuous (AC or DC coupling)
Frequency measurement range**	0.06 Hz to 100 kHz (1 MHz clock) 0.6 Hz to 100 kHz (10 MHz clock)
Filtering	
Filter type	Single-pole, low-pass RC type, programmable (filter in/out)
-3 dB cutoff frequency (f_C)	50 kHz
Time Base	
Clock rate	1 MHz or 10 MHz, programmable
Stability	±1 ppm, 0°C to +50°C ±1 ppm/year
Observation Window	From 1 ms to 1.024 s, in 1 ms increments
Counter Sizes	
Time base counter	16,777,215 (24 bits)
Input pulse counter	262,140 (18 bits)
Input Connector Types	50 Position High Density SCSI Socket Connector
Power Requirements	
+5 V	225 mA
+3.3V	755 mA
+12 V	175 mA
-12 V	175 mA

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

P635 Specifications (cont.)

Item	Specifications
Environmental and Mechanical	
Temperature range	
Operational	0°C to +50°C
Storage	-25°C to +75°C
Relative humidity	0 to 85%, non-condensing to 40°C
Cooling requirements	10 CFM
Dimensions	100 mm x 160 mm (3U PXI module)
Front-panel potential	Chassis ground

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

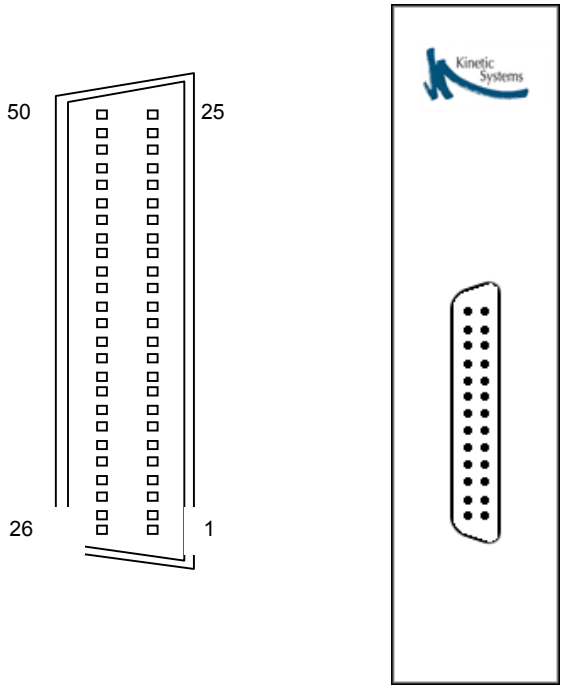
Table 1-1. Specifications

*Tested using 1kHz 50% duty cycle sine wave. P635 DC coupled, filter disabled, unity gain.
** Tested using +/-1V 50% duty cycle sine wave. P635 DC coupled, filter disabled, unity gain.

Front Panel

Connector P1

The 50 position high density SCSI connector with sockets provides the differential analog and TTL input paths for the eight channels and the "health check" (analog and TTL) input paths.



Front Panel Connector Pinout

Pin Number	Signal Description	Pin Number	Signal Description
1	Channel 1 + Input	**26**	Channel 1 - Input
2	Channel 2 + Input	**27**	Channel 2 - Input
3	Ground	**28**	Ground
4	Channel 3 + Input	**29**	Channel 3 - Input
5	Channel 4 + Input	**30**	Channel 4 - Input
6	Ground	**31**	Ground
7	Channel 5 + Input	**32**	Channel 5 - Input
8	Channel 6 + Input	**33**	Channel 6 - Input
9	Ground	**34**	Ground
10	Channel 7 + Input	**35**	Channel 7 - Input
11	Channel 8 + Input	**36**	Channel 8 - Input
12	Ground	**37**	Ground
13	Channel 1 TTL	**38**	Channel 1 TTL Ground
14	Channel 2 TTL	**39**	Channel 2 TTL Ground
15	Channel 3 TTL	**40**	Channel 3 TTLGround
16	Channel 4 TTL	**41**	Channel 4 TTL Ground
17	Channel 5 TTL	**42**	Channel 5 TTL Ground
18	Channel 6 TTL	**43**	Channel 6 TTL Ground
19	Channel 7 TTL	**44**	Channel 7 TTL Ground
20	Channel 8 TTL	**45**	Channel 8 TTL Ground
21	Tick Clock I/O	**46**	Tick Clock I/O Ground
22	Window Clock I/O	**47**	Window Clk I/OGround
23	TTL Health In	**48**	TTL Health Ground
24	Ground	**49**	Ground
25	Analog Health + In	**50**	Analog Health - In

Table 1-2. Front Panel Connector Pinout

Product Ordering Information

Model P635-AA21 8-channel, 100 kHz Frequency Counter, ±20 - 200 mV to ±20 V Range
Model P635-AB21 8-channel, 100 kHz Frequency Counter, ±100 - 1000 mV to ±20 V Range

Related Products

Chapter 2: Installation and Configuration

⚠ Do not install hardware before installing accompanying software. Installing the software before the hardware ensures that the information in the module description file is available to the operating system when it needs to identify the hardware. A brief overview of the installation steps are as follows:

1. Install software
2. Run the *Resource Manager* to register the module type with VISA
3. Power the system down
4. Install the module.
5. Power the system up. The operating system will automatically identify the new hardware and install kernel mode drivers.

Software Installation

The CP635 Plug and Play driver depends on an installed VISA layer. This procedure assumes that VISA has already been installed.

1. Insert the accompanying CD into your system and run setup.exe. This will install the Plug and Play driver code and libraries, as well as the module.ini file.
2. Run the VISA *Resource Manager* tool. The *Resource Manager* will identify the newly installed module.ini file and register the module type with VISA and build appropriate kernel mode driver files for the operating system.

Directory Structure

Software installation will place files as described below. `<VXIPNP>` denotes where VISA is installed (e.g., by default `C:\vxipnp\winnt` on a Windows based machine).

- `<VXIPNP>\include`: ks635.h (API header file)
- `<VXIPNP>\bin`: `ksp635.dll` (API library)
- `<VXIPNP>\lib\<`*format*`>`: (API lib file `ksp635.lib`, in various formats)
- `<VXIPNP\ksp635`: `ksp635.c` (API source code)

In addition, the module_cp635.ini file will be installed in the directory specified by registry setting `HKEY_LOCAL_MACHINE\SOFTWARE\PXISA\CURRENT_VERSION`, value `ModuleDescriptionFilePath`, or simply `<VXIPNP>` if the registry value is not set or does not exist.

Manual Registration with VISA

In the event that your VISA *Resource Manager* does not or cannot automatically register the CP635 with VISA via the module.ini file, you will probably need to manually register it with

VISA. This will probably be accomplished by running a tool or wizard distributed with your VISA; consult your VISA documentation for details.

To manually register the CP635 with VISA, you will need the following information:

- Module Name: "CP635"
- Module Vendor: "KineticSystems Company, LLC"
- Model Code: 0x635
- Manufacturer Code: 0x11f4
- Interrupt Detect and Quiesce: The CP635 generates 1 source of interrupt.
  - Detect: true if a 32 bit read of space BAR2, offset 0x70 returns bit 0 on.
  - Quiesce: To deassert the interrupt, perform a 32 bit write of value 0x1 to space BAR2, offset 0x70

  In PXI terms:

```
NumDetectSequences = 1
InterruptDetect0 = "C32 BAR2 0x70 0x1 0x1;"
InterruptQuiesce = "W32 BAR2 0x70 0x1;"
```

Unpacking the P635

The P635 comes in an anti-static bag to avoid electrostatic damage to the module. Please take the following precautions when unpacking the module:

- When handling module, use a grounding strap or touch a grounded object.
- Touch the anti-static package to a metal part of your PXI chassis before removing the module from the package.
- Remove the module from the package and inspect the module for damage.
- Do not install the module into the PXI chassis until you are satisfied that the module exhibits no obvious mechanical damage and is configured to conform to the desiring operating environment.

Selecting the Analog Input Termination

The analog input path for each of the eight channels has the switch-selectable option of unterminated or terminated. For most applications, the unterminated selection (the factory-set default) should be chosen. A 100Ω termination can also be selected for each of these paths for use with RS-422, RS-485 or similar differential transmitters. If this module is the only (or last) receiver node on an RS-422 or an RS-485 path, the 100Ω termination should be switched in (terminated).

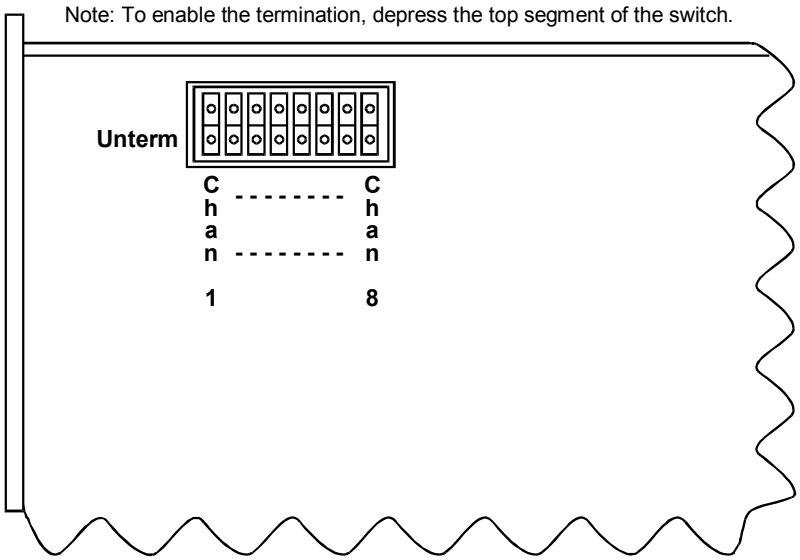


Figure 2-1. P635 Channel Termination Switch Locations

Chapter 3: Device Operation

Overview

The P635 is a single-width, 3U, PXI module with eight frequency counter channels. This counter module monitors a variety of frequency sources. Moreover, its unique circuitry allows the monitoring of a wide frequency range without changing any module settings. TTL inputs are provided as well as differential analog input circuits with filtering and hysteresis to provide high noise immunity. Figure 3-1 is a block diagram showing one channel of the module.

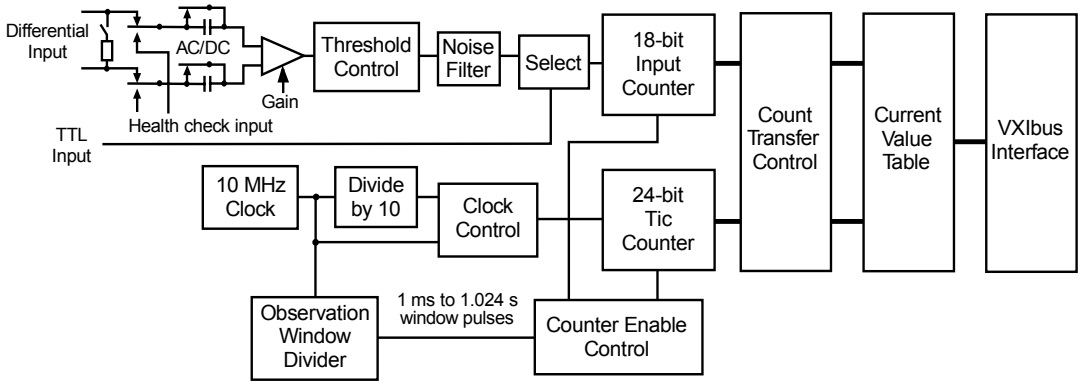


Figure 3-1. P635 Simplified Block Diagram

Basic Circuit Operation

The P635 provides for two types of signal sources: analog and TTL. The analog input paths are differential for high noise immunity. These inputs can be unterminated (high impedance) or terminated with 100Ω, selected by an on-board switch. The analog input coupling can be DC (the normal setting for most applications) or AC (with a series capacitor in each leg of the input path). Each analog path also includes programmable hysteresis control and a programmable high-frequency noise filter. The hysteresis control sets the input pulse detection threshold, while the filter provides a 3 dB rolloff at 50 kHz. The selection of analog or TTL input is programmable on a per-channel basis.

The internal clock for the P635 is driven from a Master 10 MHz clock having a 1 ppm accuracy from 10 °C to 50 °C. The internal clock can be set to 1 MHz or 10 MHz under program control. The 10 MHz clock provides an order of magnitude increase in counting resolution over the 1

MHz clock. The 1 MHz internal clock, however, allows detection of slower frequencies: 0.06 Hz for the 1 MHz clock, as compared to 0.6Hz for the 10 MHz clock

All input channels on the module are monitored over a single user-selectable period, called the observation window. The observation window is programmable from 1 millisecond to 1.024 seconds, in increments of 1 millisecond. The basic resolution of the measurement is ±1 tick clock period. As the name implies, the input signal will be monitored over the course of the observation window, the result of the measurement becoming available when the window closes. In general, measurements are more accurate with longer observation windows, since more cycles of the input are collected for the determination of the final result. Longer observation windows, however, come at the expense of the results being available less frequently (i.e., longer intervals between when new results become available), which limits the ability to track a signal whose frequency is changing faster than the observation window timespan.

While the observation window remains open, the P635 hardware counts–for each channel–the number of internal clock cycles (called the tick count) and the number of input signal cycles (called the period count) that occur. The ratio of the period count and the tick count, multiplied by the clock rate, must be the rate of the input signal:

$$frequency = clock\ \ rate \times \frac{peroid\ \ count}{tick\ \ count}$$

For example, if, in a given observation window, the input signal undergoes half as many cycles as the internal clock, the input signal must be changing at half the clock rate's frequency. When the observation window closes, these counters are latched externally so that they can be read by software, and zeroed internally when the counting begins anew. Note that the size of the observation window does not directly enter into the calculation.

As stated above, the duration of the observation window is adjustable under program control. The observation window does not officially close, however, until at least 1 cycle of the input signal is observed. This insures that the period count is always at least 1. If the input signal period is long enough, the tick count will overflow, invalidating the results for that window; more on this topic is discussed in the "Overflow" section.

All the operations described above–reading the tick and period counters, dividing them and multiplying by the clock rate, etc–are handled by the *Plug and Play* software drivers provided with the P635. The explanation provided above is only meant to describe the P635's theory of operation.

