

SIGNAL[™]

Timer System

User Guide

Version 1.0
September, 2007
Revised March, 2016

Engineering Design

This document is provided for the sole purpose of operating the **SIGNAL Timer System**. No part of this document may be reproduced, transmitted, or stored by any means, electronic or mechanical. It is prohibited to alter, modify, or adapt the software or documentation, including, but not limited to, translating, decompiling, disassembling, or creating derivative works. This document contains proprietary information which is protected by copyright. All rights are reserved.

ENGINEERING DESIGN MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THE MATERIAL CONTAINED HEREIN, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Engineering Design shall not under any conditions be liable for errors contained herein or for incidental or consequential damages arising from the furnishing, performance, or use of this material.

The information in this document is subject to change without notice.

© 2007-2016 Engineering Design, Berkeley, CA. All rights reserved.
Printed in the United States of America.

SIGNAL, Real-Time Spectrogram, RTS, Event Detector, Event Analyzer, Experiment Maker, CBDisk, DartDisk, DTDisk, NIDisk, WaveDisk are trademarks of Engineering Design.

The following are service marks, trademarks, and/or registered trademarks of the respective companies:

Communication Automation: Dart

Creative Technology: Audigy, Extigy

Data Translation: Open Layers

Hewlett-Packard: HP, LaserJet, and DeskJet

Measurement Computing Corp: Computer Boards

Microsoft: Windows, Windows 95, Windows 98, Windows 2000, Windows XP

National Instruments: NI-DAQ, NI-DAQmx

Engineering Design
262 Grizzly Peak Blvd
Berkeley, CA 94708 USA
Tel/fax 510-524-4476
Email info@engdes.com
www.engdes.com

LICENSE AGREEMENT

THIS IS A LEGAL AGREEMENT BETWEEN ENGINEERING DESIGN AND THE BUYER. BY OPERATING THIS SOFTWARE, THE BUYER ACCEPTS THE TERMS OF THIS AGREEMENT.

1. Engineering Design (the "Vendor") grants to the Buyer a non-exclusive license to operate the provided software (the "Software") on ONE computer system at a time. The Software may NOT reside simultaneously on more than one computing machine.
2. The Software is the exclusive property of the Vendor. The Software and all documentation are copyright Engineering Design, all rights reserved. The Software may be duplicated ONLY for archival back-up.
3. The Software is warranted to perform substantially in accordance with the operating literature for a period of 30 days from the date of shipment.
4. EXCEPT AS SET FORTH IN THE EXPRESS WARRANTY ABOVE, THE SOFTWARE IS PROVIDED WITH NO OTHER WARRANTIES, EXPRESS OR IMPLIED. THE VENDOR EXCLUDES ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
5. The Vendor's entire liability and the Buyer's exclusive remedy shall be, at the Vendor's SOLE DISCRETION, either (1) return of the Software and refund of purchase price or (2) repair or replacement of the Software.
6. THE VENDOR WILL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES HEREUNDER, INCLUDING, BUT NOT LIMITED TO, LOSS OF PROFITS, LOSS OF USE, OR LOSS OF DATA OR INFORMATION OF ANY KIND, ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE. IN NO EVENT SHALL THE VENDOR BE LIABLE FOR ANY AMOUNT IN EXCESS OF THE PURCHASE PRICE.
7. This agreement is the complete and exclusive agreement between the Vendor and the Buyer concerning the Software.

Table of CONTENTS

Overview	1
System Features	1
Hardware/Software Overview	2
Timer Hardware Architecture	3
Timer Software Architecture	3
Clock Mode	4
Free-Running (FREERUN)	5
Wait Interval (WAIT)	6
Wait Reference (WAITREF)	6
Gate Time (GATETIME)	7
Hardware Delay (HDELAY)	8
Event Counting Mode	9
Free-Running (FREERUN)	10
Fixed Period (PERIOD)	111
Gated Count (GATED)	122
Two-Trigger (TWOTRIG)	133
Duration Measurement Mode	144
Pulse Duration (PULSE)	144
Pulse Train Period (PERIOD)	155
Intra-Period Intervals (SEMIPER)	166
Two-Trigger Interval (TWOTRIG)	166
Signal Output Mode	177
Pulse (PULSE)	18
Pulse Sequence (PULSESEQ)	19
SIGNAL Timer Devices and Operations	20
Timer Devices	20
GETDEV Command	211
TIMER Command	211
Summary of Timer Parameters	233
Using the TIMER Command	244
Timer Examples	244
1. Software-Poll Event Time: CLOCK FREERUN	244
2. Hardware-Poll Event Time: CLOCK GATETIME	244
3. Fixed Period Event Count: COUNT PERIOD	255
4. Sequential Interval Durations: DUR SEMIPER	277
5. Differential Event Time: DUR TWOTRIG	28
6. Pulse Output: SIGOUT PULSE	29
7. Pulse Train: SIGOUT PULSESEQ	300
8. Sample Clock for Analog I/O: SIGOUT PULSESEQ	311
9. Synthesized Pulse Train: CLOCK WAITREF	322
Hardware Timer Environment	333
Overview of Timer Models	333

Timer Model Capabilities	344
Hardware Issues	355
Timer Signal Connections	355
Timer Signal Polarity	355
Timer Resolution and Accuracy	355
Timer Clock Rate: Resolution vs. Maximum Duration	366
Software vs. Hardware Timer Control	366
Cascading Timers	377
Connecting Timers and I/O Modules on NI Boards	377
Software Issues	39
Buffered Measurements	39
Process Synchronization	40
Timer Overflow and Underflow	411
SIGNAL Timer System vs. Labview	411
Future Developments	422
TIMER STAT: Timer Status	422
DT Capabilities	422

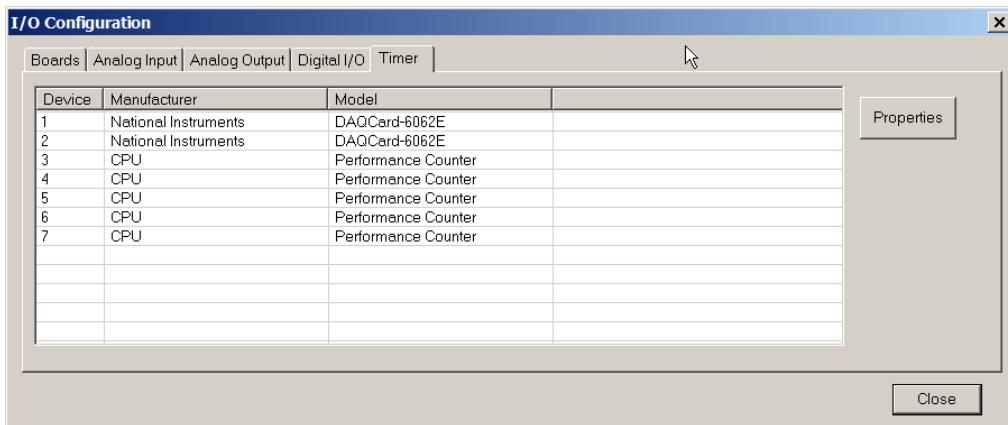
Overview

The SIGNAL Timer System provides time measurement, timing control and triggering capabilities to the **Experiment Maker™** add-on to SIGNAL, which supports real-time behavioral, operant, and neurophysiological experiments. **Capabilities not yet implemented are shown in red.**

System Features

The SIGNAL Timer System provides an extensive library of functions for controlling hardware timers at the software and hardware level. System features include:

- Individual timers automatically detected and accessed by device number:

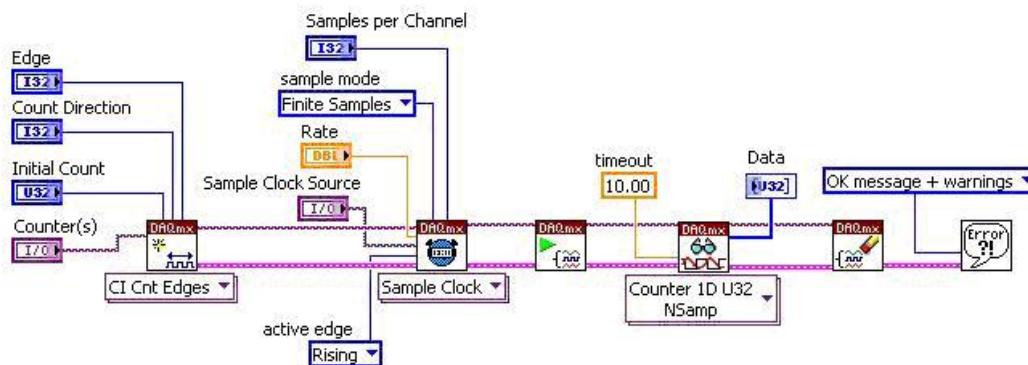


- Time resolution as fine as 12 nsec and time accuracy of 0.01%.
- Fifteen timer functions provide a wide range of timing capabilities:
 - Event timestamping
 - Event counting
 - Event and inter-event duration and frequency measurement
 - Pulse and pulse sequence generation
- Applications include:
 - Reaction times
 - Multiple event duration times, such as button presses representing duration of perceived conditions
 - Timed stimulus presentation loops
 - Precise control of inter-stimulus intervals and chaining of multiple sequential stimuli
 - Counting and timestamping events (such as button presses) over fixed or indefinite time intervals
 - Counting events (such as button presses) while a stimulus is active
 - Synchronizing multiple processes, such as stimulus presentation and response measurement
 - Precise control of external systems via timed triggers
 - Complex timing sequences & loops by cascading multiple timers

- Software interconnect of multiple timers and analog I/O (National Instruments only)
- Generation of trigger signals, gate signals and sampling clocks for external systems
- One parameter set allows easy configuration of parameters such as measurement duration, number of events to count, trigger mode, trigger polarities, pulse duration, pulse onset delay, and pulse sequence rate and duty cycle.

Parameter	Description	Purpose	Support
TIMBUF	Result buffer	Receive multiple duration values	NI
TIMPOLGAT	Gate signal polarity		NI
TIMQTY	No. pulses to measure	Measure fixed or indefinite no. pulses	NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI

- Functions provide software-level and hardware-level timer control:
 - Software control provides easy integration into SIGNAL programs for stimulus presentation, reading and writing digital control lines, and detecting and timestamping button presses with 1-10 msec accuracy.
 - Hardware control provides microsecond accuracy in timestamping and duration measurement, precise inter-event intervals, and synchronization with external events and systems.
- Timer functions support all National Instruments timers and can replace Labview timer VI's and parameters with a coherent, high-level language and easy graphical and numerical access to result data. The following Labview circuit can be replaced with a few SIGNAL program lines. See "SIGNAL Timer System vs. Labview" for details.



Hardware/Software Overview

Hardware timers are provided on analog I/O boards, timer/DIO boards and the CPU chip itself, with a wide variety of overlapping but often incomplete capabilities. Manufacturer timer libraries provide equally arbitrary software support for these capabilities.

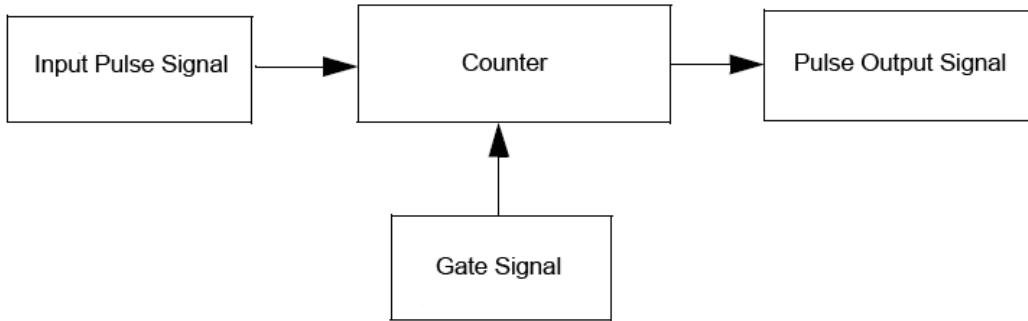
The goal of the SIGNAL Timer System is to define a coherent set of timer tasks and associated language covering timers from all supported manufacturers. As with the SIGNAL analog I/O system, manufacturer-specific hardware and software differences are subsumed by one unified command and parameter set.

Timer operation involves many different kinds of tasks and multiple parameters that can be configured for each task. This document describes how SIGNAL organizes these tasks and parameters into a user interface for timer operation.

Note: All timer functions involve some form of counting and/or timing, and different sources refer to timer devices variously as timers, timer/counters, or counters. This document refers to the physical device as a **timer** and its counting component as a counter. See "Timer Hardware Architecture" for further detail.

Timer Hardware Architecture

Fundamentally, a timer device consists of a counter, an input pulse signal, a gate line, an output pulse generator, and an internal precision timebase:



The **input pulse signal** is a stream of digital pulses which may be periodic (as in a timebase) or sporadic (originating from external events). The **counter** accumulates the number of rising (or falling) edges on the input line. The **gate signal** enables the counter to count (e.g., enabled when high) or triggers special counter operations. The **pulse output signal** is a stream of digital pulses timed by the counter or events on the gate line.

