QUALITY INSPECTION SUMMARY

We have made every effort to manufacture this instrument to the highest quality standards. All assemblies have been thoroughly tested and inspected at the factory as follows:

Initial Assembly Inspection	_____	☐
Initial QC Inspection/Calibration	_____	☐
Burn-In Cycle	_____	☐
Final Performance Inspection	_____	☐

Packaging Inspection Initials: _____ Date: _____

Items included with any catalog number may be labeled and packaged separately in shipping carton.

Description		**Quantity**	**Checked**
55-50-0B	ECM Experiment Control Module	_____	☐
	Containing:		
	Experiment Control Module	_____	☐
	Accessory Kit	_____	☐
55-00-1	12V Desktop Power Supply	_____	☐
55-AUS	Australia	_____	☐
55-CH	China	_____	☐
55-DAN	Denmark	_____	☐
55-EURO	Europe	_____	☐
55-ISR	Israel	_____	☐
55-ITA	Italy	_____	☐
55-JA	Japan	_____	☐
55-SAF	South Africa	_____	☐
55-SWI	Switzerland	_____	☐
55-UK	United Kingdom	_____	☐
55-USA	North America	_____	☐

FHC, Inc. maintains a system of traceability to allow for product notifications in the event that issues arise pertaining to the recall or failure of critical components contained in this instrument. In order that we may properly notify you, we ask you to complete the following information and return to:

Quality Assurance Department
FHC, Inc.
1201 Main Street
Bowdoin ME 04287 USA
Serial Number(s) _____ _____ _____
Installation and Functional Checkout Complete per section 2.2 and 2.4 of this manual:

Accepted by: _____

Institution: _____

Date: _____

"Innovation through collaboration"

FHC

Providing Instrumentation and Apparatus for Cellular Research, Intraoperative Recording, and Microneurography; Micro-electrodes, Micropipettes, and Needles to the Neuroscience Community for 35+ years.

ECM Experiment Control Module

50-50-0B ECM Experiment Control Module

A998 (Rev. C0, September 2011)

microTargeting™

neurocraft™

FHC Headquarters
1201 Main Street,
Bowdoin, ME, 04287 USA
Fax: 207-666-8292
E-mail: fhcinc@fh-co.com
www.fh-co.com

24 hour technical service
+1-207-666-8190
1-800-326-2905(US & Can)

FHC Europe
(TERMOBIT PROD srl)
129 Barbu Vacarescu Str,
Sector 2
Bucharest 020272
Romania

TABLE OF CONTENTS

Manual A998: ECM

0.2 *CONDITIONS FOR USE*

Intended Use
The ECM Experimental Control Module is intended to interface and coordinate the various instruments involved in awake behaving animal experiments.

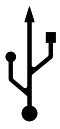
Warnings
The ECM components should not be disassembled beyond their major assemblies. Any disassembly beyond this may affect function and calibration. If repair is required please contact FHC at (207) 666-8190 for evaluation and to secure a return authorization number if necessary.

Storage Precautions
Store at normal room temperatures between -34°C (-29°F) and 57°C (135°F). Do not expose to temperatures below -39°C (-29°F) or greater than 70°C (158°F), or a relative humidity of less than 10% or more than 100%, including condensation, or an atmospheric pressure less than 500hPa or greater than 1060hPa for long-term storage.

Sterilization
None of the ECM components are designed for sterilization. Any attempt to sterilize them may result in malfunction or component failure.

Handling
While a high degree of durability has been designed into the ECM components, care should be taken not to drop them. Do not force any of the connections. Place all cables and leads where they will not be inadvertently pulled or tangled.

0.3 *SYMBOLS USED*



USB: This symbol is used to designate a standard USB connection.

1 *OPERATIONAL MANUAL*

1.1 *FEATURES*

- Centralized real time coordination of all experimental data (electrophysiological recording, behavioral, stimuli, reward) by implementing a finite state machine.
- Intuitive programming of the entire experiment as a finite state machine.
- Considerably shortens the setup time of an experiment by integrating into standard widely used applications (ex. Visual Basic, Matlab, etc) supporting ActiveX. No separate programming language to learn.
- Enables non-real time computer operating systems (like Windows) to run real time experiments.
- Triggered presentation of stimuli or reward based on behavioral data input. May be used as a training aid.
- Real-time operation insured by a dedicated, advanced 32 bit floating-point Digital Signal Processor (DSP).
- 20 - 16-bit digital inputs (20ea.) configurable as event inputs at bit level. Each input line is software configurable as active low or high.
- 19 - 16-bit three-state (high / low / high-impedance) digital output.
- Monitors and processes analog behavioral variables through the analog inputs, or devices (mice, touch screen panels) attached to serial port. Performs real-time differentiation of the analog signal, and can monitor the derivative of the input signal.
- High resolution 16 bit analog input sampling at frequencies up to 48 kHz.
- Three timers for accurate measuring of durations (one running on a per-state basis, two global).
- High speed USB interface enables instant connection to PC. No need for add-on PC boards, no interrupt or address conflicts.
- Upgradeable firmware through the user-programmable FLASH memory.
- Downloadable library of user-contributed algorithms in standard C programming language.
- Compact, modular, desktop or rack mountable.
- Channel version available for use with the neuro/Craft Electrophysiology Workstation.

1.2 *DESCRIPTION*

The ECM (Experiment Coordination Module) is a user-programmable device that is designed for real time control, and sub-millisecond synchronization, of the various instruments in an experimental setup. Rather than just a time sequencer that outputs logical time signals at predefined time intervals, the ECM continuously monitors experimental variables and adapts its state according to the user defined algorithm. In simple psychophysics/behavioral experiments, the ECM is used to enable acquisition of advanced behavioral variables, monitoring subject's responses, and synchronized stimulus presentation. The ECM together with FHC's complete line of microelectrodes, MCM Microelectrode Positioning Systems, APC Neural Spike Discriminator with Amplification, and a variety of readily available visual/auditory/electrical stimulus generators, behavioral variable monitors and reward systems

constitute a completely integrated electrophysiology experiment setup. The ECM can also be easily integrated into existing setups.

20 digital input lines and 19 digital output lines allow controlling complex experiments. Each input line is configurable as active low or high, and can trigger a transition to a different state. Information about the transitions made by the ECM during its evolution in state space is sent to the PC such that one can keep track of the course of the experiment.

ECM also performs specialized digital data acquisition, processing, and monitoring of behavioral variables (for instance eye/arm position). Behavioral monitoring devices (eye trackers, touch screens, mice, joysticks) are interfaced to the ECM through the dual-channel analog input or the serial asynchronous interface (UART). The behavioral variables are capable of triggering, in real time, events that change the state of the system (for example spike recording of only valid trials).

The core of the ECM is a powerful 32-bit Analog Devices floating-point digital signal processor (DSP). All ECM functions are implemented in the hardware and DSP firmware of the unit. The dedicated DSP allows real-time processing of input signals to control the devices in the setup with no data processing required from the host PC. That enables non-real-time computer operating systems to run real-time experiments. The host PC's role is only to do the programming of the ECM, check its status, and retrieve acquired data. The communication between ECM and host PC is via a standard USB interface, making the device easy to install, and compatible with any modern PC.

The ECM PC software is designed as an ActiveX control that is inserted in the controlling application. There is no additional programming language or graphic user interface to learn. Other hardware devices used in experimental setups (for instance visual stimulators) come with their own API/libraries. Programming a certain experiment usually requires writing some code. Instead of having two independent applications that have to be synchronized and talk to each other, one can take advantage of the COM (Component Object Model) that allows inserting the ECM ActiveX control in the original application, and make it part of it. In such a way, bidirectional communication with the ECM is extremely easy to implement by setting the control's properties, invoking its methods, and handling the events it generates. A number of Matlab and Visual Basic samples are provided for using the ActiveX control (see samples help topic for a list of samples included in the current release). Behavioral analog variables are displayed online on the host PC, which can be retrieved for further processing.

The ECM is a software-upgradeable device. Its firmware is stored on a FLASH memory that can be re-programmed with the latest code version, available on our support site www.neurocraft.com. A web-based users group is also established on the site for a free exchange of software, under an open-source policy.

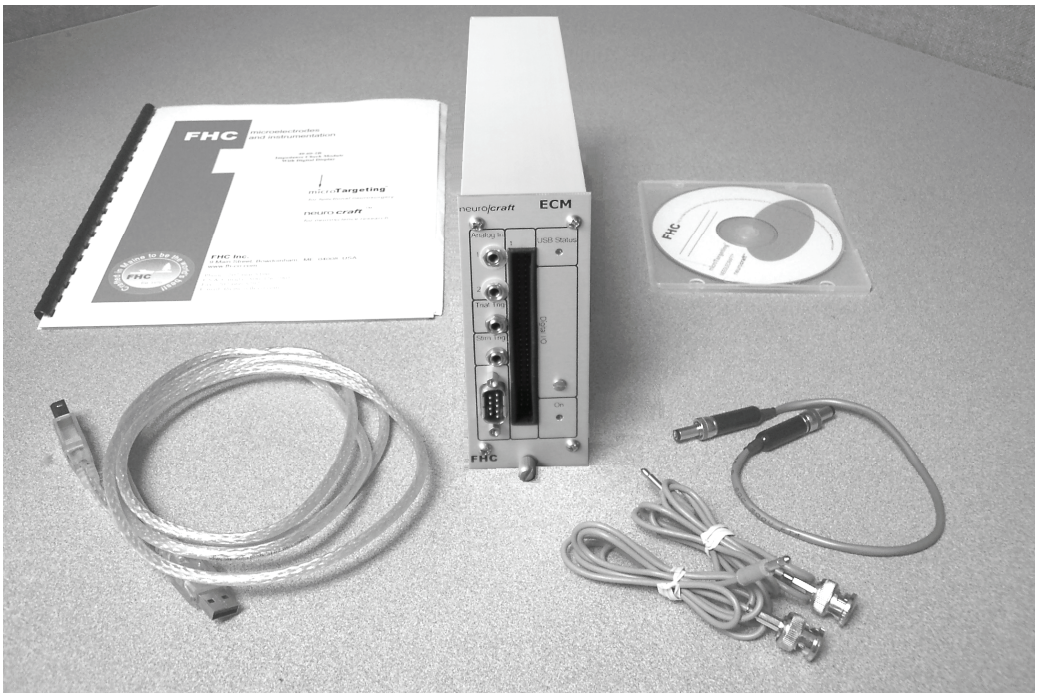
1.3 *OPERATING ENVIRONMENT*

The ECM is designed to function in a typical laboratory environment.

1.4 *INVENTORY*

1.4.1 *ITEMS DESCRIBED IN THIS MANUAL*

The following Items are included under the following catalog numbers:



1 ea. Cat. #55-50-0B ECM

Includes: ECM Module
A998 Manual
Accessory Kit
Includes:
Power Transfer Cord
Software Setup Kit
USB 2.0 Cable
BNC Adaptor Cables (4ea.)
Digital I/O Connector
Rubber Feet (not shown 4 ea. Use Optional)

1.4.2 *ADDITIONAL ITEMS REQUIRED FOR OPERATION*

The following additional items are ORDERED SEPERATELY:

1 ea. 55-00-1 12V Desktop Power Supply

1 ea. 55-XXX Line Cord (Country specific see sec 2.1.4 of this manual for catalog number)

Host PC (Pentium II at 333 MHz or higher with 64 MB RAM running Windows 98SE, Me, 2000, or XP)

Not Shown

1 ea. Cat. # 55-50-1(iin) Digital I/O Adaptor Cable (Specified when ordered)