Count Acquisition

Figure 3-2 shows a typical timing sequence for one observation window. In this example the clock is set to 10 MHz, the observation window is set to 10 milliseconds and the input signal being measured is 490 Hz. The input period counter and the tick counter are simultaneously

enabled on the first low-to-high transition of the input signal after the observation window edge, and counting of the input signal and clock ticks commence. Counting stops at the first low-to-high transition of the input signal after the next observation window edge. The period count = 5, and the tick count = 102,040. The frequency calculation is:

$$freq = 10{,}000{,}000 \times \frac{5}{102{,}040} = 490.0039 Hz$$

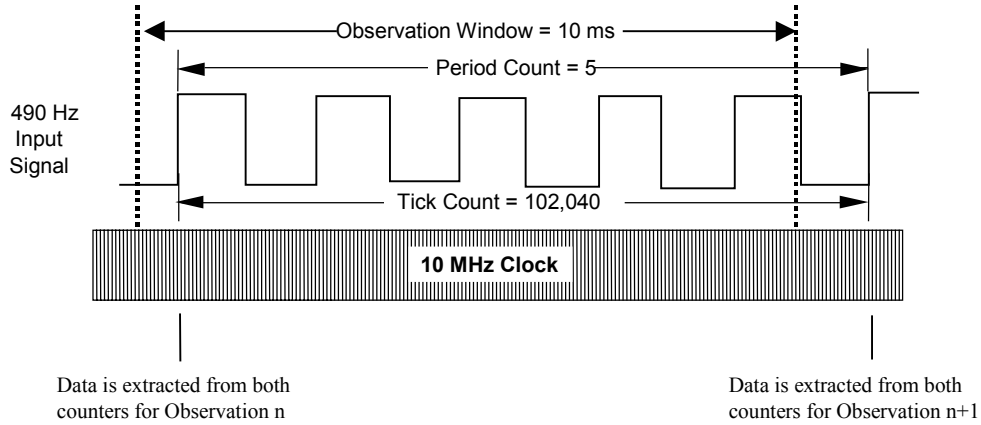


Figure 3-2. Timing Sequence for a 490 Hz Input Signal with 10 msec Observation Window

In the previous example the actual observation period approximated the observation window period. Figure 3-3 shows another timing example. In this case the input frequency is 20 Hz, with the observation window remaining at 10 milliseconds and the clock at 10 MHz. Since the actual observation period starts and ends at the first input low-to-high transition after a window edge, the counting spans multiple observation window periods. For this example the input period = 1, and the tick count = 500,000. The frequency calculation is:

$$freq = 10{,}000{,}000 \times \frac{1}{500{,}000} = 20 Hz$$

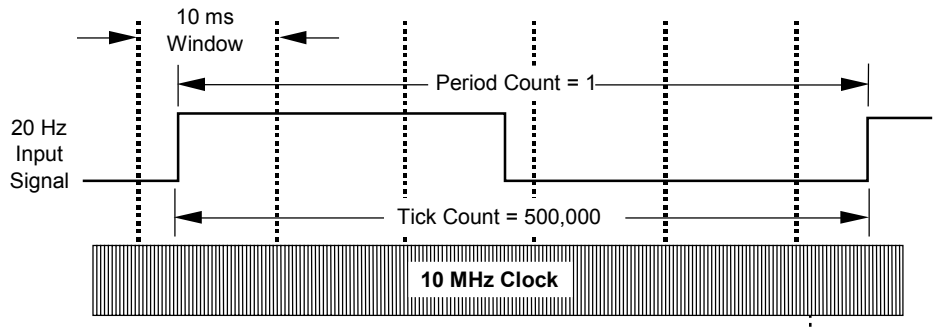


Figure 3-3. Timing Sequence for 20 Hz Input Signal and 10 msec Observation Window

Counting Accuracy

The clock rate for the module is programmable to provide a tick rate of 1 MHz or 10 MHz with a clock accuracy of ±1 ppm or ±0.0001%. The counting accuracy depends on the time base accuracy as well as the monitoring resolution. The longer the observation window, the higher the accuracy. The equation that expresses frequency could be rewritten to account for accuracy as:

$$frequency = clock\ rate \times \frac{period\ count}{tick\ count \pm 1}$$

In general, if the observation window were increased by some factor *X* (say, from 10 ms to 100 ms, for a factor of 10), then the *period count* and *tick count* would also increase by this factor *X*, but the error term +/-1 would remain the same, reducing its impact by a factor of *X*.

Excluding any inaccuracy (jitter) in the signal itself, Table 3-1 indicates the accuracy obtained for various clock rates and observation periods.

Observation Period (ms)	Accuracy (%) (1 MHz Clock)	Accuracy (%) (10 MHz Clock)
1	0.2	0.02
2	0.1	0.01
5	0.04	0.004
10	0.02	0.002
20	0.01	0.001
50	0.004	0.0007
100	0.002	0.0004
200	0.001	0.0003
500	0.0007	0.0003
1000	0.0004	0.0002

Table 3-1. Accuracy for Various Observation Periods and Internal Clock Rates

Overflow

The observation window remains open at least until the input signal completes 1 full cycle; while the observation window remains open, the Tick Counter is incrementing at the rate of the internal clock rate. Given that the Tick Counter has only 24 bits, if the input signal is slow enough the Tick Counter will overflow before the input signal completes 1 cycle. This condition is called 'OverFlow', and limits the minimum frequency that can be measured.

$$\frac{10{,}000{,}000}{2^{24}} \approx 0.6Hz$$

Minimum Frequency for 10,000,000Hz Clock

$$\frac{1{,}000{,}000}{2^{24}} \approx 0.06Hz$$

Minimum Frequency for 1,000,000Hz Clock

When Overflow occurs, the measured frequency is reported as 0 Hz. The Overflow status of a channel can be queried via *Plug and Plug* function `ks635_getCounterStatus` or read with the regular frequency data via `ks635_readFrequencyWithStatus`.

The P635 is also capable of generating a hardware interrupt on the event of a channel experiencing Overflow. *Plug and Plug* functions `ks635_setInterruptMask`, `ks635_setInterruptHandler`, and `ks635_setInterruptEnable` specify which channels are allowed to generate interrupts, install an interrupt handler routine, and enable the overall interrupt mechanism.

Overflow is not an error condition, and can occur in a system that is operating normally.

Stale Data

Instantaneous measurement updates of the current input frequency are not always available; a certain interval of time must elapse (either the observation window or 1 cycle of the input) before the results can be known. For long observation windows and/or slow input signals, it is possible that software will attempt to sample the input data faster than the data is being made available. This is perfectly acceptable; all resampled data collected while the observation window is still open is termed *stale data*. The classification of *stale* merely indicates that the measurement available from the hardware has not changed since the last time it was collected; it still represents the latest data available. When the observation window closes and new data becomes available, that sample will not be stale.

The Stale status of a channel can be queried via *Plug and Plug* function `ks635_getCounterStatus` or read with the regular frequency data via `ks635_readFrequencyWithStatus`.

Stale is not an error condition, and can occur in a system that is operating normally.

TickClock Synchronization

When a system is equipped with multiple P635 modules, it may be desirable to *synchronize* them by driving them off the same master clock source. By default, each P635 generates its own master 10 MHz clock, which drives the Tick Clock directly at ether the 10 MHz rate, or at 1 MHz. It is possible, however, for 1 P635 to feed its Tick Clock to other P635s via the PXI trigger lines so that they will all be running synchronized. This is accomplished by *Plug and Play* function calls `ks635_setTickClockToTriggerLineOut` and `ks635_setTickClockSource`. The former call would be made by the session associated with the Tick Clock Source P635 (the master), and the latter by all sessions associated with P635s wishing to synchronize their Tick Clocks with the source (the slaves). Responsibility for coordinating which P635 serves as the master and which are the slaves, as well as which PXI trigger line will be used to route the clock between modules, is left to the application; no checks are made to insure that only 1 master is chosen, or that all participating modules are connected to the same trigger line.

It is also possible for a P635 to take its timing from the PXI 10 MHz backplane clock; all P635s running in this mode will be synchronized. The PXI 10 MHz backplane clock, however, does not offer the same precision and stability as that generated by a P635, so this option would only be used if no trigger line were available.

Tick Clock synchronization might be thought of as external synchronization: all P635s are being driven by the same master clock. Internally, however, counters amongst the various P635s are incrementing asynchronously. The next section explains how to perform internal synchronization.

WindowClock Synchronization

Window Clock synchronization is similar to Tick Clock synchronization, except that the Window Clock is routed between P635s. As in Tick Clock synchronization, 1 module is chosen as the master, and all others as the slaves. *Plug and Play* functions `ks635_setWindowClockToTriggerLineOut` and `ks635_setWindowClockSource` are used to route the master window clock out and to route the window clock in on the slaves. As with Tick Clock synchronization, the onus is on the application to designate the master and slaves, and to route them all on the correct trigger line.

The observation window size of a module which is taking its Window Clock source from another module is taken from the observation window size of the source module; its own observation window size is ignored (q.v. `ksp635_setObservationWindowSize`).

Applying Tick or Window Clock synchronization (but not both) on a system may be desirable for certain applications; for a fully synchronized system, however, both Tick Clock and Window Clock synchronization should be used. 1 module should be designated as the source for both Tick and Window Clock, 1 trigger line designated for each of these 2 signals, and all the other modules serving as slaves. This is the recommended configuration for a system containing multiple P635s when synchronization is required.

Input Paths

Each of the 8 channels can monitor either analog differential input signals or TTL signals. The SCSI connector has different pins for the 2 different types for each channel. In addition to wiring the input signal to the appropriate pins on the connector, the P635 must also be programmed via the *Plug and Play* function `ks635_setChannelInputType` to the correct type.

Analog Input Paths

The analog input path associated with each channel contains a differential input for high noise immunity. Each path has the switch-selectable option of unterminated or terminated. For most applications, the unterminated selection (the factory-set default) will be chosen. A 100Ω termination can also be selected for each of these paths for use with RS-422, RS-485 or similar differential transmitters. If this module is the only (or last) receiver node on an RS-422 or an RS-485 path, the 100Ω termination should be switched in.

The analog input circuit is shown in Figure 3-4. The input circuit consists of a 47kΩ series resistor in each leg, diode clamps to ±12 volts and a pair of 1 MΩ resistors to ground. Also, AC coupling can be program selected by adding a pair of 1 microfarad capacitors in series with the inputs. This circuit drives a programmable instrumentation amplifier. The diode clamps protect the input circuit and "clip" the input signal peaks with an absolute value greater than 10 volts. The 1 MΩ input impedance decreases to 47 kΩ for voltages above ±12 volts. A 100 Ω resistor can provide termination for RS-422 or RS-485 differential paths. All of the analog inputs can be disconnected from the input connector and connected to a common "health check" bus. A known frequency source can be connected to the health check input to verify module operation.

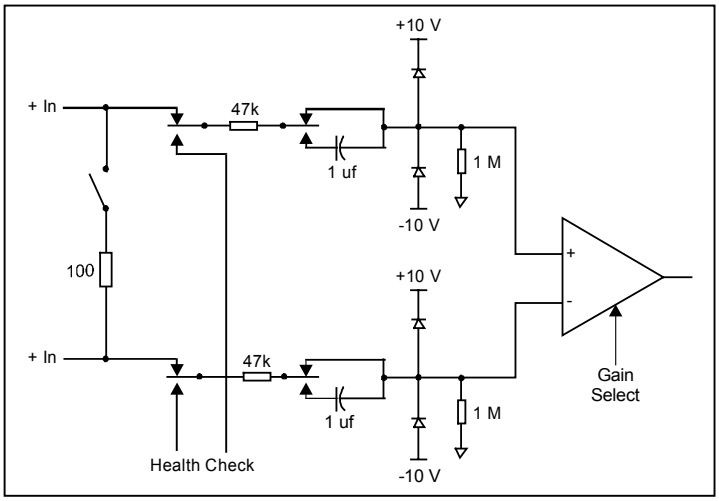


Figure 3-4. Detailed View of Analog Differential Input Circuit

Table 3-2 shows the recommended input ranges and associated input pulse detection thresholds for the P635. To provide a safety margin, the actual Pulse detection threshold is approximately 30% ± 5% of the input range minimums shown in the table (±50 to ±70 mV for the ±200 mV to

20 V range, for example). The input range from this table should be used as a guide for setting the pulse detection threshold. A threshold value higher than needed will result in a lower pulse detection threshold and poorer noise immunity. A threshold that is too low could cause input pulses to be missed. If a P635 analog channel is used as an RS-422 or RS-485 receiver, the threshold should be selected to the 200 mV to 20 V range. The threshold is selectable on a per-channel basis.

NOTE: The specified input ranges are at a nominal 1 kHz frequency. At higher frequencies the threshold should be increased. For example, at 100 kHz a threshold of 12mV is recommended for a ±200 mV analog input signal.

Gain	P635-AAyz Input Range	P635-AAyz Threshold	P635-AByz Input Range	P635-AByz Threshold
1	±200 mV to 20 V	±50 to ±70 mV	±1 V to 20 V	±250 to ±350 mV
2	±100 mV to 20 V	±25 to ±35 mV	±500 mV to 20 V	±125 to ±175 mV
5	±40 mV to 20 V	±10 to±14 mV	±200 mV to 20 V	±50 to ± 70 mV
10	±20 mV to 20 V	±5 to ±7 mV	±100 mV to 20 V	±25 to ±35 mV

Table 3-2. Recommended Input Ranges and Pulse Detection Thresholds

For maximum noise immunity, the instrumentation amplifier drives an operational amplifier that is configured for differential hysteresis: once the signal reaches the positive pulse detection threshold (+60 mV, for example), it must reach the negative pulse detection threshold (-60 mV, for example) before the output will switch again. This is shown in Figure 3-5. Note that the higher voltage threshold in (a) causes the circuit to ignore the unwanted zero-crossing, while the lower threshold voltage in (b) produces spurious results. Therefore, KineticSystems recommends the highest threshold setting that gives sufficient voltage margin. The recommended settings for various voltage ranges are shown in Table 3-2.

