Experiment Maker™

Stimulus Presentation & Control System

User Guide

Version 5.0
September, 2007
Revised March, 2016

Engineering Design

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.

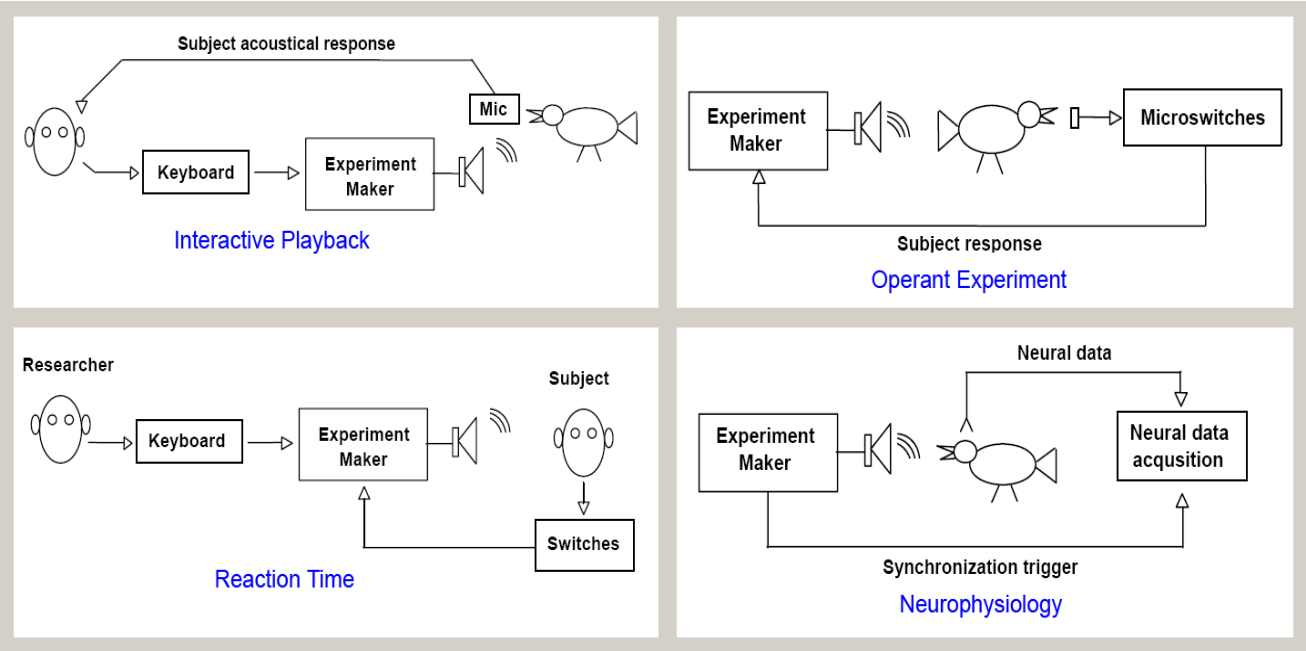
4. EXCEPT AS SET FORTH IN THE EXPRESS WARRANTY ABOVE, THE SOFTWARE IS PROVIDED WITH NO OTHER WARRANTIES, EXPRESS OR IMPLIED. THE VENDOR EXCLUDES ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

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



1. Introduction

Experiment Maker[tm] turns SIGNAL into a programmable, automated, experiment control system. With SIGNAL and Experiment Maker, the researcher can construct presentation and control systems that would otherwise require months of programming.

Experiment Maker provides tools for recording and presenting acoustic signals in real-time, precisely timing experimental events, interacting with human and animal subjects through digital switches and lights, and synchronizing with other systems through digital control signals.

Experiment Maker 5 offers capabilities for real-time behavioral, operant, and neurophysiological experiments – including simultaneous acquisition and playback, digital I/O, advanced timer measurement and control, and keyboard control. Applications include:

- Simultaneous acquisition and playback
- Precisely timed repetitive playbacks
- Adaptive playbacks
- Dynamic stimulus selection
- Reaction time measurement
- Automatic event counting
- Frequency measurement
- Trigger and pulse train generation
- Integrated acoustic and visual testing

Stimuli can be presented automatically to the subject by the computer, and stimulus presentation can be varied according to subject responses and experimental conditions.

Stimulus choice, subject response, and experimental conditions can be logged for later analysis.

Experiment Maker is implemented as an additional set of SIGNAL commands and is fully integrated with SIGNAL's analysis and programming capabilities. Analytically complex, automated experiments can be constructed as SIGNAL programs which combine Experiment Maker's interactive and real-time features with SIGNAL's analysis, execution, and programming commands.

Representative Applications

"Experiment Maker Applications" later in this guide describes six representative Experiment Maker applications:

- Interactive Playback
- Adaptive Playback
- Reaction Time
- Close-Loop Operant Testing
- Human Subject Test Station
- Neurophysiological Stimulus Delivery

How to Use this Guide

This Guide describes Experiment Maker, its commands, operation and applications. Experiment Maker provides capabilities for data acquisition, playback, digital I/O, precision timing, event synchronization, and combinations of these as simultaneous I/O processes. In practice, complete experiment programs are a combination of Experiment Maker commands for presentation and control and SIGNAL commands for measurement, analysis, programming, and display. For background on SIGNAL commands, see the **SIGNAL Reference Guide**.

2. Installing Experiment Maker

Experiment Maker (EM) is embedded in SIGNAL and requires no additional installation. However, your SIGNAL license must be activated for EM. If you purchased EM with SIGNAL, EM will be preactivated. If not, EM can be activated by email. See "Upgrading Your SIGNAL License" in the **SIGNAL Reference Guide**.

3. Simultaneous I/O Processes – Tools

Foreground and Background Tasks

Experiment Maker can run an unlimited number of I/O processes simultaneously. These include acquisition, playback, digital I/O, and timing tasks such as timebase, interval measurement, and trigger or pulse train output.

Simultaneous processes are possible due to Experiment Maker's ability to run I/O processes in "background". SIGNAL is fundamentally a command processor and the stream of commands processed (whether typed into the console, read from a comfile, or generated by menu dialogs) is called the **command stream**. SIGNAL has only one command stream. This stream represents the SIGNAL **foreground** and operations that occur in this command stream – such as screen displays, selection dialogs, actively polling digital lines for a button press, or calculating a playback stimulus – are **foreground tasks**. A **background task** runs – typically on a hardware device – independent of the SIGNAL command stream. The command stream initiates a background task, then resumes foreground processing.

Experiment Maker can launch an unlimited number of background I/O tasks. A program might launch one or more background tasks (such as an acquisition and/or playback), then (in the foreground) interact with the user, perform calculations, and monitor and interact with the background tasks.

The following sections describe some illustrative foreground/background task structures and the associated Experiment Maker commands. The next chapter describes several applications in detail.

Background Acquisition Tools

A background acquisition process runs independent of the program that initiated it, leaving that program free to launch another I/O process (such as playback), record and store other experiment data, make time measurements on experiment variables, monitor subject response, etc - all while the acquisition is executing. In practical terms, with a background acquisition, the acquisition command returns immediately after starting the acquisition, so the control program can process other tasks.

Background acquisition is specified via the **ACRTN** parameter. The **AC** command normally waits for acquisition to complete before returning to the program (ACRTN WAIT). Setting ACRTN IMMED directs AC to return immediately after starting acquisition, leaving the process running in background.

The **ACTRIG** parameter allows acquisition to be initiated immediately (ACTRIG IMMED) or by an external TTL trigger signal (ACTRIG EXT). External triggering allows for precise time synchronization (< 1 μsec) with external processes. ACTRIG is described in the **SIGNAL Reference Guide**.

The **AC STAT** command polls the status of a background acquisition to determine if the process has completed. A background acquisition can be halted via **AC STOP**. The AC command is described in the **SIGNAL Reference Guide**.

Background Playback Tools

A background playback process runs independent of the program that initiated it, leaving that program free to launch another I/O process (such as acquisition), record and store other experiment data, make time measurements on experiment variables, monitor subject response, etc - all while the playback is executing. In practical terms, with a background playback, the playback command returns immediately after starting the playback, so the control program can process other tasks.

Background playback is specified by the **PLRTN** parameter. The **PL** command normally waits for playback to complete before returning to the program (PLRTN WAIT). Setting PLTRN IMMED instead returns immediately after starting (or arming) playback, leaving the process running in background.

The **PLTRIG** parameter allows playback to be initiated immediately (PLTRIG IMMED), by PL START from an armed state (PLTRIG CMD), or by an external TTL trigger signal (PLTRIG EXT). External triggering allows for precise time synchronization (< 1 μsec) with external processes. PLTRIG is described in the **SIGNAL Reference Guide**.

The **PL STAT** command polls the status of a background playback to determine if the process has completed. A background playback can be halted via **PL STOP**. The PL command is described in the **SIGNAL Reference Guide**.

Task Number

SIGNAL assigns a unique task number to each background I/O task when it's created and sets the **IOTASK** parameter to this number. This task number should be saved by the control program to access the task later. I/O control commands will operate on the task specified by IOTASK. To access a particular task, set IOTASK to its task number, then issue commands to start, stop, or poll the task. Task numbers begin at 1 and IOTASK will be 0 when there are no background tasks active or armed for execution.

Armed Playbacks

SIGNAL buffers are stored in floating point (vs. integer) numerical format, and playing a buffer requires significant time to convert the floating point values to integers for the D/A converter before playback can begin. Conversion time is proportional to signal length.

Some applications require that playback begin immediately after an external event such as a keypress, switch closure, or vocal response. For example, **interactive playbacks** present a sound in response to subject behavior and **operant studies** present playbacks in response to switches and other controls manipulated by the subject. This requires that signal values be converted to integer format in advance. This can be accomplished in three ways:

1) Store the signal as an integer sound file, then play the sound file. See the **R** (Read) command in the **SIGNAL Reference Guide**.
2) Initiate playback from a hardware trigger. SIGNAL will convert the signal to integer, then wait for the trigger signal. See the **PLTRIG** parameter in the **SIGNAL Reference Guide**.
3) Initiate playback from a software trigger. SIGNAL will convert the signal to integer, then wait for the trigger command. This is called an **armed playback** and is described in this section.