1.4.3 *REPLACEMENT ITEMS*

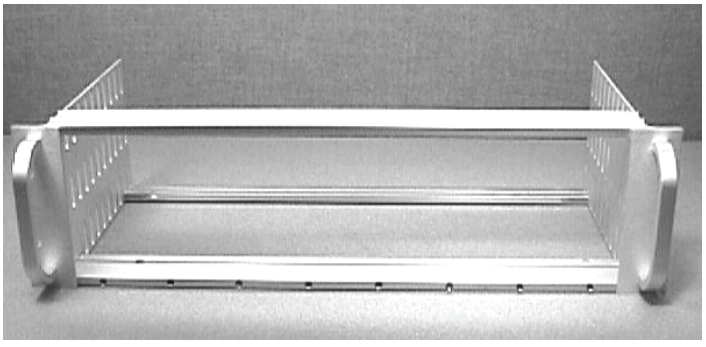
55-00-2 Power Transfer Cord

55-50-4-01 ECM Digital I/O Adaptor

55-00-5 USB 2.0 Cable

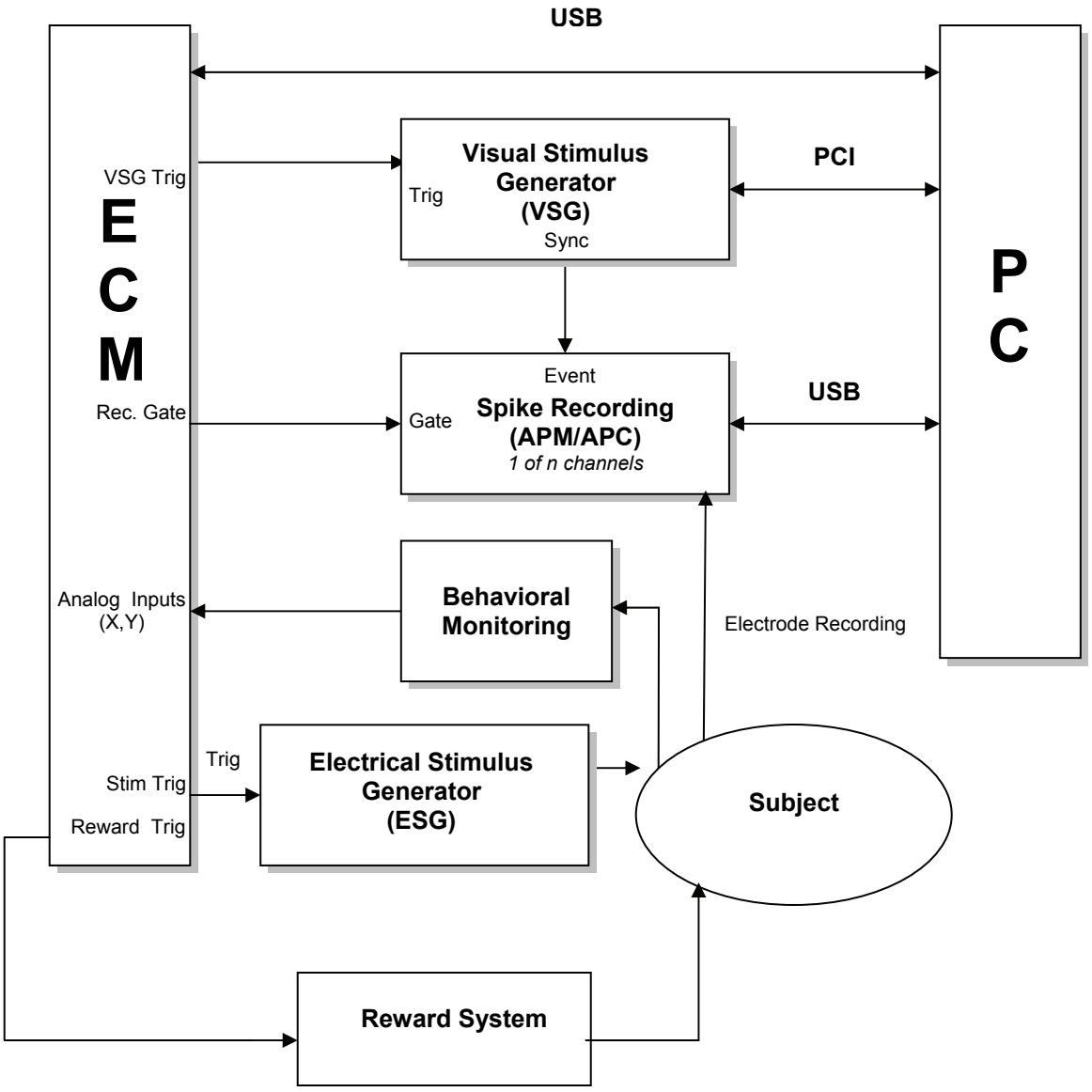
1.4.4 *OPTIONAL ACCESSORIES*

The following accessories are available:



55-11-0 SAF Rack Frame for Stand-Alone Modules

1.4.5 *SYSTEM CONFIGURATIONS*



1.5 *CONCEPTS*

1.5.1 *TERMINOLOGY*

ADC

Analog-to-Digital Converter - a device that converts continuous input signals in numeric form. The APM modules have 16-bit dual channel ADC's.

CODEC

The **CO**der/**DEC**oder is a circuit that contains a pair of analog-to-digital (ADC) and digital-to-analog (DAC) converters.

Driver

Drivers are programs that allow the operating system, and the applications running on it, to communicate with the hardware connected to the computer.

DSP

Digital Signal Processor - a microprocessor that has an architecture and instruction set optimized for real-time processing of digitized analog signals. It often has embedded memory and peripherals, to form a complete microcomputer-on-a-chip.

FLASH Memory

A read-write memory whose content is non-volatile. The programmed data is permanently stored. Re-programming the memory is performed by first erasing the chip, then writing the new data. All these functions can be performed in-circuit, by issuing a sequence of software commands.

SRAM Memory

A Static Random Access Memory that allows fast read-write operations. Being a static RAM, it does not require periodic refreshing of data, therefore data storage is obtained with no CPU overhead. A complete read/write cycle can be performed during one DSP clock cycle, with no extra wait states.

Saccade – A rapid movement of the eye as it moves from one fixation point to another.

State Machine

A model of computation consisting of a set of states, a start state, a pattern of the inputs/outputs, and a transition function that maps inputs and current states to a next state. Computation begins in the start state having a particular pattern of the inputs/outputs. It changes to new states depending on the transition function. There are many variants, for instance, machines having actions (outputs) associated with transitions (Mealy machine) or states (Moore machine), multiple start states, transitions conditioned on no input patterns (a null pattern) or more than one transition for a given symbol and state (nondeterministic finite state machine), one or more states designated as accepting states (recognizer), etc. It is also known as finite state automaton. In their book *Real-time Object-oriented Modeling*, Bran Selic & Garth Gullekson view a state machine as:

- A set of input events
- A set of output events
- A set of states
- A function that maps states and input to output

- A function that maps states and inputs to states (which is called a state transition function)
- A description of the initial state

USB

Universal Serial Bus - a standard interface for connecting peripherals to computers, that is supported by most modern hardware platforms and operating systems.

VSG

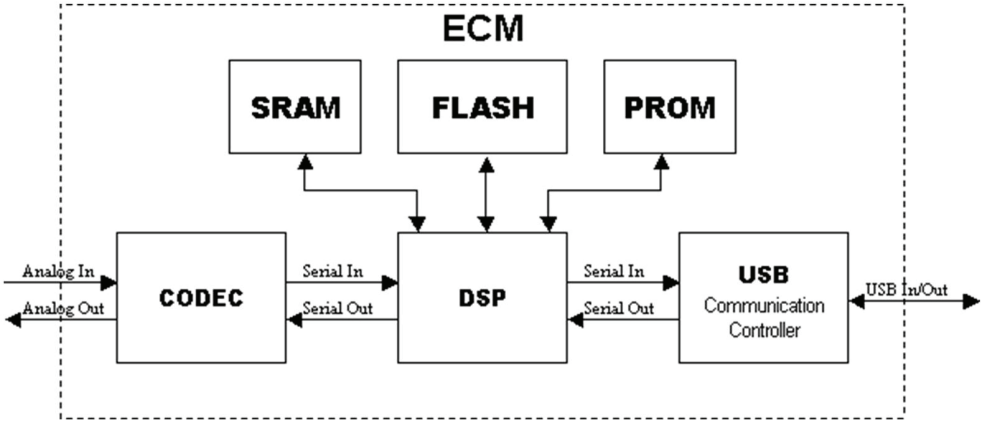
Visual stimulus generator - a device that generates visual stimuli, usually displayed on a computer monitor or projection screen during an experiment. It can be a very specialized device, or a simple VGA-compatible card in a computer.

1.5.2 *DESIGN DESCRIPTION*

Hardware Design:

The ECM hardware platform design is centered around a powerful 32-bit Digital Signal Processor (DSP) with floating point support, from Analog Devices. Analog signals are converted to digital using a 16 bit A/D converter. Functions of additional hardware modules (filters etc.) are completely implemented in the software by the DSP. The processor is powerful enough to perform complex signal conditioning and spike discrimination tasks between two samples even at the maximum sampling rate of 48kHz.

ECM hardware block diagram:

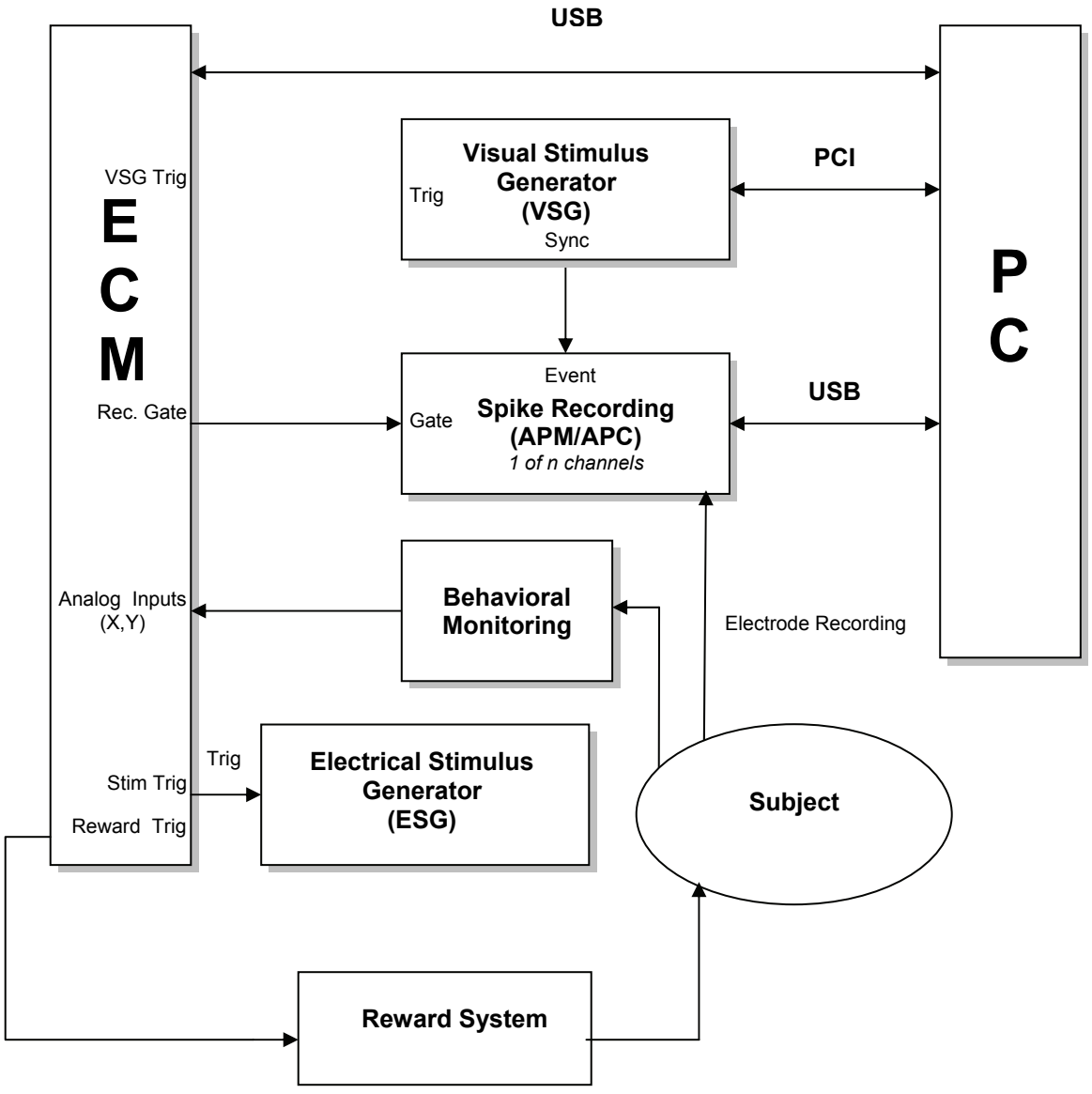


The CODEC is a circuit that contains two pairs of analog-to-digital (ADC) and digital-to-analog (DAC) converters. It sends/receives data to/from the DSP. The DSP receives the digitized input stream, and performs real-time data processing according to the algorithm stored in its internal program memory. The program is loaded into the internal memory from the PROM or FLASH memories after a board reset. The PROM holds a fixed code for restoring the module's functionality when the FLASH has been erroneously programmed by the user. The FLASH holds the actual DSP firmware that defines module's functionality, or user code. It is factory programmed, but can be re-programmed by the user with the latest firmware version, or a customized algorithm. The SRAM is a fast static memory that holds volatile data like communication and waveform data. The communication with the host PC is performed through the Universal Serial Bus (USB) using a dedicated communication adapter.

Software Design:

For simplicity, we will refer to a basic application for vision experiments in awake, behaving primates. The design of the ECM is general enough to allow using it in auditory or electrical stimulation experiments, in other behaving animals (mice, rats, birds, frogs etc.), and in humans.

The module implements a state machine with a finite, but large, amount of states. The amount is limited only by the capacity of the internal RAM memory (1.5MB). The module implements a state machine with discrete binary inputs/output, and also takes into account the value (state) of continuous behavioral inputs. Since the result would be an infinite state machine (there is an infinite number of values a continuous variable can take), the analog inputs are processed in real time and compared with fixed levels and/or templates (see Continuous Variables Analysis for details). The result of the comparison will be a binary variable that is used for determining the next state of the controller.