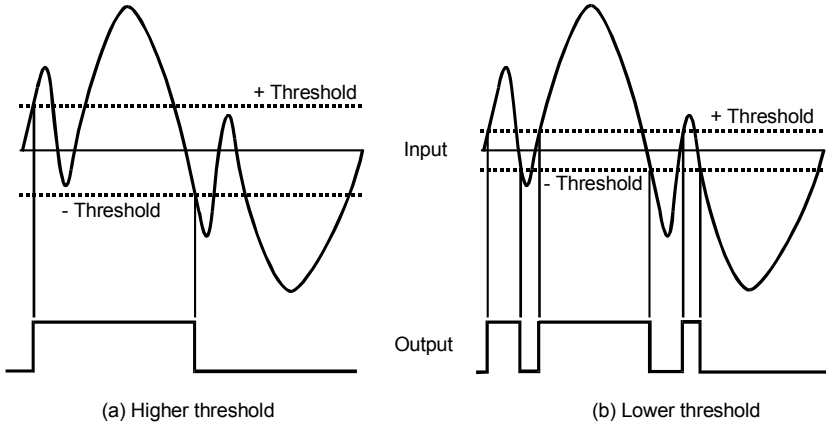


Figure 3-5. Effect of Threshold Voltage Setting with Signal Having Multiple Transitions

At high frequencies, there is an increase in the effective threshold. This is shown in Table 3-3.

Frequency	Effective Threshold Multiplier
100 kHz	5.33
75 kHz	3.83
50 kHz	3.53
20 kHz	1.42
10 kHz	1.18
5 kHz	1.00

Table 3-3 Decrease in Input Sensitivity at High Frequencies

AC Input Considerations

DC coupling is appropriate for most applications. However, if the input signal on a channel has a sufficient DC offset where there is little or no zero crossing margin, then that channel should be programmed for AC coupling. With AC coupling the input signal will "float" to a value to provide an equal integrated voltage verses time above and below zero. It will not be centered around zero if the signal is very asymmetric. Also, AC coupling increases the effective threshold voltage at very low frequencies. This is shown in Table 3-4. For example, the actual threshold voltage will be about 128% of the DC threshold voltage at 0.2 Hz for input signals approximating a sine wave.

Frequency (Hz)	Effective Threshold Multiplier
0.05	3.34
0.1	1.88
0.2	1.28
0.5	1.05
1.0	1.01
>1.0	1.00

Table 3-4. Decrease in Input Sensitivity at Very Low Frequencies with AC Coupling

Channel coupling settings are programmed via the *Plug and Play* function `ks635_setChannelCoupling`.

Filtering

A single-pole low-pass RC filter can be added to a channel path; the filter provides a -3dB cutoff frequency of 50 kHz. Filter enable/disable operations are performed via *Plug and Plug* function `ks635_setChannelFilterEnable`.

TTL Input Paths

The P635 also monitors standard digital TTL level signals. By default, input signals are assumed to be analog differential; monitoring TTL signals requires a call to the *Plug and Play* function `ks635_setChannelInputType`.

External Wiring Considerations

The recommended external wiring connections for one channel are shown in Figure 3-6. The connections from a typical ground-connected transducer to a differential analog input are shown in 3-6(a), while connections from an RS-422 or RS-485 transmitter are shown in 3-6(b). Note that the shield is connected to ground at both ends. This reduces noise pickup and does not create a "bad" ground loop because shield is not a signal-carrying conductor. For more information on ground loops, refer to the VXI Data Acquisition Handbook, by J. W. Tippie, found on the Internet at http://www.kscorp.com/www/tech_rep/pdf/prodgd.pdf.

There can be one cable shield per channel or a single overall shield for the entire cable.

Figure 3-6(c) shows the recommended connections for a TTL input. Note that each TTL input is terminated with a 220Ω/330Ω termination within the P635. This requires a minimum of 23mA current sinking capability on the external driver.

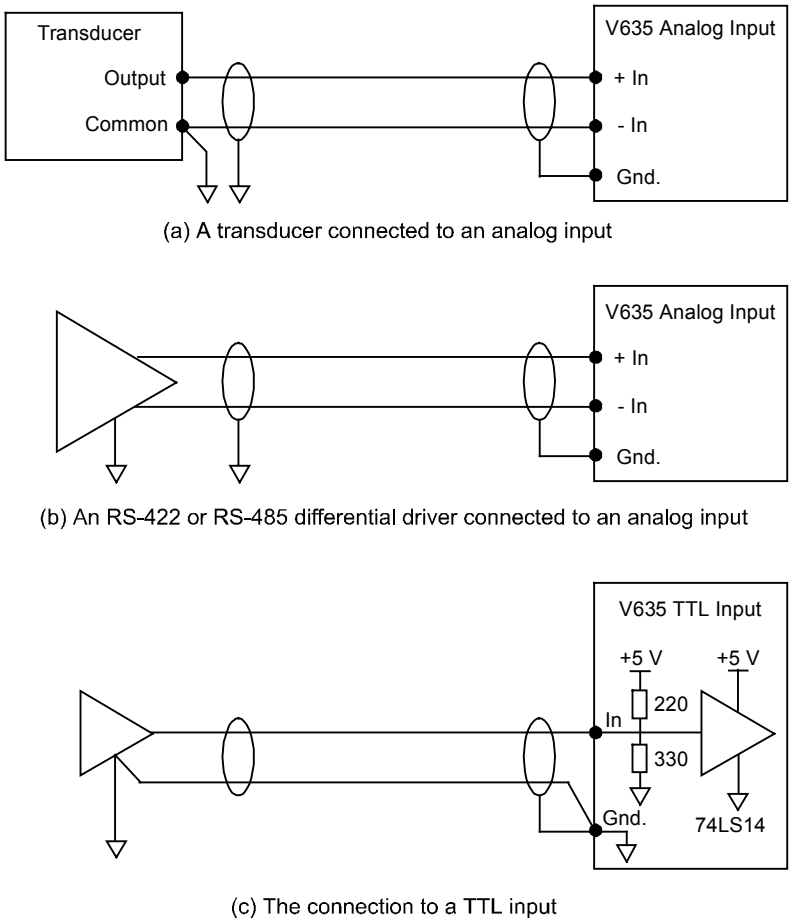


Figure 3-6. Recommended Connections for Analog and TTL Channel Input Paths

Health Check

The Health Check functionality of the P635 allows, under program control, user-provided differential or TTL precision frequency sources to replace regular channel inputs. The precision sources are connected to the Analog/TTL Health pins on the SCSI connector. The Health Check mode can either be disabled (normal operation), differential mode, or TTL mode. When differential mode is enabled, Health Check disconnects all channels (irregardless of their Channel Path settings) from their regular pins on the SCSI connector and connects them to the Analog Health +In and Analog Health -In pins; likewise, when TTL Health Check is enabled, all channels (irregardless of their Channel Path settings) are disconnected from their regular pins and connected to the TTL Health In and TTL Health Ground pins. Data collected on any channel reflects the (presumably known) precision input source, and can be used to verify channel setup and configuration. Once verified, Health Check should be disabled and regular operation resumes.

Health Check is enabled/disabled via *Plug and Play* function call `ks635_setHealthCheck`.

Chapter 4: Programming

Building an application

To build an application using the P635 Plug & Play driver, include the ks635.h header file located in the *<install_dir>\include* folder. The application must also link to a ksp635_32.lib library file appropriate for the compiler being used. The Microsoft Visual Studio lib is located in the *<install_dir>\lib\Msc* folder. The National instruments CVI lib is located in the *<install_dir>\lib\cvi* folder.

Software Driver Functions

The following is a list of software functions provided with the P635 *Plug and Play* driver.

Required Instrument Driver Functions	
Initialize	ks635_init
Reset	ks635_reset
Self Test	ks635_self_test
Error Query	ks635_error_query
Error Message	ks635_error_message
Revision Query	ks635_revision_query
Close	ks635_close

Optional Instrument Driver Functions	
Auto Connect To First	ks635_autoConnectToFirst
Auto Connect To All	ks635_autoConnectToAll

Configuration Functions	
Set Observation Window Size	ks635_setObservationWindowSize
Get Observation Window Size	ks635_getObservationWindowSize
Set Tick Clock Rate	ks635_setTickClockRate
Get Tick Clock Rate	ks635_getTickClockRate
Set Window Clock Source	ks635_setWindowClockSource
Get Window Clock Source	ks635_getWindowClockSource
Set Tick Clock Source	ks635_setTickClockSource
Get Tick Clock Source	ks635_getTickClockSource
Set Window Clock To Trigger Line Out	ks635_setWindowClockToTriggerLineOut
Get Window Clock To Trigger Line Out	ks635_getWindowClockToTriggerLineOut
Set Tick Clock To Trigger Line Out	ks635_setTickClockToTriggerLineOut
Get Tick Clock To Trigger Line Out	ks635_getTickClockToTriggerLineOut
Set Channel Switching Threshold	ks635_setChannelThreshold
Get Channel Switching Threshold	ks635_getChannelThreshold
Set Channel Filter Enable	ks635_setChannelFilterEnable
Get Channel Filter Enable	ks635_getChannelFilterEnable
Set Channel Coupling	ks635_setChannelCoupling
Get Channel Coupling	ks635_getChannelCoupling

Set Channel Input Type	ks635_setChannelInputType
Get Channel Input Type	ks635_getChannelInputType
Set Interrupt Enable	ks635_setInterruptEnable
Get Interrupt Enable	ks635_getInterruptEnable
Set Interrupt Level	ks635_setInterruptLevel
Get Interrupt Level	ks635_getInterruptLevel
Set Interrupt Mask	ks635_setInterruptMask
Get Interrupt Mask	ks635_getInterruptMask
Get Number of Channels	ks635_getNumChannels

Action and Status Functions	
Set Continuous Scan Enable	ks635_setContinuousScanEnable
Get Continuous Scan Enable	Ks635_getContinuousScanEnable
Set Health Check Enable	ks635_setHealthCheck
Get Health Check Enable	ks635_getHealthCheck
Get Counter Status	ks635_getCounterStatus
Clear Count Status	ks635_clearCountStatus

Data Functions	
Execute Single Scan	ks635_executeSingleScan
Read Frequency	ks635_readFrequency
Read Frequency With Status	ks635_readFrequencyWithStatus
Read Single Channel Frequency	ks635_readSingleChannelFrequency
Read Counters	ks635_readCounters
Read Single Channel Counters	ks635_readSingleChannelCounters

Initialization Functions

ks635_init

Syntax

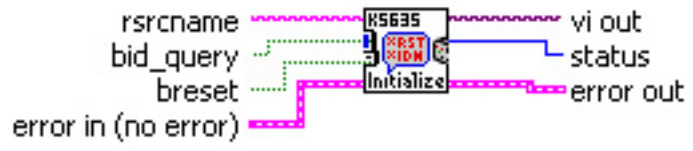
C:

```
ViStatus ks635_init (ViChar rsrcName[],
                     ViBoolean bId_query,
                     ViBoolean bReset,
                     ViPSession vi);
```

Visual Basic:

```
Function ks635_init As ViStatus (rsrcName As ViChar,
                                 ByVal bId_query As ViBoolean,
                                 ByVal bReset As ViBoolean,
                                 vi As ViSession)
```

LabVIEW:



ks635 Init.vi

Description

`ks635_init` establishes communication with a P635 and optionally resets and queries the module. Once these operations are complete, an instrument handle is returned for subsequent communication with the module. For each instrument handle returned via `ks635_init`, a corresponding `ks635_close` should be executed prior to termination of the application program.

Parameters

rsrcName specifies the name of the resource to be initialized. The format of the resource name string is as follows:

Description	Grammar
Bus/Device/Function string	PXI[*interface*]::*bus-device*[.*function*][::INSTR]
Bus/Device/Function legacy string	PXI[*bus*]::*device*[::*function*][::INSTR]

bId_query is set to either VI_TRUE or VI_FALSE. When set to VI_TRUE, a query of the hardware is executed to verify that the instrument handle is accessing a P635; if the specified resource is not a P635, no session will be opened and failure is returned. When set to VI_FALSE, the query is not performed.

bReset is set to either VI_TRUE or VI_FALSE. When set to VI_TRUE, a hardware reset operation is performed to the P635 which restores the device to its default configuration. If set to VI_FALSE, the reset operation is not performed. *(refer **to ks635_reset** function description for default device configuration)*

vi the returned instrument handle. This handle is used for subsequent communication with the device.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_SUCCESS_DEV_NPRESENT	Success, but device is not responding
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
VI_ERROR_RSRC_NFOUND	Resource Not Found
VI_ERROR_FAIL_ID_QUERY	Device Query Failed
VI_ERROR_ALLOC	Error Allocating Memory

ks635_reset

Syntax

C:

```
ViStatus ks635_reset (ViSession vi);
```

Visual Basic:

```
Function ks635_reset As ViStatus (ByVal vi As ViSession)
```

LabVIEW:

Description

`ks635_reset` resets the P635 hardware, returning it to its powerup default state:

- Observation Window size: 1ms
- Tick Clock Rate: 10 MHz
- Window Clock Source: Internal
- Tick Clock Source: Internal
- Window Clock Output: Disabled
- Tick Clock Output: Disabled
- Channel Threshold: 200 mv
- Channel Filter: Disabled
- Channel Coupling: DC Coupling
- Channel Input Type: Analog Differential
- Interrupts: Disabled
- Continuous Scan: Disabled
- Health Check: Disabled

Parameters

vi unique logical identifier to a session

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

ks635_self_test

Syntax

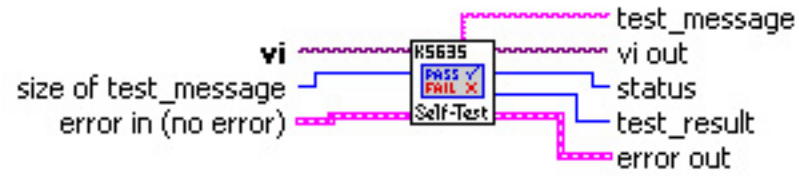
C:

```
ViStatus  ks635_self_test (ViSession vi,
                           ViInt16 *test_result,
                           ViChar test_message[]);
```

Visual Basic:

```
Function ks635_self_test As ViStatus (ByVal vi As ViSession,
                                       ByRef test_result As ViInt16,
                                       ByRef test_message As ViChar)
```

LabVIEW:



ks635 Self Test.vi

Description

`ks635_self_test` performs self test on the session. A numeric test result and the associated test string message are returned.

NOTE: The P635 can not perform self-test. This function is included because it is part of the *Plug and Play* specification; since it is not supported, it must return VI_WARN_NSUP_SELF_TEST.