These approaches differ in the latency between trigger and start of playback. Latency with a hardware trigger is < 1 μsec. Latency with a software trigger is 1-10 msec due to SIGNAL instruction execution time. File playback latency is 5-10 msec due to SIGNAL instruction execution plus reading initial playback data from the file.

Note: the following functionality is not yet implemented.

Armed Playback

An **armed playback** converts the playback signal in advance, then begins playback upon receipt of a **PL START** command. Following are the steps:

1. To create an armed playback task, set PLTRIG to CMD, then issue the playback command. SIGNAL will create a playback task, convert the signal, and return the task number in UVAR11 and the IOTASK parameter. The playback is now **armed** and ready for execution.
2. To execute an armed playback task, set IOTASK to the desired task number (if multiple tasks are armed) then issue PL START. Playback will begin immediately.

Any number of armed playbacks can be created and queued for playback. Setting the **PLTASK** parameter to DISCARD deletes playback tasks after playback completes. Setting PLTASK KEEP retains playback tasks for repeated playback, for example, to select repeatedly from a battery of playback tasks. See the example. **PL CLR ALL** is then issued to delete the tasks.

Armed Playback Example

The following example creates armed playback tasks for time buffers 1, 2, and 3. The user or program then selects one and plays it immediately.

```
new int pltask1                 ! variables to hold task nos.
new int pltask2
new int pltask3
new int choice

set pltrig cmd                  ! create armed playbacks
set pltask keep                 ! retain tasks after playback

pl t 1                          ! create playback task
pltask1 = uvar11                ! save task no.
```

```
pl t 2
pltask2 = uvar11

pl t 3
pltask3 = uvar11

label 1000
choice = <1, 2, or 3 selected by user or program algorithm>

if choice eq 1 then                          ! select playback task
  set iotask pltask1
elseif choice eq 2 then
  set iotask pltask2
elseif choice eq 3 then
  set iotask pltask3
else
  goto 2000
endif

set plrtn wait                               ! wait for playback to complete
pl start                                     ! start playback

goto 1000                                    ! repeat

label 2000
pl clr all                                   ! clear all playback tasks
```

Sound File Playback

Armed playbacks can be closely approximated by playing integer sound files. Integer sound files do not require real-to-integer conversion and playback can begin almost immediately.

Sound File Playback Example

In the following example, the user or program selects one of three sound files and then plays it immediately.

```
new int choice

label 1000
choice = <1, 2, or 3 selected by user or program algorithm>

if choice eq 1 then                          ! select playback file
  set plfname "c:\myfile1"
elseif choice eq 2 then
  set plfname "c:\myfile2"
elseif choice eq 3 then
  set plfname "c:\myfile3"
else
  goto 2000
```

```
endif

set plrtn wait                  ! wait for playback to complete
pl file                         ! start playback

goto 1000                       ! repeat

label 2000                      ! done
```

COMMANDS

PL — Play back analog data

PL is documented in the **SIGNAL Reference Guide**.

SWITCHES & PARAMETERS

ACRTN — Acquisition return mode

Syntax: SET ACRTN *mode* Default: WAIT

Settings:

IMMED	Return after acquisition begins
WAIT	Return after acquisition is complete

Function: Set return mode for the AC command. WAIT waits for acquisition to complete before returning, while IMMED launches the acquisition as a "background" process, returning immediately so the "foreground" process can proceed in parallel.

Example: `>set acrtn immed` Return after acquisition begins

IOTASK — I/O task number

Syntax: SET IOTASK *num* Default: 0

Function: SIGNAL assigns a unique task number when creating a background or armed I/O task and returns this number in UVAR11 and the **IOTASK** parameter. I/O control commands operate on the task specified by IOTASK. To access a particular task, set IOTASK to its

task number, then issue commands to start, stop, or poll the task. Task numbers begin at 1 and IOTASK will be 0 when there are no tasks active in the background or armed for execution.

Example: `>set iotask 2` Direct task control cmds to I/O task #2.

PLRTN Playback return mode

Syntax: SET PLRTN *mode* Default: WAIT

Settings:

IMMED	Return after playback begins
WAIT	Return after playback is complete

Function: Set return mode for the PL command. WAIT waits for the playback to finish before returning, while IMMED launches the playback as a "background" process, returning immediately so the "foreground" process can proceed in parallel.

Example: `>set plrtn immed` Return after playback begins.

4. Simultaneous I/O Processes – Applications

The ability to perform acquisition and/or playback in background – and therefore simultaneously – opens up many powerful experimental approaches. This chapter describes several applications.

Background Playback

A background playback returns to the controlling program immediately after playback starts. The program can then launch other processes, perform calculations, store results, etc. while the playback is executing.

Goal: Perform a background playback.

Example: Launch a background playback, then perform foreground processing until the playback completes.

```
set plrtn immed                    ! start playback in background
pl t 1

label 1000                         ! wait for playback to complete
[perform some processing]
```

```
pl stat                          ! get playback status
if uvar11 ne 0 goto 1000         ! not done yet
```

Simultaneous Acquisition and Playback

Experiment Maker allows the researcher to perform simultaneous, synchronized acquisition and playback processes. The two processes may be synchronized in software or hardware, depending on the required timing accuracy. See "Hardware vs. Software Synchronization" for background, and compare the examples. Applications include:

- Present an auditory stimulus while simultaneously acquiring the subject's acoustic response. See "Adaptive Playbacks" below.
- Present an auditory stimulus while simultaneously acquiring synchronized neurophysiological or other experimental data.

Goal: Perform simultaneous, synchronized acquisition and playback.

Example 1 – software synchronization: Run AC and PL tasks in background with a foreground task allowing the operator to close the trial and process results. Start AC and PL processes from software for msec-level synchronization.

```
set plrtn immed                  ! perform playback in background
pl t 1

set acrtn immed                  ! perform acquisition in background
ac t 2

wait                             ! operator hits <enter> to proceed
[process results]
```

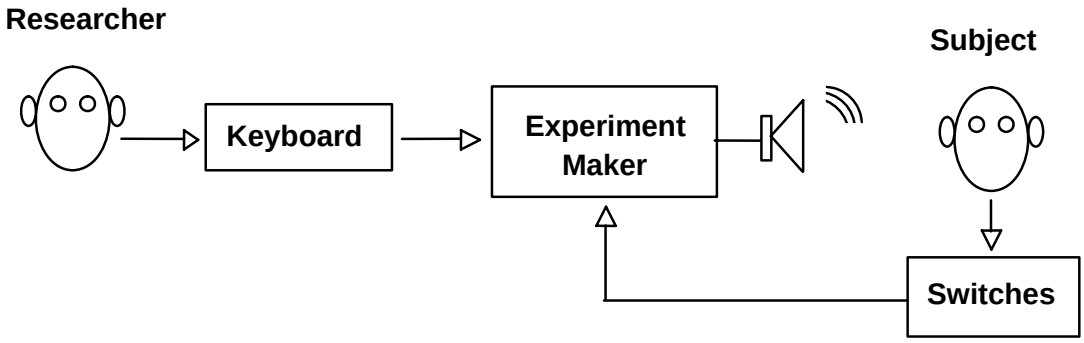
Example 2 – hardware synchronization: Same as Example 1, but start AC and PL processes from a hardware trigger for μsec-level synchronization.

```
set pltrig ext                   ! start on external trigger
set plrtn immed                  ! perform playback in background
pl t 1                           ! arm for trigger

set actrig ext                   ! start on external trigger
set acrtn immed                  ! perform acquisition in background
ac t 2                           ! arm for trigger

[issue start trigger]
wait                             ! operator hits <enter> to proceed
[process results]
```

Example 3: Run Example 1 as an automated experiment by running AC in foreground as task control. See the "Adaptive Playbacks" example.

Reaction time

Experiment Maker allows the researcher to measure **reaction time** – the time between stimulus onset and subject response – with software accuracy (msec) or hardware accuracy (μsec).



Goal: The researcher selects an auditory stimulus using the keyboard, Experiment Maker presents the stimulus, records stimulus onset time, then waits for the subject to activate a switch signaling stimulus recognition. EM then records switch activation time and calculates the latency between stimulus onset and key press.

Implementation: A playback task and timer task are launched synchronously in background to provide the stimulus and a synchronized timebase. A foreground task monitors the keyboard and when a keystroke occurs gets a timestamp from the timer task. The experiment can be implemented using software or hardware synchronization, which present a tradeoff between hardware complexity and measurement accuracy. See "Hardware vs. Software Synchronization" for background.