These components can be combined in various configurations to provide different timer modalities, based on counting edges on the input line while enabled by the gate line and optionally producing state transitions on the output line. For example:

1. When the timer simply counts the timebase, it provides an elapsed-time clock for timestamps or wait intervals.
2. If the timer counts the timebase and records timestamps at each pulse on the gate line, it functions as a timestamping event recorder.
3. If the timer counts the timebase only while enabled by the gate line, it effectively measures duration, providing pulse-width and period measurement of the gate signal.
4. If the timer counts the edges of an arbitrary input signal while enabled by the gate for a known duration (such as 1 sec), it provides frequency measurement (e.g., for rotating machinery).
5. If the timer counts the internal timebase and produces output transitions (low to high or high to low) at pre-specified counts, it functions as a programmable pulse or pulse train generator.

Note that examples 3 and 4 are symmetrical: in example 3, the timer uses a known timebase on the input line to measure an unknown duration on the gate line, while in example 4 the timer uses a known duration on the gate line to measure an unknown frequency on the input line.

Timer Software Architecture

The SIGNAL Timer System provides a unified timer task set and timer language covering timers from various manufacturers. In this architecture, timer operations are grouped into four modes – clock, event counting, duration measurement and signal output – and are configured in three layers: timer mode, task type within that mode, and timer parameters to configure task operation.

Timer mode is selected by the **TIMMOD** parameter:

```
SET TIMMOD { CLOCK      }  
            { COUNT      }  
            { DUR       }  
            { SIGOUT    }
```

Timer task is selected by the **TIMTASK** parameter. TIMTASK settings are detailed in the following sections.

```
SET TIMTASK tasktype
```

Finally, each task is configured by timer parameters – such as TIMTRIG (triggering), TIMRATE (clock rate), and TIMDUR (output pulse duration or event counting interval).

Following is a summary of timer modes and tasks.

Mode	Task	Description
CLOCK		Clock mode
	FREERUN	Free-running
	WAIT	Wait interval
	WAITREF	Wait reference
	GATETIME	Gate time
	HDELAY	Hardware delay
COUNT		Event counting mode
	FREERUN	Free-running
	PERIOD	Fixed period
	GATED	Gated count
	TWOTRIG	Two-trigger
DUR		Duration measurement mode
	PULSE	Pulse duration
	PERIOD	Pulse train period
	SEMIPER	Intra-period intervals
	TWOTRIG	Two-trigger interval
SIGOUT		Signal output mode
	PULSE	Pulse
	PULSESEQ	Pulse sequence

The following sections describe each timer mode, its defined tasks, and the parameters recognized by each task.

Clock Mode

Clock mode provides a standalone timebase which can be used for wait intervals between events, loop intervals for repeated events and timestamps for external events. Typical timebase resolution is 50-100 nsec – see "Timer Resolution and Accuracy". Clock mode provides software and/or hardware timer control, depending on the task, as noted below – see "Software vs. Hardware Timer Control" for background.

To set the timer for Clock mode, enter:

```
SET TIMMOD CLOCK
```

Five tasks are defined – free running clock, wait interval, wait reference, triggered timestamp, and hardware delay:

```
SET TIMTASK { FREERUN }
{ WAIT }
```

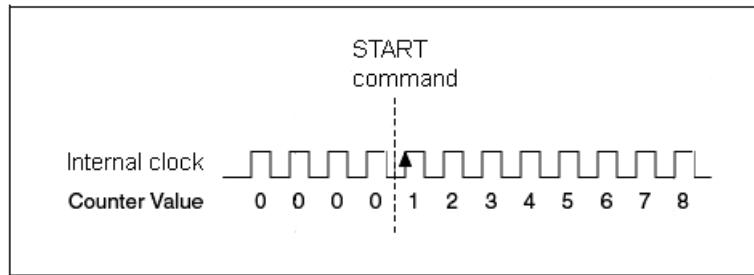
```
{ WAITREF }
{ GATETIME }
{ HDELAY }
```

These are described in the following sections.

Free-Running (FREERUN)

Function

Provide a continuous, pollable timebase for manually timestamping events. The timebase continues indefinitely, hence "free-running". Initiated via software command or hardware trigger if supported by the timer device. Polled via software.



Applications

Timestamping external events from software.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. To start from hardware trigger, set TIMTRIG = EXT and set TIMPOLIN to trigger polarity. **Not supported on NI E-series or any DT boards.**
3. Issue TIMER START to start timebase or arm for hardware trigger. Return immediately.
4. To read current time, issue TIMER READ.

Parameters

Parameter	Description	Purpose	Support
TIMPOLIN	Start trigger polarity		NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	DT, NI
TIMTRIG	Trigger mode	Optional hardware start trigger	NI

Connections

DT: inter-timer connection required: rate gen out to event counter in.

NI: Gate = start trigger [optional] – M-series and TIO-series only.

CPU: none.

Supported Hardware

DT: software start and poll on all models (via rate gen + event count).

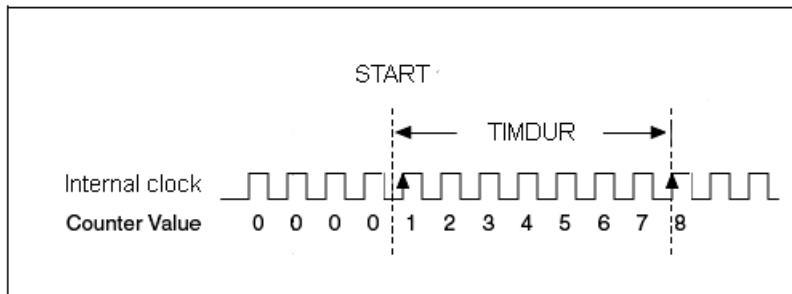
NI: software start and poll on all models. Hardware start on M-series and TIO-series only.

CPU: software start and poll on all models.

Wait Interval (WAIT)

Function

Wait for a specified interval, then resume execution. Software controlled. The Interval task (see below) provides the same functionality in hardware. TIMER START can be called multiple times, and each time will wait for the specified interval.



Applications

Insert a fixed time delay in a command stream.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMDUR to desired wait interval.
3. Issue TIMER START to begin wait interval. Return after specified interval is complete.

Parameters

Parameter	Description	Purpose	Support
TIMDUR	Duration (sec) of wait interval		DT, NI, CPU
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	DT, NI

Connections

DT: inter-timer connection required: rate gen out to event counter in.

NI, CPU: none.

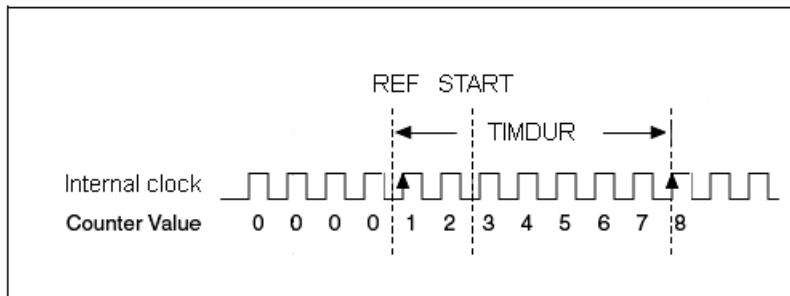
Supported Hardware

DT, NI, CPU: supported on all models.

Wait Reference (WAITREF)

Function

Wait for a specified interval since a reference time, then resume execution. Software controlled. TIMER START can be called multiple times, and each time will wait for another multiple of the specified interval.



Applications

- Create a software loop interval, for example, to initiate a playback every 10 seconds including processing time between playbacks.
- For a hardware-controlled loop interval, use a pulse train at the desired loop interval to trigger the playback (see Signal Output mode) instead.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMDUR to desired wait interval.
3. Issue TIMER REF to set reference time.
4. Perform desired processing, such as playback.
5. Issue TIMER START to begin waiting for remainder of specified interval. Return after interval is complete.
6. Typically, return to TIMER REF step for another iteration.

Parameters

Parameter	Description	Purpose	Support
TIMDUR	Duration (sec) of wait interval		DT, NI, CPU
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	DT, NI

Connections

DT: inter-timer connection required: rate gen out to event counter in.
NI, CPU: none.

Supported Hardware

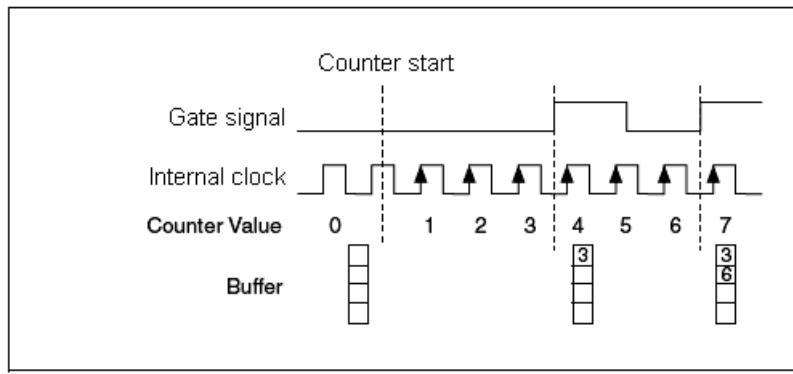
DT, NI, CPU: supported on all models.

Gate Time (GATETIME)

Function

Provide a continuous timebase for digitally timestamping events. Initiated via software command or hardware trigger if supported by the timer device. Records a timestamp at each rising (or falling) edge on the gate line. Read stored timestamps via software into a SIGNAL buffer for analysis.

Note that with a finite TIMQTY, CLOCK | GATETIME can provide a "COUNT | WAIT" functionality, waiting for a specified number of events, analogous to CLOCK | WAIT waiting for a specified time duration.



Applications

Timestamp a sequence of digital events.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMQTY = number of events to timestamp, or 0 to timestamp indefinitely.
3. Set TIMBUF = time buffer to receive multiple timestamps.
4. Set TIMPOLGAT = polarity of event stream to be timestamped.
5. To start from hardware trigger, set TIMTRIG = EXT and set TIMPOLIN to trigger polarity. **Not supported on NI E-series boards.**
6. Issue TIMER START to start timebase or arm for hardware trigger. Return immediately.
7. Issue TIMER READ to get measurement results.
8. **Important:** see "Buffered Measurements" for details on TIMQTY and TIMER READ.

Parameters

Parameter	Description	Purpose	Support
TIMBUF	Timer buffer	Destination buffer for event timestamps	NI
TIMPOLGAT	Gate signal polarity	Polarity of event stream	NI
TIMPOLIN	Start trigger polarity		NI
TIMQTY	No. events to timestamp	Timestamp fixed or infinite events	NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI
TIMTRIG	Trigger mode	Optional hardware start trigger	NI

Connections

Aux = start trigger [optional] – M-series and TIO-series only.

Gate = digital event stream to be timestamped.

Supported Hardware

DT: not available.

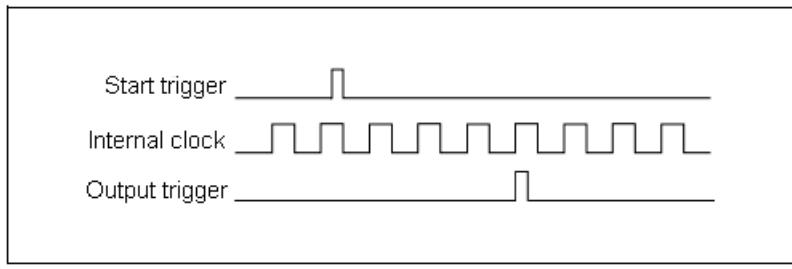
NI: supported on all models.

CPU: not available.

Hardware Delay (HDELAY)

Function

Issue an output trigger a specified interval after receiving a start trigger. Implemented internally as a triggered output pulse with initial delay equal to the specified wait interval (see Signal Output mode).



Applications

Precise inter-event delay between two hardware events, becoming part of a chain of hardware triggers.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMDUR to desired wait interval.
3. Set start trigger polarity TIMPOLIN and output trigger polarity TIMPOLOUT.
4. Issue TIMER START to arm for start trigger. Return immediately.

Parameters

<u>Parameter</u>	<u>Description</u>	<u>Purpose</u>	<u>Support</u>
TIMDUR	Duration (sec) of wait interval		NI
TIMPOLIN	Start trigger polarity		NI
TIMPOLOUT	Output trigger polarity		NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI

Connections

Gate = start trigger

Output = output trigger

Supported Hardware

DT: available (via one-shot + duty cycle) but not implemented.

NI: supported on all models.

CPU: not available.

Event Counting Mode

In event counting mode, the timer counts the number of events (rising or falling edges) occurring within some specified interval. This interval may be defined by software start and read commands, a fixed time period, rising and falling edges of a gate signal, or start and stop triggers.