Parameters

vi unique logical identifier to a session

test_result code indicating result of test

test_message message indicating result of test

Return Values

Return Value	Description
VI_WARN_NSUP_SELF_TEST	Not supported

ks635_error_query

Syntax

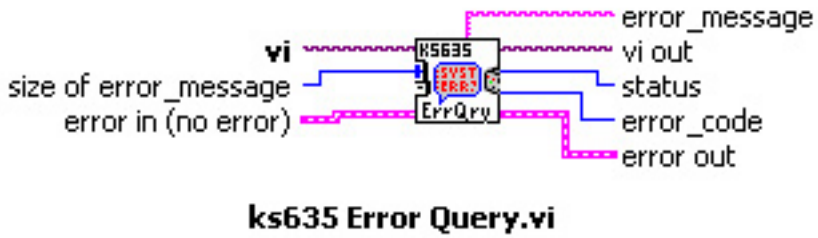
C:

```
ViStatus ks635_error_query (ViSession vi,
                            ViInt16 *error_code,
                            ViChar error_message[]);
```

Visual Basic:

```
Function ks635_error_query As ViStatus (ByVal vi As ViSession,
                                        ByRef error_code As ViInt16,
                                        ByRef error_message As ViChar)
```

LabVIEW:



ks635 Error Query.vi

Description

`ks635_error_query` queries the instrument and returns instrument-specific error information.

NOTE: The P635 can not perform error-query. This function is included because it is part of the *Plug and Play* specification; since it is not supported, it must return VI_WARN_NSUP_ERROR_QUERY.

Parameters

vi unique logical identifier to a session

error_code code indicating last error encountered

error_message description of error

Return Values

Return Value	Description
VI_WARN_NSUP_ERROR_QUERY	Not supported

ks635_error_message

Syntax

C:

```
ViStatus ks635_error_message (ViSession vi,
                              ViStatus error_code,
                              ViChar error_message[])
```

Visual Basic:

```
Function ks635_error_message As ViStatus (ByVal vi As ViSession,
                                          ByVal error_code As ViStatus,
                                          ByRef error_message As ViChar)
```

LabVIEW:

Description

`ks635_error_message` takes an error_code–presumably the result of some other *Plug and Play* function call–and translates it to a text message.

Parameters

vi unique logical identifier to a session

error_code code to be translated.

error_message description of error

Return Values

Return Value	Description
VI_SUCCESS	Success
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_WARN_UNKNOWN_STATUS	Unknown error_code

ks635_revision_query

Syntax

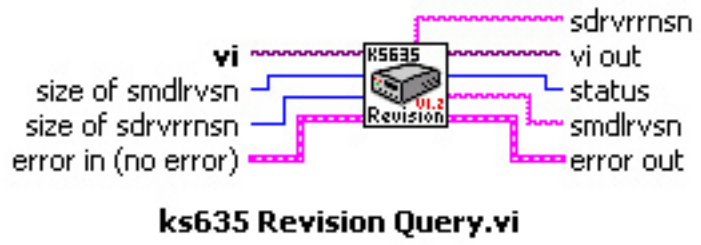
C:

```
ViStatus ks635_revision_query (ViSession vi,
                               ViChar sMdlRvsn[],
                               ViChar sDrvrRvsn[]);
```

Visual Basic:

```
Function ks635_revision_query As ViStatus (ByVal vi As ViSession,
                                           ByRef sMdlRvsn As ViChar,
                                           ByRef sDrvrRvsn As Char)
```

LabVIEW:



ks635 Revision Query.vi

Description

`ks635_revision_query` returns the current hardware and driver revision numbers.

Parameters

vi unique logical identifier to a session.

sMdlRvsn character array allocated by the caller and populated by this function with the current revision number of the P635 connected to this session. This array should be at least 16 bytes long.

sDrvrRvsn character array allocated by the caller and populated by this function with the current revision number of this Plug & Play driver. This array should be at least 16 bytes long.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_WARN_NSUP_REV_QUERY	Instrument revision (sMdlRvsn) not available

ks635_close

Syntax

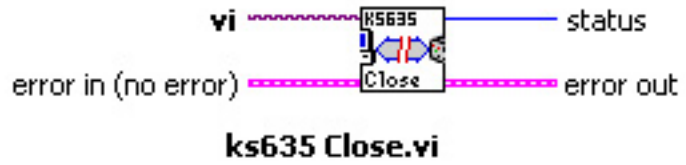
C:

```
ViStatus ks635_close (ViSession vi);
```

Visual Basic:

```
Function ks635_close As ViStatus (ByVal vi As ViSession)
```

LabVIEW:



Description

`ks635_close` closes out the VISA session and frees any associated allocated memory.

Parameters

vi unique logical identifier to the session to be closed.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

Optional Instrument Driver Functions

ks635_autoConnectToFirst

Syntax

C:

```
ViStatus ks635_autoConnectToFirst (ViSession *vi);
```

Visual Basic:

```
Function ks635_autoConnectToFirst As ViStatus (ByRef vi As ViSession)
```

LabVIEW:

Description

`ks635_autoConnectToFirst` returns a session to the first P635 module as found via VISA's `viFindRsrc` function. This function may be used in conjunction with `ks635_autoConnectToNext`.

Parameters

vi the returned instrument handle. This handle is used for subsequent communication with the device.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_RSRC_NFOUND	No more P635 modules found
VI_ERROR_ALLOC	Error Allocating Memory

ks635_autoConnectToAll

Syntax

C:

```
ViStatus ks635_autoConnectToAll (ViSession vi[],
                                 ViInt16 iArrayLength,
                                 ViInt16 *iNumFound);
```

Visual Basic:

```
Function ks635_autoConnectToAll As ViStatus (ByRef vi As ViSession,
                                             ByVal iArrayLength As ViInt16,
                                             ByRef iNumFound As ViInt16)
```

LabVIEW:

N/A

Description

`ks635_autoConnectToAll` returns a list of sessions to each P635 module found via VISA's `viFindRsrc`/`viFindNext` functions.

Parameters

vi an array of ViSessions allocated by the user, and populated by this function.

iArrayLength the allocated size of the **vi** argument. If there are more P635 modules than specified by this argument, only this many will be returned.

iNumFound the number of P635 module session pointers returned in argument **vi**.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_MORE_INST_PRESENT	Instruments were found that could not communicate

Configuration Functions

ks635_setObservationWindowSize

Syntax

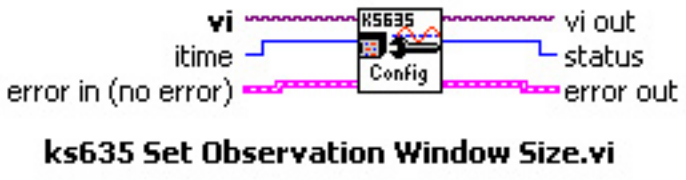
C:

```
ViStatus ks635_setObservationWindowSize (ViSession vi,
                                         ViInt16 iTime);
```

Visual Basic:

```
Function ks635_setObservationWindowSize As ViStatus (ByVal vi As ViSession,
                                                     iTime As ViInt16)
```

LabVIEW:



ks635 Set Observation Window Size.vi

Description

`ks635_setObservationWindowSize` sets the size of the observation window. The observation window is the amount of time that the P635 allows for a sampling interval. See Chapter 3, "Basic Circuit Operation", for details on how to configure the Observation Window.

Parameters

vi unique logical identifier to a session.

iTime is the observation window size, in milliseconds. The range is from 1 to 1024 milliseconds.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_WINDOW	Observation Window Specification (iTime) Out Of Range

ks635_getObservationWindowSize

Syntax

C:

```
ViStatus ks635_getObservationWindowSize (ViSession vi,
                                         ViInt16 *iTime);
```

Visual Basic:

```
Function ks635_getObservationWindowSize As ViStatus (ByVal vi As ViSession,
                                                     iTime As ViInt16)
```

LabVIEW:

ks635 Get Observation Window Size.vi

Description

`ks635_getObservationWindowSize` returns the current size of the observation window.

Parameters

vi unique logical identifier to a session.

iTime pointer to a 16 bit integer to be populated with the current Observation Window size of the P635 associated with this session.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

ks635_setTickClockRate

Syntax

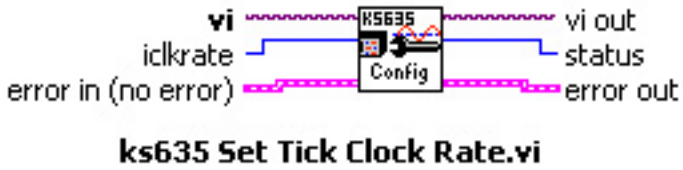
C:

```
ViStatus ks635_setTickClockRate (ViSession vi, ViInt16 iClkRate);
```

Visual Basic:

```
Function ks635_setTickClockRate As ViStatus (ByVal vi As ViSession,
                                             ByVal iClkRate As ViInt16)
```

LabVIEW:



ks635 Set Tick Clock Rate.vi

Description

`ks635_setTickClockRate` sets the frequency of the tick clock. The tick clock is used as a measurement standard for determining the frequency of the signal on a channel. See Chapter 3, "Basic Circuit Operation", for details on how to configure the tick clock.

This function must be called when an external tick clock source is used. Even though the module itself is not utilizing its own internal tick clock in external mode, the driver must be informed as to the rate at which the source tick clock is operating. See Chapter 3, "Tick Clock Synchronization", for details.

Parameters

vi unique logical identifier to a session.

iClkRate can be either KSP635_10MHZ_TICK_CLOCK for the clock to run at 10 MHz, or KS635_1MHZ_TICK_CLOCK for the clock to run at 1 MHz.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_PARAMETER2	Invalid iClkRate value

ks635_getTickClockRate

Syntax

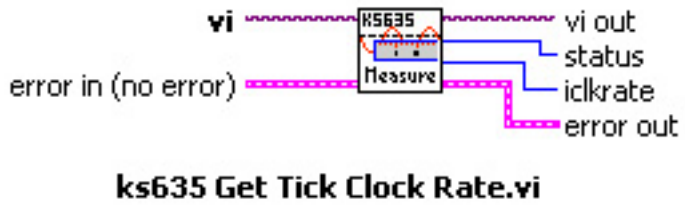
C:

```
ViStatus  ks635_getTickClockRate (ViSession vi, ViInt16 *iClkRate);
```

Visual Basic:

```
Function ks635_setTickClockRate As ViStatus (ByVal vi As ViSession,
                                              iClkRate As ViInt16)
```

LabVIEW:



ks635 Get Tick Clock Rate.vi

Description

`ks635_getTickClockRate` returns the current frequency of the tick clock. The tick clock is used as a measurement standard for determining the frequency of the signal on a channel. See Chapter 3, "Basic Circuit Operation", for details on configuring the tick clock.

Parameters

vi unique logical identifier to a session.

iClkRate returns KS635_10MHZ_TICK_CLOCK if the tick clock is at 10 MHz and KS635_1M_TICK_CLOCK for the 1 MHz tick clock.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

ks635_setWindowClockSource

Syntax

C:

```
ViStatus  ks635_setWindowClockSource (ViSession vi, ViInt16 iWclkSrc);
```

Visual Basic:

```
Function ks635_setWindowClockSource As ViStatus (ByVal vi As ViSession,
                                                  ByVal iWclkSrc As ViInt16)
```

LabVIEW:

ks635 Set Window Clock Source.vi

Description

`ks635_setWindowClockSource` specifies the source of the window clock. The window clock can either be generated internally or received from a PXI trigger line. See Chapter 3, "Window Clock Synchronization", for details on how to configure the Window Clock.

Parameters

vi unique logical identifier to a session.

iWclkSrc specifies the source of the Window Clock. The Window Clock can be generated internally or received from a PXI trigger line. The following table shows the index values and the corresponding source of the Window Clock.

Window Clock Source	IWclkSrc Constant
Internal	KS635_INTERNAL_WINDOW
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
VI_ERROR_PARAMETER2	Invalid setting for iWclkSrc

ks635_getWindowClockSource

Syntax

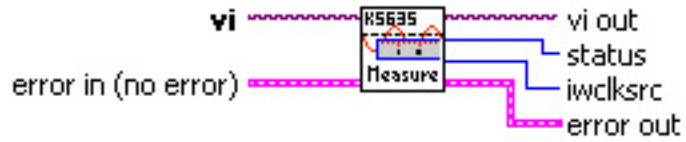
C:

```
ViStatus ks635_getWindowClockSource (ViSession vi, ViInt16 *iWclkSrc);
```

Visual Basic:

```
Function ks635_getWindowClockSource As ViStatus (ByVal vi As ViSession,
                                                  iWclkSrc As ViInt16)
```

LabVIEW:



ks635 Get Window Clock Source.vi

Description

`ks635_getWindowClockSource` returns the current source of the window clock. The window clock can either be generated internally or received from a PXI trigger line. See Chapter 3, "Window Clock Synchronization", for details on how to configure the Window Clock.

Parameters

vi unique logical identifier to a session.

iWclkSrc returned value indicating the source of the Window Clock. The following table shows the index values and the corresponding source of the Window Clock:

Window Clock Source	IWclkSrc Constant
Internal	KS635_INTERNAL_WINDOW
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_setTickClockSource

Syntax

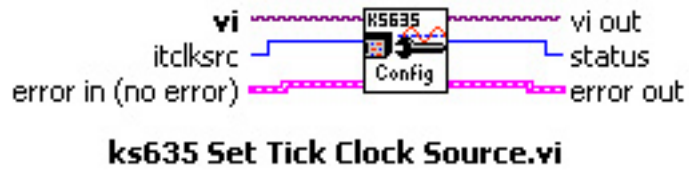
C:

```
ViStatus ks635_setTickClockSource (ViSession vi, ViInt16 iTclkSrc);
```

Visual Basic:

```
Function ks635_setTickClockSource As ViStatus (ByVal vi As ViSession,
                                                ByVal iTclkSrc As ViInt16)
```

LabVIEW:



ks635 Set Tick Clock Source.vi

Description

`ks635_setTickClockSource` specifies the source of the tick clock. The tick clock can either be generated internally, received from a PXI trigger line or received from the PXI 10 MHz backplane clock. See Chapter 3, "Tick Clock Synchronization", for details on how to configure the Tick Clock.

Parameters

vi unique logical identifier to a session.

iTclkSrc specifies the source of the 10 MHz tick clock used as the frequency measurement standard. The following table shows the index values and the corresponding source of the 10 MHz clock.