Example 1 – software synchronization: This approach uses only built-in components (CPU timer, sound chip and keyboard), so external hardware and cabling are not required. The timebase is software-sychronized with stimulus playback and software-polled for a typical accuracy of 1-10 msec.

```
set timmod clock                ! create timer task = free-running timebase
set timtask freerun
timer 1 open                    ! open timer & start it
timer 1 start

set plrtn immed                 ! start playback in background
pl t 1

twait key                       ! wait for keystroke

timer 1 read                    ! read timer
xtime = uvar14                  ! get timestamp
timer 1 close                   ! release timer
```

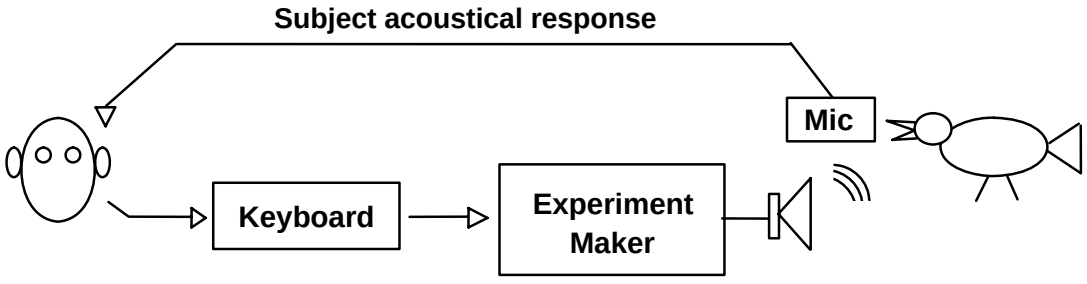
Example 2 – hardware synchronization: This approach uses a hardware timer, a human subject button box with TTL output, hardware triggering and timer cabling. An external start

trigger (which could be supplied by another timer) is connected to the D/A and timer, to hardware-sychronize stimulus playback and timebase. The timebase is hardware-polled for an accuracy of < 1 μsec. Button box output is connected to timer gate. TIMER READ waits for the first timestamp signaling the first button press.

```
set timmod clock                ! create timer task = gated timestamps
set timtask gatetime
set timqty 1                    ! measure one button press
set timpolgat pos               ! button box output polarity
set timtrig ext                 ! start timer on hdwr trigger
timer 1 open                    ! open timer & arm for hardware trigger
timer 1 start

set plrtn immed                 ! start playback in background
set pltrig ext                  ! start playback on hardware trigger
pl t 1                          ! arm for trigger

[issue start trigger]

timer 1 read                    ! wait for timestamp
xtime = uvar14                  ! get timestamp
timer 1 close                   ! release timer
```

Interactive Playbacks

In an **interactive playback** experiment, the researcher presents an auditory stimulus, observes subject response, then selects the next stimulus from a predetermined set based on that response. The goal is to investigate the subject's perception by varying stimulus selection in directed ways. An interactive playback puts the researcher at the center of the stimulus selection "loop". "Adaptive Playbacks" presents an automated approach to the same paradigm, replacing the researcher's observation and selection by computer-based data acquisition and a stimulus selection algorithm.



Goal: Select the next playback stimulus from a predetermined stimulus set based on subject response.

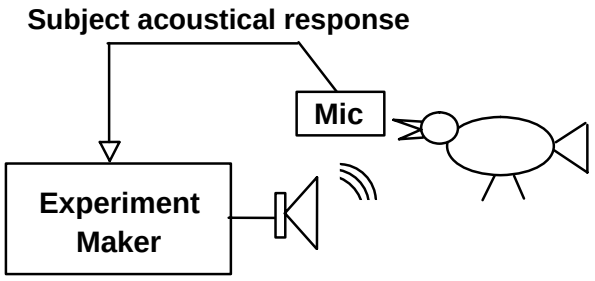
Example: Play one of 40 sound buffers selected by operator keystroke. The keyboard is used as a 40-button panel, where different keys represent different playback buffers. See "Keystroke Events" for keyboard layout and other technical details.

```
new int mybuf
set plrtn immed                    ! perform playback in background
mybuf = <initial stimulus no.>
label 1000
pl t mybuf                         ! play selected sound buffer
twait key                          ! wait for keystroke
mybuf = uvar12                     ! buffer number = keyboard location
goto 1000                          ! play new selection
```

Adaptive Playbacks

An **adaptive playback** experiment presents an auditory stimulus and simultaneously records the subject's vocal response. The program then analyzes the response and selects or generates a new stimulus. By coding subject response as "responsive" vs. "non-responsive", an iterative process can map perceptual boundaries by incrementally varying the stimulus along multiple dimensions (pitch, duration, etc.). For example, the original stimulus can be varied successively in pitch upward and downward to determine perceptual frequency range.

Adaptive playbacks are powerful because their automation can gather a large amount of data unattended, allowing researchers to map many perceptual dimensions at fine scale.



Goal: Calculate the next playback based on the subject's acoustic reponse to the current playback.

Implementation: Launch playback and acquisition tasks simultaneously, to present a stimulus and record subject's overlapping response. When these processes finish, the program analyzes the response, calculates the next stimulus, and performs another trial. The paradigm loops through incremental stimulus variations along multiple dimensions until the entire perceptual space has been mapped.

Example: Set acquisition duration to playback duration + post-stimulus duration, then run AC in foreground as task control, to create an automated presentation loop.

```
label 1000
new float xdur
bd t 1                             ! get PL signal duration
xdur = uvar14 / 1000 + 5           ! AC duration = PL duration + 5 sec
set acdur xdur                     ! set AC duration
```

```
set plrtn immed                    ! start playback in background
pl t 1

set acrtn wait                     ! start acquisition in foreground
ac t 2

[process acquired signal & generate next stimulus]
goto 1000                          ! do another trial
```

5. Digital I/O

Experiment Maker provides the capability to read and write to the digital I/O ports on analog I/O and timer-DIO boards. The researcher can use these digital lines to interact with the research subject and to synchronize or exchange information with external events and systems. Here are some DIO applications:

- Digital outputs can be used to switch visual stimuli on and off in a timed fashion, synchronized if desired with auditory stimuli from the PL command.
- Subject responses can be read by digital inputs from microswitches (animal or industrial process) or push buttons (human), and these responses can be time-logged using Experiment Maker's precision timing capabilities (see below).
- Stimulus synchronization signals and stimulus selections can be read from an external control system through digital inputs.
- External devices such as programmable attenuators can be controlled through digital outputs.

Hardware Support

SIGNAL DIO commands can utilitize DIO hardware on analog I/O boards as well as dedicated timer-DIO boards. Two board manufacturers are supported – Data Translation (DT) and National Instruments (NI).

Hardware vs. Software DIO Architecture

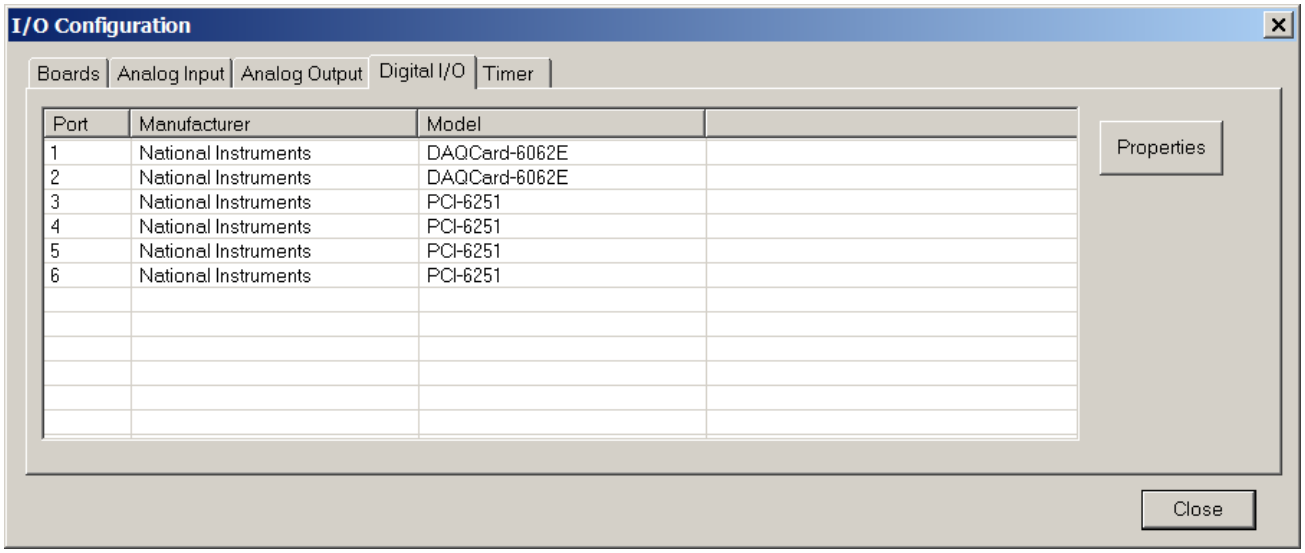
Hardware DIO Architecture

Physical DIO devices are organized into multiple DIO **ports**, where a port is a set of digital **lines** that can be read or written individually (line-level I/O) or collectively (port-level I/O). A line is a single TTL input or output, and hardware ports typically contain 8 or 16 lines. Ports have **directionality** – input and output – which is usually software configurable. I/O boards vary in the number of ports and lines per port. A line is addressed relative to its port, e.g., line 2 of port 3.

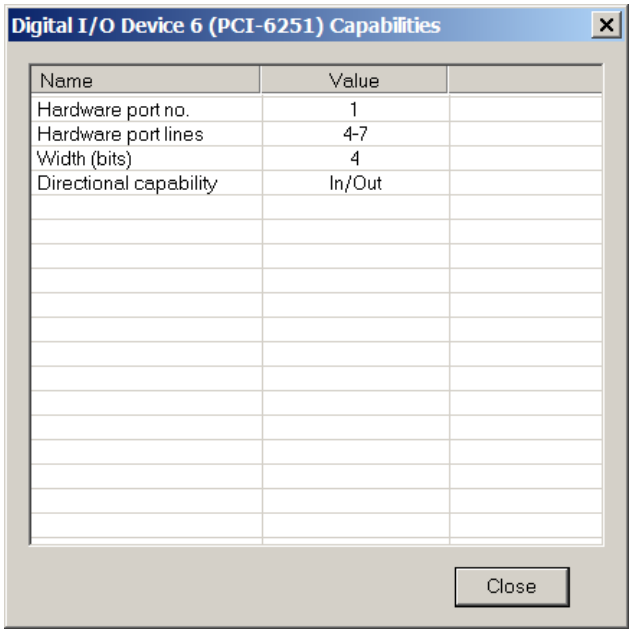
Software DIO Architecture

In order to present a uniform interface across different manufacturers and board models, SIGNAL presents a virtual **software DIO architecture** to the user. This is also organized into ports and lines, but in order to maintain consistency across all board models, software port numbers and sizes may differ from the physical hardware of a particular model. For example, SIGNAL port numbers begin at 1 and increase consecutively across **all** DIO devices in the system, and SIGNAL port widths are typically 4 lines rather than 8 or 16 (to accommodate the narrowest physical ports).