A typical system configuration consists of the ECM module connected to a host PC (for user interface), Visual Stimulus Generator (VSG, for presentation of visual stimulus in typical awake animal experiments), Neural Spike Recording System (FHC APM/APC system), behavioral monitoring device (eye tracker for instance), reward system, and if applicable, an Electrical Stimulus Generator (for

triggering electrical microstimulation currents). The ECM can trigger these related instruments as well as perform real-time evaluation and validation.

The most complex task of this type of experiment is not programming the controller, but programming the visual stimuli, and ensuring that it is displayed in real-time. No matter how fast today's computers are getting, their displays are still subject to delays and jitters. The reason most of us don't notice is that there's a great amount of buffering in the video adapters. However, that buffering introduces some delays from the time the software decides to update the display to the time it actually gets updated. Even one-frame buffering at a frequency of 60Hz (16.7 ms) may be a huge delay in some experiments. In one of FHC's reference configurations, the visual stimulator is a VSG2/5 board (Cambridge Research Systems, Cambridge, UK,) that supports hardware triggering of visual stimuli presentation. This board is equipped with its own DSP processor that can run scripts for stimuli presentation. Once the board is programmed, it runs nearly independent of the host PC. With real-time hardware triggers coming from the ECM module (see ECM Stim.Trig to VSG Trig connection in the above figure), the whole system can run in real-time, even if the host PC is running under a non-real-time operating system. This reduces the amount of jitter and delay of the stimuli since it is not dependent on the host PC.

Another issue with the normal VGA adapters is the number of colors that can be displayed simultaneously. No matter how colorful your display may look, it is still in most cases the result of a combination of red, green, and blue (RGB) signals that are each represented with 8 bits per color. So if one wants to display pure red, it only has 2^8=256 levels of red available, which is not enough for a good number of vision experiments. In the mentioned reference configuration, the VSG boards from CRS provide 14-bit (16384 levels) video DACs for each color channel.

The VSG is not usually programmed using a graphic interface, but is instead programmed using software libraries that support C, Visual Basic and Matlab. Since the visual stimuli presentation requires code programming, the corresponding supported method of programming the ECM would be a software library that can be accessed from the same languages, and in general, from any application that supports ActiveX integration. An example of this would be programming the VSG and ECM through Visual Basic. (see ECM ActiveX reference later in this document).

The stimulus presentation is organized in a sequence that has a finite duration. During this sequence, the system evolves from the initial state to an end state. This is defined as a trial. A trial is successful if the path from the initial state to the final state closely matches a desired path, as defined by some constraints. If the path is not close to the defined path, it is unsuccessful, and is usually aborted as soon as the state of the system goes beyond the defined constraints. For example, if an eye movement is executed to a designed target within a specified interval of time, a trial is considered successful. If either the eye movement is spatially accurate, but too slow, or fast enough but not spatially accurate, the trial is aborted. Usually a successful completion of the trial is signaled by a special stimulus (an auditory or visual signal), or by dispensing some reward (a drop of juice or food pellets) to the subject of the experiment. If the experiment design does not require breaking it up into trials, the whole recording session, from beginning to the end, can be regarded as a single trial.

The programming of a trial requires programming the VSG board for presenting the visual stimuli, and the ECM to monitor the behavioral variables and trigger the VSG presentation of various visual stimuli that make the trial. The VSG board is configured to wait for hardware trigger signals from the ECM, instead of starting presentation of the stimuli immediately. A trial is started in this manner by using a software or hardware trigger (subject pressing a bar, hitting a button or directing the gaze to a point on the display), and trigger signals are sent from the ECM to the VSG in real-time, independently of the processes running on the host PC.

The triggering of the VSG board can be performed at fixed intervals of time, as defined by a fixed sequence of ECM states, or it can be triggered from the state of the behavioral variables. The behavioral variables can be simple mechanical/optical switches or push-buttons (push bar), that are binary variables that are directly taken into account into the state of the ECM. However, the analog behavioral variables of the subject have to be converted and processed in real time in order to drive the transitions in the state of the ECM, and ultimately the stimuli presentation.

The relationship between stimuli presentation, behavior and neural activity during a trial is the ultimate goal of an experiment. Therefore, the recording of the neural activity has to be accurately synchronized

with the stimuli presentation and the behavioral activity. The neural spike discrimination and recording is performed by an APM/APC system where synchronization with the ECM is achieved by gating the recording system for the duration of a trial (the ECM Rec.Gate to APM Gate line).

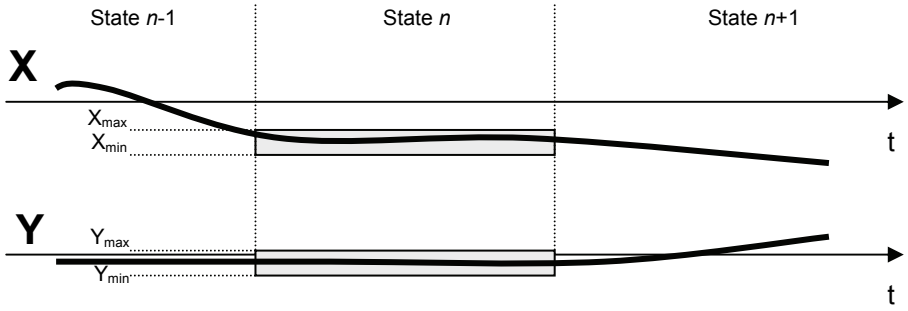
The computer or projection systems display information frame by frame. If a hardware trigger comes from the ECM in the interval between two frames, the display is going to be updated with the next frame. In order to keep track of the actual times when the display has been updated, frame synchronization events are recorded by the APM/APC system. This is another point where the VSG's board unique combination of features proves to be very useful, since there is a hardware frame sync signal that is connected to the APM/APC system's Event inputs (the VSG Sync to APM Event line).

Another reference implementation uses the Psychophysics toolbox (see psychtoolbox.org) for Matlab, and a photodiode circuit (supplied by FHC's MSU Monitor Sync Unit) that detects flashes in one corner of the display to perform synchronization between stimulus display and the ECM. Psychophysics toolbox samples (for instance RFMapping.m) are located in the Matlab subdirectory of your installation.

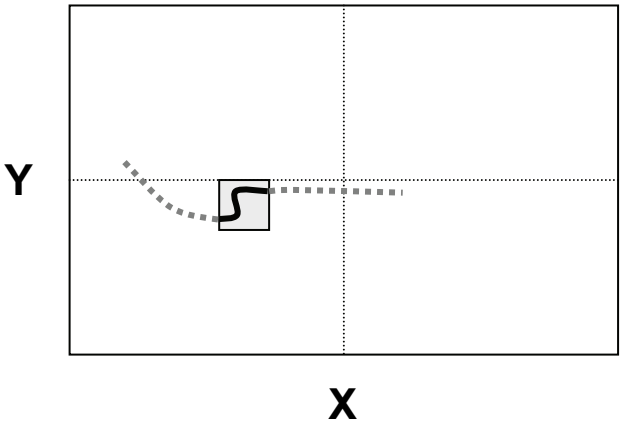
Continuous Variables Analysis:

The processing of analog variables (X and Y position for instance) is performed in real time, and is aimed at transforming the continuous input into discrete variables with a finite number of states. This is very much akin to the underlying principle of spike discrimination, where the waveform of an action potential of a neuron is compared against templates or voltage levels. The result of this comparison is a binary variable (spike/no-spike). There are two sorts of analysis that are performed in real time:

1. Comparing position. Testing the X and Y inputs against two min/max levels is equivalent with checking that the behavioral variable (for instance the eye gaze position) lies within a square defined by the fixed levels on X and Y coordinates.

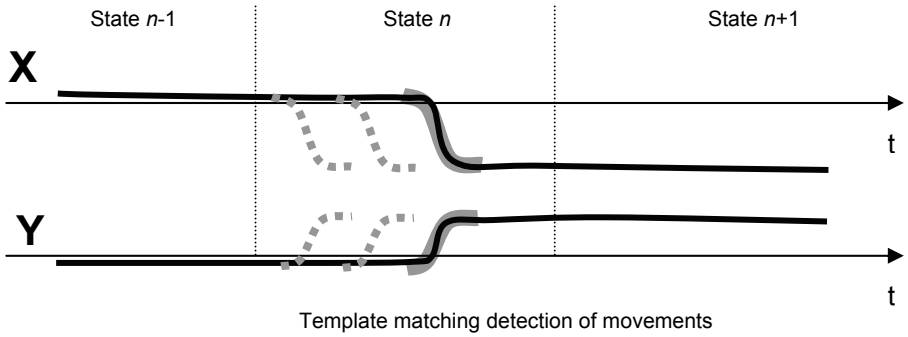


X-t, Y-t representation of the behavioral variables



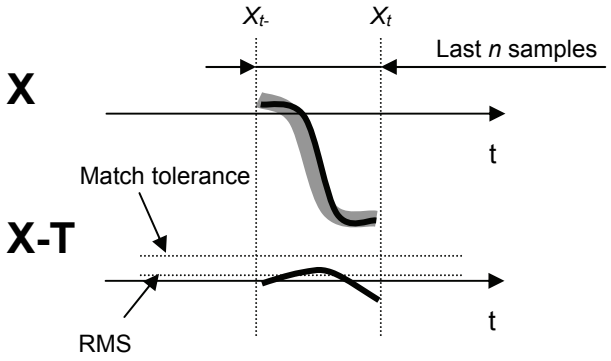
X-Y representation of the behavioral variables

2. Detecting movements, which show as transients in the behavioral variables. This can be achieved by **a**) comparing the X and Y against a pre-defined template. If eye gaze position is monitored, rapid changes in the eye position, known as saccades, can be detected in real-time. In the case where saccades have a large scatter in space, templates are not a reliable method for their detection. Therefore, **b**) analysis of the behavioral variable velocities Vx and Vy is used; The velocities are computed by numerical differentiation of the input signal using a FIR (finite-impulse response) digital filter. A movement is asserted whenever the velocities exceed certain levels, defined by the user.



Template matching detection of movements

In the template matching method of detecting movements, a template, shown in the above figure as the thick gray line, is compared with the most recent analog data every time a new analog sample is acquired. The acceptance criterion is the RMS value of the difference between the template (T) and the analog data (X or Y) (see figure below).

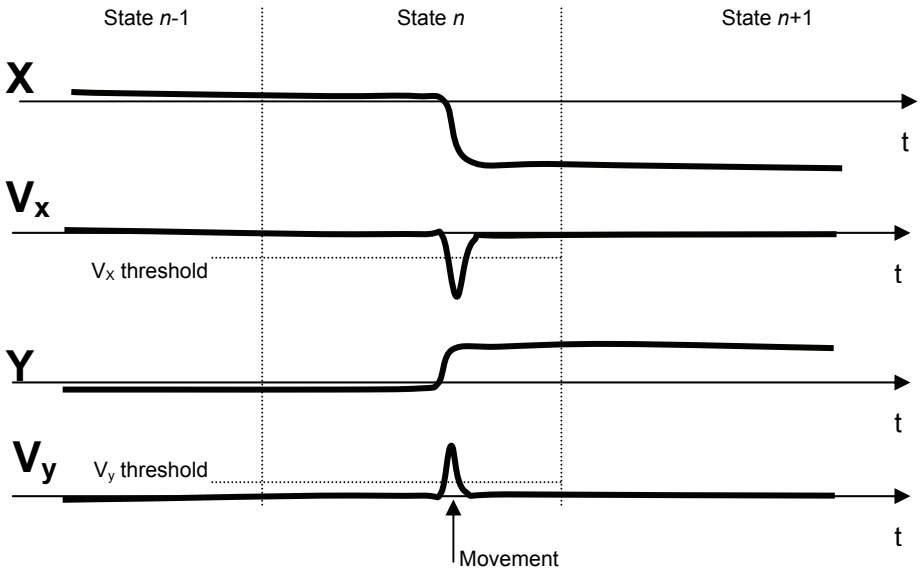


When the last *n* analog samples match the *n* values of the template, the difference between them is small. Therefore, if its RMS value, calculated using the formula below is less than the match tolerance, a saccade detection event is asserted.

$$\delta_{RMS} = \sqrt{\sum_{i=1}^{n} (X_{t-n+i} - T_i)^2}$$

It may be possible, when higher tolerances are set, that the matching criterion is met for several consecutive samples. In this case, only the first successful saccade detection in the set will generate an event.

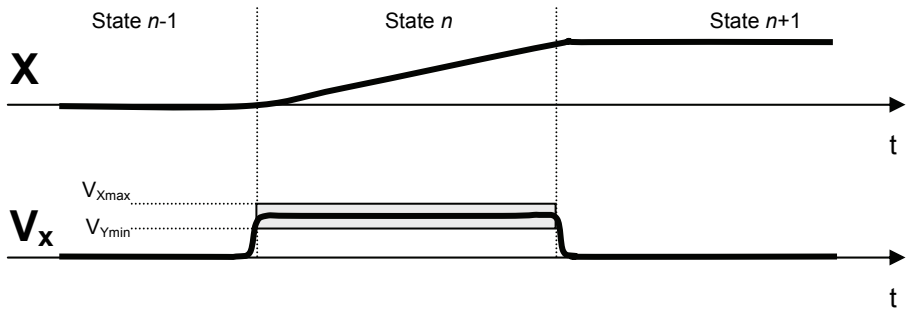
A simpler method for detecting the movements is analyzing the derivative of the primary position input, which is the velocity. The speed for a typical eye movement is shown in the next figure.