Tick Clock Source	ITclkSrc Constant
Internal	KS635_INTERNAL_TICK
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR
PXI 10 MHz System Clock	KS635_10MHZ_SYSTEM_CLOCK

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_PARAMETER2	Invalid Tick Clock Source Selection

ks635_getTickClockSource

Syntax

C:

```
ViStatus ks635_getTickClockSource (ViSession vi, ViInt16 *iTclkSrc);
```

Visual Basic:

```
Function ks635_getTickClockSource As ViStatus (ByVal vi As ViSession,
                                                iTclkSrc As ViInt16)
```

LabVIEW:

ks635 Get Tick Clock Source.vi

Description

`ks635_getTickClockSource` returns the current source of the Tick Clock. The Tick Clock can either be generated internally, received from a PXI trigger line or received from the PXI 10 MHz system clock. See Chapter 3, "Tick Clock Synchronization", for details on how to configure the Tick Clock.

Parameters

vi unique logical identifier to a session.

iTclkSrc returned value indicating the source of the 10 MHz Tick Clock used as the frequency measurement standard. The following table shows the index values and the corresponding source of the 10 MHz clock:

Tick Clock Source	ITclkSrc Constant
Internal	KS635_INTERNAL_TICK
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR
PXI 10 MHz System Clock	KS635_10MHZ_SYSTEM_CLOCK

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

ks635_setWindowClockToTriggerLineOut

Syntax

C:

```
ViStatus ks635_setWindowClockToTriggerLineOut (ViSession vi,
                                               ViInt16 iWclkTrg);
```

Visual Basic:

```
Function ks635_setWindowClockToTriggerLineOut
                                  As ViStatus (ByVal vi As ViSession,
                                               ByVal iWclkTrg As ViInt16)
```

LabVIEW:

ks635 Set Window Clk To Trig Line Out.vi

Description

`ks635_setWindowClockToTriggerLineOut` connects the P635 Window Clock output to a PXI Trigger Line. This mode is used to chain multiple P635's together and drive them with the same Window Clock. See Chapter 3, "Window Clock Synchronization", for details on how to configure the Window Clock.

Parameters

vi unique logical identifier to a session.

iWclkTrg specifies which PXI trigger line to connect to the Window Clock output of the P635. The following table shows the index values and the corresponding connections to the trigger lines.

PXI Trigger Line Connection	IWclkTrg Constant
No Connection	KS635_TRIGGER_DISCONNECTED
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
VI_ERROR_PARAMETER2	Invalid Trigger Setup

ks635_getWindowClockToTriggerLineOut

Syntax

C:

```
ViStatus ks635_getWindowClockToTriggerLineOut (ViSession vi,
                                               ViInt16 *iWclkTrg);
```

Visual Basic:

```
Function ks635_getWindowClockToTriggerLineOut
                                        As ViStatus (ByVal vi As ViSession,
                                                     iWclkTrg As ViInt16)
```

LabVIEW:

ks635 Get Window Clk To Trig Line Out.vi

Description

`ks635_getWindowClockToTriggerLineOut` returns the PXI trigger line to which the Window Clock is connected. See Chapter 3, "Window Clock Synchronization", for details on how to configure the Window Clock.

Parameters

vi unique logical identifier to a session.

iWclkTrg returned value indicating the PXI trigger line connected to the Window Clock output of the P635. The following table shows the index values and the corresponding connections to the trigger lines.

PXI Trigger Line Connection	IWclkTrg Constant
No Connection	KS635_TRIGGER_DISCONNECTED
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_setTickClockToTriggerLineOut

Syntax

C:

```
ViStatus ks635_setTickClockToTriggerLineOut (ViSession vi,
                                             ViInt16 iTclkTrg);
```

Visual Basic:

```
Function ks635_setTickClockToTriggerLineOut As ViStatus
                                            (ByVal vi As ViSession,
                                             iTclkTrg As ViInt16)
```

LabVIEW:

ks635 Set Tick Clk To Trig Line Out.vi

Description

`ks635_setTickClockToTriggerLineOut` connects the P635 Tick Clock output to a PXI Trigger Line. This mode is used to chain multiple P635's together and drive them with the same Tick Clock. See Chapter 3, "Tick Clock Synchronization", for details on how to configure the Tick Clock.

Parameters

vi unique logical identifier to a session.

iTclkTrg specifies which PXI trigger line to connect to the Window Clock output of the P635. The following table shows the index values and the corresponding connections to the trigger lines.

PXI Trigger Line Connection	ITclkTrg Constant
No Connection	KS635_TRIGGER_DISCONNECTED
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_INV_TRIG_SETUP	Invalid Trigger Setup

ks635_getTickClockToTriggerLineOut

Syntax

C:

```
ViStatus ks635_getTickClockToTriggerLineOut (ViSession vi,
                                             ViInt16 *iTclkTrg);
```

Visual Basic:

```
Function ks635_getTickClockToTriggerLineOut
                                    As ViStatus (ByVal vi As ViSession,
                                                 iTclkTrg As ViInt16)
```

LabVIEW:

ks635 Get Tick Clk To Trig Line Out.vi

Description

`ks635_getTickClockToTriggerLineOut` returns the PXI trigger line to which the Tick Clock output is currently connected. See Chapter 3, "Tick Clock Synchronization", for details on how to configure the Tick Clock.

Parameters

vi unique logical identifier to a session.

iTclkTrg returned value indicating the PXI trigger line connected to the Tick Clock output of the P635. The following table shows the index values and the corresponding connections to the trigger lines.

PXI Trigger Line Connection	iTclkTrg Constant
No Connection	KS635_TRIGGER_DISCONNECTED
PXI Trigger Line 0	KS635_TRIGGER_LINE0
PXI Trigger Line 1	KS635_TRIGGER_LINE1
PXI Trigger Line 2	KS635_TRIGGER_LINE2
PXI Trigger Line 3	KS635_TRIGGER_LINE3
PXI Trigger Line 4	KS635_TRIGGER_LINE4
PXI Trigger Line 5	KS635_TRIGGER_LINE5
PXI Trigger Line 6	KS635_TRIGGER_LINE6
PXI Trigger Line 7	KS635_TRIGGER_LINE7
PXI Star Trigger Line	KS635_TRIGGER_STAR

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_setChannelThreshold

Syntax

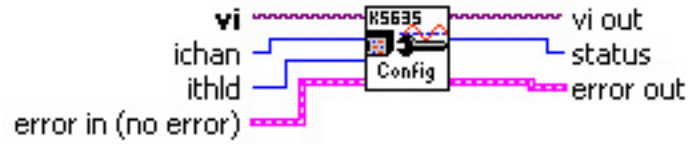
C:

```
ViStatus ks635_setChannelThreshold (ViSession vi,
                                    ViInt16 iChan,
                                    ViInt16 iThld);
```

Visual Basic:

```
Function ks635_setChannelThreshold As ViStatus (ByVal vi As ViSession,
                                                 ByVal iChan Av ViInt16,
                                                 ByVal iThld As ViInt16)
```

LabVIEW:



ks635 Set Channel Threshold.vi

Description

`ks635_setChannelThreshold` specifies the voltage switching threshold for the specified channel. The threshold can be configured for 20, 40, 100 or 200 millivolts. See Chapter 3, "Analog Input Paths", for details.

Parameters

vi unique logical identifier to a session.

iChan is the channel on which the threshold is to be altered. Channel numbers range from 1 to 8.

iThld specifies the switching threshold for the channel. The following table shows the available threshold selections for the given index.

Switching Threshold	IThld Constant
20 millivolts	KS635_THLDINDX_20
40 millivolts	KS635_THLDINDX_40
100 millivolts	KS635_THLDINDX_100
200 millivolts	KS635_THLDINDX_200

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_GAIN	Invalid Gain Index Selection
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_getChannelThreshold

Syntax

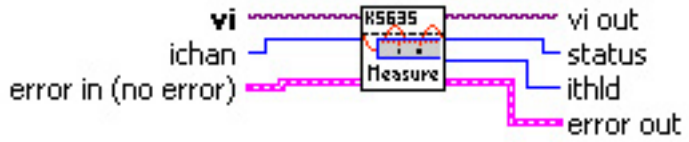
C:

```
ViStatus ks635_getChannelThreshold (ViSession vi,
                                    ViInt16 iChan,
                                    ViInt16 *iThld);
```

Visual Basic:

```
Function ks635_getChannelThreshold As ViStatus (ByVal vi As ViSession,
                                                ByVal iChan Av ViInt16,
                                                iThld As ViInt16)
```

LabVIEW:



ks635 Get Channel Threshold.vi

Description

`ks635_getChannelThreshold` gets the current voltage switching threshold for a given input channel. The threshold can be configured for 20, 40, 100 or 200 millivolts. See Chapter 3, "Analog Input Paths", for details.

Parameters

vi unique logical identifier to a session.

iChan is the channel on which the threshold is to be returned. Channel numbers range from 1 to 8.

iThld returned value indicating the current voltage threshold setting as shown in the following table.

Switching Threshold	iThld Constant
20 millivolts	KS635_THLDINDX_20
40 millivolts	KS635_THLDINDX_40
100 millivolts	KS635_THLDINDX_100
200 millivolts	KS635_THLDINDX_200

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_setChannelFilterEnable

Syntax

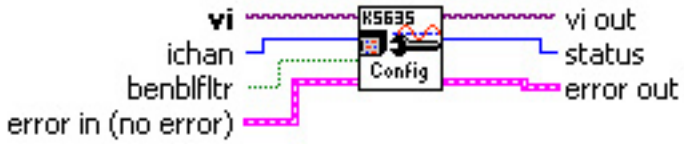
C:

```
ViStatus ks635_setChannelFilterEnable (ViSession vi,
                                       ViInt16 iChan,
                                       ViBoolean bFltEna);
```

Visual Basic:

```
Function ks635_setChannelFilterEnable As ViStatus(ByVal vi As ViSession,
                                                  ByVal iChan As ViInt16,
                                                  ByVal bFltEna As ViBoolean)
```

LabVIEW:



ks635 Set Channel Filter Enable.vi

Description

`ks635_setChannelFilterEnable` enables or disables a 50 kHz low pass filter on a channel's path. See Chapter 3, "Filtering", for details.

Parameters

vi unique logical identifier to a session.

iChan is the channel on which the filter is to be enabled or disabled. Channel numbers range from 1 to 8.

bFltEna set to VI_FALSE to remove the filter from the input channel or VI_TRUE to enable the filter.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified
VI_ERROR_PARAMETER3	Invalid Filter Enable Value Specified

ks635_getChannelFilterEnable

Syntax

C:

```
ViStatus ks635_getChannelFilterEnable (ViSession vi, ViInt16 iChan,
                                       ViBoolean *bFltEna);
```

Visual Basic:

```
Function ks635_getChannelFilterEnable As ViStatus (ByVal vi As ViSession,
                                                   ByVal iChan As ViInt16,
                                                   bFltEna As ViBoolean)
```

LabVIEW:

ks635 Get Channel Filter Enable.vi

Description

`ks635_getChannelFilterEnable` determines whether a filter is enabled or disabled on a channel. See Chapter 3, "Filtering", for details.

Parameters

vi unique logical identifier to a session.

iChan is the channel to be checked to see if the filter is enabled or disabled. Channel numbers range from 1 to 8.

bFltEna returned as VI_FALSE when the filter is disabled or VI_TRUE when the filter is enabled.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_setChannelCoupling

Syntax

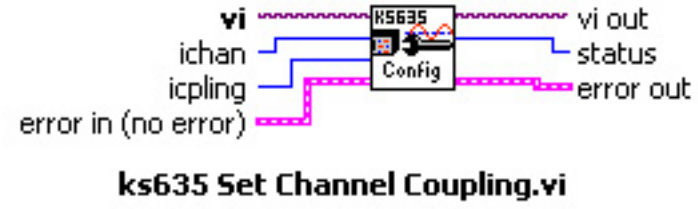
C:

```
ViStatus ks635_setChannelCoupling (ViSession vi,
                                   ViInt16 iChan,
                                   ViInt16 iCpling);
```

Visual Basic:

```
Function ks635_setChannelCoupling As ViStatus (ByVal vi As ViSession,
                                               ByVal iChan As ViInt16,
                                               ByVal iCpling As ViInt16)
```

LabVIEW:



ks635 Set Channel Coupling.vi

Description

`ks635_setChannelCoupling` specifies the manner in which the input signal is coupled in to the P635. The two selections include AC and DC coupling. See Chapter 3, "AC Input Considerations", for details concerning channel coupling.

Parameters

vi unique logical identifier to a session.

iChan is the channel on which the coupling it to be selected. Channel numbers range from 1 to 8.

iCpling set to KS635_DC_COUPLING to select DC coupling or KS635_AC_COUPLING for AC coupling.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified
VI_ERROR_PARAMETER3	Invalid iCpling Specified

ks635_getChannelCoupling

Syntax

C:

```
ViStatus ks635_getChannelCoupling (ViSession vi,
                                   ViInt16 iChan,
                                   ViInt16 *iCpling);
```

Visual Basic:

```
Function ks635_getChannelCoupling As ViStatus (ByVal vi As ViSession,
                                               ByVal iChan As ViInt16,
                                               iCpling As ViInt16)
```

LabVIEW:

ks635 Get Channel Coupling.vi

Description

`ks635_getChannelCoupling` determines if the input signal on a given P635 channel is AC or DC coupled. See Chapter 3, "AC Input Considerations", for details.

Parameters

vi unique logical identifier to a session.

iChan is the channel to be checked to see if either AC or DC coupling is selected. Channel numbers range from 1 to 8.

iCpling returned as KS635_DC_COUPLING indicating DC coupling or KS635_AC_COUPLING indicating AC coupling.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_setChannelInputType

Syntax

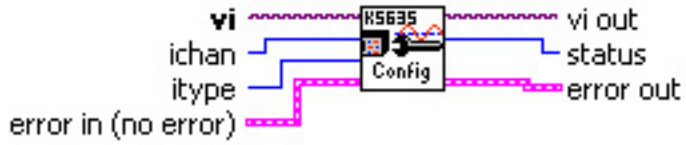
C:

```
ViStatus ks635_setChannelInputType (ViSession vi,
                                    ViInt16 iChan,
                                    ViInt16 iType);
```

Visual Basic:

```
Function ks635_setChannelInputType As ViStatus (ByVal vi As ViSession,
                                                 ByVal iChan As ViInt16,
                                                 ByVal iType As ViInt16)
```

LabVIEW:



ks635 Set Channel Input Type.vi

Description

`ks635_setChannelInputType` configures a given channel on the P635 to use either TTL level input signals or differential input signals. See Chapter 3, "Input Paths", for details.