Applications include subject testing, where events might be button presses, and industrial processes, where events might be threshold exceedances (quality control) or periodic pulses from rotating machinery (frequency measurement). Event counting is performed entirely in hardware.

To set the timer for Event Counting mode, enter:

SET TIMMOD COUNT

Four tasks are defined – the number of events between start and read commands, during a fixed time period, during a gate pulse, or between start and stop triggers.

```

SET TIMTASK { FREERUN    }
             { PERIOD     }
             { GATED      }
             { TWOTRIG   }

```

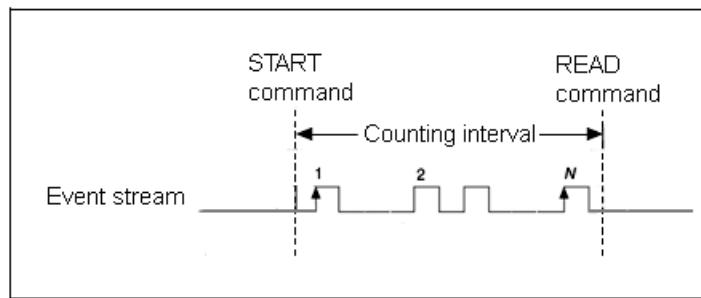
These are described in the following sections.

Free-Running (FREERUN)

Function

Count the number of rising (or falling) edges on the input line since TIMER START. Counting is performed without gating, hence "free-running", and continues until stopped, analogous to CLOCK | FREERUN. Initiated via software command or hardware trigger if supported by the timer device. Polled via software command.

To wait for a finite number of events (an "event count wait" function), use CLOCK | GATETIME with a finite TIMQTY.



Applications

Count an event stream allowing the user to poll current event count in software.

Operation

1. Set TIMPOLIN to polarity of event stream to be counted.
2. To start from hardware trigger, set TIMTRIG = EXT and set TIMPOLIN to trigger polarity. **Not supported on NI E-series or any DT boards.**
3. Issue TIMER START to start counting or arm for hardware trigger. Return immediately.
4. Issue TIMER READ any number of times to get the cumulative number of events since TIMER START. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMPOLIN	Event stream polarity		DT, NI
TIMTRIG	Trigger mode	Optional hardware start trigger	NI

Connections

Input = event stream.

Gate = start trigger [optional] – M-series and TIO-series only.

Supported Hardware

DT: available but not implemented.

NI: supported on all models.

CPU: not available.

Implementation

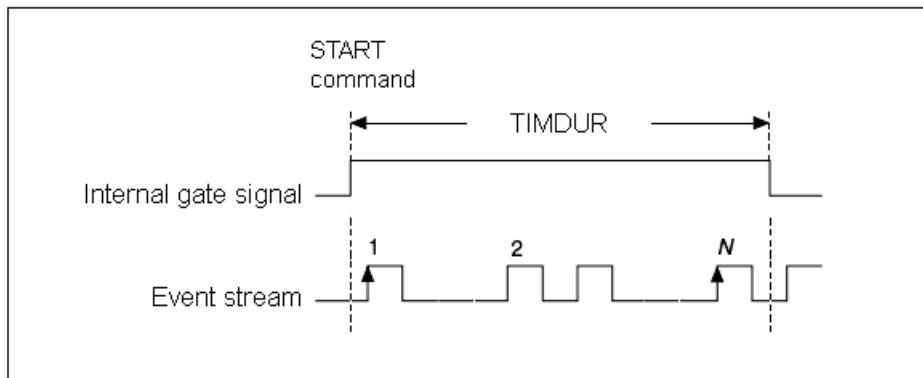
NI: edge counting on event stream.

DT: ungated event count.

Fixed Period (PERIOD)

Function

Count the number of rising (or falling) edges on the input line within a specified time period. Initiated via software command or hardware trigger if supported by the timer device.



Applications

- Count the number of events (such as button presses) occurring within a specific time interval.
- Measure the frequency of a continuous pulse train (such as machine rotational speed).

Operation

1. Set TIMDUR to desired measurement period.
2. Set TIMDELAY to desired delay before measurement period.
3. Set TIMPOLIN to polarity of event stream to be counted.
4. To start from hardware trigger, set TIMTRIG = EXT and set TIMPOLGAT to trigger polarity.
Supported on NI M-series and TIO boards only.
5. Issue TIMER START to begin counting or arm for hardware trigger. Return immediately.
6. Issue TIMER READ to wait for measurement result. See "Buffered Measurements" for details.
7. **Note: If this task is performed on timer devnum, SIGNAL will use timer devnum+1 internally to define the measurement period, and will issue an error if this timer is unavailable.**

Parameters

Parameter	Description	Purpose	Support
TIMDELAY	Onset delay (sec)	Delay interval before measurement period	NI
TIMDUR	Measurement period (sec)		NI
TIMPOLGAT	Start trigger polarity		NI
TIMPOLIN	Event stream polarity		NI
TIMTRIG	Trigger mode	Optional hardware start trigger	NI

Connections

Input = event stream.

Aux = start trigger [optional] – M-series and TIO-series only.

DT: inter-timer connection required: one-shot out to event counter gate.

Supported Hardware

DT: available but not implemented.

NI: supported on all models.

CPU: not available.

Implementation

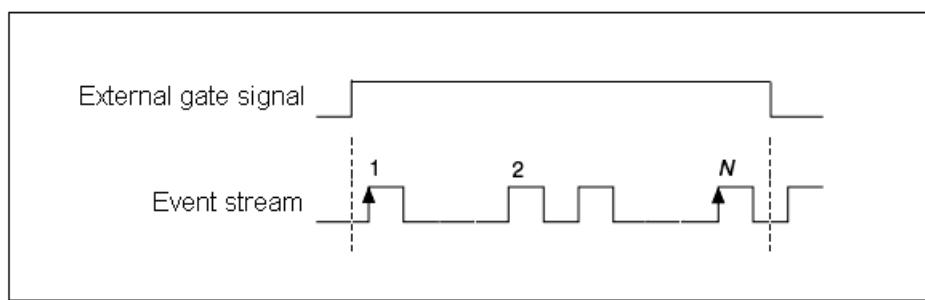
NI: timer 1 = buffered pulse-width measurement, qty=1, w/ event stream as clock, timer 2 = fixed-duration output pulse connected to timer 1 gate.

DT: gated event count on timer 1, one-shot pulse output on timer 2, cabled to timer 1 gate.

Gated Count (GATED)

Function

Count the number of rising (or falling) edges on the input line while an external gate signal is active. The gate signal may go active for one or more periods, and SIGNAL will count events during each period separately.



Applications

Count the number of button presses occurring while a stimulus is active, or more generally, the number of events occurring while another event is active.

Operation

1. Set TIMQTY = number of gate pulses over which to count, or 0 to measure over infinite pulses.
2. Set TIMBUF = time buffer to receive multiple event counts.
3. Set TIMPOLIN to polarity of event stream to be counted.
4. Set TIMPOLGAT to polarity of counting control signal.
5. Issue TIMER START to begin counting. Return immediately.
6. Issue TIMER READ to wait for measurement result. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMBUF	Timer buffer	Destination buffer for event counts	DT, NI
TIMPOLGAT	Gate signal polarity	Polarity of counting control signal	DT, NI
TIMPOLIN	Event stream polarity		NI
TIMQTY	No. gate pulses to count over	Count fixed or infinite gate pulses	NI

Connections

Input = event stream.

Gate = gate signal.

Supported Hardware

DT: available but not implemented.

NI: supported on all models.

CPU: not available.

Implementation

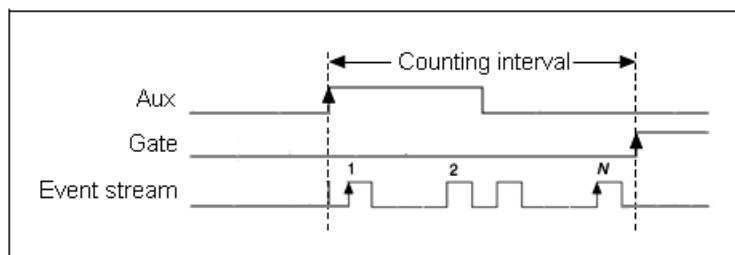
NI: buffered pulse-width measurement w/ event stream as clock and external signal as gate.

DT: gated event count.

Two-Trigger (TWOTRIG)

Function

Count the number of rising (or falling) edges on the input line between a start trigger and stop trigger on two different lines. To count events between successive triggers on the same line, perform this task with the line connected to Aux and Gate inputs.



Applications

Count the number of events between start and stop triggers.

Operation

1. Set TIMPOLGAT to polarity of aux and gate triggers.
2. Set TIMPOLIN to polarity of event stream to be counted.
3. Issue TIMER START to arm timer for start trigger. Return immediately.
4. Issue TIMER READ to wait for measurement result. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMPOLGAT	Aux and Gate trigger polarities	Polarity of triggers	NI
TIMPOLIN	Event stream polarity		NI

Connections

Input = event stream.

Aux = start trigger.

Gate = stop trigger.

Supported Hardware

DT: not available.

NI: supported on M-series and TIO-series only. **NOT IMPLEMENTED**

CPU: not available.

Implementation

NI: buffered two-signal measurement.

Duration Measurement Mode

In duration measurement mode, the timer counts its own timebase to measure the time duration of one or more events in an external signal. Measurements include the time between the rising and falling edges of one signal (pulse width), between successive rising edges of one signal (period measurement), the sequence of durations between successive rising and falling edges on one signal ("semi-period" measurement), and between the rising edges on two different lines (inter-event or "two-trigger" interval).

To set the timer for Duration Measurement mode, enter:

```
SET TIMMOD DUR
```

Four tasks are defined – pulse width, pulse period, intra-period intervals (between successive pulse edges), and duration between two triggers:

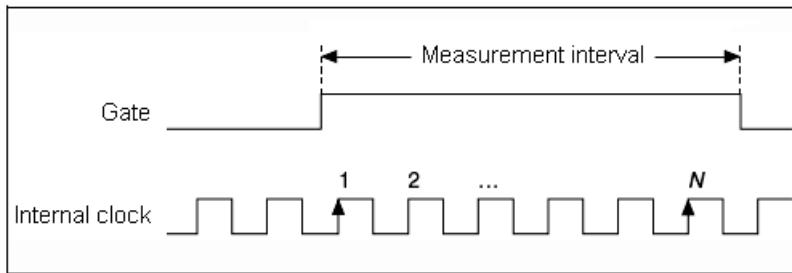
```
SET TIMTASK { PULSE      }
             { PERIOD     }
             { SEMIPER    }
             { TWOTRIG   }
```

These are described in the following sections.

Pulse Duration (PULSE)

Function

Measure duration between rising and falling edges of gate signal.



Applications

Measure the time duration of a digital pulse (representing, for example, event activity).

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMPOLGAT = polarity of pulse to be measured.
3. Set TIMQTY = number of pulses to measure, or 0 to measure indefinitely.
4. Set TIMBUF = time buffer to receive multiple duration values.
5. Issue TIMER START to begin measuring. Return immediately.
6. Issue TIMER READ to get measurement results. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMBUF	Result buffer	Receive multiple duration values	NI
TIMPOLGAT	Gate signal polarity	Polarity of pulse to be measured	NI
TIMQTY	No. pulses to measure	Measure fixed or infinite no. pulses	NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI

Connections

Gate = pulse to be measured.

DT: inter-timer connection required: rate gen out to event counter in.

Supported Hardware

DT: available (via rate gen + gated event count) but not implemented.

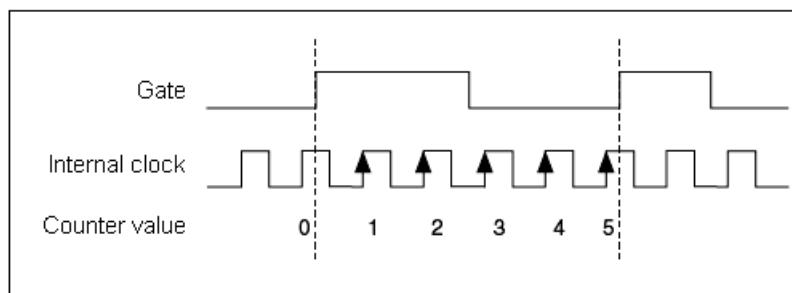
NI: supported on all models.

CPU: not available.

Pulse Train Period (PERIOD)

Function

Measure duration between successive rising edges of gate signal.



Applications

- Measure reaction time = time between stimulus trigger and subject response trigger on same line.
- Measure period of pulse train.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMPOLGAT = polarity of pulse train to be measured.
3. Set TIMQTY = number of periods to measure, or 0 to measure indefinitely.
4. Set TIMBUF = time buffer to receive multiple duration values.
5. Issue TIMER START to begin measuring. Return immediately.
6. Issue TIMER READ to get measurement results. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMBUF	Result buffer	Receive multiple duration values	NI
TIMPOLGAT	Gate signal polarity	Polarity of pulse signal to be measured	NI
TIMQTY	No. periods to measure	Measure fixed or infinite no. periods	NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI

Connections

Gate = pulse train to be measured.