To see the software ports available in SIGNAL, select I/O | Configure | Digital I/O from the menu. The following shows that SIGNAL has configured 6 virtual ports from the DIO devices available on two analog I/O boards (DAQCard-6062E and PCI-6251):



To see the properties of a particular port, click Properties:



This shows that SIGNAL port 6 is 4 bits wide, uses hardware lines 4-7 of hardware port 1 on the PCI-6251 board, and can be configured for input or output.

You can use the DIO SHOW command to load similar information for a specified port into result variables for use in programs (see the DIO command definition).

Digital Levels

A low TTL level on the digital input (DIO IN) is returned to SIGNAL as 0 and a high TTL level is returned as 1. Similarly, on output (DIO OUT), a 0 value is output as a low TTL level, while any non-zero value (positive or negative) is output as a high TTL level.

DIO Command

Digital I/O is performed by the **DIO** command. Every DIO command includes a port number specifying which port to control. Software port numbers are assigned by SIGNAL at startup and are sequential **beginning with 1**. They can be obtained from I/O | Configure | Digital I/O on the menu (see the figure above) or the GETDEV command.

SIGNAL can read or write one digital line or all lines in the selected port. Line numbers are sequential **beginning with 0**. Data is read into a variable and written from a variable or numerical constant. To read or write a single digital line, specify a line number. For example, "dio 1 in 2 lineval" reads line 2 from port 1 into variable *lineval*. To read or write the entire port, specify the keyword ALL. For example, "dio 1 in all portval" reads the entire port 1 into variable *portval*.

Port-wide I/O can be thought of as an arithmetic operation in which the port lines are read or written as a binary number. This can be used to exchange numerical data (such as stimulus selection or programmable attenuator gain) with another system.

The DIO command provides the following operations:

- **OPEN** reserves the specified digital port for the specified operation (input or output). Opening a port for output resets the output lines to low.
- **CLOSE** releases the specified digital port. The port must be closed before it can be opened again.
- **IN** reads one digital line or the entire port into the specified variable.
- **OUT** writes the value in the specified variable to one digital line or the entire port.
- **SHOW** displays manufacturer and model number and loads port properties – such as on-board port no., no. digital lines, port directionality, etc. – into result variables.

To perform a digital I/O operation, first open the port (OPEN). This reserves the port and sets and validates the requested port directionality (in or out). Then perform the desired operations (IN or OUT). Then close the port (CLOSE), which releases it for reuse. The port must be closed before it can be opened again. For example:

```
dio 1 open in                    ! open port 1 for output
dio 1 in 3 line3                 ! read line 3 into variable line3
```

```
dio 1 in 4 line4          ! read line 4 into variable line4
dio 1 close               ! release port 1
```

Performing DIO using the Engineering Design NI I/O Panel

Users can perform DIO through the digital I/O ports on a National Instruments (NI) analog I/O board using an Engineering Design National Instruments I/O panel, if that panel contains DIO connectors.

Virtual DIO port width in SIGNAL is 4 bits for NI analog I/O boards and 8 bits for dedicated NI Timer-DIO boards. To determine virtual port width and on-board port bit numbers, select I/O | Configure | Digital I/O from the menu, select the desired port, and click Properties. For hardware details, see "Engineering Design National Instruments I/O Panel" below.

Following are the required panel connectors and software settings for digital I/O using an ED NI panel. For example, to read digital line 3, connect the signal of interest to DIO connector 3, then read line 3 of port 2. This corresponds to hardware lines 4-7 of hardware port 0 (see "Engineering Design National Instruments I/O Panel" below).

Line-Width DIO

Panel connection = DIO connectors 0-3

Software configuration = port 2, lines 0-3

Port-Width I/O

Panel connection = DIO connectors 0-3

Software configuration = port 2

Engineering Design National Instruments I/O Panel

The following table shows the assignment of DIO/PFI lines to the DIO, trigger, and timer connectors on the Engineering Design NI I/O panel. Not all ED NI I/O panels include DIO connectors.

Panel Connector	Hardware DIO Port		PFI Pin
Digital I/O			
Line 0	Port 0	Bit 4	
Line 1	Port 0	Bit 5	
Line 2	Port 0	Bit 6	
Line 3	Port 0	Bit 7	
Analog I/O triggers			
AC In	Port 1	Bit 0	0
PL In	Port 1	Bit 1	1
AC Out	Port 0	Bit 0	
PL Out	Port 0	Bit 1	

Timer 1			
In	Port 2	Bit 0	8
Gate	Port 2	Bit 1	9
Aux	Port 2	Bit 2	10
Out	Port 2	Bit 4	12
Timer 2			
In	Port 1	Bit 3	3
Gate	Port 1	Bit 4	4
Aux	Port 2	Bit 3	11
Out	Port 2	Bit 5	13

Notes

1. The "Hardware DIO Port" column uses **hardware port nos. (0-based) and widths (8 bits)**. SIGNAL uses software port nos. (1-based) and 4-bit virtual port widths for analog I/O boards and 8-bit virtual port widths for dedicated Timer-DIO boards.
2. The NI-6062 has only hardware port 0. Hardware ports 1 and 2 are supported on the NI-6251.
3. PFI lines 10 and higher are defined on the NI-6251 and software-assigned to timer functions. The same pins on the NI-6062 are hardwired to timer functions and may not function as PFI's.

COMMANDS

DIO — Digital I/O control

Syntax:

DIO *devnum* { OPEN } { IN }
{ OUT }
{ CLOSE }
{ IN } { *iline* } *var*
{ ALL }
{ OUT } { *iline* } *var*
{ ALL }
{ SHOW }

Function:

The DIO command controls the digital I/O ports on analog I/O and timer-DIO boards. Digital I/O operations include open, close, read, and write. Every DIO command includes a port number specifying which digital port to control. Port numbers are assigned by SIGNAL at startup and are sequential **beginning with 1**. They can be obtained from I/O | Configure | Digital I/O on the menu or the IOCFG or GETDEV commands.

DIO can read or write one digital line (DIO…*iline)* or all lines (DIO…ALL) in the specified port. See the examples. Line numbers

are sequential **beginning with 0**. Data is read into or written from a specified variable.

Port-wide I/O can be thought of as an arithmetic operation in which the port lines are read or written as a binary number, and can be used to exchange numerical data (such as stimulus index) with another system.

OPEN reserves the specified digital port for the specified operation (input or output). Opening a port for output resets the output lines to low.

CLOSE releases the specified digital port. The port must be closed before it can be opened again.

IN reads one digital line or the entire port into the specified variable.

OUT writes the value in the specified variable to one digital line or the entire port.

SHOW displays manufacturer and model number and loads port properties – such as on-board port no., no. digital lines, port directionality, etc. – into result variables.

Result Vars:

DIO SHOW

UVAR11	Port no. in I/O table (1-based)
12	Board index within manufacturer (1-based)
13	Port no. within board (0-based)
14	Low-order line no. within board (0-based)
15	Directional capability 0 = In 1 = Out 2 = In or Out
16	Port width (bits)
17	No. DIO ports in I/O table
ULAB11	Manufacturer name
12	Manufacturer abbreviation
13	Model name

Example:

```
>dio 2 open in                Open digital port for input
>dio 2 in 3 lineval           Read line 3 into lineval
>dio 2 in all portval         Read entire port into portval
>dio 2 close                  Close digital port
```

6. Timer

Experiment Maker includes a precision timing system capable of timestamping events, pausing for time intervals, measuring time durations, measuring frequency, counting events,

and generating triggers and pulse trains. The timing system can be used for the timed presentation of auditory stimuli and digitally controlled events and for time-stamping external events detected through digital input lines.

This section describes SIGNAL timer commands and options, and provides an overview of timer features, operating modes, architecture and technical operating issues. **See the SIGNAL Timer System Guide for complete coverage of timer commands, operating modes, hardware and software architecture, and technical operating issues.**

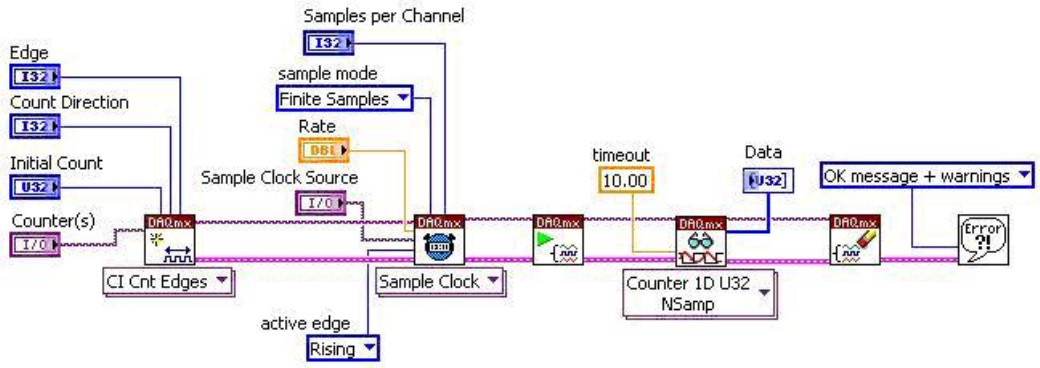
Features

Timing system features include:

- 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.



Timer Operating Modes

The SIGNAL Timer System provides 15 different timer functions or "tasks", which are organized into 4 operating modes.

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. Clock mode provides software and/or hardware timer control, depending on the task.

Clock mode provides 5 tasks: Free Running, Wait Interval, Wait Reference, Gate Time and Hardware Delay.

Event Counting

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.

Event Counting mode provides 4 tasks: Free Running, Fixed Period, Gated Count and Two-Trigger.

Duration Measurement

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).

Duration Measurement mode provides 4 tasks: Pulse Duration, Pulse Train Period, Intra-Period Intervals and Two-Trigger Interval.

Signal Output

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.

Signal Output mode provides 2 tasks: Pulse and Pulse Sequence.