Parameters

vi unique logical identifier to a session.

iChan is the channel on which the input type is to be configured. Channel numbers range from 1 to 8.

iType set to KS635_DIFFERENTIAL_INPUT for analog differential input or KS635_TTL_INPUT for TTL level input.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified
VI_ERROR_PARAMETER3	Invalid Input Type Specified

ks635_getChannelInputType

Syntax

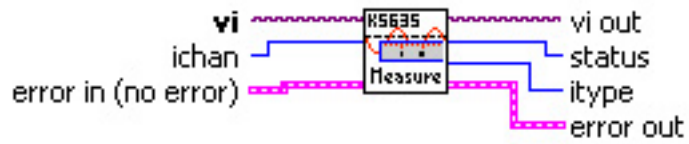
C:

```
ViStatus ks635_getChannelInputType (ViSession vi,
                                    ViInt16 iChan,
                                    ViInt16 *iType);
```

Visual Basic:

```
Function ks635_getChannelInputType As ViStatus (ByVal vi As ViSession,
                                                 ByVal iChan As ViInt16,
                                                 iType As ViInt16)
```

LabVIEW:



ks635 Get Channel Input Type.vi

Description

`ks635_getChannelinputType` determines whether a given P635 channel is configured for analog differential input or TTL level input.

Parameters

vi unique logical identifier to a session.

iChan is the channel on which the input type is to be determined. Channel numbers range from 1 to 8.

iType returned as KS635_DIFFERENTIAL_INPUT for analog differential input or KS635_TTL_INPUT for TTL level input.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_setInterruptEnable

Syntax

C:

```
ViStatus ks635_setInterruptEnable (ViSession vi, ViBoolean bIntEna);
```

Visual Basic:

N/A

LabVIEW:

ks635 SetInterruptEnable.vi

Description

`ks635_setInterruptEnable` enables or disables the generation of PCIbus interrupts when a channel overflows. Overflow occurs when an input signal is too slow to be measured, as explained in Chapter 3, "Overflow".

In order for the interrupt to occur, 1 or more channels must be enabled via `ks635_setInterruptMask` and an interrupt handler must be installed via ks635_`setInterruptHandler`.

Note that events _must_ be enabled first in VISA via viEnableEvent() **before** this function (`ks635_enableEvent`) is called. Enabling events via the Pnp driver before enabling in VISA may cause a system hang!

Parameters

vi unique logical identifier to a session.

bIntEna set to VI_FALSE to disable interrupt generation or VI_TRUE to enable interrupts.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_getInterruptEnable

Syntax

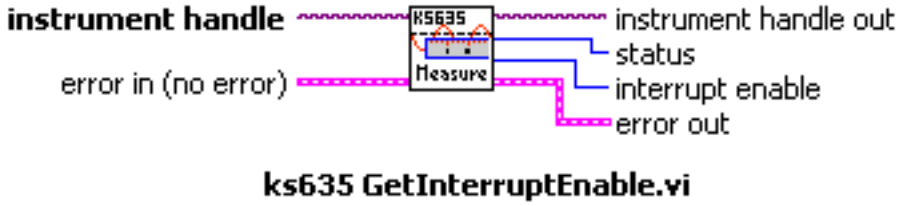
C:

```
ViStatus ks635_getInterruptEnable (ViSession vi, ViBoolean *bIntEna);
```

Visual Basic:

N/A

LabVIEW:



ks635 GetInterruptEnable.vi

Description

`ks635_getInterruptEnable` determines if PCIbus interrupts are enabled or disabled.

Parameters

vi unique logical identifier to a session.

bIntEna returned as VI_FALSE when interrupts are disabled or VI_TRUE when interrupts are enabled.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_setInterruptLevel

Syntax

C:

```
ViStatus ks635_setInterruptLevel (ViSession vi, ViInt16 iLevel);
```

Visual Basic:

N/A

LabVIEW:

Description

`ks635_setInterruptLevel` sets the interrupt IRQ Level.

NOTE: This function is not supported for P635 and thus returns KS635_WARN_NSUP_OPER.

Parameters

vi unique logical identifier to a session.

iLevel the IRQ Level which ranges from 1 to 7.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_WARN_NSUP_OPER	Operation not supported (on P635)

ks635_getInterruptLevel

Syntax

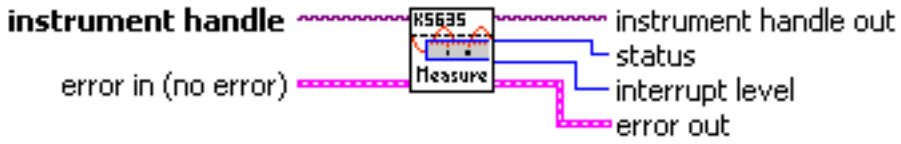
C:

```
ViStatus ks635_getInterruptLevel (ViSession vi, ViInt16 *iLevel);
```

Visual Basic:

N/A

LabVIEW:



ks635 GetInterruptLevel.vi

Description

`ks635_getInterruptLevel` returns the current interrupt IRQ Level.

NOTE: This function is not supported for P635 and thus returns KS635_WARN_NSUP_OPER.

Parameters

vi unique logical identifier to a session.

iLevel populated with the IRQ Level which ranges from 1 to 7.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_WARN_NSUP_OPER	Function not supported for device

ks635_setInterruptMask

Syntax

C:

```
ViStatus  ks635_setInterruptMask (ViSession vi, ViInt16 channelMask);
```

Visual Basic:

N/A

LabVIEW:

Description

`ks635_setInterruptMask` defines the set of channels capable of generating an interrupt. See Chapter 3, "Overflow", for details.

Parameters

vi unique logical identifier to a session.

channelMask a mask in which bits 0-7 represent channels 1 through 8. If a bit is **"1"**, then interrupts are enabled on the associated channel; if a bit is **"0"**, then interrupts are disabled on the associated channel.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_getInterruptMask

Syntax

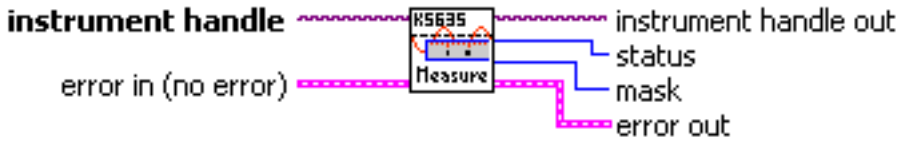
C:

```
ViStatus ks635_getInterruptMask (ViSession vi, ViInt16 *channelMask);
```

Visual Basic:

N/A

LabVIEW:



ks635 GetInterruptMask.vi

Description

`ks635_getInterruptMask` returns a mask of channels for which interrupts are enabled.

Parameters

vi unique logical identifier to a session.

channelMask returned mask in which bits 0-7 represent channels 1 through 8. If a bit is **"1"**, then interrupts are enabled on the associated channel; if a bit is **"0"**, then interrupts are disabled on the associated channel.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_getNumChannels

Syntax

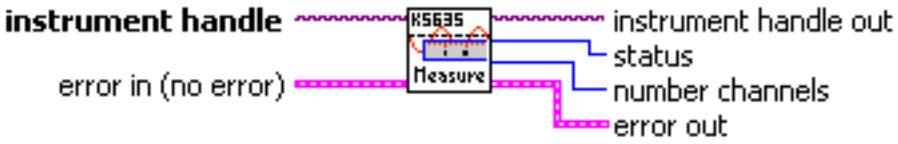
C:

```
ViStatus ks635_getNumChannels (ViSession vi, ViInt16 *numChannels);
```

Visual Basic:

N/A

LabVIEW:



ks635 GetNumChannels.vi

Description

ks635_getNumChannels returns the number of channels the instrument was equipped with at the factory. See the Product Ordering Information section for information on various ordering options.

Parameters

vi unique logical identifier to a session.

numChannels reference to a variable to be populated with the number of channels.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

Action and Status Functions

ks635_setContinuousScanEnable

Syntax

C:

```
ViStatus ks635_setContinuousScanEnable (ViSession vi, ViBoolean bScanEna);
```

Visual Basic:

```
Function ks635_setContinuousScanEnable As ViStatus(ByVal vi As ViSession,
                                                   ByVal bScanEna As ViBoolean)
```

LabVIEW:

ks635 Set Continuous Scan Enable.vi

Description

`ks635_setContinuousScanEnable` starts or stops the P635's continuous free-running mode. While in continuous mode, data can be collected via functions such as `ks635_readFrequency` and `ks635_readSingleChannelFrequency`.

Parameters

vi unique logical identifier to a session.

bScanEna set to VI_TRUE to enter continuous mode or VI_FALSE to stop continuous mode.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_getContinuousScanEnable

Syntax

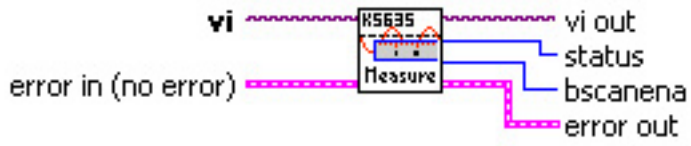
C:

```
ViStatus ks635_getContinuousScanEnable (ViSession vi, ViBoolean *bScanEna);
```

Visual Basic:

```
Function ks635_getContinuousScanEnable As ViStatus (ByVal vi As ViSession,
                                                    bScanEna As ViBoolean)
```

LabVIEW:



ks635 Get Continuous Scan Enable.vi

Description

`ks635_getContinuousScanEnable` determines if the P635 currently is or is not in continuous free-running mode.

Parameters

vi unique logical identifier to a session.

bScanEna returned as VI_TRUE if continuous mode is enabled or VI_FALSE if continuous mode is disabled.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_setHealthCheck

Syntax

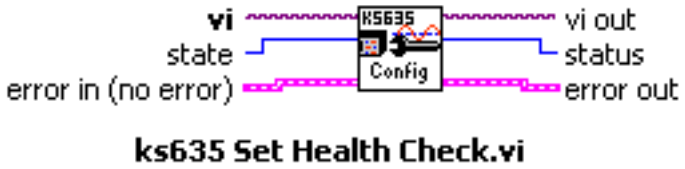
C:

```
ViStatus ks635_setHealthCheck (ViSession vi,
                               ViInt16 state);
```

Visual Basic:

```
Function ks635_setHealthCheck As ViStatus (ByVal vi As ViSession,
                                           ByVal state As ViInt16)
```

LabVIEW:



ks635 Set Health Check.vi

Description

`ks635_setHealthCheck` sets Health Check mode. While Health Check mode is enabled, all channels ignore their regular inputs and instead take their inputs from the Health Check inputs; see Chapter 3, "Health Check", for details.

This facility is used for verifying channel path integrity.

Parameters

vi unique logical identifier to a session.

state is KS635_TTL_HEALTHCHECK to enable TTL health check, KS635_DIFFERENTIAL_HEALTHCHECK to enable differential health check, KS635_DISABLE_HEALTHCHECK to disable health check.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_getHealthCheck

Syntax

C:

```
ViStatus ks635_getHealthCheck (ViSession vi,
                               ViInt16 *state);
```

Visual Basic:

```
Function ks635_getHealthCheckEnable As ViStatus (ByVal vi As ViSession,
                                                 state As ViInt16)
```

LabVIEW:

ks635 Get Health Check.vi

Description

`ks635_getHealthCheck` determines the current Health Check mode of the P635.

Parameters

vi unique logical identifier to a session.

state is either KS635_TTL_HEALTHCHECK, KS635_DIFFERENTIAL_HEALTHCHECK, or KS635_DISABLE_HEALTHCHECK (meaning health check is currently disabled).

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_getCountStatus

Syntax

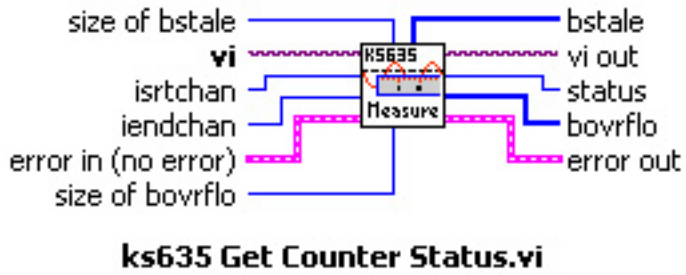
C:

```
ViStatus ks635_getCountStatus (ViSession vi,
                               ViInt16 iStrtChan,
                               ViInt16 iEndChan,
                               ViBoolean bOvrflo[],
                               ViBoolean bStale[]);
```

Visual Basic:

```
Function ks635_ getCountStatus As ViStatus (ByVal vi As ViSession,
                                            ByVal iStrtChan As ViInt16,
                                            ByVal iEndChan As ViInt16
                                            bOvrflo As ViBoolean,
                                            bStale As ViBoolean)
```

LabVIEW:



ks635 Get Counter Status.vi

Description

`ks635_getCountStatus` reads the Stale and Overflow Flags for a range of channels. The Stale Flag is set when two or more successive reads of a channel occur before new measurements are available from the hardware; the Overflow Flag is set when a channel has overflowed due to a slow input signal. See Chapter 3, "Overflow" and "Stale", for details.

Parameters

vi unique logical identifier to a session.

iStrtChan specifies the first channel in the range of channels to be queried. Channel numbers range from 1 to 8.

iEndChan specifies the last channel in the range of channels to be queried. Channel numbers range from 1 to 8.

bOvrflo an array of elements which reflect the state of the requested channel(s) Overflow Flag; VI_TRUE indicates that the channel is in Overflow. The user must provide an array of ViBooleans sufficiently large to accommodate the number of channels specified.

bStale an array of elements which reflect the state of the requested channel(s) Stale Flag; VI_TRUE indicates that the channel is Stale. The user must provide an array of ViBooleans sufficiently large to accommodate the number of channels specified.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_clearCountStatus

Syntax

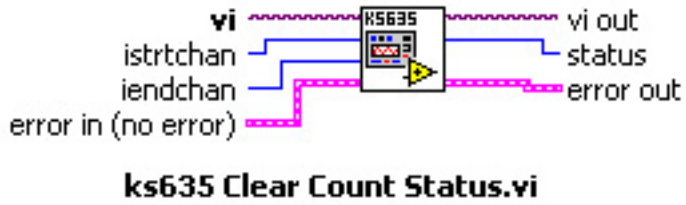
C:

```
ViStatus ks635_clearCountStatus (ViSession vi,
                                 ViInt16 iStrtChan,
                                 ViInt16 iEndChan);
```

Visual Basic:

```
Function ks635_clearCountStatus As ViStatus (ByVal vi As ViSession,
                                             ByVal iStrtChan As ViInt16,
                                             ByVal iEndChan As ViInt16)
```

LabVIEW:



ks635 Clear Count Status.vi

Description

`ks635_clearCountStatus` clears the Stale and Overflow Flags for a range of channels. See Chapter 3, "Overflow" and "Stale", for details.