Supported Hardware

DT: not available.

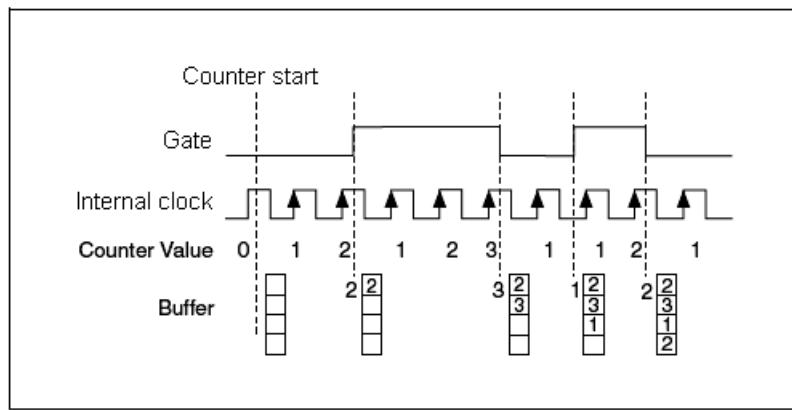
NI: supported on all models.

CPU: not available.

Intra-Period Intervals (SEMIPER)

Function

Measure sequence of durations between successive rising and falling edges of gate signal.



Applications

Record the time profile of multiple presses of a single button, where each press represents the time duration of a condition perceived by the subject.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMPOLGAT = polarity of pulse train to be measured.
3. Set TIMQTY = number of intervals to measure, or 0 to measure indefinitely.
4. Set TIMBUF = time buffer to receive multiple duration values.
5. Issue TIMER START to begin measuring. Return immediately.
6. Issue TIMER READ to get measurement results. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMBUF	Result buffer	Receive multiple duration values	NI
TIMPOLGAT	Gate signal polarity	Polarity of pulse signal to be measured	NI
TIMQTY	No. intervals to measure	Measure fixed or infinite no. intervals	NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI

Connections

Gate = pulse train to be measured.

Supported Hardware

DT: not available.

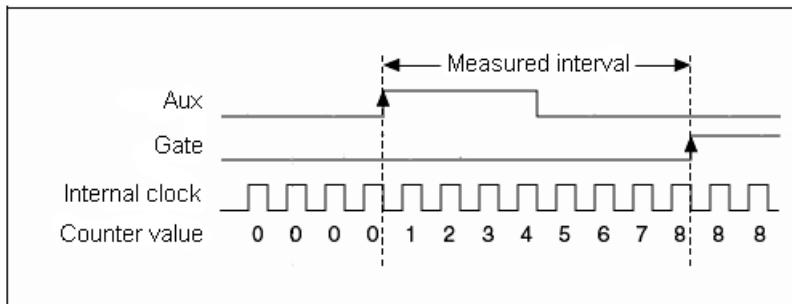
NI: supported on all models.

CPU: not available.

Two-Trigger Interval (TWOTRIG)

Function

Measure duration between start and stop triggers on two different lines.



Applications

Measure reaction time = time between stimulus trigger and subject response trigger on separate lines.

Operation

1. Set TIMRATE for desired resolution and duration – see "Timer Clock Rate: Resolution vs. Maximum Duration".
2. Set TIMPOLGAT to polarity of aux and gate triggers.
3. Issue TIMER START to arm timer for start trigger. Return immediately.
4. Issue TIMER READ to wait for measurement result. See "Buffered Measurements" for details.

Parameters

Parameter	Description	Purpose	Support
TIMPOLGAT	Aux and Gate trigger polarities	Polarity of triggers	NI
TIMRATE	Requested timebase frequency	Determines timer resolution and max duration	NI

Connections

Aux = start trigger.

Gate = stop trigger.

Supported Hardware

DT: not available.

NI: supported on M-series and TIO-series only. **NOT IMPLEMENTED**

CPU: not available.

Implementation

NI: buffered two-signal measurement.

Signal Output Mode

Signal Output mode generates TTL output signals consisting of individual pulses or periodic pulse trains. Pulses are produced by counting the internal timebase and producing output transitions (low to high or high to low) at pre-specified counts, and pulse trains are produced by doing this at a specified repetition interval.

Individual pulses can be configured for onset delay, duration, and polarity. Pulse trains can be configured for pulse frequency, duty cycle, number of pulses (for finite trains), and polarity.

Signal Output mode can be used to generate trigger signals for another process (such as acquisition or playback), enable signals to gate another timer, pulse trains for use as sample clocks (in acquisition or playback), and compound pulse trains (a pulse sequence repeated at some interval), for example to

perform analog I/O at widely spaced intervals. Compound pulse trains are created by cascading multiple timers – see "Cascading Timers".

To set the timer for Signal Output mode, enter:

SET TIMMOD SIGOUT

Two tasks are defined – single pulse output and pulse sequence output:

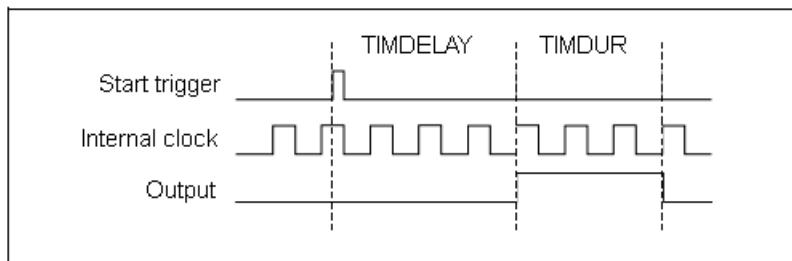
SET TIMTASK { PULSE }
{ PULSESEQ }

These are described in the following sections.

Pulse (PULSE)

Function

Generate a single output pulse configurable for onset delay, duration and polarity. Initiated by software command or hardware trigger. Following is a hardware-triggered pulse:



Applications

- Trigger another timer or an external system.
- Provide a gate control pulse for another timer.

Operation

1. Set TIMDELAY to desired delay between start command or trigger and pulse onset.
2. Set TIMDUR to pulse duration.
3. Set TIMPOLOUT to active pulse state (high or low). E.g., TIMPOLOUT POS starts low, goes high while active, then returns low.
4. To start from hardware trigger, set TIMTRIG = EXT and set TIMPOLIN to trigger polarity.
5. Issue TIMER START to start immediately or arm for hardware trigger. Return immediately.

Parameters

Parameter	Description	Purpose	Support
TIMDELAY	Onset delay (sec)	Delay interval between start command or trigger and first pulse	NI
TIMDUR	Pulse duration (sec)		DT, NI
TIMPOLIN	Start trigger polarity		DT, NI
TIMPOLOUT	Output pulse polarity	Active pulse state (low or high)	DT, NI
TIMRTN	Timer return mode	Whether TIMER START returns immediately or waits for task to complete	NI
TIMTRIG	Trigger mode	Optional hardware start trigger	DT, NI

Connections

Gate = start trigger [optional]

Output = pulse output

Supported Hardware

DT: available (via one-shot + duty cycle) but not implemented.

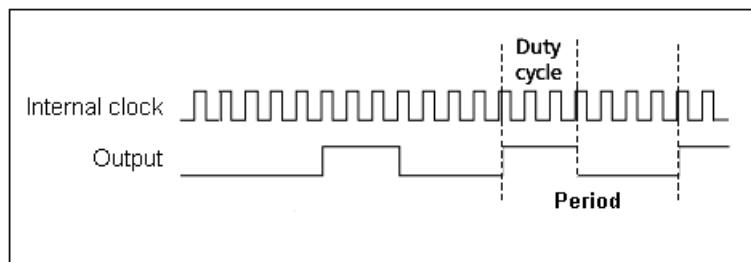
NI: supported on all models.

CPU: not available.

Pulse Sequence (PULSESEQ)

Function

Generate a train of equally spaced pulses, configurable for pulse frequency, duty cycle, number of pulses (for finite trains), and polarity. Initiated by software command or hardware trigger.



Applications

- Provide the clock source for another timer or analog I/O process.
- Provide a repetitive trigger for timed process loop such as repeated playbacks.

Operation

1. Set TIMRATE to desired pulse rate.
2. Set TIMDELAY to desired delay between start command or trigger and first pulse onset.
3. Set TIMCYCLE to pulse duty cycle = fractional pulse period. E.g., TIMCYCLE = 0.25 goes active for 25% of the period, where period = 1 / TIMRATE.
4. Set TIMQTY to number of output pulses or 0 for a continuous pulse train.
5. Set TIMPOLOUT to active pulse state (high or low). E.g., TIMPOLOUT POS starts low, goes high for the specified duty cycle, then returns low.
6. To start from hardware trigger, set TIMTRIG = EXT and set TIMPOLIN to trigger polarity.
7. Issue TIMER START to start immediately or arm for hardware trigger. Return immediately.
8. TIMQTY > 0 with NI-6062 (and possibly other NI devices) invisibly uses a second timer device for output control.

Parameters

Parameter	Description	Purpose	Support
TIMCYCLE	Pulse duty cycle	Fraction (0-1) of pulse period (= 1 / TIMRATE) when pulse is active	DT, NI
TIMDELAY	Onset delay (sec)	Delay interval between start command or trigger and first pulse	NI
TIMPOLIN	Start trigger polarity		DT, NI
TIMPOLOUT	Output pulse polarity	Active pulse state (low or high)	DT, NI
TIMQTY	No. pulses to generate	Output fixed or infinite no. pulses	NI
TIMRATE	Pulse train frequency (Hz)	No. pulses / sec	DT, NI
TIMRTN	Timer return mode	Whether TIMER START returns immediately or waits for task to complete	NI
TIMTRIG	Trigger mode	Optional hardware start trigger	DT, NI

Connections

Gate = start trigger [optional]

Output = pulse output

Supported Hardware

DT: available (via rate gen) but not implemented. Not all parameters supported.

NI: supported on all models.

CPU: not available.

SIGNAL Timer Devices and Operations

SIGNAL **timer devices** represent the physical timers in the system. These may reside on analog I/O boards, timer/DIO boards and the CPU board. At startup, SIGNAL scans the system for timers and assigns device numbers.

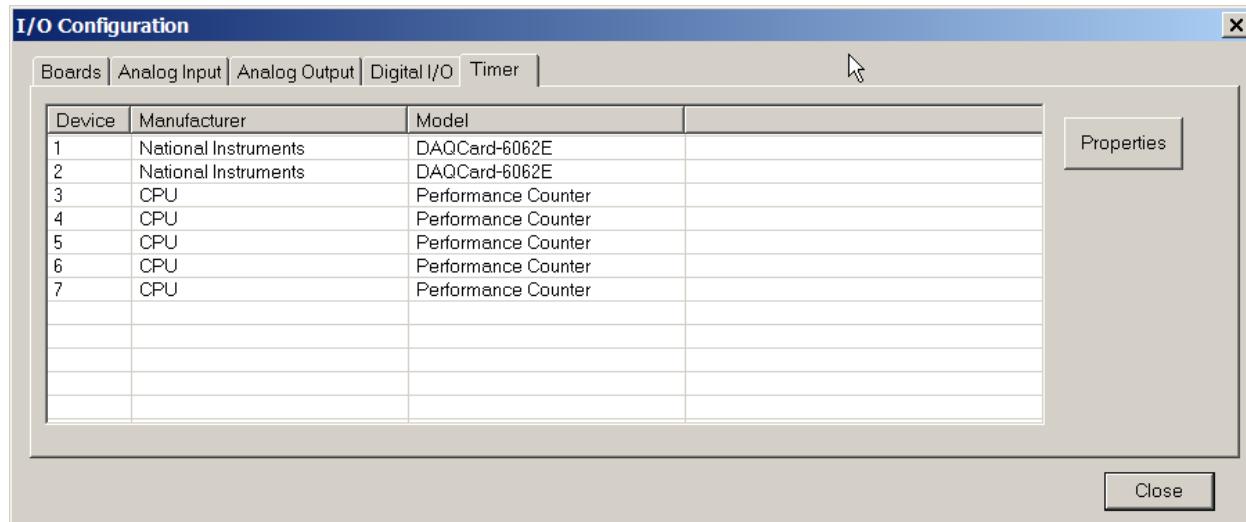
SIGNAL **timer operations** are configured by the TIMMOD, TIMTASK, and timer operational parameters (TIMDUR, TIMRATE, TIMTRIG, etc.), then performed by the TIMER command.