Velocity level detection of movements

As seen in the figure, the movements can be detected by checking the value of the velocity signal against a threshold set by the user. It may also be possible to check the velocity against two min/max levels, (like checking the eye position described earlier). This method ends up being faster than the template matching, since a saccade event is asserted as soon as the trigger level is achieved. In the other method (template matching), a saccade could only be asserted when at least *n* samples (the template length) are acquired, possibly introducing a delay (which could be as high as *n* samples).

The velocity analysis just described can be generalized by introducing the concept of *virtual analog channels*, or *processed channels*, where as a result of some numerical computation, the primary position input is transformed into a processed input that is taken into account when applying the standard two-level (min max) checking. This offers great advantages not only for fast transient detection, but also for monitoring more complex paths in space. To be more specific, consider a vision experiment in which the subject has to fixate on a stationary target, then track it on the screen as it starts moving with constant speed. To check that the eye gaze is close to the moving target, a moving window has to be implemented. Checking just the eye position would require defining a state with different min/max levels for every new analog sample. Therefore, a trial definition would require defining a large number of states, making its programming difficult, time- and memory-consuming. Using a processed velocity signal, a movement with constant speed would show up as a constant level, therefore one could check if the velocity is within the upper/lower levels, using the same procedure as described for the position signal.



Using processed velocity channel to monitor continuous movements

1.6 *TECHNICAL SUMMARY*

1.6.1 *SPECIFICATIONS*

ECM:

Analog Inputs: 2 - 3.5mm mono jacks; alternately available as 2 pins on the front panel DB60 connector.

Digital Inputs: 16 TTL pins on the front panel 60 pin connector.

Digital Outputs: 16 TTL pins on the front panel 60 pin connector. 2 alternately available as 3.5mm mono jacks on the front panel. (Trial Trigger and Stim Trigger)

Analog Input Impedance: 20 kOhm

Analog Input Dynamic Range: -0.5V to +0.5V, or –10V to +10V; jumper selectable

A/D Sampling Resolution: 16 bits

Frequency Response: DC to 20 kHz

Memory: 1.5MB zero wait state fast SRAM data buffers, 256kB FLASH

Display: XY plot of the current value of the analog inputs

Computer Interface: Full Speed USB 1.1 (12Mbps)

Dimensions: 2.05" (5cm) width X 5.22" (13cm) height X 9.6" (24cm) depth

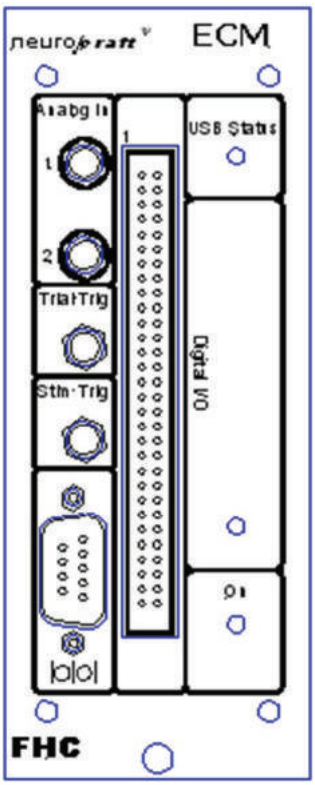
Weight: 1.30 lbs (.59 Kg)

Power Requirements:

Power Supply: 115/230VAC, 50-60Hz, 9W max. Country specific line cord.

ECM: 12V, 350mA

1.6.2 ***CONTROLS/CONNECTORS***



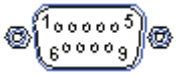
Connections – ECM Front Panel:

Analog Inputs (1,2): 3.5mm mono jacks for analog input. (For instance, the X,Y outputs of a position monitoring device (eye tracker, touch screen etc).

Trial Trigger: 3.5mm mono jack for TTL triggering of data recording in a spike recording system. (ex. FHC's APC)

Stim Trigger: 3.5mm mono jack for hardware triggering of a visual, auditory or electrical stimulator.

Serial: DB9 UART connector for interface with a serial mouse or touch screen



Pinout

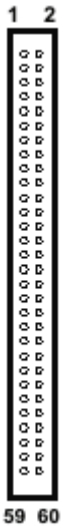
1. 1NC Not connected
2. RxD Serial data from mouse/touch panel to ECM
3. TxD Serial data from ECM to mouse/touch panel (when using a mouse, used only for powering it)
4. DTR Data Terminal Ready, positive voltage to mouse/touch panel and reset/detection
5. Signal Ground
6. DSR Data Set Ready

7. RTS Request to Send (when using a mouse, used only for powering it)
8. CTS Clear to Send
9. NC Not connected

Shell - Protective Ground

Software access to the asynchronous serial port is performed through the properties/methods/events: UARTSettings, UARTWrite, TouchDevice, TouchSamplingFrequency, TouchDataAcquire, TouchData, GetTouchData, TouchCalibration, TouchAcquire, TouchMonitorActive, TouchMonitorActiveInbound, TouchMonitorNextState, TouchMonitorSource, TouchXLowerBound, TouchXUpperBound, TouchYLowerBound, TouchYUpperBound

Digital I/O: 60-pin connector containing all signals above, as well as a number of other digital input-output lines (see pinout below):



Pinout:

1. Digital Input 0
2. Digital Input 1
3. Digital Input 2
4. Digital Input 3
5. Digital Input 4
6. Digital Input 5
7. Digital Input 6
8. Digital Input 7
9. Digital Input 8
10. Digital Input 9
11. Digital Input 10
12. Digital Input 11
13. Digital Input 12

14. Digital Input 13
15. Digital Input 14
16. Digital Input 15
17. Strobe In - All digital inputs are locked in their current state when this is high
18. Reserved
19. Reserved
20. Reserved
21. Reserved
22. Digital Ground
23. Digital Output 0
24. Digital Output 1
25. Digital Output 2
26. Digital Output 3
27. Digital Output 4
28. Digital Output 5
29. Digital Output 6
30. Digital Output 7
31. Digital Output 8
32. Digital Output 9
33. Digital Output 10
34. Digital Output 11
35. Digital Output 12
36. Digital Output 13
37. Digital Output 14
38. Digital Output 15
39. Digital Ground
40. Strobe Out - Goes high to mark the time period between states
41. Not Connected
42. Not Connected
43. Analog Ground
44. Analog Input 1
45. Analog Ground
46. Analog Input 0

47. Not Connected
48. Not Connected
49. Digital Ground
50. Event Input 1
51. Event Input 2
52. Event Input 3
53. Event Input 4
54. Digital Ground
55. Digital Ground
56. Reward Trigger
57. ESG (Electrical Stimulus Generator) Trigger
58. VSG (Visual Stimulus Generator) Trigger
59. Record Gate Trigger
60. +5V

Software access to digital inputs (pins 1 to 17) is performed through the properties/methods/events: DigitalInputsActive, DigitalInputsActiveLow, DigitalInputsNextState, GetDigitalInputsNextState, SetDigitalInputsNextState, DigInputChanged

Software access to digital outputs (pins 23 to 40): DigitalOutputs, DigitalOutputsActive

Software access to analog inputs (pins 44 and 46): VoltageCalibration, XYAcquire, XInputCalibration, XInputOffset, YInputCalibration, YInputOffset, XYSamplingFrequency, XYViewInterval, XYDataAcquire, XYMonitorActive, XYMonitorActiveInbound, XYMonitorNextState, XYMonitorSource, XLowerBound, XUpperBound, YLowerBound, YUpperBound

Software access to additional event inputs (pins 50 to 53): Event1InputActive, Event1InputActiveLow, Event1InputNextState, Event2InputActive, Event2InputActiveLow, Event2InputNextState, Event3InputActive, Event3InputActiveLow, Event3InputNextState, Event4InputActive, Event4InputActiveLow, Event4InputNextState

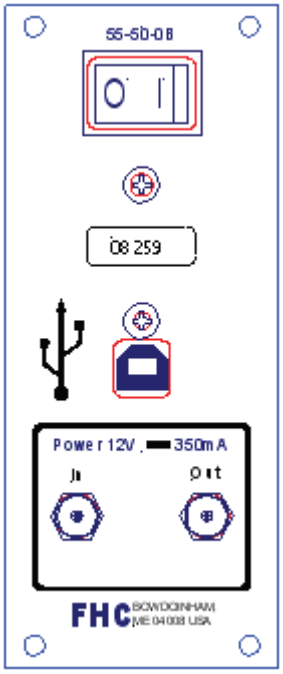
Software access to dedicated digital outputs (pins 56 to 59): ESGTrigger, ESGTriggerDuration, RecordGate, RewardTrigger, RewardTriggerDuration, VSGTrigger, VSGTriggerDuration

Display - ECM Front Panel:

USB Status: Tri-color LED indicates status of the USB port.

- **Green:** Indicates communication ready to send and receive.
- **Amber:** Indicates module is not yet recognized/initialized as an USB device. This happens primarily at startup. LED changes from amber to green when device is recognized/initialized by the host computer.
- **Red:** Indicates communication is busy.

On: Green LED indicates power is on.



Controls – ECM Back Panel:

0|I: Rocker switch used to activate power

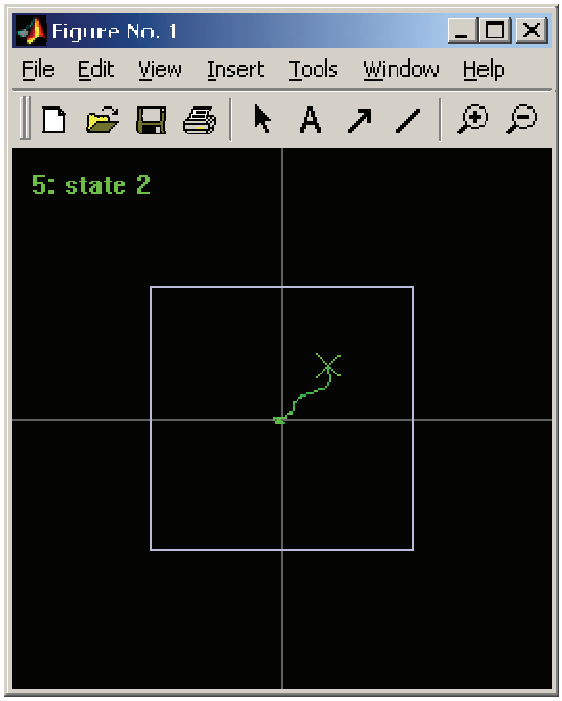
Connections – ECM Back Panel:

USB: High speed USB 2.0 type “B” port for interfacing with host computer.

Power In: 2.1mm female socket. 12VDC 2A input.

Power Out: 2.1mm female socket. 12VDC pass-through outlet for daisy chaining of modules.

Controls – ECM Main Panel:



The windows application that communicates with the ECM module is designed in the form of an ActiveX control, to be inserted in the application that controls your experimental setup. The main reason for this choice is that other programmable devices in a setup come with their own applications or libraries, and it is much more efficient to insert the ECM control in the code that's controlling the other devices, and synchronize their operation, than have two or more distinct application exchanging data with each other. The ECM ActiveX control can be used in any application (Matlab, Visual Basic, Visual C etc) supporting COM (Component Object Model) integration, and even in web pages.

The ECM ActiveX control displays the analog data in a simple X-Y plot. The most recent position is shown as the X marker, while the previous positions are displayed as the green irregular line trailing behind the marker. The window against which the position is checked is represented as the white square. Status messages are displayed in the top left corner of the window.

1.6.3 *COMPATIBILITIES*

The ECM hardware is designed for compatibility with most available components of typical awake animal experiments. Please contact Technical Services at (207) 666-8190 for details. The ECM software is designed to run as an Active-X component under Matlab or similar software suites that support Active-X. Several programming examples are currently available for Matlab and Visual Basic.