Parameters

vi unique logical identifier to a session.

iStrtChan specifies the first channel in the range of channels to have flags cleared. Channel numbers range from 1 to 8.

iEndChan specifies the last channel in the range of channels to be have flags cleared. Channel numbers range from 1 to 8.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

Data Functions

ks635_executeSingleScan

Syntax

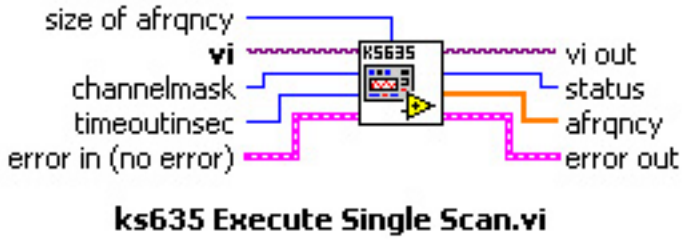
C:

```
ViStatus ks635_executeSingleScan (ViSession vi,
                                  ViInt16 channelMask,
                                  ViUInt32 timeoutInSec,
                                  ViReal64 aFrequency[]);
```

Visual Basic:

```
Function ks635_executeSingleScan As ViStatus (ByVal vi As ViSession,
                                              ByVal channelMask As ViInt16,
                                              ByVal timeoutInSec As ViUInt32,
                                              ByRef aFrequency As ViReal64)
```

LabVIEW:



ks635 Execute Single Scan.vi

Description

`ks635_executeSingleScan` collects and returns 1 sample of frequency from each channel specified in the **channelMask**. All counters are cleared, and the function does not return until a sample is available on each channel specified. The time required to acquire a channel's frequency depends on the size of the Observation Window (q.v. `setObservationWindowSize`), the rate of the Tick Clock (q.v. `setTickClockRate`), and the rate of the input signal: the input signal is monitored for the Observation Window period, or until the signal completes 1 cycle, whichever takes longer. This time is limited by the amount of time it takes the channel to Overflow, and by the time specified in the **timeoutInSec** parameter. The function will not return until all channels specified in the **channelMask** have met 1 of these conditions. In general, the slower the input signal, the longer it will take to measure that input. See Chapter 3, "Basic Circuit Operation", for details on how the Observation Window, Tick Clock rate and other factors impact a channel's measurement.

As a side effect, `ks635_executeSingleScan` will take the P635 out of continuous scan mode (refer to `ks635_setContinuousScanEnable`).

Parameters

vi unique logical identifier to a session.

channelMask specifies which channels to read. Bit 0 is channel 1 whereas Bit 7 is channel 8. If a bit is **"1"**, then the associated channel is read. Because this function is only as fast as the slowest channel, only channels of actual interest should be included in the mask.

timeoutInSec an overall timeout value, in seconds. A value of zero sets the timeout period to a value appropriate for the current Tick Clock rate, and in nearly all cases is the preferred value.

aFrequency an array which will be populated with the measurements of the selected channels. The user is responsible for providing an array of sufficient size to accommodate the number of channels specified in **channelMask**.

Return Values

Return Value	Description
VI_SUCCESS	Successful
KS635_SUCCESS_BUT_OVERFLOW	1 or more channels met an Overflow condition, probably the result of their input being too slow to measure.
KS635_SUCCESS_BUT_STALE	1 or more channels had no measurable input within the timeout interval, probably the result of no signal or DC signal.
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_readFrequency

Syntax

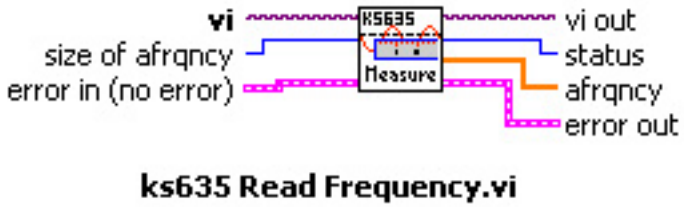
C:

```
ViStatus ks635_readFrequency (ViSession vi, ViReal64 aFrqncy[]);
```

Visual Basic:

```
Function ks635_readFrequency As ViStatus (ByVal vi As ViSession,
                                          aFrqncy As ViReal64)
```

LabVIEW:



Description

`ks635_readFrequency` reads all channels and returns their calculated frequencies in an array. If any channel is in overflow condition, this function will return KS635_SUCCESS_BUT_OVERFLOW. Details on the set of channels in Overflow can be found from `ks635_getCountStatus`.

This function assumes P635 is already in continuous mode via `ks635_setContinuousScanEnable`

Parameters

vi unique logical identifier to a session.

aFreqncy an array which will be populated with the frequency measurements of all channels. The user is responsible for providing an array of sufficient size to accommodate 8 ViReal64 values.

Return Values

Return Value	Description
VI_SUCCESS	Successful
KS635_SUCCESS_BUT_OVERFLOW	1 or more channels met an Overflow condition, probably the result of their input being too slow to measure.
KS635_SUCCESS_BUT_STALE	Valid, but no new measurements were available on 1 or more channels since the last read.
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_readFrequencyWithStatus

Syntax

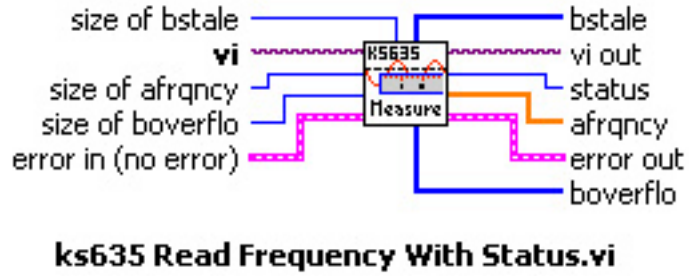
C:

```
ViStatus ks635_readFrequencyWithStatus (ViSession vi,
                                        ViReal64 aFrqncy[],
                                        ViBoolean bOverflo[],
                                        ViBoolean bStale[]);
```

Visual Basic:

```
Function ks635_readFrequencyWithStatus As ViStatus (ByVal vi As ViSession,
                                                    aFrqncy As ViReal64,
                                                    bOverflo As ViBoolean,
                                                    bStale As Boolean)
```

LabVIEW:



ks635 Read Frequency With Status.vi

Description

`ks635_readFrequencyWithStatus` reads all channels and returns their calculated frequencies in an array. Additionally, the Overflow and Stale conditions for each channel is returned in bOverflo and bStale, respectively.

If any channel has an overflow condition, the channel frequency will be 0 and this function will return KS635_SUCCESS_BUT_OVERFLOW; otherwise, if any channel is stale, it will return KS635_SUCCESS_BUT_STALE.

This function assumes P635 is already in continuous mode via `ks635_setContinuousScanEnable`

Parameters

vi unique logical identifier to a session.

aFreqncy an array which will be populated with the frequency measurements of all channels. The user is responsible for providing an array of sufficient size to accommodate 8 ViReal64 values.

bOverflo an array which will be populated with the Overflow condition all channels. The user is responsible for providing an array of sufficient size to accommodate 8 ViBoolean values.

bStale an array which will be populated with the Stale condition all channels. The user is responsible for providing an array of sufficient size to accommodate 8 ViBoolean values.

Return Values

Return Value	Description
VI_SUCCESS	Successful
KS635_SUCCESS_BUT_OVERFLOW	1 or more channels met an Overflow condition, probably the result of their input being too slow to measure.
KS635_SUCCESS_BUT_STALE	Valid, but no new measurements were available on 1 or more channels since the last read.
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_readSingleChannelFrequency

Syntax

C:

```
ViStatus ks635_readSingleChannelFrequency (ViSession vi,
                                           ViInt16 iChan,
                                           ViReal64 *aFrqncy);
```

Visual Basic:

```
Function ks635_readSingleChannelFrequency As ViStatus (ByVal vi As ViSession,
                                                       ByVal iChan As iChan,
                                                       aFrqncy As ViReal64)
```

LabVIEW:

ks635 Read Single Channel Frequency.vi

Description

`ks635_readSingleChannelFrequency` returns the current frequency value of a single channel.

When reading multiple channels, it is more efficient to use `ks635_readFrequency` or `ks635_readFrequencyWithStatus` instead of calling `ks635_readSingleFrequency` multiple times.

This function assumes P635 is already in continuous mode via `ks635_setContinuousScanEnable`

Parameters

vi unique logical identifier to a session.

iChan specifies the channel to read. Channel numbers range from 1 to 8.

aFrqncy pointer to a ViReal64 which will be populated with the frequency measurement of the channel.

Return Values

Return Value	Description
VI_SUCCESS	Successful
KS635_SUCCESS_BUT_OVERFLOW	Overflow condition, probably the result of input being too slow to measure.
KS635_SUCCESS_BUT_STALE	Valid, but no new measurements were available since the last read on this channel
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified

ks635_readCounters

Syntax

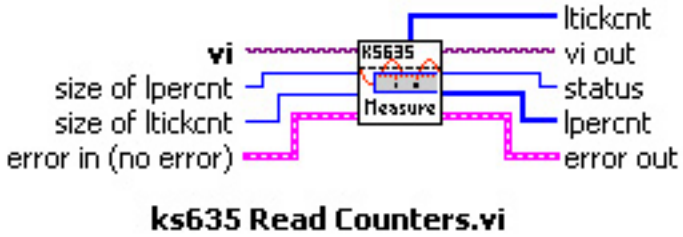
C:

```
ViStatus ks635_readCounters (ViSession vi,
                             ViUInt32 lPerCnt[],
                             ViUInt32 lTickCnt[]);
```

Visual Basic:

```
Function ks635_readCounters As ViStatus (ByVal vi As ViSession,
                                         lPerCnt As ViUInt32,
                                         llTickCnt As ViUInt32)
```

LabVIEW:



Description

`ks635_readCounters` returns raw Period Count and Tick Count values for all channels. See Chapter 3, "Basic Circuit Operation", for details on how to interpret these values.

This function assumes P635 is already in continuous mode via `ks635_setContinuousScanEnable`

Parameters

vi unique logical identifier to a session.

lPerCnt an array which will be populated with the Period Counts of all channels. The user is responsible for providing an array of sufficient size to accommodate 8 ViUInt32 values.

lTickCnt an array which will be populated with the Tick Counts of all channels. The user is responsible for providing an array of sufficient size to accommodate 8 ViUInt32 values.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle
VI_ERROR_INV_SETUP	Invalid Setup Information

ks635_readSingleChannelCounters

Syntax

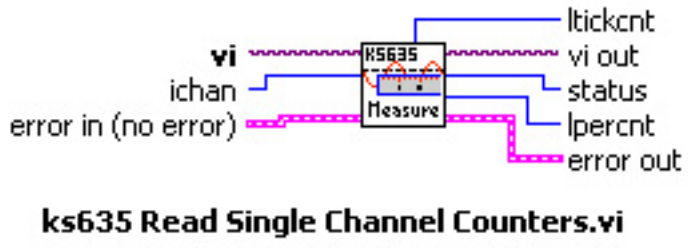
C:

```
ViStatus ks635_readSingleChannelCounters (ViSession vi,
                                          ViInt16 iChan,
                                          ViUInt32 *lPerCnt,
                                          ViUInt32 *lTickCnt);
```

Visual Basic:

```
Function ks635_readSingleChannelCounters As ViStatus (ByVal vi As ViSession,
                                                      ByVal iChan As ViInt16,
                                                      lPerCnt As ViUInt32,
                                                      lTickCnt As ViUInt32)
```

LabVIEW:



Description

`ks635_readSingleChannelCounters` returns raw Period Count and Tick Count values for a given channel. See Chapter 3, "Basic Circuit Operation", for details on how to interpret these values.

This function performs 1 I/O operation each time it is called. For reading 2 or more channels, it will probably be more efficient to call `ks635_readCounters` which performs 1 I/O operation for the block of channels.

This function assumes P635 is already in continuous mode via `ks635_setContinuousScanEnable`

Parameters

vi unique logical identifier to a session.

iChan specifies the channel to read. Channel numbers range from 1 to 8.

lPerCnt pointer to a ViUInt32 which will be populated with the channel's Period Count.

lTickCnt pointer to a ViUInt32 which will be populated with the channel's Tick Count.