Timer Devices

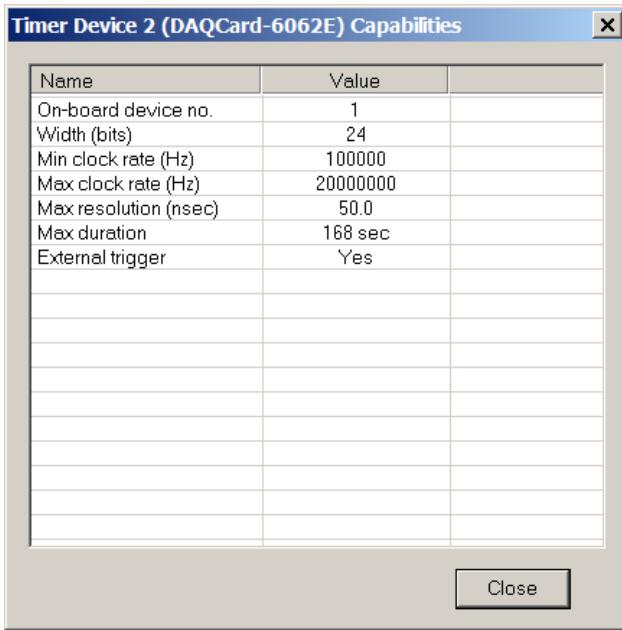
Physical timers may reside on analog I/O boards, timer/DIO boards and the CPU board and an individual board usually contains multiple timers. At startup, SIGNAL scans the system for installed timers and assigns sequential timer device numbers beginning with 1. See the screenshot below.

SIGNAL device numbers are used to address a specific timer in the TIMER command. Thus "timer 2 open" opens the second timer, which in the screenshot refers to the second timer on the DAQCard-6062E card. Note that SIGNAL device numbers do not generally represent the timer's device number within its own board, which is shown in the Properties page.

I/O | Configure | Timer from the SIGNAL menu displays a summary of installed timers and their device numbers. In the following, the DAQCard-6062E contains 2 timers and the CPU board contains 5. Programs can obtain SIGNAL device numbers from the GETDEV command – see below.



The Properties button displays properties of an individual timer. Following is the Properties display for timer device 2. Its on-board device number is 1 because internal device numbers begin with 0.



GETDEV Command

For automation and platform portability, command files can obtain SIGNAL timer device numbers from the **GETDEV** command:

GETDEV devtyp mfr boardnum [devnum]

where:

devtyp = device type = AC, PL, DIO, TIMER
mfr = manufacturer = DART, DT, NI, WAVE
boardnum = installed board no. from this manufacturer [beginning with 1]
devnum = device no. on this board [beginning with 1]

boardnum allows for multiple installed boards from the specified manufacturer, and uses the board numbers displayed in I/O | Configure | Boards. *devnum* allows for multiple devices on the specified board, which is common among DIO and TIMER devices. GETDEV displays device information and returns the SIGNAL device number in UVAR11. Thus "getdev timer ni 1 2" sets UVAR11 to the SIGNAL device number for the second timer on the first NI board.

TIMER Command

All timer operations are performed by the **TIMER** command with various options. *devnum* is the sequential device number assigned by SIGNAL – see "SIGNAL Timer Devices and Operations".

```

TIMER devnum { OPEN }
{ CLOSE }
{ START }
{ STOP }
{ REF }
{ STAT }
{ READ }
{ SHOW }

```

Most systems have multiple timers (from analog I/O board, timer/DIO board, and CPU board) and may use several simultaneously, for example, to trigger a process and count the resulting digital events. Therefore the **TIMER** command specifies a particular timer device in *devnum*.

The following sections describe TIMER command options.

OPEN: Open Timer

TIMER OPEN reserves the specified hardware timer. It reads TIMMOD and TIMTASK and creates the requested timer task type. It then reads and stores the values of all timer parameters required by the task. **Note:** task parameters are only read once, so **timer parameters must be set before issuing TIMER OPEN**.

CLOSE: Close Timer

TIMER CLOSE releases the specified hardware timer and all task resources. **The timer must be closed before it can be opened for another task.**

START: Start Timer

Depending on the task and task attributes, **TIMER START** may start the timer, arm it for a trigger, prepare it for gate-enabled counting, etc. See the "Operation" section of each task description for details.

STOP: Stop Timer

TIMER STOP stops the current timer operation while keeping the timer open. All task attributes are retained so another TIMER START can be issued. With the exception of CLOCK | WAIT and WAITREF, **the timer must be stopped before it can be started again.**

REF: Reference Time

TIMER REF sets a reference time for use by the Wait Reference task in Clock mode. See that task for details.

STAT: Timer Status

TIMER STAT returns timer status in UVAR11 and timer resolution (in usec) in UVAR12.

UVAR11	0 = timer done, stopped, or aborted by error 1 = timer active = counting or waiting for a trigger or gate signal
12	Timer resolution (usec)

READ: Read Timer

TIMER READ gets timer values and status information from the timer. Status information is returned as with TIMER STAT.

Depending on the task, **TIMER READ** may return a single value or an array of values. All time values are in seconds. The number of values is returned in UVAR13.

- Single values are returned in UVAR14. Single values are returned by Clock | Freerun or by Event Counting and Duration tasks when TIMQTY = 1.
- Value arrays are returned in the time buffer specified by the **TIMBUF** parameter. Value arrays are returned by Clock|Gate Time and by Event Counting and Duration tasks when TIMQTY=0 or is greater than 1.

Depending on the task and TIMQTY setting, **TIMER READ** may return immediately or wait for measurement results. See "Buffered Measurements" for background and task descriptions for details.

SHOW: Show Timer Properties

TIMER SHOW displays the timer manufacturer and model number and loads timer properties – such as on-board device no., bit width, minimum and maximum clock rate, etc. – into result variables:

UVAR11	Device index in I/O table (1-based)
12	Board index within manufacturer (1-based)
13	Device index within board (0-based)
14	Bit width
15	Min clock rate
16	Max clock rate
17	Triggering available
18	Total no. installed timer devices
ULAB11	Manufacturer name
12	Manufacturer code
13	Model name

Summary of Timer Parameters

The following table shows all SIGNAL timer parameters, definitions and default values. The same parameters may configure multiple tasks and their meanings can vary between tasks, as shown.

Parameter	Description	Purpose	Default
TIMMOD	Timer mode	Select timer operating mode	CLOCK
TIMTASK	Timer task	Select task within timer mode	FREERUN
TIMBUF	Result buffer	Destination buffer for timestamps, event counts, or duration values	1
TIMCYCLE	Pulse duty cycle	Duty cycle of output pulse sequence (0-1)	0.5
TIMDELAY	Onset delay (sec)	Delay interval before measurement period Delay between start command and pulse output	0
TIMDEVAUX	Aux device connection	Connect output of specified device to aux input of current timer	0 = none
TIMDEVGAT	Gate device connection	Connect output of specified device to gate of current timer	0 = none
TIMDEVIN	Input device connection	Connect output of specified device to input of current timer	0 = none
TIMDUR	Duration (sec)	Wait interval, counting period, pulse duration	1.0
TIMERR	Timer error handling	Response to timer errors	STOP
TIMPOLGAT	Gate signal polarity	Polarity of event stream to be timestamped, counting control signal, gate and aux triggers, pulse signal to be measured	POS
TIMPOLIN	Start trigger polarity	Polarity of start trigger or event stream to be counted	POS
TIMPOLOUT	Output pulse polarity	Polarity of output trigger or output pulse signal	POS
TIMQTY	Quantity	Events to timestamp, events to count, gate pulses to count over, pulses/periods/intervals to measure, output pulses	1
TIMRATE	Requested timebase frequency Pulse output frequency	Determines timer resolution and max duration Sets pulse rate	Note 1
TIMRTN	Timer return mode	Whether TIMER START returns immediately or waits for task to complete	IMMED
TIMTRIG	Trigger mode	Optional hardware start trigger	IMMED

Notes

1. Default timebase frequency depends on available timebase values, which varies with timer model. See "Timer Clock Rate: Resolution vs. Maximum Duration" for details.

Using the TIMER Command

Time Units

All time values provided to the timer (e.g., parameters such as TIMDUR or TIMDELAY) or read from the timer by TIMER READ are **expressed in seconds**.

Timer Examples

1. Software-Polled Event Time: CLOCK | FREERUN

Application: Reaction time measurement

Tested: 6-2-07

This example uses free-running clock mode to record the time of occurrence of observational (rather than digital) events. The timebase is initiated manually in sync with an experimental stimulus, then polled manually, yielding a typical accuracy of 1-10 msec. The example uses only built-in components (CPU timer, sound chip and keyboard), so no external hardware or timer cabling is required. The result is a simple and versatile experiment engine.

The example performs an auditory discrimination experiment. The stimulus is presented and the subject hits one of several keys indicating (for example) perceived sound type. The program records the choice and the reaction time between stimulus onset and subject response. Using built-in components and software polling eliminates the need for an electronic button box, timer board and timer cabling.

Operation

- Optionally connect analog Out 1 to a audio monitor.

```
new int tdev 1                      ! set timer device
new int xkey                         ! key code
new float xtime                       ! elapsed time (sec)
r t 1                                ! read stimulus sound file
myfile
set timmod clock                      ! set timer mode & task
set timtask freerun
timer tdev open                       ! open timer
timer tdev start                      ! start timebase
set plrtm immed                      ! return immediately after starting playback
pl t 1                                ! start playback
twait key                            ! wait for keystroke
xkey = uvar12                         ! save key code
timer tdev read                       ! read timer
xtime = uvar14                         ! save timestamp
timer tdev close                      ! release timer
typ xkey xtime                        ! display key code and timestamp
```

2. Hardware-Polled Event Time: CLOCK | GATETIME

Application: Precision reaction time measurement

Tested: 6-2-07

This example is similar to "Software-Polled Event Time" but uses a hardware timer, electronic button box, hardware triggering and hardware timer control to increase measurement accuracy from msec to microsecond accuracy.

The button box contains one button and emits a TTL level when that button is pressed. It is connected to the timer Gate input.

The example performs an auditory perception experiment. The stimulus is presented and the subject presses the button each time a certain feature is perceived. The timer timestamps each button press.

Button times are synchronized with playback onset by starting both with a hardware trigger signal. This trigger could be generated externally and connected to the timer's Aux input, or provided by another timer (see the "Pulse Output" example) and connected internally via the TIMDEVAUX parameter.

Operation

- To test the example, generate a 1 Hz pulse train on another timer to simulate multiple button presses and connect internally to the timer **gate** as described in the example "Pulse Sequence Output".
- Alternately, generate a pulse train on external hardware and connect physically to the appropriate timer **gate** terminal (such as PFI-9 or PFI-4 on NI boards).
- Provide the external trigger or disable the TIMTRIG and PLTRIG code lines.
- Optionally connect analog Out 1 to a audio monitor.

```
new int tdev 1           ! set timer device
r t 1                  ! read stimulus sound file
myfile
set timmod clock        ! set timer mode & task
set timtask gatetime
set timqty 0            ! measure timestamps indefinitely
set timpolgat pos      ! polarity of event signal
set timbuf 2            ! store timer results in buffer 2
set timtrig ext         ! start timer on hardware trigger
timer tdev open          ! open timer with specified task params
timer tdev start         ! arm timebase for trigger
set plrtn wait          ! wait for playback to finish, then return
set pltrig ext          ! start playback on hardware trigger
pl t 1                  ! arm playback for trigger
timer tdev read          ! read timestamps
timer tdev close         ! release timer
list t 2                ! display timestamps
```

3. Fixed Period Event Count: COUNT | PERIOD

Application: Frequency measurement

Tested: 6-2-07

This example uses event counting over a fixed period (1 sec) to measure the frequency of a digital signal such as the shaft monitor on a rotating machine. The measurement period is controlled with microsecond accuracy. The frequency measurement has a mathematical accuracy of $(\pm \frac{1}{2} \text{ event}) / (\text{events in measurement period})$, so accuracy increases with period duration.

Note that COUNT | PERIOD uses two timers – the requested timer N and timer N+1 to control the measurement period.

Example 3A: Single period count

Operation

- To test the example, generate a 1000 Hz pulse train on another timer to simulate an event stream and connect internally to the timer **input** as described in the example "Pulse Sequence Output".
- Note that COUNT | PERIOD uses two timers – the requested timer N and timer N+1 to control the measurement period, so timer 2 is not available for pulse train generation.
- Alternately, generate a pulse train on external hardware and connect to the timer **input** terminal (such as PFI-8 or PFI-3 on NI boards).

```
new int tdev 1           ! set timer device
new float xfreq          ! measured frequency (Hz)
set timmod count         ! set timer mode & task
set timtask period
set timqty 1             ! make 1 measurement
set timdur 1.0            ! period = 1 sec
set timpolgat pos        ! polarity of event signal
timer tdev open          ! open timer with specified task params
timer tdev start          ! start timer
timer tdev read           ! wait for period, then read timer
xfreq = uvar14            ! save event count
timer tdev close          ! release timer
typ xfreq                ! display event count = frequency
```

Example 3B: Repeated period count

The following program runs the above example program once per minute for 24 hours for continuous process monitoring, using a CPU timer to control the loop interval. Each frequency measurement is immediately checked for validity (e.g., process malfunction) and then stored.