1.7 *ILLUSTRATIVE PROCEDURE*

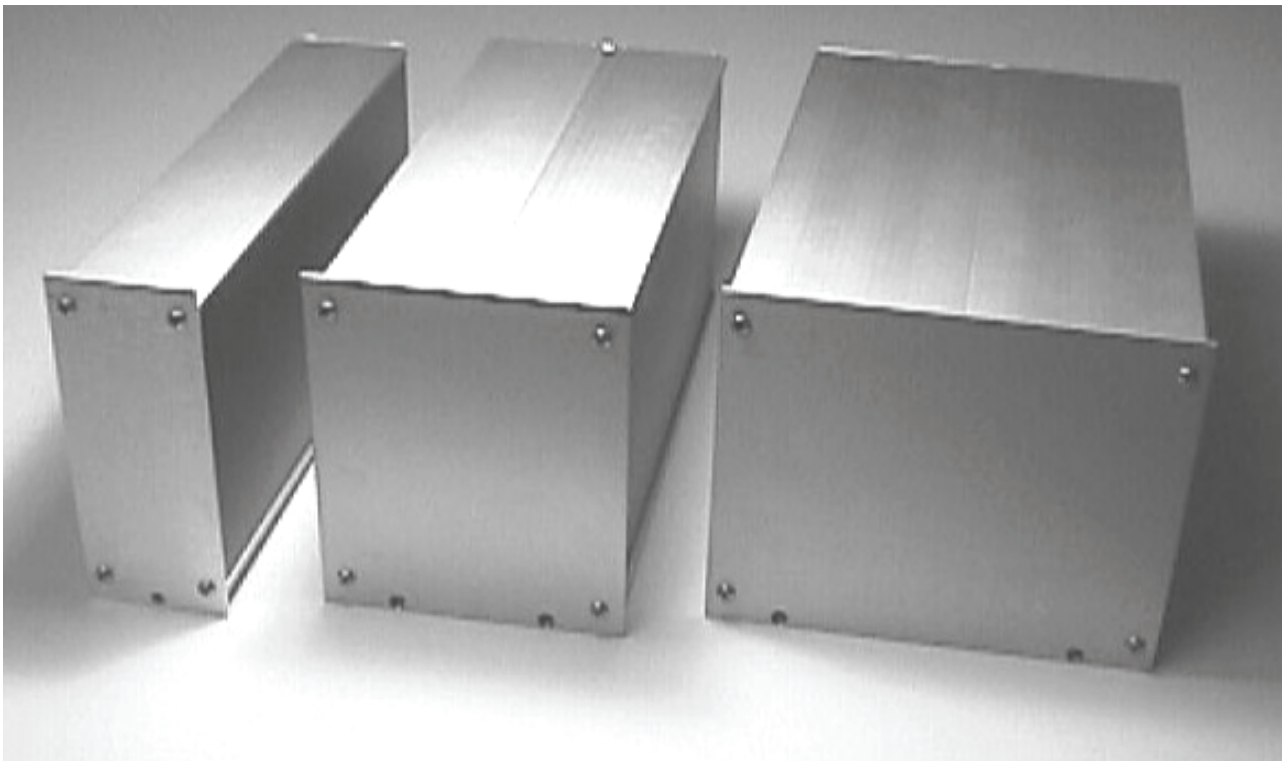
1. Install (first time only) and power up the ECM. Once initialized, the **USB Status** LED will turn from amber to green.
2. Connect all peripheral equipment (neural recording, visual stimulus, reward, etc.) to the relevant inputs/outputs on the ECM
3. Design experiment as a group of states. Design includes presentation of all types of stimulus, behavioral response definition, neural recording gates, reward guidelines, etc.
4. Open Active-X supported programming software. (Matlab etc.)
5. Program experiment based on state machine design.
6. Conduct experiment.

2 *REFERENCE MANUAL*

2.1 *REFERENCE INFORMATION*

2.1.1 *PACKAGING*

The stand-alone modules of the neuro/craft™ series instruments are packaged in metal cases, which consist of standard 5.25" high front panels. Front panel widths are specified as Type 2 modules (2.05" actual), Type 4 modules (4.15" actual), and Type 6 modules (6.25" actual) Front panels are mounted on extruded top and bottom panels. Flat side panels slide into slots in the extrusions, and are held in place when the back panel is secured into the extrusion. All modules are 9.75" in depth.

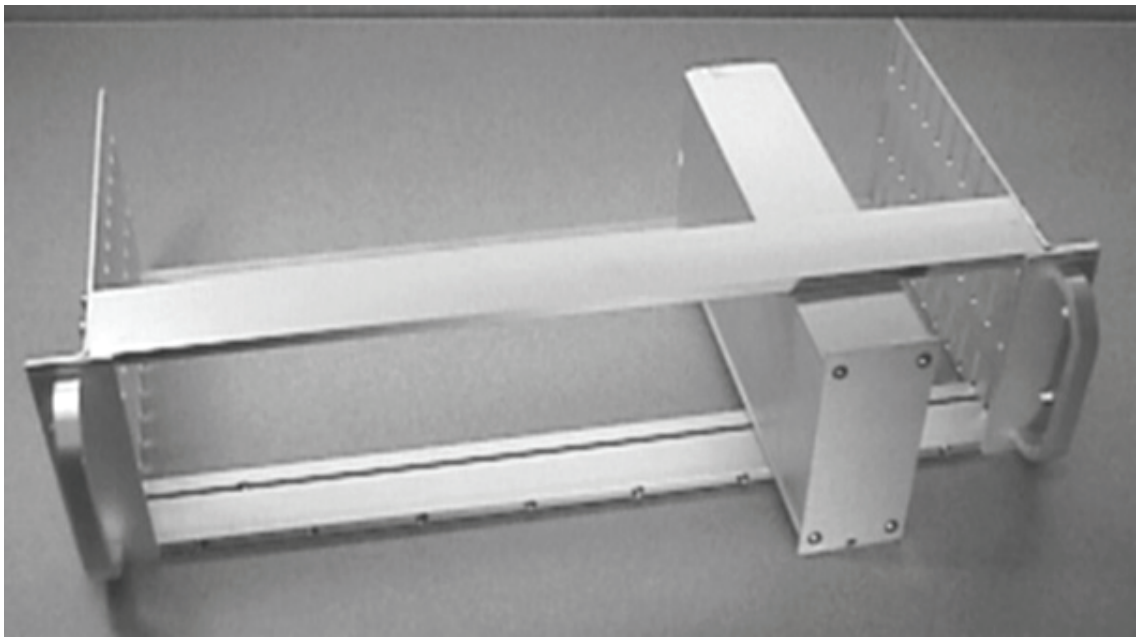


Type 2 Module **Type 4 Module** **Type 6 Module**

2.1.2 *MOUNTING*

All stand-alone modules are completely encased and can be used without further mounting or hardware. Provided rubber feet may be used to protect surfaces from scratching. However, it may be suitable to group modules, and we have made provision for several configurations. The SAF Rack Frame for Stand-Alone Modules (cat #55-11-0) will hold up to eight Type 2 modules, four Type 4 modules, or two Type 6 modules and 2 ea. Type 2 Dress Panels (cat #55-11-1, use optional), while occupying only 3 rack units (5.25") vertically on a standard 19” instrument rack. Several combinations are available for all of the neuroCraft

series stand-alone modules. For example an SAF-08 frame could accommodate 3-Type 2, 1-Type 4, and 1-Type 6 within its 16" of horizontal rack space.



SAF Rack Frame For Stand-Alone Modules
(Shown with a neuro/craft™ Type 2 Module)

Dress Panels for SAF (Ordered Separately):

- 55-11-1 Type 2 Dress Panel

2.1.3 ***INSPECTION***

FHC Modules are factory checked and calibrated but should be carefully inspected upon receipt, before using, or activating power. If any exterior damage to the shipping carton is noted, the instrument(s) should be inspected for obvious physical damage. The contents of each package should be physically checked against the inventory list (sec. 1.3) to determine shortages or errors in inventory.

2.1.4 ***POWER CONNECTIONS***

All of the stand-alone modules in the neuro/craft™ series are powered by a desktop 12V power supply. (input:100-240VAC, 50-60Hz, 1.7A output: +12VDC,5000mA) (Cat. # 55-00-1) An international pattern Line Cord (not shown) is ordered separately, and is specified by country per the catalog number. (See table below for catalog numbers.) Additionally, the power transfer cord (not shown) supplied with the NeuroCraft stand-alone modules can be used to "daisy-chain" the power between other instruments in the series from one power supply. The amount of modules powered from one supply is determined by the amount of current drawn by each module. Contact Technical Services at (207) 666-8190 for assistance.

55-AUS	Australia
55-CH	China
55-DAN	Denmark
55-EURO	Europe
55-ISR	Israel
55-ITA	Italy
55-JA	Japan
55-SAF	South Africa
55-SWI	Switzerland
55-UK	United Kingdom
55-USA	North America

2.1.5 ***WARRANTY***

All FHC products are unconditionally guaranteed against defects in workmanship for one year from date of shipment as long as they have been exposed to normal and proper use. Although the one-year warranty may have expired, please contact our Service Department before attempting any repairs or alterations. Many of these repairs will still be performed at the factory at no charge to the customer.

2.1.6 ***POLICIES***

1.TECHNICAL SUPPORT: It is our policy to provide our customers with the most comprehensive technical support in the industry. If any questions arise or problems occur, we encourage you to call

or write and we promise to promptly and comprehensively respond to your requirements.

2.TRADE-UP POLICY: It is our policy to offer customers trade-up ability as new and/or expanded capabilities for their instruments are announced. In many cases, full credit will be given. In general, we will allow 100% credit for two years and depreciate 20% per year thereafter. Please contact our Marketing Department for information relating to your particular situation.

2.1.7 *SERVICE*

Should service be required, please contact our Service Department for a return authorization number and instructions (207-666-8190). Please have the model and serial number on hand (Both are located on the back panel). Carefully pack the instrument before returning.

Please include a note indicating:

1. The model number and serial number of the instrument
2. The person to contact if questions arise
3. The "symptoms" indicating that repair is necessary
4. Statement of Decontamination

If the instrument is not covered by the warranty, a quotation will be forwarded to the sender detailing the repairs necessary and charges, before repair is begun.

2.2 *INSTALLATION*

2.2.1 *HARDWARE INSTALLATION*

1. Ensure that the ECM is placed in a convenient out of the way spot. Alternately, install the ECM in the SAF Rack Frame for Stand-Alone Units.
2. Ensure power switch on the back panel is in the **Off** position. (Indicated by the **O** side of the rocker switch pressed in.)
3. If powering from the desktop power supply:
    - Plug the power supply line cord into a properly grounded wall receptacle.
    - Install the 2.1mm plug from the power supply into the **In** jack of the **Power** section of the back panel.

   If powering from another module:
    - Install the power transfer cord from the **Out** jack of the other module, to the **In** jack of this module.
4. Connect the USB cable from the USB jack on the back panel of the ECM to the USB jack of the host computer.
5. Connect the cables from the instruments to be controlled in your experimental setup to the **Digital I/O** connector.
6. Connect the output of your behavioral monitoring system (eye tracker, analog touch screen etc) to the **Analog Inputs** or **IOIOI** (serial asynchronous interface) jacks on the front panel.
7. Route all wires as to prevent them from being pulled or tangled.
8. Perform software installation as described in section 2.2.2 before powering up the module.

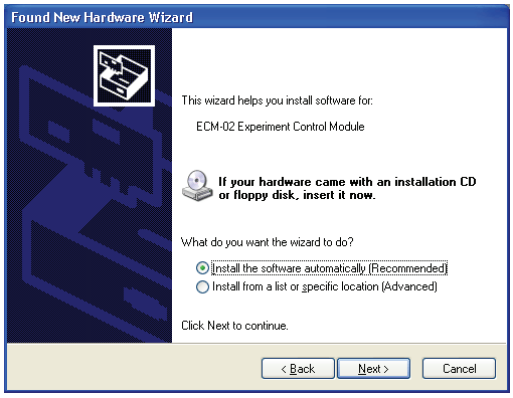
2.2.1 *SOFTWARE INSTALLATION*

1. Insert the application CD and run Setup.exe. If you have downloaded an updated setup kit from the web, run setup from the directory where you have unpacked the files. There are no options to select during install, except for the destination directory.

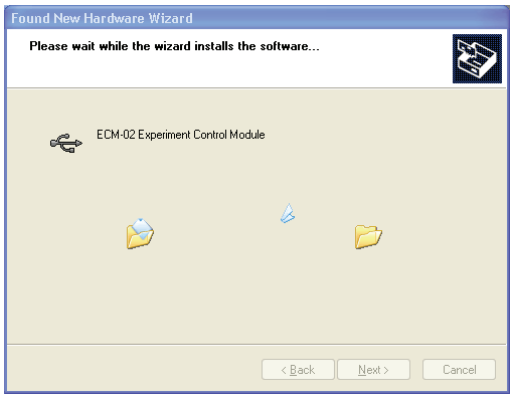
2. Upon successful installation, apply power by pressing in the **I** side of the rocker switch located on the back panel

3. The ECM device is plug-and-play, and Windows operating systems will automatically recognize it after you turn the power on. The operating system goes through the new hardware installation with little or no user intervention required. The system may prompt you with a message that is operating-system dependent (shown below is the WinXP version).