Return Values

Return Value	Description
VI_SUCCESS	Successful
VI_ERROR_INV_OBJECT	Invalid Instrument Handle

VI_ERROR_INV_SETUP	Invalid Setup Information
KS635_ERROR_INV_CHNLNMBR	Invalid Channel Number Specified
KS635_SUCCESS_BUT_STALE	A channel contains stale data
KS635_SUCCESS_BUT_OVERFLOW	A channel overflow has occurred

Chapter 5: Application Examples

Typical Device Configuration Example

This example sets up the P635 then reads the module for period count and tick count data. The setup conditions are:

Continuous scan
10 MHz clock
100 ms observation window
50 kHz filters for all channels
DC coupling
Differential input
Operating range = ±100 mV to 20 V operating range: Internal gain = 2

Module Setup	
Initialize P635	ks635_init();
Set the Observation Window to 100 ms.	ks635_setObservationWindowSize();

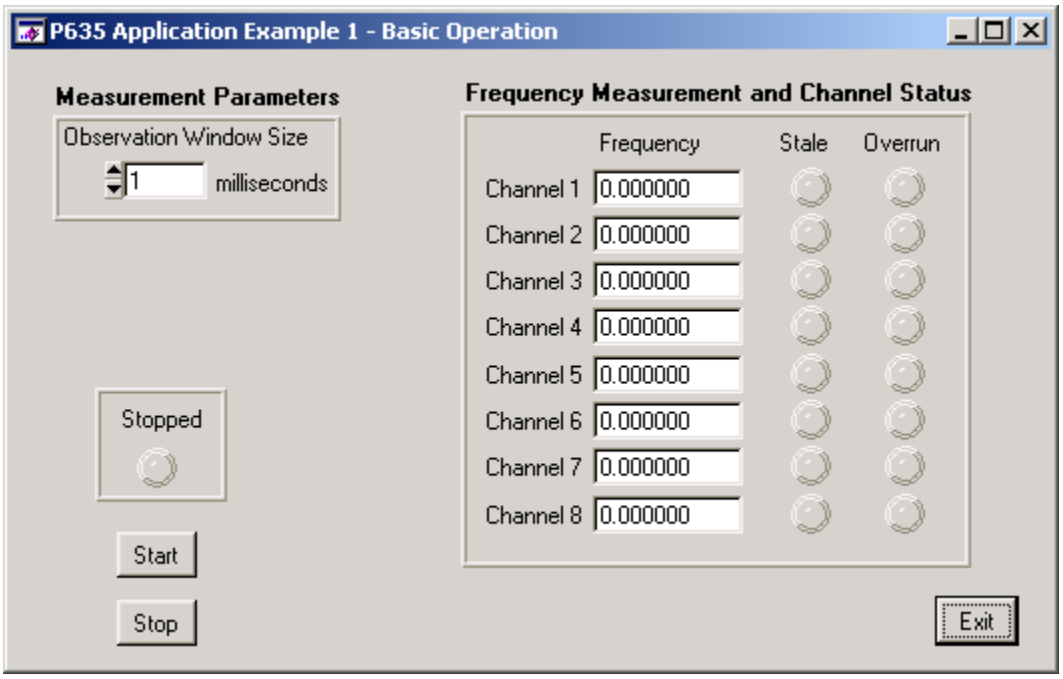
Set 10 MHz Clock.	ks635_setWindowClockSource();
	ks635_setTickClockSource();
	ks635_setTickClockRate();

Set the 50 kHz filters IN for all channels.	ks635_setChannelFilterEnable();
Select DC coupling for all channels.	ks635_setChannelCoupling();
Select differential inputs for all channels.	ks635_setChannelInputType();
Select an internal gain of 2 for all channels.	ks635_setChannelThreshold();
Set Continuous scan.	ks635_setContinuousScanEnable();

Read Period and Tick Count Data	
Read the Count Status data.	ks635_getCounterStatus();
Read the Period Count data for Channel 1.	ks635_readSingleChannelCounters();
Read the Tick Count data for Channel 1.	ks635_readSingleChannelFrequency();
Read the data for the remaining channels that are in use.	ks635_readFrequency(); ks635_readCounters();

Application Example 1

This application example shows the basic functionality of the hardware and PNP driver. All 8 channels are displayed, as well as indications for Overflow and Stale Data. The user can start and stop the acquisition, as well as adjust the Observation Window.

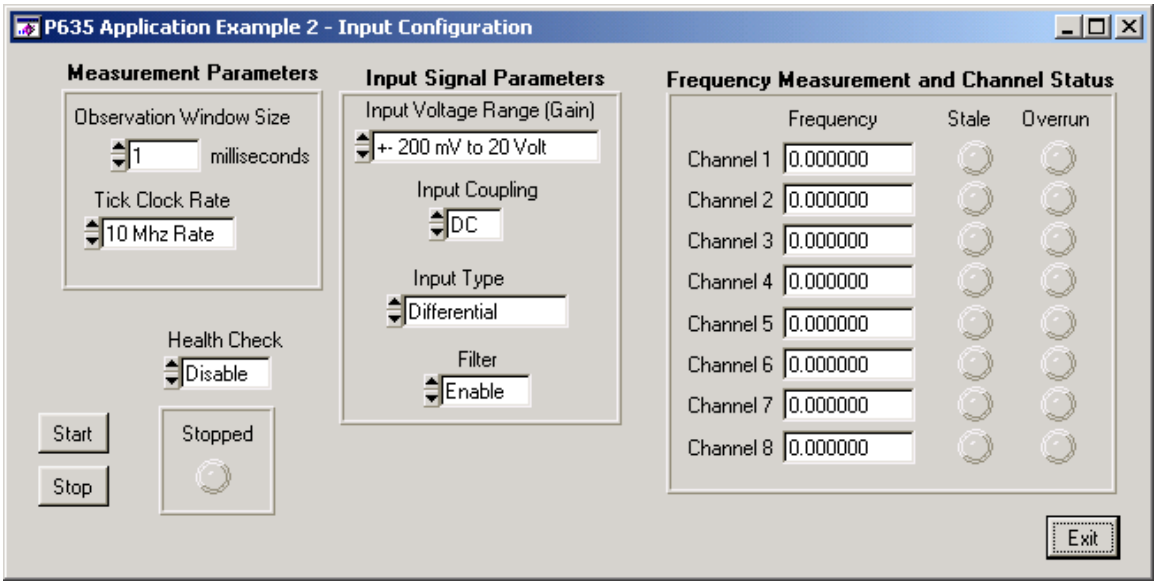


Application Example 2

This application example shows the next level of functionality by allowing the user to modify various settings within the module. The example does not allow all properties of a given channel to be independently modified, but does allow all the various settings to be changed on a global basis. Capabilities supported are:

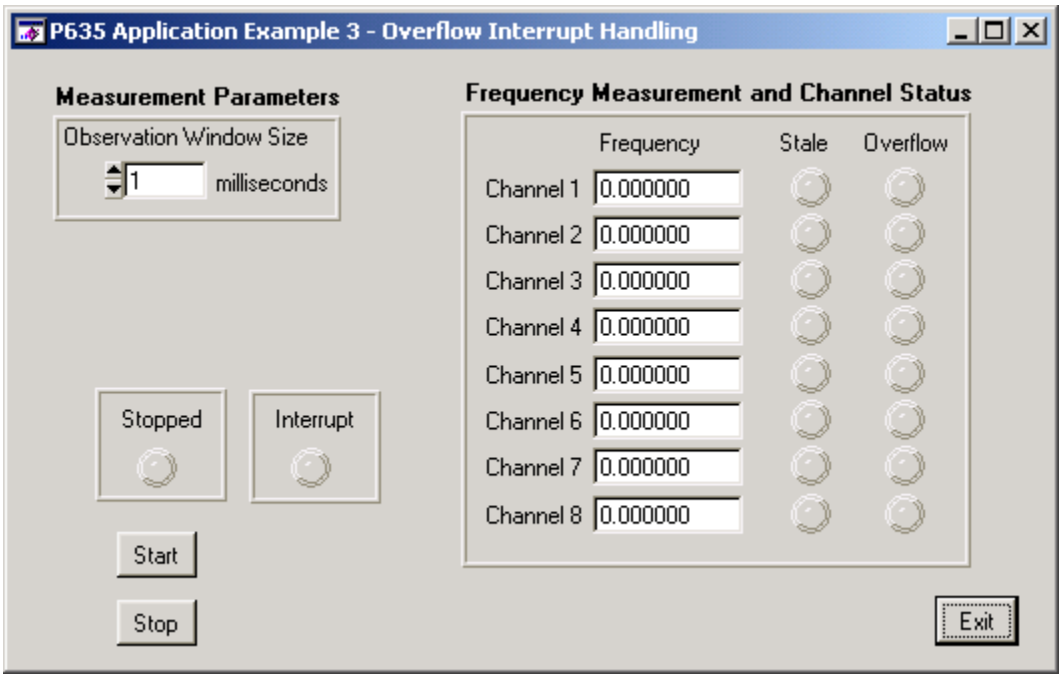
- Start and Stop Acquisition
- Set the input full scale range for all channels
- Set the Observation Window
- Enable/Disable Health check
- Set inputs to AC or DC coupling for all channels
- Set inputs to TTL or Differential for all channels
- Enable/Disable input Filtering for all channels
- Select the Tick Clock Rate

Once acquisition is enabled, the UI will display each channel's current frequency as well as the status of Stale Data and Overflow flags.



Application Example 3

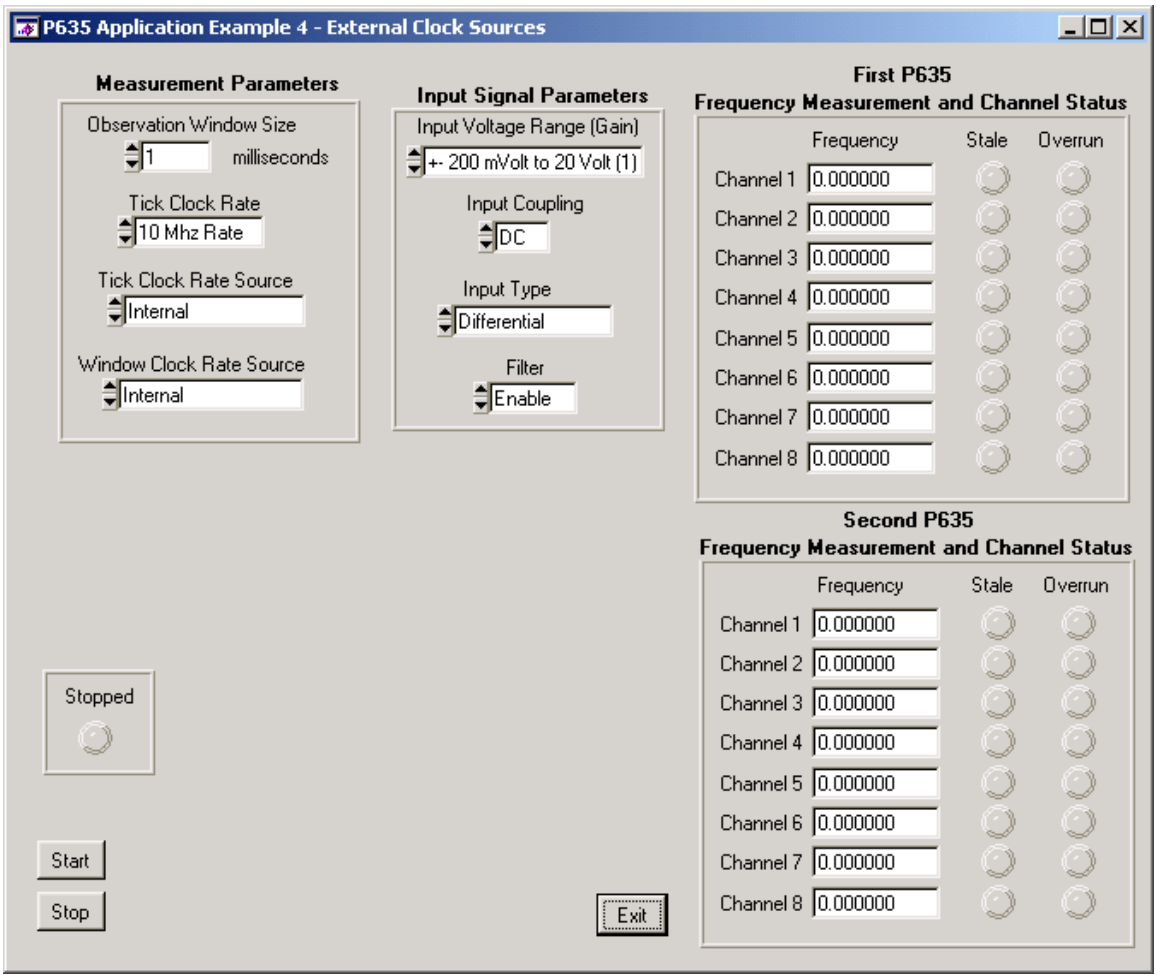
This application example demonstrates the use of interrupts with the P635. The UI allows the user to start and stop the acquisition and control the Observation Window. All 8 channels of frequency data are displayed, as well as indicators for the Stale Data and Overflow flags. An interrupt service routine is used to inform the application that an overflow condition has occurred.



Application Example 4

This application example demonstrates the use of the PXI trigger lines to distribute the Window Clock and/or Tick Clock between modules. This example is only applicable for the PXI platform since the CompactPCI platform does not support trigger lines.

This example allows the user to start and stop acquisition and specify which trigger lines are to be used, and displays all frequency channels, as well as stale and overflow indications.



Appendix A

Technical Support and Warranty

KineticSystems warrants its standard hardware products to be free of defects in workmanship and materials for a period of one year from the date of shipment to the original end user. KineticSystems warrants its software products to conform to the software description applicable at the time of purchase for a period of ninety days from the date of shipment. Products purchased for resale by KineticSystems carry the original equipment manufacturer's warranty.

KineticSystems will, at its option, either repair or replace products that prove to be defective in materials or workmanship during the warranty period.

Transportation charges for shipping products to KineticSystems are prepaid by the purchaser, while charges for returning the repaired product to the purchaser, if located in the United States, are paid by KineticSystems. Return shipments are made by UPS, where available, unless the purchaser requests a premium method of shipment at his expense. The selected carrier is not the agent of KineticSystems, and KineticSystems assumes no liability relating to the services provided by the carrier.

The product warranty may vary outside the United States and does not include shipping, customs clearance or any other charges. Consult your local authorized representative for more information regarding specific warranty coverage and shipping details.

Product specifications and descriptions in this document subject to change without notice.
KineticSystems specifically makes no warranty of fitness for a particular purpose or any other warranty either expressed or implied, except as is expressly set forth herein. This warranty does not cover product failures created by unauthorized modifications, product misuse or improper installation.

Products are not accepted for credit or exchange without prior written approval. If it is necessary to return a product for repair replacement or exchange, a Return Authorization (RA) Number must first be obtained from the Repair Service Center before shipping the product to KineticSystems.

Please take the following steps if you are having a problem and feel you may need to return a product for service:

Contact KineticSystems and discuss the problem with a Technical Service Engineer.
Obtain a Return Authorization (RA) Number.
Initiate a purchase order for the estimated repair charge if the product is out of warranty.
Include with the product a description of the problem and the name of the technical contact person at your facility.
Ship the product prepaid with the RA Number marked on the outside of the package to:

DynamicSignals, LLC
Repair Service Center
900 North State Street
Lockport, IL 60441

Telephone: (815) 838-0005
Fax: (815) 838-4424

Feedback

The purpose of this manual is to provide you with the information you need to make the P635 as easy as possible to understand and use. It is very important that the information is accurate, understandable and accessible. To help us continue to make this manual as "user friendly" as possible, we hope you will fill out this form and Fax it back to us at (815) 838-4424. Or mail a copy to DynamicSignals, LLC 900 N. State, Lockport, IL 60441. Your input is very valuable.

Please rate each of the following.

The information in this manual is:

	Yes									No
Accurate	10	9	8	7	6	5	4	3	2	1
Readable	10	9	8	7	6	5	4	3	2	1
Easy to find	10	9	8	7	6	5	4	3	2	1
Well organized	10	9	8	7	6	5	4	3	2	1
Sufficient	10	9	8	7	6	5	4	3	2	1

We would appreciate receiving any thoughts you have about how we can improve this user's manual:

(Include additional sheets if needed)

Name Phone Company