Hardware Support

The timing system can utilitize hardware timers on analog I/O boards as well as dedicated timer-DIO boards. Three board manufacturers are supported – Data Translation (DT), National Instruments (NI), and the precision timer on the CPU board. Among these, NI boards support most or all of the Experiment Maker timing capabilities (depending on board model), DT supports a limited set of timing capabilities, and the the CPU timer supports only the timebase capability.

Hardware vs. Software Timer Architecture

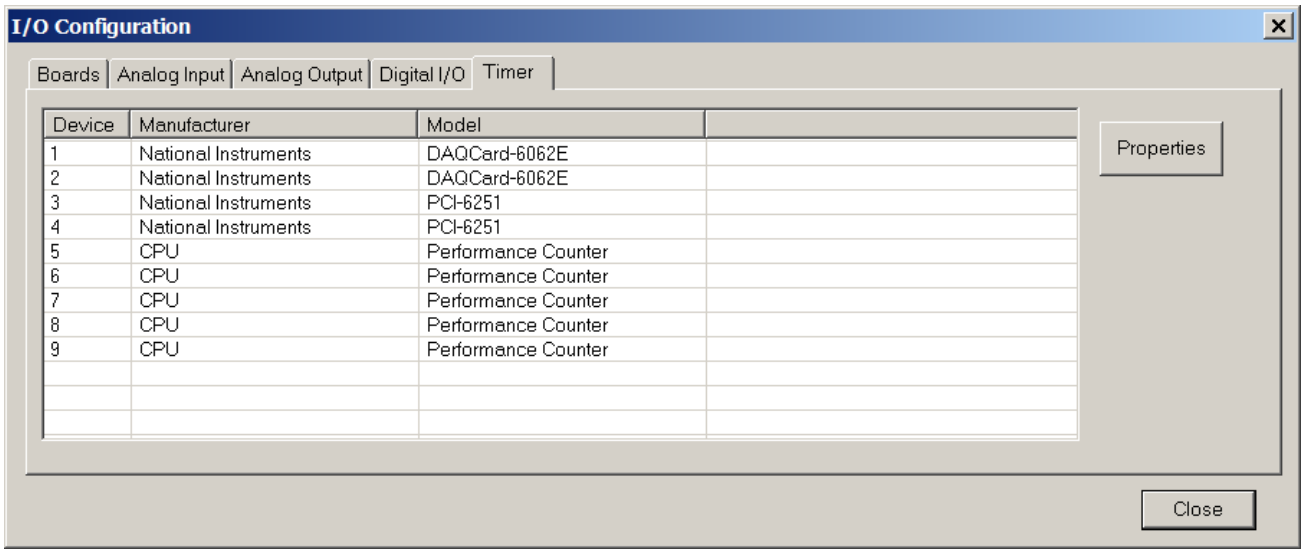
Hardware Timer Architecture

Physical timer devices are typically 16, 24 or 32 bits wide. I/O boards vary in the number of timers provided.

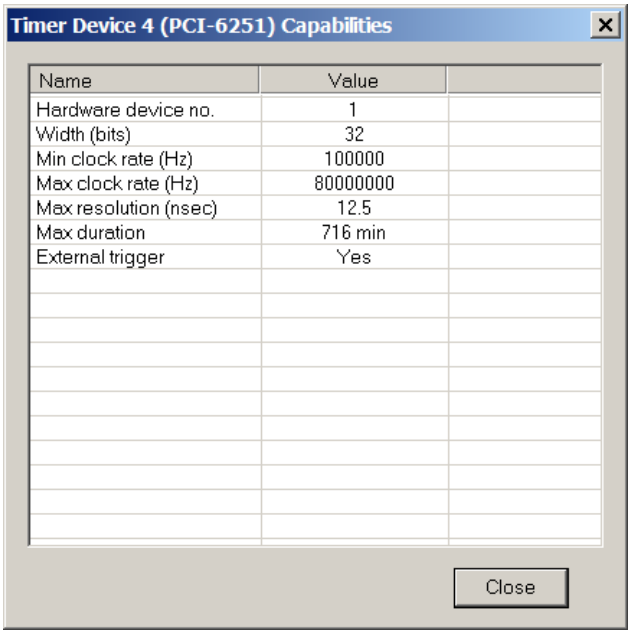
Software Timer Architecture

In order to present a uniform interface across different manufacturers and board models, SIGNAL presents a virtual **software timer architecture** to the user. In order to maintain consistency across all board models, software timer numbers and bit widths may differ from the physical hardware of a particular model. For example, SIGNAL may aggregate two 16-bit timers into one virtual 32-bit timer to achieve a wider counting range. Timer numbers begin at 1 and increase consecutively across **all** timer devices in the system.

To see the software timers available in SIGNAL, select I/O | Configure | Timer from the menu. The following shows that SIGNAL has configured 9 virtual timers from the timer devices available on two analog I/O boards (DAQCard-6062E and PCI-6251) plus the CPU precision timer ("Performance Counter"):



To see the properties of a particular timer, click Properties:



This shows that SIGNAL timer 4 is 32 bits wide, uses hardware device 1 on the PCI-6251 board, provides clock rates between 100 kHz and 80 MHz, has a maximum resolution of 12.5 nsec and maximum duration of 716 minutes, and can be externally triggered.

You can use the TIMER SHOW command to load similar information for a specified timer into result variables for use in programs (see the TIMER command definition).

Timer Clock Rate, Resolution, Accuracy and Duration

Clock rate is the timer's underlying timebase, established by an on-board crystal. The NI PCI-6251 board provides clock rates of 100 kHz, 20 MHz and 80 MHz.

Timer resolution expresses the granularity of the internal timebase and equals 1 / (clock rate). Increasing clock rate improves resolution. The typical timer timebase of 20 MHz has a granularity of 50 nsec and a resolution of ±25 nsec.

Timer accuracy represents the accuracy of the internal timebase: how many ticks does it actually deliver per second? Accuracy is expressed as fractional error in ±ppm (parts per million). Timers on the NI PCI-6251 board provide an accuracy of ±50 ppm (±0.005%).

Maximum duration is determined jointly by timer width (in bits) and clock rate. An n-bit timer counts to 2^n, for a max duration of 2^n / (clock rate). Increasing clock rate improves resolution and decreases max duration. For example, a 32-bit timer has a resolution of 25 nsec and max duration of 215 sec at 20 MHz clock rate and a resolution of 5 µsec and max duration of 11.9 hours at 100 kHz.

"Timer Model Capabilities" in the **SIGNAL Timer System Guide** summarizes the capabilities of timer devices from Data Translation, National Instruments, and the precision timer on the CPU board.

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.

1) Timer operations are performed by the **TIMER** command (see "TIMER Command" below).

2) Timer mode is selected by the **TIMMOD** parameter – CLOCK, COUNT, DUR and SIGOUT.

3) Timer task is selected by the **TIMTASK** parameter – FREERUN, WAIT, PULSE, PERIOD, etc.

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

Summary of Timer Modes and Tasks

See the **SIGNAL Timer System Guide** for details.

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

TIMER Command

Timing capabilities are provided by the **TIMER** command. Every TIMER command includes a device number specifying which timer to control. Timer numbers are assigned by SIGNAL at startup and are sequential **beginning with 1**. They can be obtained from I/O | Configure | Timer on the menu (see the figure above) or the GETDEV command.

TIMER provides the following operations:

- **OPEN** reads TIMMOD and TIMTASK and creates the requested timer task, then reads and stores the values of all task parameters, then reserves the specified hardware device. **Note:** task parameters are only read once, so set them before issuing TIMER OPEN.
- **CLOSE** releases the specified hardware timer and all task resources. The timer must be closed before it can be opened for another task.
- **START** may start the timer, arm it for a trigger, prepare it for gate-enabled counting, etc., depending on the task and task attributes.
- **STOP** stops the current timer operation while keeping the timer open. 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** sets a reference time for use by the Wait Reference task in Clock mode.
- **STAT** returns timer status and timer resolution in result variables.
- **READ** gets timer values and status information from the timer. TIMER READ may return a single value or an array of values, depending on the task. The number of values is returned in UVAR13. Single values are returned in UVAR14 and multiple values are returned in the time buffer specified by TIMBUF parameter. All time values are in seconds.
- **SHOW** displays 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.

Performing Timer Operations using the Engineering Design NI I/O Panel

Experiment Maker can access the hardware timers on a National Instruments (NI) analog I/O board using an Engineering Design National Instruments I/O panel, if that panel contains timer connectors. Typical connectors provide access to two timers with four connections per timer: Input, Gate, Auxillary input, and Output. The **SIGNAL Timer System Guide** describes their functions.

Connecting Timers Internally

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.

Example: Precision Reaction Time Measurement

This example performs an auditory reaction time experiment. A stimulus is presented and the subject presses a button each time a certain feature is perceived. The timer timestamps each button press. The timestamp is synchronized with μsec accuracy to the stimulus onset.

The example uses a hardware timer, electronic button box, hardware triggering and hardware timer control to achieve μsec 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.

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 and connected internally to the timer via the TIMDEVAUX parameter and to the playback via PLTRIGDEV.

```
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
```

COMMANDS

TIMER Timer control

Syntax:

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

Function:

The TIMER command controls the timer devices found on analog I/O boards, timer-DIO boards and the CPU chip. Timer operations include open, close, start, stop, read, etc. Every TIMER command includes a device number specifying which timer to control. Device numbers are assigned by SIGNAL at startup and are sequential beginning with 1. They can be obtained from I/O | Configure on the menu or the GETDEV command.

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

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

START may start the timer, arm it for a trigger, prepare it for gate-enabled counting, etc., depending on the task and task attributes.

STOP stops the current timer operation while keeping the timer open. 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 sets a reference time for use by the Wait Reference task in Clock mode.

STAT returns timer status and timer resolution in result variables.