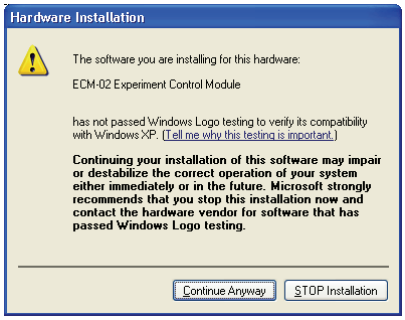
4. Select “No, not at this time”, then click “Next”



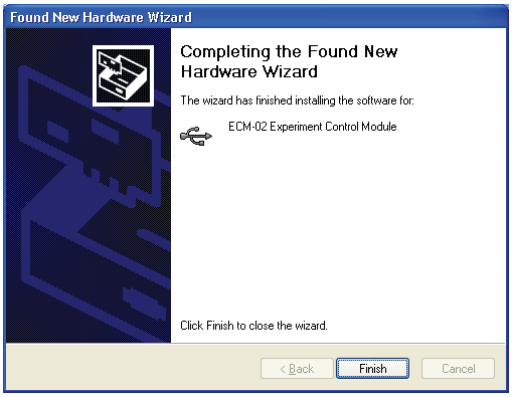
5. Select **Install software automatically**, then click **Next** button.



6. The Windows XP operating system starts installing the software, and prompts you with another message, that refers to the driver digital signature. Click **Continue Anyway**



7. If driver installation has been successful, you will get the acknowledge message shown below. Click the **Finish** button.



Through this procedure the driver for ECM module has been installed. If you install more than one ECM module to a host computer, the operating system will launch the "Add New Hardware" wizard for each ECM.

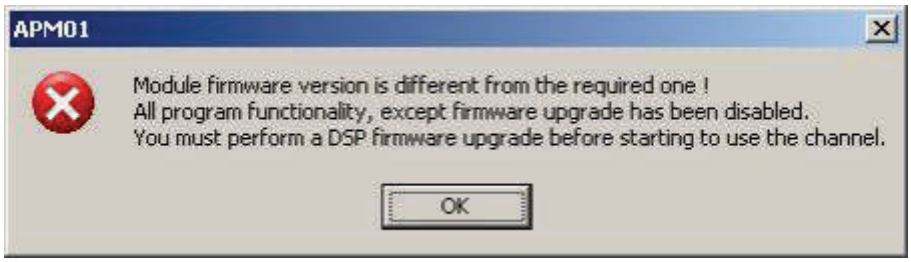
At this point you can start using the module, by inserting the ActiveX control that's handling the communication with it into your application that's running the experiment.

2.2.1 *FIRMWARE UPGRADE*

The ECM offers the powerful option to keep up-to-date with the latest code developments, on both the PC application and the ECM firmware. A PC application upgrade, and/or a DSP firmware upgrade can be easily performed.

Upgrading the PC application:

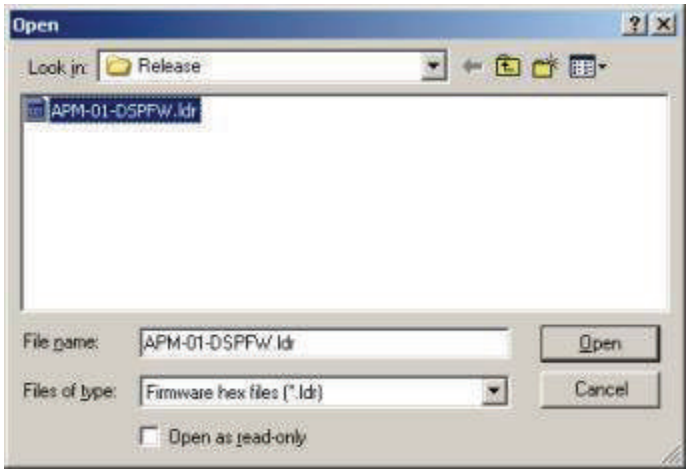
- Download from www.neurocraft.com (downloads section) the zip archive containing the updated software package
- Unpack the zip archive to a directory on the computer;
- Run **Setup.exe** from the folder where you unpacked the files, and follow the installation instructions on the screen; **DO NOT** run Setup.exe directly from the zipped folder, it will not work.
- Power on the ECM, and start the application where you have inserted the ActiveX control (ex. MATLAB, Visual Basic, etc.); If the firmware needs to be upgraded, you will be informed through a pop-up dialog.



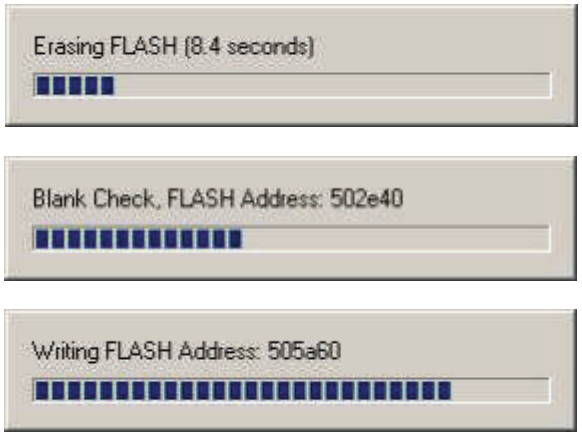
The firmware upgrade requires erasing the FLASH memory that holds the DSP code, blank-checking it, uploading the new firmware, programming the new code into the FLASH, and verification of the programmed data. This procedure is automated, requiring little intervention from the user.

Upgrading the firmware:

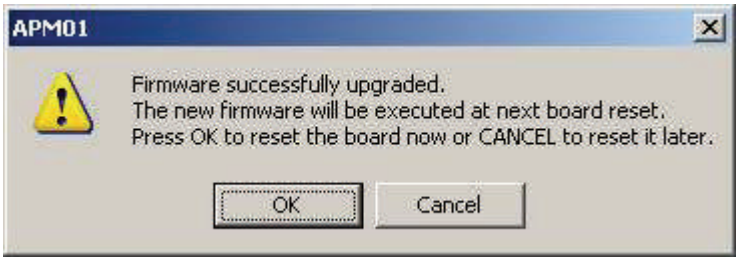
- Power up the ECM, and start the application where you have inserted the ActiveX control; A sample Matlab script that performs the firmware upgrade, ECMUpgrade.m, is included in the **Matlab\Firmware Upgrade** subdirectory of your installation (typically C:\Program Files\FHC\ECM\).
- Select the appropriate virtual comm port by setting the CommPort property of the ECM ActiveX control, then open communications by setting PortOpen property to TRUE;
- Perform the upgrade by invoking the method UpgradeFirmware with arguments: 1. The desired firmware file having the extension .ldr (default is ECM-02-DSPFW.ldr), 2. The map of variable addresses in DSP memory (default is ECM-02-DSPFW.map).



- After invoking this method, the firmware process automatically starts. First of all the FLASH memory is erased, then its contents are checked to confirm that they are blank. Next the new code is uploaded to the ECM, programmed and verified in a single apparent step.



- After a firmware upgrade and reset, the DSP executes the initialization (programming serial ports, selecting inputs, LED test etc) code, therefore you will not be able to communicate with the module for a few seconds, before initialization is completed. It is recommended that you wait for the end of the reset cycle, before you attempt to use the ECM.



- It is highly recommended that you quit and restart the PC application after a firmware upgrade. This ensures that the newly read variable addresses from the .map file are stored into the system registry.

2.3 *FUNCTIONAL CHECKOUT*

Testing the module functionality:

1. Ensure that the hardware and software are installed correctly, as described in the Installation instructions.
2. Start Visual Basic or Matlab. Change path to the VisualBasic or Matlab directory of your installation (default is C:\Program Files\FHC\ECM).
3. Start the Visual Basic sample Blink.vb or run the Blink.m Matlab script. You should see the on the display that ECM bounces between state 0 and state 1 every second.
4. At this point you can start connecting the various pieces of equipment to ECM and programming more complex sequences of states. There are not other relevant tests you can perform on it as a stand-alone device.

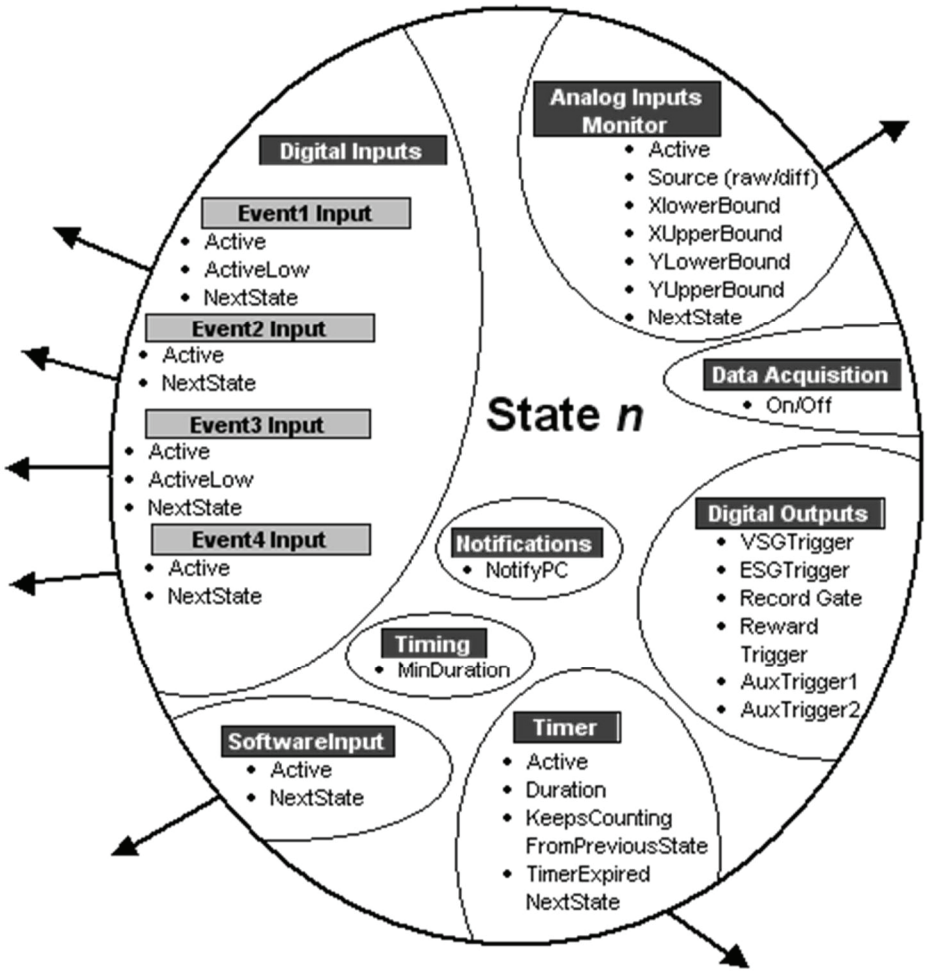
Calibrating the analog inputs:

1. Connect a DC voltage source at both analog inputs, in parallel with a calibrated multimeter.
2. To visualize the output values of the ADC converter, you can use the scripts you have specifically designed for your task, or use the ADCInput.m sample Matlab script in the Matlab subdirectory of your installation.
3. Set the **XInputCalibration** property of the ECM ActiveX control to 1.0.
4. Set the input voltage to 0 V. Adjust the XInputOffset and YInputOffset properties of the ECM ActiveX control until you obtain near zero indication at the output for both channels.
5. If the analog input range is set to +/- 10V, apply a +10V voltage at the input, otherwise apply +0.5V.
6. Adjust the **VoltageCalibration** property of the ECM ActiveX control until you obtain the same indication at the X output as the value displayed on the multimeter.
7. Adjust the **YInputCalibration** property of the ECM ActiveX control until you obtain the same indication at the Y output as the value displayed on the multimeter.
8. If the analog input range is set to +/- 10V, apply a -10V voltage at the input, otherwise apply -0.5V. Check that the displayed value is correct for both channels, otherwise adjust the **VoltageCalibration** property and repeat steps 4 to 8.

2.4 *OPERATIONAL INFORMATION*

Experiment Design:

ECM operation is based on a particular implementation of an abstract state machine. An experiment is represented as a sequence of states. Various triggering events cause transitions between the states depending on the constraints defined by the user. Before programming the ECM, one has to clearly define the "state chart" of the experiment. In such a chart, the states are represented as boxes or balloons, while transitions between them are represented as arrows. The source variable that triggers a transition is represented as an inset in the state balloon. Below is a representation of a single state implemented by the ECM, with its possible variables and transitions.



The possible transitions toward other states are represented as outbound arrows. The transition source variable can be any, single or combination, of the following (labeled in the dark boxes above): Digital Inputs, Software Input, Timer, and Analog Inputs Monitor. These variables transition (output) to other states rather than output to other instruments.

The possible output sources are the insets above that don't have outbound arrows. These sources output directly to other instruments or the host PC, triggering them to perform some function.

For each state, the condition of all of the output sources has to be specified. The states of the transition source variable are optional, however, at least one transition source must be active or the state machine will freeze at the current state.