Operation

- Operating instructions are the same as the previous example.

```
new int tdev1 1           ! set timer devices
new int tdev2 3
new int iloop              ! loop variable
new float xfreq            ! measured frequency (Hz)
***** set up CPU timer as loop timer
set timmod clock          ! set timer mode & task
set timtask waitref
set timdur 60               ! period = 60 sec
timer tdev2 open           ! open timer 2 with specified task params
***** set up hdwr timer for event counting
set timmod count           ! set timer mode & task
set timtask period
set timqty 1                 ! make 1 measurement
set timdur 1.0                ! period = 1 sec
set timpolgat pos           ! polarity of event signal
timer tdev1 open             ! open timer 1 with specified task params
***** counting loop
```

```

timer tdev2 ref           ! set loop time origin
doloop 1000 iloop 1 1440   ! 1 meas per min for 24 hrs
timer tdev1 start          ! start timer
timer tdev1 read            ! wait for period, then read timer
xfreq = uvar14              ! save event count
typ xfreq                  ! display event count = frequency
timer tdev1 stop             ! stop timer
[store, validate, issue alarm, etc.]
timer tdev2 start          ! wait for remainder of loop interval
enddo 1000
!***** end of event counting loop
timer tdev1 close           ! release timers
timer tdev2 close

```

4. Sequential Interval Durations: DUR | SEMIPER

Applications: Measure duration of perceived subject conditions
Measure duty cycle of periodic signals

Tested: 6-2-07

This example measures the intervals between successive rising and falling edges in an event stream and reports the complete sequence of "high" and "low" intervals. One application is behavioral experiments in which the response button is pressed for the duration of a perceived condition, and another is monitoring the duty cycle of a periodic signal, such as in industrial process control.

The example uses a hardware timer, electronic button box, hardware triggering and hardware timer control to deliver microsecond accuracy. The button box contains one button and emits a TTL level when that button is pressed. It is connected to the timer Gate input.

The example performs an auditory perception experiment. The stimulus is presented and the subject presses and holds the button for the duration over which a certain feature is perceived. The button can be pressed multiple times during the stimulus. The timer measures the duration of each button press and the intervals in between.

Button times are synchronized with playback onset by starting both with a hardware trigger signal. This trigger could be generated externally and connected to the timer's Aux input, or provided by another timer (see the "Pulse Output" example) and connected internally via the TIMDEVAUX parameter.

Note the similarity of the following program to the example "Hardware-Polled Event Time", even though the functionality is different.

Operation

- To test the example, generate a 1 Hz pulse train with a 25% duty cycle on another timer to simulate multiple button presses and connect internally to the timer **gate** as described in the example "Pulse Sequence Output".
- Alternately, generate a pulse train on external hardware and connect physically to the appropriate timer **gate** terminal (such as PFI-9 or PFI-4 on NI boards).
- Provide the external trigger or disable the TIMTRIG and PLTRIG code lines.
- Optionally connect analog Out 1 to a audio monitor.

```

new int tdev 1           ! set timer device
r t 1                   ! read stimulus sound file
myfile
set timmod dur           ! set timer mode & task

```

```

set timtask semiper
set timqty 0                                ! measure timestamps indefinitely
set timpolgat pos                            ! polarity of event signal
set timbuf 2                                  ! store timer results in buffer 2
set timtrig ext                               ! start timer on hardware trigger
timer tdev open                             ! open timer with specified task params
timer tdev start                            ! arm timebase for trigger
set plrtn wait                                ! wait for playback to finish, then return
set pltrig ext                               ! start playback on hardware trigger
pl t 1                                         ! arm playback for trigger
timer tdev read                                ! read timestamps
timer tdev close                               ! release timer
list t 2                                       ! display timestamps

```

5. Differential Event Time: DUR | TWOTRIG

Application: Precision reaction time measurement

Not tested

This example measures the interval between the two digital events. One application is to measure reaction time between stimulus onset and a button press from the subject.

The example uses a hardware timer, electronic button box and hardware timer control to deliver microsecond accuracy. The button box contains two buttons, one to trigger the stimulus and the other for subject response and two output TTL lines representing the two buttons. One line is connected to the timer Aux input and the other to the timer Gate input.

The example performs an auditory perception experiment. The subject presses the playback button to initiate playback and start the timer, then presses the response button when a certain feature is perceived. The timer measures the time difference between the two button presses.

Note: TWOTRIG tasks are supported only NI M-series and TIO-series boards. This capability can also be achieved by using CLOCK | GATETIME with hardware triggering. See the example "Hardware-Polled Event Time".

Operation

- To test the example, generate a 1 Hz pulse train with a 25% duty cycle on another timer to generate the two triggers and connect internally to the timer **aux** and **gate** as described in the example "Pulse Sequence Output".
- Alternately, generate two sequential triggers externally and connect to the appropriate timer **aux** terminal (such as PFI-10 or PFI-11 on NI boards) and **gate** terminal (such as PFI-9 or PFI-4 on NI boards).
- Connect this signal also to the playback trigger input, or disable the PLTRIG code line.
- Optionally connect analog Out 1 to a audio monitor.

```

new int tdev 1                                 ! set timer device
new float xtime                                ! time difference (sec)
r t 1                                           ! read stimulus sound file
myfile
set timmod dur                                ! set timer mode & task
set timtask twotrig
set timpolgat pos                            ! polarity of event signal
timer tdev open                             ! open timer with specified task params
timer tdev start                            ! arm timebase for trigger

```

```

set plrtn wait           ! wait for playback to finish, then return
set pltrig ext           ! start playback on hardware trigger
pl t 1                   ! arm playback for trigger

timer tdev read          ! read time difference
xtime = uvar14            ! save timestamp
timer tdev close          ! release timer

typ xtime                ! display time difference

```

6. Pulse Output: SIGOUT | PULSE

Applications: Generate gate signal for external device control
Generate trigger signal for system synchronization

Tested: 6-2-07

This example generates a single output pulse of precise duration and timing. This can be used to control an external system, for example, to deliver a stimulus for a precise interval or enable data collection. Another use is providing a trigger signal (perhaps delayed by a precise interval) to synchronize an external system. Pulse duration and timing deliver microsecond accuracy.

Example 6A: Single pulse output

The following example outputs a single 1-second pulse.

Operation

- Connect timer output to an oscilloscope for monitoring, if desired.

```

new int tdev 1             ! set timer device
set timmod sigout          ! set timer mode & task
set timtask pulse

set timdur 1.0              ! pulse duration
set tempolout pos           ! pulse polarity
set timrtn wait              ! wait for pulse to complete

timer tdev open             ! open timer with specified task params
timer tdev start             ! start pulse output
timer tdev close              ! release timer

```

Example 6B: Cascaded pulse output with double triggering

The following example cascades two pulse outputs from two timers to provide a synchronized timing offset between two systems. The first timer delivers a short trigger that starts playback and the second timer. The second timer waits 1 sec, then delivers a 1-second duration pulse which controls the time and duration of stimulus delivery in an external system. Task parameters are repeated between the two timers for clarity. The first timer output is connected to the other timer and playback device entirely in software (via TIMDEVGAT and PLTRIGDEV parameters – NI devices only) -- **no cabling is required!**

Operation

- Connect timer outputs 1 and 2 to an oscilloscope for monitoring, if desired.
- Optionally connect analog Out 1 to a audio monitor.

```

new int tdev1 1             ! set timer devices
new int tdev2 2

r t 1                      ! read stimulus sound file
myfile

new float stimDur

```

```

bd t 1
stimDur = uvar14 / 1000           ! stimulus duration (sec)

***** set timer mode & task

set timmod sigout
set timtask pulse

***** set up timer 1 for short trigger pulse

set timdelay 0                   ! no onset delay
set timdur 0.001                 ! pulse duration
set timpolout pos               ! pulse polarity
set timtrig immed               ! begin output on TIMER START

timer tdev1 open                ! open timer 1 with specified task params

***** set up timer 2 for 1-sec pulse after 1-sec delay

set timdelay 1.0                 ! begin output 1 sec after receiving trigger
set timdur 1.0                   ! pulse duration
set timpolout pos               ! pulse polarity
set timtrig ext                 ! start timer on hardware trigger
set timdevgat 1                  ! get trigger from timer 1 output

timer tdev2 open                ! open timer 2 with specified task params
timer tdev2 start               ! arm timer 2 for trigger

***** set up playback on external trigger

set plrtn immed                ! return immediately after arming playback
set pltrig ext                 ! start playback on hardware trigger
set pltrigpol pos              ! start on rising edge of trigger
set pltrigdev 1                 ! get trigger from timer 1
pl t 1                          ! arm playback for trigger

***** perform the process

timer tdev1 start               ! start pulse output on timer 1
twait stimDur                   ! wait for playback to complete
timer tdev1 close               ! close timers
timer tdev2 close

```

7. Pulse Train: SIGOUT | PULSESEQ

Applications: Hardware loop control
External clock signal

Tested: 6-2-07

This example generates a pulse train which can be used to trigger another timer (or other process) for precision loop timing, or as a clock signal for an analog I/O process (see the example "Sample Clock for Analog I/O").

A pulse train can be used to simulate an input event stream for other examples in this guide. To do this:

- Insert the shaded lines before the main timer code in the desired example. This will generate the pulse train.
- Add a parameter to connect the pulse train output internally to the main timer.
 - To connect to the timer **input** (Clock mode), add "set timdevin xtdev".
 - To connect to the timer **gate** (Count and Duration mode), add "set timdevgat xtdev".
 - To connect to the timer **aux** (Two-Trigger tasks), add "set timdevaux xtdev".
- Add "timer xtdev close" after the main timer code to release the pulse train timer when done.

For example:

[shaded code]	! set up & start pulse output timer
[setup code for main timer]	! from desired example
set timdevgat xtdev	! connect pulse output to main timer
[execution code for main timer]	! from desired example
timer xtdev close	! close pulse output timer

Operation

This example will output a 1000 Hz pulse train with a 25% duty cycle for 10 secs.

- Connect timer 1 output to an oscilloscope for monitoring, if desired.
- Run the example.

new int xtdev 1	! set timer device
set timmod sigout	! set timer mode & task
set timtask pulseseq	
set timrate 1000	! pulse rate = 1000 Hz
set timcycle 0.25	! duty cycle = 25%
set timqty 0	! generate pulse train until stopped
timer xtdev open	! open timer with specified task params
timer xtdev start	! start pulse train
twait 10	! wait 10 sec
timer xtdev close	! close timer

8. Sample Clock for Analog I/O: SIGOUT | PULSESEQ

Application: Generate pulse train to achieve non-standard analog I/O sample rate

Tested: 6-2-07

This example generates a pulse train clock signal for an analog I/O process, and connects it internally to the analog I/O unit. This can provide an exact sample rate – such as 44,100 Hz – unavailable from the internal analog I/O clock.

Background: internal analog I/O clocks are typically restricted to sample rates arising from integer divisors of the timebase. Timers, on the other hand, can generate clock signals with arbitrary pulse rates and predictable jitter. For example, a 44,100 Hz pulse train can be generated from an 80 MHz timebase with a mean error of 0 and max jitter of $\pm 0.5 * (\text{pulse rate}) / (\text{timebase frequency}) = \pm 0.03\%$, while the nearest internal analog I/O sample rate is 44.053, an fixed error of 0.11%.

Operation

- Connect an analog signal to analog input 1.
- Connect timer 1 output to an oscilloscope for monitoring, if desired.
- Run the example.

new int tdev 1	! set timer device
***** set up clock signal & start it	
set timmod sigout	! set timer mode & task
set timtask pulseseq	
set timrate 44100	! pulse rate = 44,100 Hz
set timcycle 0.5	! duty cycle = 50%
set timqty 0	! generate pulse train until stopped
timer tdev open	! open timer with specified task params
timer tdev start	! start pulse train
***** perform analog I/O	

```

set acdur 5           ! acquire for 5 secs
set acrate 44100      ! so SIGNAL can label the data
set acclkdev tdev      ! get clock signal from timer
ac t 1                ! perform acquisition
g t 1                ! display acquired signal
timer tdev close       ! close timer

```

9. Synthesized Pulse Train: CLOCK | WAITREF

Application: Generate pulse train without using a hardware timer

Tested: 6-2-07

This example produces an output square wave without using a hardware timer. Instead, a built-in CPU timer controls a digital I/O line. The timer provides a wait interval that times the low and high portions of the waveform. An asymmetrical duty cycle can be achieved using two timers to provide two different wait intervals. This can be useful to provide a pulse train when all hardware timers are in use.

Frequency stability is provided by WAITREF, which absorbs the accumulated time error every half-cycle. Frequency stability and maximum bandwidth are limited by software latency (see "Software vs. Hardware Timer Control"), which can extend the wait interval when the task is not receiving CPU attention. The TIMERR parameter should be set to prevent the task from halting due to occasional timer underflow.

Pulse rates up to 1000 Hz can be achieved with reasonable quality. For a precision square wave, use a hardware timer as described in the "Pulse Train Output" example. This process runs in foreground, unlike a hardware timer, and may need to be integrated with other I/O processes.