READ gets timer values and status information from the timer. TIMER READ may return a single value or an array of values, depending on the task. The number of values is returned in UVAR13. Single values are returned in UVAR14 and multiple values are returned in the time buffer specified by TIMBUF parameter. All time

values are in seconds.

SHOW displays 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.

Result Vars:

TIMER STAT

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

TIMER READ

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

TIMER SHOW

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 abbreviation
13	Model name

Example:

```
>set timmod clock          Configure timer as
>set timtask freerun       free-running timebase
>timer 2 open              Open timer
>timer 2 start             Start timebase
 [perform some processing]
>timer 2 read              Get elapsed time
```

SWITCHES & PARAMETERS

TIMBUF — Timer destination buffer

Syntax: SET TIMBUF *bufnum* Default: 1

Function: Set the destination buffer number for the TIMER READ command. A timer task may produce a single result value (e.g., a pulse duration) or an array of values (e.g., durations of a sequence of pulses). Single values are returned in a user variable and arrays in a SIGNAL time buffer. TIMBUF specifies this buffer.

Example: `>set timbuf 3` Return timer values in T buf 3.

TIMCYCLE Timer duty cycle

Syntax: SET TIMCYCLE *fraction* Default: 0.5

Function: Specify the duty cycle of the output pulse sequence as a decimal fraction (0-1). For example, if the duty cycle is 0.25, each pulse will be active (e.g., high) for 25% of the cycle and inactive (e.g., low) for 75% of the cycle.

Example: `>set timcycle .25` Duty cycle = 25%.

TIMDELAY Timer delay

Syntax: SET TIMDELAY *sec* Default: 0

Function: Specify the delay in seconds before a measurement period begins (COUNT mode) or between the start command or trigger and first pulse output (SIGOUT mode).

Example: `>set timdelay 1.0` Delay = 1 sec.

TIMDEVAUX Timer auxillary input device

Syntax: SET TIMDEVAUX *devnum* Default: 0 = none

Function: Connect the output of timer device *devnum* to the auxillary input of the current timer, allowing one timer to trigger another. The connection is made internally without physical cables. **National Instruments boards only.**

Example: `>set timdevaux 2` Connect timer 2 out to aux input.

TIMDEVGAT Timer gate device

Syntax: SET TIMDEVGAT *devnum* Default: 0 = none

Function: Connect the output of timer device *devnum* to the gate of the current timer, allowing one timer to trigger or gate another. The connection is made internally without physical cables. **National Instruments boards only.**

Example: `>set timdevgate 2` Connect timer 2 out to gate.

TIMDEVIN Timer input device

Syntax: SET TIMDEVIN *devnum* Default: 0 = none

Function: Connect the output of timer device *devnum* to the input of the current timer, allowing one timer to provide the clock signal for another. The connection is made internally without physical cables. **National Instruments boards only.**

Example: `>set timdevin 2` Connect timer 2 out to input.

TIMDUR Timer duration

Syntax: SET TIMDUR *sec* Default: 1.0

Function: Specify the duration of the wait interval (CLOCK mode), counting period (COUNT mode), or pulse duration (SIGOUT mode).

Example: `>set timdur .1` Duration = 0.1 sec.

TIMERR Timer error handling

Syntax: SET TIMERR *mode* Default: STOP

Settings:

CONTINUE	Ignore timer errors
STOP	Stop on timer errors

Function: Specify the response to timer errors. This parameter currently handles only timer underflow.

Example: `>set timerr continue` Ignore timer underflow errors.

TIMMOD Timer mode

Syntax: SET TIMMOD *mode* Default: CLOCK

Settings:

CLOCK	Clock (timebase)
COUNT	Event counting
DUR	Duration measurement
SIGOUT	Signal output

Function: Set the timer's operating mode. Each mode provides several tasks, which are selected by the TIMTASK parameter.

Example: `>set timmod sigout` Select signal output mode.

TIMPOLGAT Timer gate signal polarity

Syntax: SET TIMPOLGAT *polarity* Default: POS

Settings:

POS	HI level or rising (LO-to-HI) edge
NEG	LO level or falling (HI-to-LO) edge

Function: Specify gate signal polarity, representing event stream to be timestamped (CLOCK mode), counting control signal (COUNT mode), gate and/or aux triggers (COUNT mode and DUR mode), and pulse signal to be measured (DUR mode). For example, specifying POS directs DUR|PULSE to measure the duration of the next positive pulse, i.e., between successive rising and falling edges.

Example: `>set timpolgat neg` Set polarity to LO or HI-to-LO

TIMPOLIN Timer input signal polarity

Syntax: SET TIMPOLIN *polarity* Default: POS

Settings:

POS	HI level or rising (LO-to-HI) edge
NEG	LO level or falling (HI-to-LO) edge

Function: Specify input signal polarity, representing start trigger (CLOCK and SIGOUT modes) or event stream to be counted (COUNT mode). For example, specifying POS directs COUNT mode tasks to count the number of rising edges.

Example: `>set timpolin neg` Set polarity to LO or HI-to-LO

TIMPOLOUT Timer output signal polarity

Syntax: SET TIMPOLOUT *polarity* Default: POS

Settings:

POS	HI level or rising (LO-to-HI) edge
NEG	LO level or falling (HI-to-LO) edge

Function: Specify output signal polarity, representing output trigger or pulse signal (SIGOUT mode). For example, specifying POS directs SIGOUT mode tasks to output LO-to-HI triggers or positive (HI level) pulses.

Example: `>set timpolout neg` Set polarity to LO or HI-to-LO

TIMQTY Timer quantity

Syntax: SET TIMQTY *num* Default: 1

Function: Specify the number of events to timestamp (CLOCK mode), events to count or gate pulses to count over (COUNT mode), pulses / periods / intervals to measure (DUR mode), or pulses to output (SIGOUT mode).

Example: `>set timqty 10` Quantity = 10.

TIMRATE Timer rate

Syntax: SET TIMRATE *rate* Default: [see note]

Function: Specify timebase frequency (CLOCK mode) or pulse output frequency (SIGOUT mode). **Note:** default timebase frequency depends on available timebase values, which varies with timer model.

Example: `>set timrate 1000` Frequency = 1000.

TIMRTN Timer return mode

Syntax: SET TIMRTN *mode* Default: IMMED

Settings:

IMMED	Return after timer task begins
WAIT	Return after timer task is complete

Function: Set timer return mode. WAIT waits for the timer task to complete before returning, while IMMED launches the task as a "background" process, returning immediately so the "foreground" process can proceed in parallel.

Example: `>set timrtn wait` Wait for task to complete.

TIMTASK Timer task

Syntax: SET TIMTASK *task* Default: FREERUN

Function: Select the timer task within the timer mode specified by the TIMMOD parameter. *task* values vary with timer mode. See the **SIGNAL Timer System Guide** for details.

Example:
`>set timmod count` Count no. events
`>set timtask period` in a fixed period.

TIMTRIG Timer trigger mode

Syntax: SET TIMTRIG *mode* Default: IMMED

Settings:

IMMED	Begin timer task immediately
EXT	Begin timer task after trigger signal

Function: Set timer trigger mode. IMMED begins the timer task immediately. EXT begins the task when the timer receives an external TTL trigger signal. **Not all timer boards support external triggering** – see board properties in I/O | Configure or the I/O board hardware manual.

Example: `>set timtrig ext` Begin timer task after receiving ext trigger signal.

7. Keystroke Events

Experiment Maker provides two keyboard control functions:

- TWAIT KEY waits for the next keystroke
- IF KEY polls for whether a keystroke has occurred since the last IF KEY

Keyboard-Based Wait: TWAIT KEY

TWAIT KEY is a form of the TWAIT statement that waits for the next keystroke. See the **SIGNAL User's Guide** for the complete TWAIT command description.

TWAIT KEY forces a comfile to halt execution until the next keystroke is detected (vs. IF KEY). When the keystroke occurs, the comfile can parse the keystroke and take appropriate action. TWAIT can be aborted at any time by hitting <esc>.

TWAIT KEY returns information about the key that was struck in result variables. See "Keystroke Codes" below for a description.

Example: Interactive Playback

The following program can play any of 40 stored sound buffers using a single keystroke. This can be used to conduct an **interactive playback** experiment, in which the the next playback is selected based on subject response to the previous playback.

```
new int mybuf
set plrtn immed          ! continue execution after starting playback
label 1000
pl t mybuf               ! play selected sound buffer
twait key                ! wait for keystroke
mybuf = uvar12           ! selected buffer = keystroke
goto 1000                ! play new selection
```

Keyboard-Based Conditional: IF / IFNOT KEY

IF KEY is a form of the IF statement that evaluates TRUE if a key has been pressed since the last IF KEY. **IFNOT KEY** evaluates TRUE if a key has **not** been pressed. When IF or IFNOT evaluates TRUE, the dependent statement, branch, or statement block is executed. See the **SIGNAL User's Guide** for the complete IF command description.

IF KEY allows a comfile to detect a keystroke without halting execution (vs. TWAIT KEY). If a keystroke has occurred, the comfile can parse the keystroke and take appropriate action.

IF KEY returns information about the key that was struck in result variables. See "Keystroke Codes" below for a description.

Example: Sequential Playback

The following program plays 10 stored sound buffers in sequence. Each buffer is played repeatedly until the operator advances to the next buffer by hitting any key. Hitting <esc> aborts the experiment.

```
new int mybuf
set plrtn immed          ! continue execution after starting playback
mybuf = 1
label 1000
pl t mybuf               ! play sound buffer
if key then              ! check for keystroke
```

```
   if uvar11 eq 27 goto 2000          ! abort
   mybuf = mybuf + 1                  ! advance buffer no.
   if mybuf gt 10 goto 2000           ! check if this was last buffer
endif
goto 1000                             ! do another playback
label 2000
stop
```

Keystroke Codes

TWAIT KEY and IF KEY return the following information about the key that was struck in result variables.

UVAR11	ASCII character code
12	Keyboard location code
ULAB11	Text character

Thus the "2" key returns UVAR11 = 50 (ASCII code), UVAR12 = 2 (location code), and ULAB11 = "2" (text character).