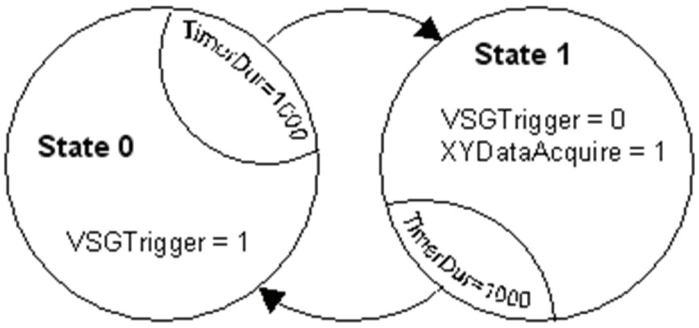
Another feature of the digital outputs not represented in the figure is programmability. The output source signal can be programmed to be high for a limited duration as opposed to the duration of the whole state. All outputs emulate such a "monostable" behavior, except RecordGate. The script properties that set the length of the output pulse are: VSGTriggerDuration, ESGTriggerDuration, RewardTriggerDuration, AuxnTriggerDuration.

An essential inset in the above diagram is the behavioral data acquisition, labeled as Analog Inputs Monitor above. The main advantages of having this built in a device that also implements the state machine are: a) you can monitor the analog position variables against a window then trigger transitions based on the match, and b) you can acquire behavioral data in perfect synchrony with the evolution of the experiment through different states. In fact, data is stored and sent to the PC as triplets (state, X, Y) for every sampling point, so that one can match the data with the states of the experiment.

In order to design an experiment or convert an existing one to run in the new software/hardware configuration, one must start "thinking in state machine", or rather in its particular ECM configuration. The following two examples illustrate this. The examples graduate from simple to complex.

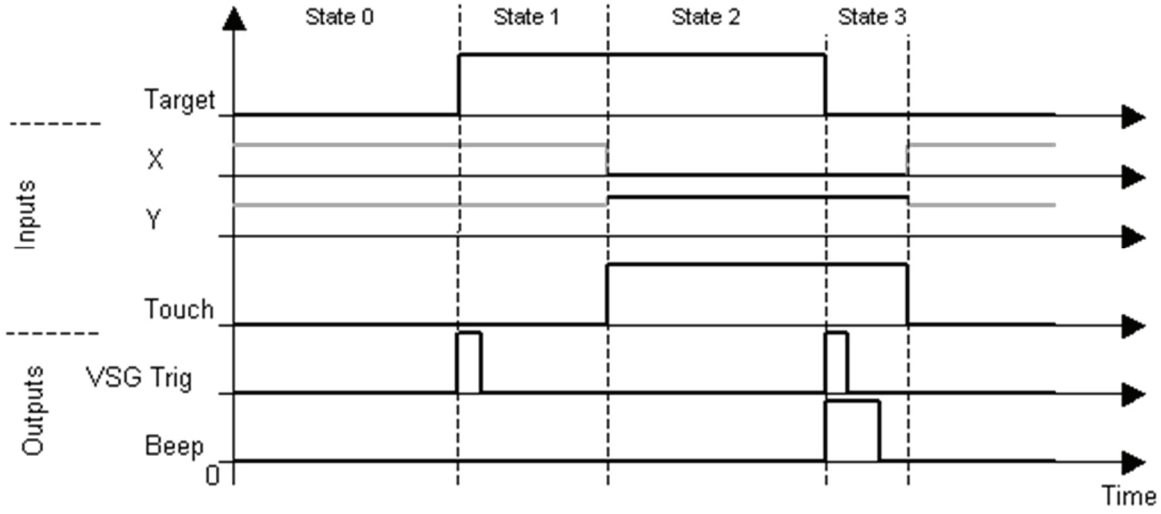
Simple Two State Experiment:

In this example the state machine bounces between the two states every 1000 milliseconds, toggling the output source VSG Trigger, regardless of the state of any other inputs. The associated Matlab code is blink.m in the Matlab samples directory of the ECM program file folder, and will be shown in the following **ECM Programming** section.



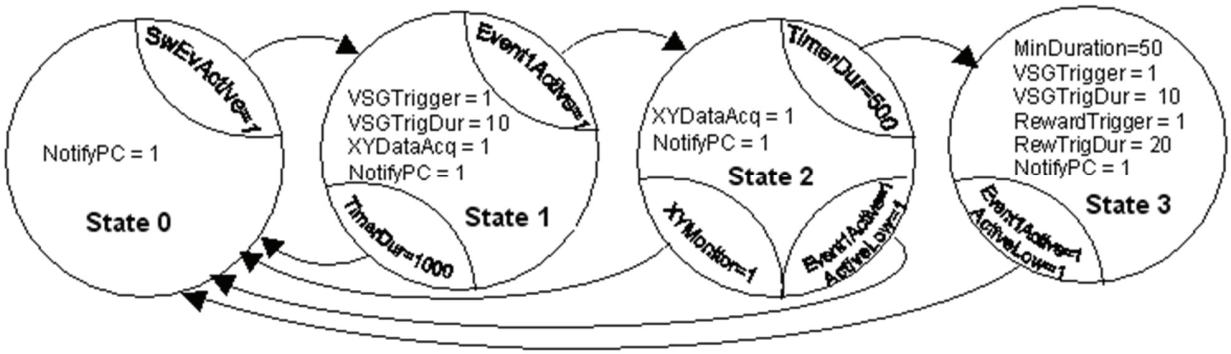
Four State Touch Experiment

In this more realistic experiment, a subject has to touch and hold a target presented on a touch-screen monitor for 500 milliseconds. If the target is touched within 1000 milliseconds with enough positional accuracy, and this XY position is maintained for another 500 milliseconds interval, the trial is validated, and a sound generator is triggered by the TrialValidate/RewardTrigger output to mark the successful completion of the trial. A standard representation of the experiment would be as a set of time plots for each variable or stimulus:



The hardware would be set up by connecting the X and Y analog outputs of the touch screen to the X and Y analog inputs of the ECM, and the Touch recognition signal connected to the Event1Input. The VSGTrigger output is connected to the visual stimuls generator, and the RewardTrigger output to a tone generator.

This representation can be translated to a state chart as shown below (some of the long property names have been abbreviated). By default, all outputs are zero and transition triggers inactive, unless otherwise specified. Because of this default, we can represent only variables that take values different from their defaults thereby simplifying the state chart.

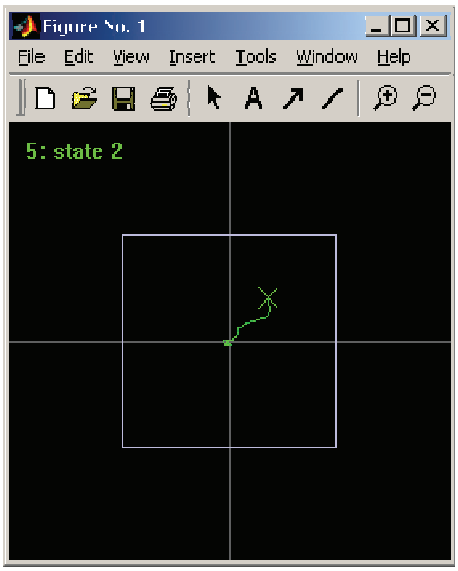


In the initial state (0), the ECM waits for a software trigger from the host PC to start the trial. In state 1, the ECM triggers the presentation of the visual target, and starts acquisition of the XY and Touch data. If the

screen has been touched (Event1 input goes high) within 1000 ms, a transition is made to state 2. If the 1000 ms timer expires, the system goes back to the initial state to start over, and this trial is aborted. In state 2 the XY monitor checks the touch XY position for 500 ms. If the touch location goes outside the defined bounds, or touch is released, this trial is aborted and the system returns to the initial state. After 500 ms of successfully maintaining touch position, the timer generates a transition to state 3, in which the target is turned off by the VSGTrigger, and a sound is generated by the RewardTrigger. The ECM remains in state 3 for at least 50 ms, until the subject stops touching the screen, signaling that it's ready for a new trial. The associated Matlab code is Touch.m in the Matlab samples directory of your software installation.

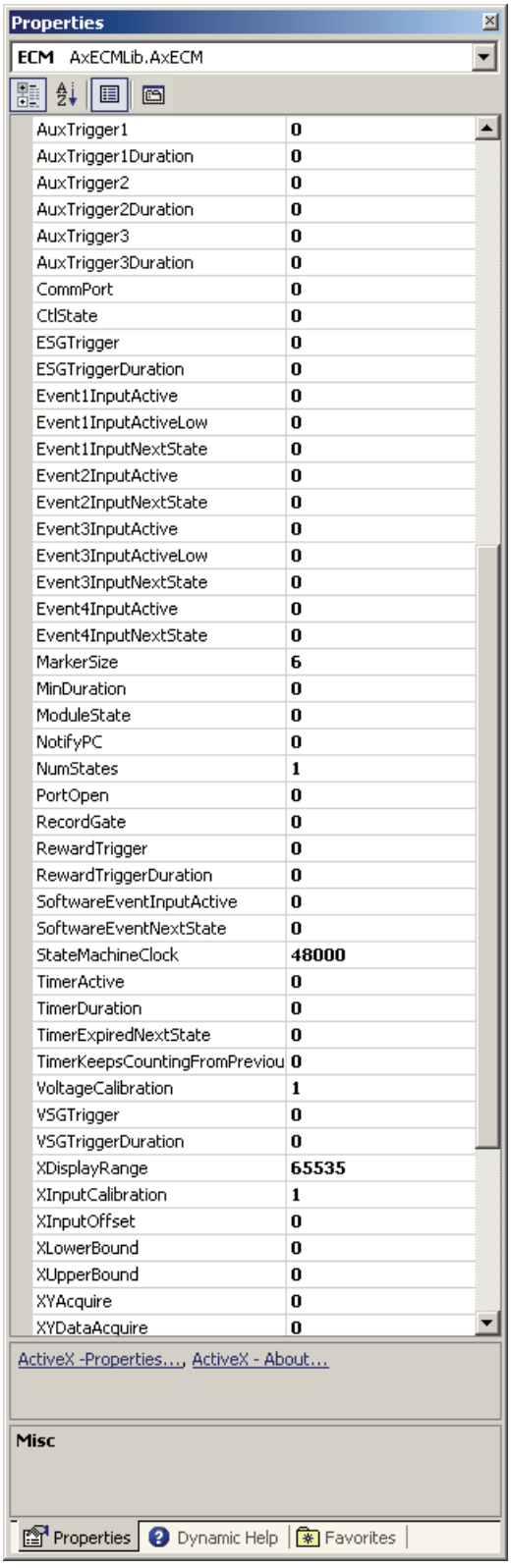
ECM Programming:

The windows application on the host PC that communicates with the ECM module is designed in the form of an ActiveX control, to be inserted in the software that controls your experimental setup. The main reason for this choice is that other programmable devices in a setup (for example the visual stimulus generator) come with their own applications or libraries, and it is much more efficient to insert the ECM control in the code that's controlling the other devices and synchronize their operation, than to have two or more distinct control applications exchanging data with each other. The ECM ActiveX control can be used in any application (Matlab, Visual Basic, Visual C etc) that supports COM (Component Object Model) integration, and even in web pages.



The ECM ActiveX (shown here in Matlab) control displays the analog data in a simple X-Y plot. The most recent position is shown as the X marker, while the previous positions are displayed as the green irregular line trailing behind the marker. The boundary window against which the position is checked is represented as the white square. Status messages are displayed in the top left corner of the window.

This is a very simple graphical interface. The main user interface will be whichever control software (Matlab, Visual Basic) editor that you will be using. Shown here is a view of the Visual Basic property editor with a partial list of the available properties.



Programming the property settings in code is straightforward. For instance in Matlab, setting the **AuxTrigger1** property to 1 can be done in the following line of code:

```
ECM.AuxTrigger1 = 1;
```

Several Matlab and VB programming examples are given later in this section that also show how to; create ECM objects, invoke methods, and handle events. The complete listing of the properties and methods, with programming examples, can be found in the Reference section.

ECM ActiveX Control

Following Microsoft's tradition of documenting ActiveX controls, only VisualBasic syntax is used in the Reference list of properties, methods, and events. Other applications/languages may use a slightly different syntax, but the essential information about the properties, methods, and events are as described in the examples. Sample Visual Basic projects that illustrate the full use of the control are located in the VisualBasic subfolder of the ECM program folder. A number of other examples for using ECM control in Matlab are located in the Matlab subfolder.

Using Matlab

Once the state chart of the experiment has been drawn, the programming of the experiment is relatively straightforward. Below is the Matlab version of the code that implements the simple two-state example shown in the **Experiment Design** section:

Blink.m

```
close all;
% Set up figure 1, where the ECM ActiveX control will be inserted.
f=figure(1);
set(f,'Pos',[100 200 250 250]);

% Create ActiveX control for communicating with the EC Module
ECM=actxcontrol('ECM.ECMCtrl.1',[0 0 250 250],f,{});

% Set control's appearance
ECM.BackColor = 0;
ECM.ForeColor = int32(hex2dec('00FF00'));

% Open communication with ECM
ECM.CommPort = 0;
ECM.PortOpen = 1;

% Define the two states of the experiment
% Initial state. All outputs are in reset state.
ECM.State = 0;
ECM.NotifyPC = 1;
ECM.TimerActive = 1;
ECM.TimerDuration = 48000;
ECM.TimerExpiredNextState = 1;
ECM.UpdateState;    % Send new state properties to ECM
% First state; VSGTrigger on; RecordGate on; duration 1 s;
ECM.State = 1;
ECM.VSGTrigger = 1;
ECM.RecordGate = 1;
ECM.XYDataAcquire = 1;
ECM.NotifyPC = 1;
ECM.TimerActive = 1;
ECM.TimerDuration = 48000;
ECM.TimerExpiredNextState = 0;
```

```
ECM.UpdateState;    % Send new state properties to ECM
% Done configuring state machine !

% Start ECM
ECM.Run;
disp('Started ECM.');

% Wait here until the user presses a key.
disp('Press any key to stop running.');
pause;

ECM.Stop;    % Stop the state machine if for some reason it hasn't been
stopped
ECM.PortOpen = 0; % Close comm port
close(1);
```

This is the Matlab implementation of the four state touch experiment from the **Experiment Design** section.