Operation

- Connect timer output to an oscilloscope for monitoring, if desired.
- Run the example.
- Hit <esc> to halt the timer.

```

new int tdev 1          ! set timer device
set timmod clock        ! set timer mode & task
set timtask waitref
set timdur 0.005         ! half cycle = 5 msec => output freq = 100 Hz
set timerr continue      ! don't halt on timer underflow
timer tdev open          ! open timer with specified task params
dio 1 open out           ! open DIO device 1 for output
timer tdev ref            ! set time origin
label 1000
dio 1 out 0 1             ! set DIO line 0 to HI
timer tdev start          ! wait for remainder of this half-cycle
dio 1 out 0 0             ! set DIO line 0 to LO
timer tdev start          ! wait for remainder of this half-cycle
goto 1000

```

Hardware Timer Environment

Overview of Timer Models

The SIGNAL Timer System supports timers from three manufacturers – Data Translation (DT), National Instruments (NI), and the high-precision timer built into most CPU boards. This section provides an overview of their features and capabilities. For details, see the task descriptions, which describe the support provided by different manufacturers and models for different functionalities and timer parameters.

Data Translation

Data Translation boards provide limited timer capabilities. Following are some important limitations:

- Timers cannot count their own internal timebase, so a single timer cannot provide a free-running clock (CLOCK | FREERUN). SIGNAL provides this capability with DT boards by internally configuring a spare timer to generate a pulse train output to be used as a clock signal by the desired counter. The user must physically connect the pulse output to the counter's clock input.
- Counting operations can be gated but not triggered.
- Pulse output operations can be triggered in a limited manner from the gate line.

DT boards allow finely-graded timer clock rates, so the user can optimize timer resolution / max duration. See "Timer Clock Rate: Resolution vs. Maximum Duration".

National Instruments

National Instruments timer capabilities range from good to extensive, depending on timer family. Following are points regarding these differences. See "Timer Model Capabilities" for details.

- Timers on E-series analog I/O boards do not support hardware-triggering on CLOCK and COUNT tasks (see individual task descriptions). These tasks can only be software triggered. However, CLOCK | FREERUN and GATETIME tasks can be hardware-triggered by generating a hardware-triggered pulse train output and connecting this to the timer's clock input via TIMDEVIN.
- Timers on E-series analog I/O boards do not provide two-trigger operations, so two-trigger event counting and duration measurement are not supported on these devices.
- M-series analog I/O boards and TIO (timer I/O) boards are nearly identical, and **support all functionalities in the SIGNAL Timer System**.

NI boards provide only two timer clock rates, so the timer resolution / max duration tradeoff must be considered. See "Timer Clock Rate: Resolution vs. Maximum Duration".

High-Precision CPU Timer

Intel Pentium and AMD Athlon processors contain one 64-bit counter known as a "performance counter". This counter increments continuously at the clock rate of the CPU and can be read but not reset. It is available under Windows XP and probably Vista.

SIGNAL wraps this counter into five virtual timers – see the figure in "SIGNAL Timer Devices and Operations" – using the counter value as a timebase. These timers are of course software-only – lacking trigger, gate, and pulse output capabilities – so they support a few of the Clock mode functionalities and none of the other SIGNAL timer modes.

The counter's high clock rate and high bit width provide high resolution and long duration. For example, a 3 GHz CPU provides 0.30 nsec resolution and max duration > 100 years. However, because the timer can only be read in software, it is subject to Windows latency, reducing its overall accuracy. See "Timer Resolution and Accuracy" and "Software vs. Hardware Timer Control" for background.

The timer has a potential problem on dual-processor systems (such as Pentium D chip released 2007). Because the counter belongs to the processor rather than the timer process, a context switch on a dual-processor could move the timer process to the other processor. If the counter registers on the two processors are not identical, timer values would only be valid on the processor active when the timer was initialized via TIMER START. This has not been investigated.

Timer Model Capabilities

SIGNAL supports timers from three manufacturers – Data Translation (DT), National Instruments (NI), and the built-in CPU timer – but not all models support all functionalities of the SIGNAL Timer System. Following is a summary of timer characteristics by manufacturer and hardware family within the manufacturer. N/S = not specified by manufacturer.

DT timer families

<u>Hardware family</u>	<u>Width (bits)</u>	<u>Avail clock rates</u>	<u>Max duration</u>	<u>Accuracy</u>	<u>Triggered CLOCK & COUNT</u>	<u>Triggered SIGOUT</u>	<u>TWOTRIG tasks</u>
3010/31016 analog I/O board	32 (1)	(2)	215 sec	100 ppm	No	Yes	No
9834 USB analog I/O unit	32	(2)	215 sec	100 ppm	No	Yes	No

Notes

1. SIGNAL presents 32-bit timers created by cascading two native 16-bit timers.
2. Clock rate is user-specifiable, derived from 20 MHz internal timebase.

NI timer families

<u>Hardware family</u>	<u>Width (bits)</u>	<u>Avail clock rates</u>	<u>Max duration</u>	<u>Accuracy</u>	<u>Triggered CLOCK & COUNT</u>	<u>Triggered SIGOUT</u>	<u>TWOTRIG tasks</u>
E-series analog I/O board [STC timer chip]	24	100 kHz 20 MHz	168 secs 840 msec	100 ppm	No	Yes	No
M-series analog I/O board [STC2 timer chip]	32	100 kHz 20 MHz 80 MHz	11.9 hrs 215 sec 54 sec	50 ppm	Yes	Yes	Yes
660x timer I/O board [TIO timer chip]	32	100 kHz 20 MHz 80 MHz	11.9 hrs 215 sec 54 sec	50 ppm	Yes	Yes	Yes

Notes

1. NI timers present 2 or 3 fixed clock rates, depending on timer model.

High-Precision CPU timer

<u>Hardware family</u>	<u>Width (bits)</u>	<u>Avail clock rates</u>	<u>Max duration</u>	<u>Accuracy</u>	<u>Triggered CLOCK & COUNT</u>	<u>Triggered SIGOUT</u>	<u>TWOTRIG tasks</u>
	64	(1)	> 100 yrs	N/S	No	No	No

Notes

1. Clock rate is fixed but depends on operating system and CPU chip. Pentium chips under Windows XP provide timer clock rate equal to CPU clock rate, e.g., 3.2 GHz on a P4/3.2 GHz CPU.

2. This timer has no external connections and provides only Clock mode functionalities: Elapsed Time, Wait, and Wait Reference.

Hardware Issues

Timer Signal Connections

Basic timer signals are input, gate, output, and auxillary input. The **input** line delivers a periodic timebase (clock signal) or sporadic external events. The **gate** line provides a count enable or triggers timer operations such as start counting or save current count. The **output** line can deliver a pulse, pulse train, or output trigger. The **auxillary input** line is used with the input line to measure two-signal time differences. The following table shows the terminology used by each manufacturer for these signals. N/A = signal not available.

	SIGNAL terminology			
	Input	Gate	Output	Auxillary
Manufacturer				
DT 9834	Clock	Gate	Out	N/A
DT 3010/3016	Clock Input	External Gate	Counter Output	N/A
NI	Source	Gate	Out	Aux
CPU	N/A	N/A	N/A	N/A

Timer Signal Polarity

The polarity of input, gate and output signals can be specified separately, for design flexibility and compatibility with external systems. These are specified by the **TIMPOLIN**, **TIMPOLGAT**, and **TIMPOLOUT** parameters, respectively:

```
SET { TIMPOLIN } { POS }
    { TIMPOLGAT } { NEG }
    { TIMPOLOUT }
```

Incoming timer signals consist of input and gate signals, and may be either edges (e.g., triggers) or levels (e.g., gate enable signals). The same polarity names refer to edges and levels. POS means a low-to-high edge or a high level and NEG means a high-to-low edge or a low level. Thus TIMPOLIN POS configures a triggered task to respond to a low-to-high trigger.

Outgoing timer signals may be triggers or output pulses. TIMPOLOUT configures the polarity of output triggers (e.g., the Interval task in Clock mode) or output pulses (all tasks in Signal Output mode). POS and NEG configure output triggers as just described. For pulse outputs, POS configures the idle and active states of the pulse(s) to be low and high, respectively, while NEG configures idle and active states to be high and low, respectively. Thus TIMPOLOUT NEG configures a pulse output that begins high, goes low for the specified pulse duration, then returns to high.

Timer Resolution and Accuracy

Timebase resolution expresses the granularity of the internal timebase and equals $1 / (\text{clock rate})$. Thus a 20 MHz timebase has a granularity of 50 nsec, i.e., measured intervals and generated pulses are resolved to 50 nsec clock ticks. The resolution of this timebase is ± 25 nsec.

Timebase accuracy (also referred to as stability) represents the accuracy of the internal timebase: how many tick does it actually deliver per second? Accuracy is expressed as fractional error in $\pm \text{ppm}$ (parts per million). Thus an accuracy of ± 50 ppm ($\pm .005\%$) means that a 20 MHz timebase will actually tick

between 19.999 MHz and 20.001 MHz. Measured or generated durations are subject to the same fractional error, as shown in the following example.

The **total accuracy** of a measurement or pulse generation is the sum of these two error components. Following are examples for two quite different durations (which might involve measurement or pulse output), based on a 20 MHz timebase. Short intervals are dominated by timer resolution, while long ones are dominated by timebase accuracy.

- 1) **10 usec duration:** Resolution = ± 25 nsec. Accuracy = $10 \text{ usec} * .005\% = \pm 0.5 \text{ nsec}$. Therefore, total accuracy = ± 25 nsec.
- 2) **1 sec duration:** Resolution = ± 25 nsec. Accuracy = $1 \text{ sec} * .005\% = \pm 50 \text{ usec}$. Therefore, total accuracy = ± 50 usec.

Timer Clock Rate: Resolution vs. Maximum Duration

Timers present a tradeoff between timer resolution and maximum duration, both determined by the timer's underlying timebase or clock rate.

Timer resolution equals $1 / (\text{clock rate})$. Thus increasing clock rate improves resolution.

But clock rate also affects the maximum duration the timer can handle. An N-bit timer can count up to $2^{** N}$, for a maximum duration of $(2^{** N}) / (\text{clock rate})$. Thus a 24-bit timer with a 20 MHz timebase has a max duration of 840 msec. So increasing clock rate decreases the timer's maximum duration. "Timer Model Capabilities" shows the max duration of different timer models.

The timer's "maximum count" places a limitation on each timer mode. In Clock mode, it determines the maximum elapsed time. In event counting mode, it is the maximum number of events that can be counted. For duration measurement, it determines the longest measureable pulse, period, or semi-period. For pulse generation, it is the longest initial delay, low-level time, or high-level time in any single pulse.

Clock rate is specified by the TIMRATE parameter, which should be specified to optimize the resolution-duration tradeoff. DT timers can synthesize a variable clock rate and will select the rate = $(20 \text{ MHz} / M)$, where M is an integer, closest to TIMRATE.

NI timers provide two or three fixed clock rates – 100 kHz, 20 MHz and 80 MHz (M-series and TIO series only) – so the tradeoff is less flexible. SIGNAL will select the 100 kHz timebase for TIMRATE's up to 100 kHz **or when the timer has less than 32 bits** (see "Timer Model Capabilities"), the 20 MHz timebase for TIMRATE's between 100 kHz and 20 MHz, and the 80 MHz timebase for TIMRATE's above 20 MHz. Arbitrary clock rates can be obtained by creating a pulse output sequence on a spare timer and connecting this output to the clock input of the target timer.

TIMRATE is specified only for Clock and Duration Measurement tasks. Event Counting tasks do not use an internal timebase – their "clock" is the event stream they're counting. Signal Output tasks select their timebase internally, based on the maximum duration (initial delay, low duration, and high duration) in the signal being generated.

Software vs. Hardware Timer Control

Timers can be started, stopped and read by software commands or hardware triggers. Hardware triggers require additional driver setup and usually additional cabling (see "Connecting Multiple Timers" below). Hardware control increases timer accuracy from msec to usec range. This is due to software command latency, which on current 3 GHz CPU's is due less to instruction execution times (typically < 1 usec) than to the multi-tasking nature of Windows.

A multi-tasking operating system executes multiple tasks simultaneously, including many background tasks invisible to the user. Since a single-CPU system can actually execute only one task at a time, it switches the processor successively among the multiple "active" tasks, providing a short time slice to each (typically 15 msec in Windows XP). Software commands in a particular task therefore have a

potential latency of tens of msec, representing the delay until that task's next execution cycle. In practice in Windows, this latency is more often only a few msec.

For a timer, this latency might occur if a different task is executing when an event of interest occurs. Thus if a software command to read the current time doesn't execute for 10 msec, the time value read will be inaccurate by 10 msec. Hardware trigger signals connect directly to the timer's hardware logic and have a typical latency of 20-50 nsec.

The SIGNAL timer provides software and/or hardware timer control, depending on the particular task, and this document refers to "software accuracy" (a few msec) and "hardware accuracy" (< 1 usec).