An experiment control program can use the ASCII code to determine which key was struck. Note that control keys (notably <esc>) are supported.

The key location code turns the keyboard into a selection device for use by the researcher in Interactive Playbacks or by a human subject in behavioral experiments.

Keyboard Location Code

The keyboard location code represents the key's physical location on the keyboard:

Keyboard Layout

```
1  2  3 ... 8  9  0
 q  w  e ... i  o  p
  a  s  d ... k  l  ;
   z  x  c ... ,  .  /
```

➔

Key Location Code

```
 1   2   3 ...  8   9   10
 11  12  13 ... 18  19  20
 21  22  23 ... 28  29  30
 31  32  33 ... 38  39  40
```

The user can hit any of the 40 keys shown and retrieve a value between 1 and 40 from UVAR12. All other keys return UVAR12 = 0. This allows the keyboard to be used as a 40-button behavioral experiment panel. The user can, for example, use the keyboard to select one of 40 possible signals for playback.

ASCII Character Code

Following are the ASCII character codes associated with the keyboard keys. SIGNAL keystroke codes do not distinguish upper and lower case – see below.

ASCII Code	Key	ASCII Code	Key	ASCII Code	Key
48	0	65	A	79	O
49	1	66	B	80	P
50	2	67	C	81	Q
51	3	68	D	82	R

52	4	69	E	83	S
53	5	70	F	84	T
54	6	71	G	85	U
55	7	72	H	86	V
56	8	73	I	87	W
57	9	74	J	88	X
		75	K	89	Y
13	<enter>	76	L	90	Z
27	<esc>	77	M		
32	<space>	78	N		

Supported Keys

IF KEY and TWAIT KEY support only the 40 keys listed under "Keyboard Location Codes", plus <enter>, <esc> and <space>. Other keys, including function keys, <bksp>, <shift>, <ctrl>, etc. return 0 for the ASCII and location code and a blank in ULAB11. <enter>, <esc> and <space> return a blank in ULAB11.

SIGNAL keystroke codes do not distinguish upper and lower case, i.e., "a" vs. "A". Actually, IF KEY and TWAIT cannot receive upper case characters because the initial <shift> would be detected as the keystroke.

Examples:

Key	UVAR11	UVAR12	ULAB11
1	49	1	"1"
A	65	21	"A"
<enter>	13	0	""
<F1>	0	0	""

COMMANDS

IF, IFNOT Conditional branch

IF / IFNOT is documented in the **SIGNAL Reference Guide**.

TWAIT Wait for time interval

TWAIT KEY is documented in the **SIGNAL Reference Guide**.

8. Event Synchronization

Experiment Maker can synchronize acquisition and playback processes with external events and systems. "Hardware vs. Software Synchronization" discusses the implications of hardware-level vs. software-level synchronization.

SIGNAL provides two facilities for hardware synchronization:

- Initiate an analog I/O process in response to an external trigger signal.
- Issue a digital control signal indicating when an analog I/O process begins and ends.

For example:

- SIGNAL issues an experimental acoustic stimulus then triggers an external system to begin recording the experimental response.
- An external system issues an experimental stimulus, then triggers SIGNAL to begin recording the acoustic response.
- An external system begins neural data acquisition, collects pre-trigger data for 5 seconds, then triggers SIGNAL to begin playback.

Not all I/O boards support hardware synchronization, i.e., they cannot accept hardware triggers or issue digital control signals. These include the Dart I/O card and general sound cards. All boards from Data Translation and National Instruments include digital control capabilities. To determine your board's capabilities, select I/O | Configure on the menu, display board properties and check the "External trigger" property. Or note whether External Triggering is enabled in the Advanced acquisition or playback properties dialog.

Hardware vs. Software Synchronization

Some experiments require the researcher to control and know the time relationship between multiple I/O processes. For example, a reaction time experiment measures the time difference between stimulus onset and the subject's button press, while a neurophysiology experiment might measure a subject's synchronized neurophysiological response to an auditory stimulus.

Synchronization accuracy is the accuracy with which the two processes can be synchronized to the same timebase. In the reaction time experiment, this is the playback stimulus onset relative to the button press timebase, while in the neurophysiology experiment, it is auditory stimulus onset relative to the neurophysiology data acquisition timebase. If the stimulus occurs at $t=0 \pm T$ in the measurement timebase, then all time measurements are limited to an accuracy of $\pm T$.

In general, I/O processes can be synchronized either in software or hardware. With **software synchronization**, I/O processes are initiated and polled by SIGNAL commands. This provides a synchronization accuracy of 1-10 msec, representing the time required to interpret the command and set up the I/O devices. Msec-level accuracy is often adequate for the presentation and detection of behavioral stimuli and responses. In the following example, acquisition begins slightly after playback, delayed by the time required to process the AC command and set up the acquisition process on the A/D hardware.

```
set plrtn immed                 ! perform playback in background
pl t 1

set acrtn immed                 ! perform acquisition in background
ac t 2
```

With **hardware synchronization**, I/O processes are initiated and polled by TTL trigger signals from the experiment subject, other I/O devices, and external systems. This provides a synchronization accuracy of < 1 μsec (typical trigger latency is 20-50 nsec). The command is interpreted, the I/O device is set up, then the I/O process begins immediately upon receipt of the trigger signal. μsec-level accuracy is often required for neurophysiology, both human and non-human (such as bat neuro systems). In the following example, acquisition and playback are triggered by the same signal and begin simultaneously.

```
set pltrig ext                  ! start on external trigger
set plrtn immed                 ! perform playback in background
pl t 1                          ! arm for trigger

set actrig ext                  ! start on external trigger
set acrtn immed                 ! perform acquisition in background
ac t 2                          ! arm for trigger

[issue start trigger]
```

In the reaction time experiment above, two trigger signals are required for hardware synchronization. An external trigger initiates stimulus playback and the timebase used to measure the button press. Then the button press sends a second trigger to the button timebase to obtain a timestamp.

The tradeoff between software and hardware synchronization is time accuracy vs. hardware complexity. Hardware synchronization requires cabling synchronization signals between devices and sometimes generating them (which can often be done using extra timers).

External Trigger – Acquisition and Playback

SIGNAL can begin acquisition or playback processes upon receipt of an external TTL trigger signal.

See "Digital Control of Acquisition and Playback" in Chapter 10, "Advanced Acquisition and Playback" of the **SIGNAL Reference Guide** for details.

External Trigger – Timer

Timer tasks can be initiated, polled, or stopped by an external TTL trigger signal. To start a timer process via an external trigger, use the TIMTRIG parameter. Polling or stopping the timer via an external trigger is task specific – see the **SIGNAL Timer System Guide** for details.

Synchronization Signals – Acquisition and Playback

SIGNAL provides digital control signals to synchronize external events and systems with SIGNAL acquisition and playback processes. **Note: synchronization signals are issued with manual triggering only, not with external triggering.**

See "Digital Control of Acquisition and Playback" in Chapter 10, "Advanced Acquisition and Playback" of the **SIGNAL Reference Guide** for details.

Analog I/O Sample Clock – Internally Routed

It is sometimes desirable to synchronize not only the start time of two I/O processes (such as acquisition and playback) but their sampling clocks. When both processes are performed on the same analog I/O board, both are driven by the board's master timebase and sampling clock synchronization is implicit. However, when the processes occur on different boards, the phase and even frequency relationship between the two sampling clocks is unknown. This can be solved by driving the I/O processes from the same **external sample clock**.

This capability is supported on Data Translation (DT) and National Instruments (NI) boards, but Experiment Maker currently supports an external sample clock on NI boards only. The user routes the output from any NI timer device to the external sample clock input of any NI analog I/O board. No cabling is required. The **ACCLKDEV** and **PLCLKDEV** parameters specify the timer device number to route to the current acquisition or playback process, respectively.

Analog I/O Trigger – Internally Routed

Experiment Maker also allows the user to route the output from any NI timer device to the trigger input of any NI analog I/O board. No cabling is required. The **ACTRIGDEV** and **PLTRIGDEV** parameters specify the timer device number to route to the current acquisition or playback process, respectively.

For example, in a reaction time experiment the user can use a spare timer to generate a trigger signal, then route that signal internally to the playback process using PLTRIGDEV and to the timer process measuring the button press using TIMDEVAUX. See the example "Precision Reaction Time Measurement" above.

SWITCHES & PARAMETERS

ACCLKDEV Acquisition sample clock source

Syntax: SET ACCLKDEV *devnum* Default: 0

Function: Acquisition sampling can be controlled by an external clock rather than the default internal acquisition clock. ACCLKDEV uses the output signal from timer *devnum* (residing on the same board as the A/D) as the acquisition sample clock. The signal is connected internally without cabling. The default setting of 0 uses the internal sample clock. **National Instruments devices only.**

Example: `>set acclkdev 2` Get sample clock from timer 2.

ACTRIGDEV Acquisition trigger source

Syntax: SET ACTRIGDEV *devnum* Default: 0

Function: An external acquisition trigger (see ACTRIG) is normally connected externally, but it can also be provided internally by a timer on the same board as the A/D. ACTRIGDEV uses the output signal from timer *devnum* as the acquisition trigger. The default setting of 0 specifies an external trigger connection. **National Instruments devices only.**

Example: `>set actrigdev 2` Get ext trigger from timer 2.

PLCLKDEV Playback sample clock source

Syntax: SET PLCLKDEV *devnum* Default: 0

Function: Playback sampling can be controlled by an external clock rather than the default internal playback clock. PLCLKDEV uses the output signal from timer *devnum* (residing on the same board as the D/A) as]the playback sample clock. The signal is connected internally without cabling. The default setting of 0 uses the internal sample clock. **National Instruments devices only.**

Example: `>set plclkdev 2` Get sample clock from timer 2.

PLTRIGDEV Playback trigger source

Syntax: SET PLTRIGDEV *devnum* Default: 0

Function: An external playback trigger (see PLTRIG) is normally connected externally, but it can also be provided internally by a timer on the same board as the D/A. PLTRIGDEV uses the output signal from timer *devnum* as the playback trigger. The default setting of 0

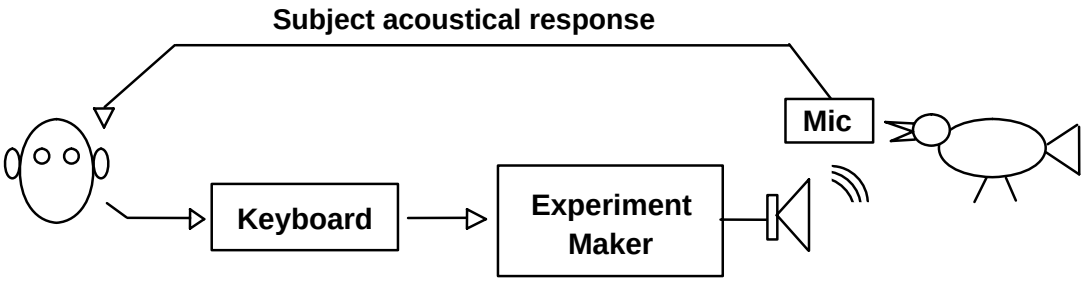
specifies an external trigger connection. **National Instruments devices only.**

Example: `>set pltrigdev 2` Get ext trigger from timer 2.

9. Experiment Maker Applications

This section describes some Experiment Maker applications.

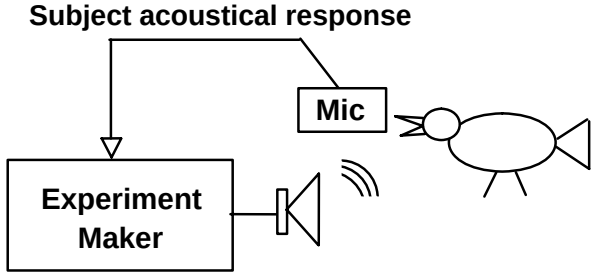
Interactive Playback



This configuration implements **interactive playback**. The researcher uses the keyboard to select an acoustic stimulus from a pre-determined stimulus set. Experiment Maker presents the stimulus to the subject. The researcher observes subject response and selects the next stimulus accordingly.

Interactive playback allows the researcher to investigate perception by varying the stimulus in directed ways. The experiment can "walk" different dimensions of perceptual space (such as pitch, duration, repetition rate, and note order), mapping subject response along these dimensions.

Adaptive Playback



This configuration implements a technique known as **adaptive playback (AP)**. An experiment control program presents an auditory stimulus and simultaneously records the subject's vocal response. After presenting the stimulus, the control program analyzes the response. The program might then either

1) Select the next stimulus from a pre-determined stimulus set, using a programmed selection algorithm, or

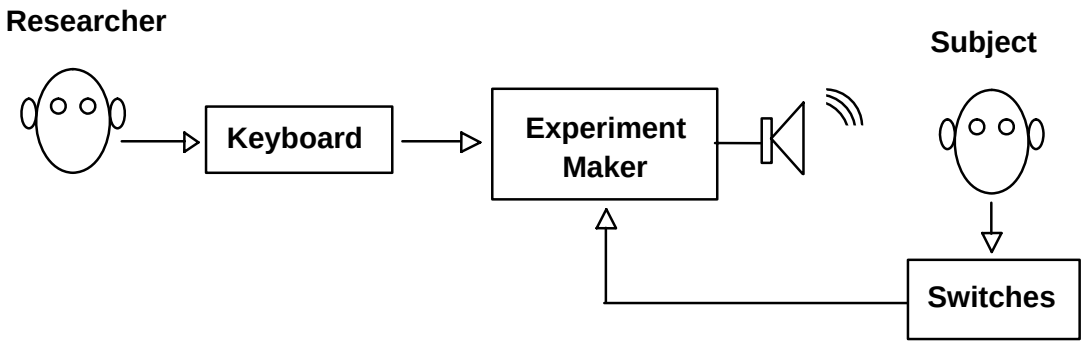
2) Generate the next stimulus parametrically based on pitch, duration, repetition rate, note order, etc., using an algorithm for varying these parameters

By coding subject response as "responsive" vs. "non-responsive", an iterative process can map perceptual boundaries by incrementally varying the stimulus along multiple dimensions (pitch, duration, etc.). For example, the original stimulus can be varied successively in pitch upward and downward to determine perceptual frequency range.

Adaptive playbacks are powerful because their automation can gather a large amount of data unattended, allowing the researcher to map many perceptual dimensions at fine scale.

Note that adaptive playback is an automated version of interactive playback (IP). IP uses the researcher to select the next stimulus, while AP uses a control program with a pre-determined algorithm for stimulus selection, generation, or mutation.

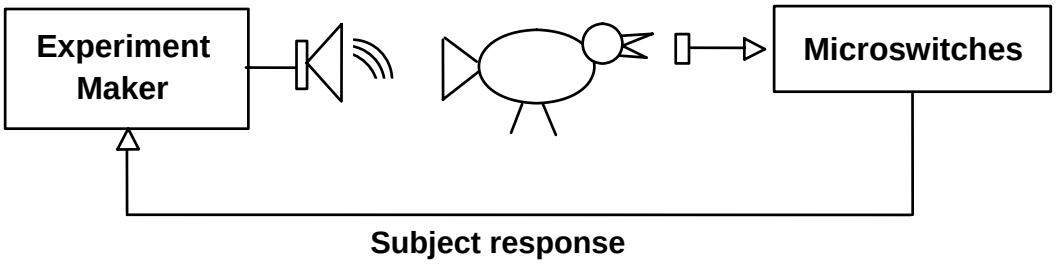
Reaction Time



This configuration measures **reaction time**, defined as the time between stimulus onset and subject response, to study variation with different stimulus features. Depending on hardware, reaction time can be measured with either software (msec) or hardware (μsec) accuracy.

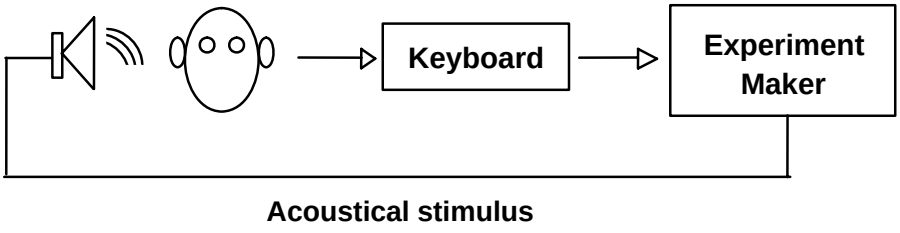
The researcher selects an auditory stimulus using the keyboard, Experiment Maker presents the stimulus, records stimulus onset time, then waits for the subject to activate a switch signaling stimulus recognition. EM then records switch activation time and calculates the latency between stimulus onset and key press.

Operant Testing



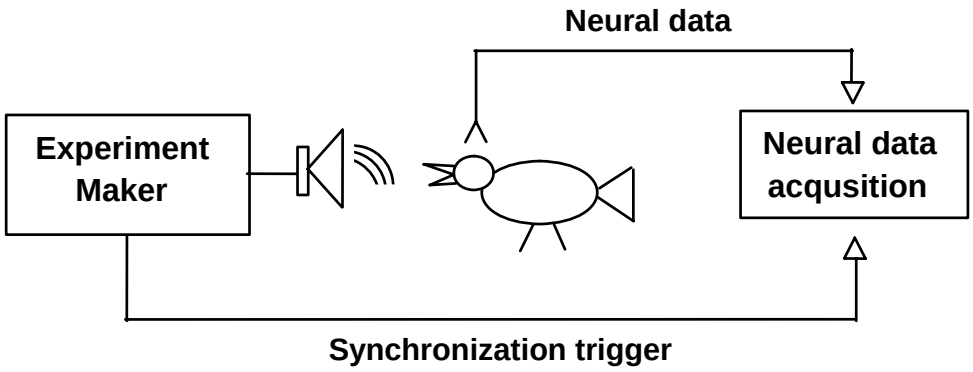
In this **operant behavioral test**, Experiment Maker presents an acoustic stimulus, receives the subject's response through digital switches, then selects the next stimulus by applying a pre-determined algorithm to the subject response. Experiment Maker records stimuli and responses in a log file. The entire process proceeds automatically, without intervention from the researcher.

Human Subject Test Station



Experiment Maker presents acoustic and/or visual cues to a **human subject**. The subject responds via the keyboard (such as key presses or menu choices). The control program receives subject responses, varies stimuli according to a pre-determined algorithm, and records experimental results.

Neurophysiological Stimulus Delivery



A **neurophysiological stimulus delivery** system can be built using Experiment Maker connected through digital control lines to a neurophysiology data acquisition and display system. Experiment Maker performs stimulus presentation while the data acquisition system gathers neural data and displays neural responses in real-time.

The Experiment Maker control program selects, generates, and/or modifies stimulus and presentation details. It presents the stimulus and provides a synchronization trigger (typically some "pre-trigger" interval before stimulus onset) to the neural system to begin data acquisition. It can also optionally transmit a stimulus ID via digital I/O lines to the neural system for storage in the neural data record.

In the diagram, the Experiment Maker (EM) system is the "master", selecting stimuli and determining presentation timing, and the neural data acquisition system is the "slave", receiving synchronization signals from the master. The reverse configuration is also possible: the neural data acquisition system can select a stimulus, transmit stimulus ID to EM, start pre-trigger data acquisition, then issue a trigger to EM to begin stimulus presentation.

Acknowledgements

The original version of Experiment Maker was dubbed BDACS (Behavioral Data Acquisition System) and was conceived in 1991 by Evan Balaban, then at Harvard University and now at McGill University. It was designed jointly by Evan Balaban and Kim Beeman of Engineering Design. Aniruddh Patel of The Neurosciences Institute contributed further suggestions and careful testing.

Index

R

T