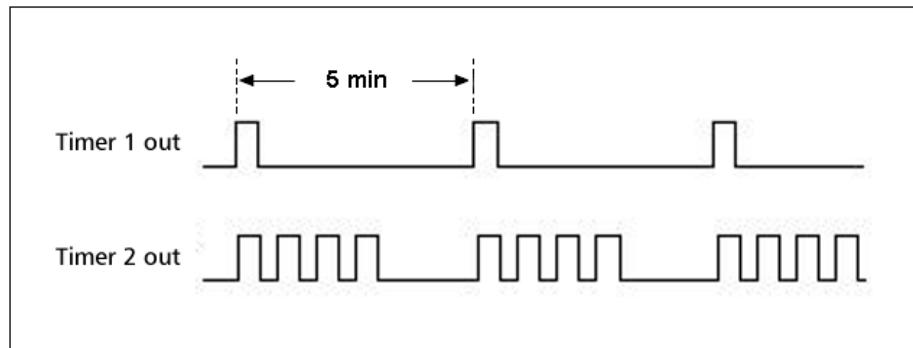
Cascading Timers

Timers can be cascaded for multiple purposes.

One is to produce a timer with greater bit width. For example, SIGNAL internally cascades two 16-bit timers on the DT3010/3016 board to produce a 32-bit timer, because the 16-bit counting range is too small. In this application, one timer is configured to produce an output pulse after counting its entire range, then this pulse is used as the input clock to another timer. The 32-bit count of the cascaded timer is the bitwise concatenation of the two 16-bit count registers.

Another purpose is to produce complex timing patterns, by using the pulse output of one timer as the input, trigger or gate for another. For example, one timer producing a 1-sec output pulse once per minute can be used to gate another timer to perform one frequency measurement per minute. As another example, one timer producing a pulse every 5 minutes can be used to trigger another timer to produce a specified number of pulses at a specified pulse rate, then that signal can be used as a sampling clock for analog input, to perform an acquisition every 5 minutes, as shown in the figure.

National Instruments timers can be connected in software without physical cables – see "Connecting Timers and I/O Modules on NI Boards" for details.



Connecting Timers and I/O Modules on NI Boards

National Instruments analog and timer boards provide a powerful capability. Timing signals can be routed between various analog, digital, and timer terminals via software commands without physical cables. For example, the output of one timer can be routed to the gate of another timer to trigger it or to the clock input of an analog I/O board to provide a custom sample clock for an analog I/O process. See "Cascading Timers" for details.

Timer outputs can be connected to terminals on other timers using the **TIMDEVAUX**, **TIMDEVGAT** and **TIMDEVIN** parameters:

```
SET { TIMDEVAUX } devnum
    { TIMDEVGAT }
    { TIMDEVIN }
```

connects the output of timer device *devnum* to the input (TIMDEVIN), gate (TIMDEVGAT), or auxillary input (TIMDEVAUX) of the current timer. "SIGNAL Timer Devices and Operations" describes timer device numbers.

Timer outputs can be connected to the trigger input of an NI analog I/O board using the **ACTRIGDEV** and **PLTRIGDEV** parameters:

```
SET { ACTRIGDEV } devnum  
      { PLTRIGDEV }
```

Timer outputs can be connected to the sample clock input of an NI analog I/O board using the **ACCLKDEV** and **PLCLKDEV** parameters:

```
SET { ACCLKDEV } devnum  
      { PLCLKDEV }
```

NI boards support most but not all possible software connections. For example, the NI-6062 can route Timer0 output to Timer1 input or Timer1 gate, and it can route Timer1 output to Timer0 gate but **not** Timer0 input. Also, some NI software connections are internally routed through on-board devices (such as another timer), rendering that device unavailable to other processes.

The NI device manager program Measurement & Automation Explorer (MAX) provides a routing matrix showing supported connections and internally utilized devices. To display this, launch MAX, open Devices and Interfaces | NI-DAQmx Devices, select your device, then click on the Device Routing tab at the bottom of the screen. The figure below shows the routing matrix for the NI-6062.

Notes

1. When connecting two timers in software, the **source timer must be opened (via TIMER OPEN) before the destination timer**, so the destination timer can read the source timer's attributes.
2. NI-6062 cannot route Timer1 output to Timer0 input.
3. Data Translation analog and timer boards do not support software connections, and must be connected by physical cables.



Software Issues

Buffered Measurements

Duration measurement and event counting tasks can be configured to process multiple events in the same event stream – for example, to measure the widths of a succession of pulses in a pulse train (or sequence of button presses). Measurements (or event counts) are stored in an internal buffer and then read by the program.

The **TIMQTY** parameter sets the number of events to process. **TIMQTY > 0** will process the specified number of events, while **TIMQTY = 0** processes the event stream indefinitely. The internal buffer can hold 10,000 measurements.

The **TIMER READ** command reads the measurement buffer:

- If **TIMQTY = 0**, **TIMER READ** can be issued multiple times, and will return immediately with all measurements acquired since **TIMER START** or the previous **TIMER READ**.
- If **TIMQTY > 0**, **TIMER READ** can be issued once, and will wait for **TIMQTY** measurements to complete, then return their values.

If the timer buffer contains only one value, TIMER READ returns it in UVAR14; otherwise multiple values are returned in the SIGNAL buffer specified by **TIMBUF**. TIMER READ loads the following UVAR's:

UVAR11	Timer status 0 = timer done, stopped, or aborted by error 1 = timer active = counting or waiting for a trigger or gate signal
12	Timer resolution (usec)
13	No. values returned
14	Measurement value, if single value

The following tasks perform buffered measurements:

- **Clock | Gate Time** stores timestamps of multiple events.
- **Measure | Pulse, Measure | Period, Measure | Semi-Period, and Measure | Two-Trigger Interval** measure multiple event durations.
- **Count | Fixed Period, Count | Gated Count, and Count | Two-Trigger** count the events in multiple time periods.

Note that Event Counting tasks return only one value (an event count) and therefore don't use TIMQTY.

Buffered measurement tasks are performed in "background", allowing the calling program to perform other tasks at the same time. That is, TIMER START initiates the task and returns immediately. Later, TIMER READ can read the measurement data. See "Process Synchronization" for details.

Process Synchronization

It is often desirable to have timer operations – such as waiting for a pulse in order to measure its duration – run in the "background" while the main program performs "foreground" tasks such as displaying selections to the user, polling digital lines for a button press, or calculating a new signal for playback. Organizing and temporally coordinating program tasks into background and foreground processes is **process synchronization**.

Process synchronization concerns the timing relationships between multiple processes within a program. Once initiated, an **asynchronous** process runs in background, leaving the program free to continue its foreground execution flow. A **synchronous** process executes in foreground and program flow pauses while the process completes.

A SIGNAL timer command is synchronous if it does not return until the operation it initiates has completed, and asynchronous if it initiates an operation, then returns to the program while the operation runs independently.

For example, the free-running timebase task (CLOCK | FREERUN) is asynchronous. The following program starts the timebase, which runs in background as a pollable time reference, then waits for a button press in foreground, then reads the timebase:

<start timebase task>	! asynchronous => continue execution
<wait for button press>	! synchronous => wait for button press
<get current time from timebase>	! synchronous => return with timebase value
<display time value>	

In the following example, timer 1 measures the duration of a pulse delivered by timer 2. Pulse measurement (DUR | PULSE) and pulse output (SIGOUT | PULSE) are asynchronous operations, while the read command (TIMER READ) is synchronous, returning only after data has been read. This allows the program to arm the measurement on timer 1 (asynchronous – the measurement will wait in background for the pulse to arrive), initiate the pulse output on timer 2 (asynchronous – the command will return before the pulse completes), wait for the pulse measurement to complete (synchronous), then display the measured value.

<start pulse measurement on timer 1>	! asynchronous => continue execution
<start pulse output on timer 2>	! asynchronous => continue execution
<read pulse measurement>	! synchronous => wait for measurement to complete
<display measurement value>	

In SIGNAL, most timer start commands are asynchronous – TIMER START starts the task and returns immediately to the calling program. Exceptions are CLOCK | WAIT and CLOCK | WAITREF. TIMER READ, on the other hand, can be asynchronous or synchronous, depending on the task attributes. Each task description describes the return behavior of TIMER START, and "Buffered Measurements" describes the possible return behaviors of TIMER READ.

Timer Overflow and Underflow

Timer overflow occurs when the timer counts beyond the capacity of its counter register, "rolls over" and continues counting again from 0. Subsequent timer readings will be invalid. This can be avoided by selecting timers with appropriate capacity and clock rates that allow for required durations.

An N-bit timer can count up to $2^{**} N$, for a maximum duration of $(2^{**} N) / (\text{clock rate})$. Thus a 24-bit timer with a 20 MHz timebase has a max duration of 840 msec and can easily overflow at that clock rate. Thus clock rate should be selected for each task to accommodate the maximum expected timebase duration, measurement duration, output pulse width or (rarely) event count. "Timer Model Capabilities" lists the duration capacities of supported timer models, also available from I/O | Configure | Timer | Properties on the SIGNAL menu. See "Timer Clock Rate: Resolution vs. Maximum Duration" for further technical discussion.

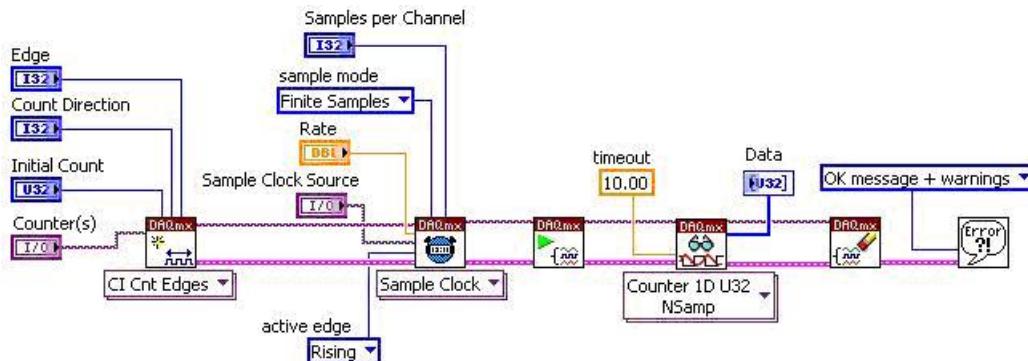
NI timers monitor their count register for overflow. SIGNAL checks this status at every timer reading and reports overflow as a run-time error, so that invalid "rolled-over" readings are never returned. Unfortunately, **DT timers** do not provide this capability and overflows are not reported. **CPU timers** have a duration capacity of > 100 years and are unlikely to overflow in the user's lifetime.

Timer underflow occurs in Clock | Wait and Clock | Waitref tasks, when the first reading from the timer already exceeds the requested wait interval. This can occur in Waitref tasks with very short wait intervals (a few msec) where multiple commands intervene between TIMER REF and TIMER START. SIGNAL checks for condition and reports a run-time error. This condition is not always objectionable and users can set the **TIMERR** parameter to CONTINUE to ignore these errors:

```
SET TIMERR { STOP      }
            { CONTINUE }
```

SIGNAL Timer System vs. Labview

The SIGNAL Timer System supports all National Instruments timers, and its coherent, high-level language can replace a complex Labview topology with a few program lines, and provide easy graphical and numerical access to the result data. For example, the following Labview program performs "Finite Buffered Event Counting" <http://zone.ni.com/devzone/cda/tut/p/id/5404>, storing the cumulative number of edges on the source line at each rising edge on the gate line. If the source is a timebase, the result will be the timestamps of the gate edges.



The following SIGNAL program performs the same function. It timestamps the first 100 gate events, stores the timestamps in a SIGNAL buffer, and displays the timestamps on the screen.

set timmod clock	! set timer mode & task
set timtask gatetime	
set tempolgat pos	! gate event stream polarity
set timqty 100	! timestamp first 100 gate edges
set timbuf 1	! store timestamps in SIGNAL buffer 1
timer 1 open	! open timer
timer 1 start	! start measuring
timer 1 read	! wait for 100 events to be measured
timer 1 close	! close timer
list t 1	! display list of timestamps

Future Developments

TIMER STAT: Timer Status

Timer status (active vs. done) is not really implemented. Currently implemented only for Clock | Freerun, which delivers no information (user already knows whether timebase is running).

Future possibilities:

- New mode = Clock | Period. Clock runs for specified period then stops. Asynchronous version of Clock | Wait, i.e., runs in background vs. foreground. Program can poll task status. Not a high priority because Clock | Freerun can be used to provide same functionality by polling its elapsed time.
- For buffered tasks, TIMER STAT rtns no. vals currently in measurement buffer in UVAR13. This allows user to poll the process asynchronously, in contrast to TIMER READ which (when TIMQTY > 0) will wait until TIMQTY measurements are ready.

DT Implementation

Many timer functionalities are listed as available on DT but not supported. This is a low priority because DT timer capabilities are so limited. Serious timer applications require NI hardware, the preferred platform for the SIGNAL Timer System.