Touch.m

```
% Global parameters
slowdown = 1;   % The playback speed.

% Set up figure 1, where the ActiveX control will reside
f=figure(1);
set(f,'Pos',[100 200 250 250]);

% Create ActiveX control for communicating with the EC Module
ECM=actxcontrol('ECM.ECMCtrl.1',[0 0 250 250],f,{});

% Set control's appearance
ECM.BackColor = 0;
ECM.ForeColor = int32(hex2dec('00FF00'));
ECM.XYTraceNumPoints = 80;

% Open communication with ECM
ECM.CommPort = 0;
ECM.PortOpen = 1;

% First, deal with XY data acquisition functions
% Calibration stuff
%ECM.XInputOffset = -1000;
ECM.XYSamplingFrequency = 200/slowdown; % 200 Hz
% Turn on XY eye position acquisition (required for monitoring the eye
position)
ECM.XYAcquire = 1;
% Set XY eye postion preview interval, in ms
ECM.XYViewInterval = 100;


% Second, deal with the state machine
% Define all possible states of the experiment

% State 0. Wait for software trigger.
ECM.State = 0;
ECM.NotifyPC = 1;
ECM.SoftwareEventInputActive = 1;
ECM.SoftwareEventNextState = 1;
```

```
ECM.UpdateState;    % Send new state properties to ECM

% State 1. Display target and wait for touch
ECM.State = 1;
ECM.VSGTrigger = 1;
ECM.VSGTriggerDuration = 100;
ECM.XYDataAcquire = 1;
ECM.NotifyPC = 1;
ECM.Event1InputActive = 1;
ECM.Event1InputNextState = 2;
ECM.TimerActive = 1;
ECM.TimerDuration = 48000;      % 1 sec
ECM.TimerExpiredNextState = 0;
ECM.UpdateState;    % Send new state properties to ECM

% State 2. Check touch for 500 ms
ECM.State = 2;
ECM.NotifyPC = 1;
ECM.XYDataAcquire = 1;
ECM.Event1InputActive = 1;
ECM.Event1InputActiveLow = 1;
ECM.Event1InputNextState = 0;
ECM.TimerActive = 1;
ECM.TimerDuration = 24000;      % 500 msec
ECM.TimerExpiredNextState = 3;
ECM.XYMonitorActive = 1; % Check touch position; place window in the
center of the display
ECM.XUpperBound = 8000;
ECM.XLowerBound = -8000;
ECM.YUpperBound = 8000;
ECM.YLowerBound = -8000;
ECM.XYMonitorNextState = 0;
ECM.UpdateState;    % Send new state properties to ECM

% State 3. Turn off target and give a beep. Trial successfuly ended.
ECM.State = 3;
ECM.MinDuration = 10000;
ECM.VSGTrigger = 1;
ECM.VSGTriggerDuration = 100;
ECM.RewardTrigger = 1;
ECM.RewardTriggerDuration = 5000;
ECM.NotifyPC = 1;
ECM.Event1InputActive = 1;
ECM.Event1InputActiveLow = 1;
ECM.Event1InputNextState = 0;
ECM.UpdateState;    % Send new state properties to ECM
% Done configuring state machine !

ECM.Run;    % Start ECM !
disp('ECM Running.');

disp('Press any key over figure1 to exit.');
trial = 1;
while (length(get(gcf,'CurrentCharacter'))==0)
    % Kick in the trial
    disp(sprintf('Starting trial %d ...',trial));
    trial=trial+1;
    ECM.SoftwareTrigger;
```

```
    pause(5*slowdown);
end;

ECM.Stop;    % Stop the state machine if for some reason it hasn't been
stopped
ECM.XYAcquire = 0; % Turn off XY eye position acquisition
pause(0.2);
ECM.PortOpen = 0; % Close comm port
close(1);
```

The next example shows how to use the ECM in conjunction with Psychophysics toolbox (see psychtoolbox.org) for Matlab.

RFMapping.m

```
%  RFMapping.m
%  This program displays a white square in the center of a the screen,
%  and briefly flashes a target in eight different locations evenly
distributed
%  around the central fixation point. The subject is required to fixate
the
%  square in the center of the screen. The script can be used for mapping
the
%  receptive fields of neurons, in vision experiments.
%
%  The mechanism of synchronization between ECM and visual stimulus
presentation
%  is either software of hardware. The hardware mechanism insures maximum
%  timing accuracy, and requires a photodiode circuit to be connected to
%  Event Input 2. A small white square is flashed in the top left corner
%  of the screen, in order to drive the photodiode. To switch between
%  software/hardware synchronization, set the usePhotodiode variable to
0/1.
%
%  To run this program, set the scrWidth and scrHeight variables to
%  match the resolution of your screen, and make sure psychtoolbox
%  is installed. Toolbox can be downloaded from the web address
%  http://psychtoolbox.org/download.html.  It is better to start
%  Matlab using the 'Matlab -nojvm' shorcut created by the
%  psychtoolbox setup program.
%
%  AB 11-17-2003
%

SetPsychToolboxPath;

% Global switches
usePhotodiode = 0;  % Set to 0 when using in demo mode, or 1 when
photodiode is connected to event input 2.

% Display parameters
colordef(0,'black');
state_colors = {[1 1 1],[1 1 0],[0 0.5 1]};
%scrWidth = 1600;
%scrHeight = 1200;
scrWidth = 1024;
scrHeight = 768;
```

```
% Trial parameters (durations etc)
slowdown = 1;
stateMachineClock = 48000;   % Use maximum available clock frequency
minDuration = 0.100;    % 100 msec minimum state duration (the photodiode
circuit may be sending several trig pulses if the computer is slowed down
in between two frame presentations)
inter_trial         = 0.1 * slowdown;    % 0.1 sec
fix_to_target_delay = 1 * slowdown;      % 1 sec
target_on_duration  = 0.5 * slowdown;    % 0.5 sec
reward_duration     = 0.5 * slowdown;    % 0.5 sec
error_timeout       = 1 * slowdown;      % 1 sec

dirs = 0:45:335;    % the eight possible directions of the flashed target
tarEcc = 0.4*min(scrWidth,scrHeight);   % the target eccentricity
tarSize = 40;       % target size
fixSize = 10;       % central fixation point size
flashSize = 40;     % white square for providing the sync signals to the
photodiode circuit.

% State definitions
INTER_TRIAL = 0;    % Short inter-trial interval to reset the APM
counters
FIX_ON      = 1;    % Fixation point comes on
TARGET_ON   = 2;    % Target comes on
REWARD      = 3;    % Monkey rewarded if maintained fixation throughout
the previous state
TRIAL_ERROR = 4;    % Add a state for situations where monkey breaks
fixation

% Flags
trialRewarded = false;

% Set up ECM first, then the visual stimulus
close all;
% Set up figure 1, where the ECM ActiveX control will be inserted.
f=figure(1);clf;
%set(f,'Pos',[-500 50 250 250]);
set(f,'Pos',[500 50 250 250]);
set(f,'MenuBar','none');

% Set up figure 2, where the XY data will be displayed.
f2=figure(2);clf;
%set(f2,'Pos',[-500 350 450 450]);
set(f2,'Pos',[500 310 450 450]);
set(f2,'MenuBar','none');

% Create ActiveX control for communicating with the EC Module
ECM=actxcontrol('ECM.ECMCtrl.1',[0 0 250 250],f,{});

% Set control's appearance
ECM.BackColor = 0;
ECM.ForeColor = int32(hex2dec('00FF00'));

% Open communication with ECM
ECM.CommPort = 0;
ECM.PortOpen = 1;
pause(0.5);
```

```
ECM.XYSamplingFrequency = 500/slowdown; % A/D sampling frequency for the
X and Y inputs.
ECM.XYAcquire = 1;
ECM.XYViewInterval = 50;    % Monitor the eye position every 50
milliseconds, in states where data acquisition is turned off
ECM.VoltageCalibration = 1.0;
ECM.YInputCalibration = -1.0;   % Use a negative value for reversed input
voltages (reversed Y axis)
ECM.XInputOffset = 4400;
ECM.YInputOffset = 3800;
ECM.GetXYData;  % Flush the input buffer

% Define the six states of the experiment, before starting the trial and
% drawing the visual stimuli

% State 0. Brief inter-trial interval that is meant to just reset the
Trial
% Gate signal
ECM.State = INTER_TRIAL;
ECM.MinDuration = minDuration*stateMachineClock;
ECM.RecordGate = 0;
ECM.NotifyPC = 1;
if (~usePhotodiode)
    ECM.SoftwareEventInputActive = 1;  % For now, use software triggering
instead of photodiode triggering
    ECM.SoftwareEventNextState = FIX_ON;
end;
ECM.Event2InputActive = 1;      % Trial Gate photodiode must be hooked up
to Event Input 2
ECM.Event2InputNextState = FIX_ON;
ECM.UpdateState;    % Send new state properties to ECM

% State 1; Fixation comes on; Recording on;
ECM.State = FIX_ON;
ECM.MinDuration = minDuration*stateMachineClock;
ECM.RecordGate = 1;
ECM.XYDataAcquire = 1;      % Turn on data acquisition
ECM.NotifyPC = 1;
ECM.SoftwareEventInputActive = 1;  % For now, use software triggering
instead of photodiode triggering
ECM.SoftwareEventNextState = TARGET_ON;
ECM.Event2InputActive = 1;      % Event photodiode must be hooked up to
Event Input 2
ECM.Event2InputNextState = TARGET_ON;
ECM.UpdateState;    % Send new state properties to ECM

% State 2; Target is presented; Recording on; Check eye position
ECM.State = TARGET_ON;
ECM.MinDuration = minDuration*stateMachineClock;
ECM.RecordGate = 1;
ECM.XYDataAcquire = 1;
ECM.NotifyPC = 1;
ECM.SoftwareEventInputActive = 1;  % For now, use software triggering
instead of photodiode triggering
ECM.SoftwareEventNextState = REWARD;
ECM.Event2InputActive = 1;      % Event photodiode must be hooked up to
Event Input 2
ECM.Event2InputNextState = REWARD;
```

```
ECM.XYMonitorActive = 1;
% Window boundaries are just some numbers between +/- FS (full scale),
which is +/-32767.
% You must properly set the VoltageClibration property, in order to have
something like volts or degrees of visual angle, instead.
ECM.XUpperBound = 4000;
ECM.XLowerBound = -4000;
ECM.YUpperBound = 4000;
ECM.YLowerBound = -4000;
ECM.XYMonitorNextState = TRIAL_ERROR;
ECM.UpdateState;    % Send new state properties to ECM

% State 3; Reward; Recording on;
ECM.State = REWARD;
ECM.MinDuration = minDuration*stateMachineClock;
ECM.RecordGate = 1;
ECM.RewardTrigger = 1;
ECM.RewardTriggerDuration = 0.1 * stateMachineClock;
ECM.XYDataAcquire = 1;
ECM.NotifyPC = 1;
ECM.SoftwareEventInputActive = 1;  % Use software triggering for the
terminal states to ensure video/ECM synchronization
ECM.SoftwareEventNextState = INTER_TRIAL;
ECM.UpdateState;    % Send new state properties to ECM

% State 4; Error; Beep; Recording on;
ECM.State = TRIAL_ERROR;
ECM.MinDuration = minDuration*stateMachineClock;
ECM.RecordGate = 1;
ECM.AuxTrigger1 = 1;
ECM.XYDataAcquire = 1;
ECM.NotifyPC = 1;
ECM.SoftwareEventInputActive = 1;  % Use software triggering for the
terminal states to ensure video/ECM synchronization
ECM.SoftwareEventNextState = INTER_TRIAL;
ECM.UpdateState;    % Send new state properties to ECM

% Done configuring state machine !

screen('CloseAll')  % closes window and all offscreen windows
[window] = screen(0,'OpenWindow',BlackIndex(0),[],32); % opens the main
window placed
% in object 'window'

f = screen(window,'OpenOffscreenWindow',BlackIndex(0),[],32); % Makes
offscreen window
% in variable 'f'
screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
pause(0.5);  %  Waits 1 second, for the screen to flash while
initializing windows

% Start ECM
ECM.Run;
disp('Started ECM.');
pause(0.1); % Waits just a little bit to the Run command to be sent over
USB to ECM, before first sync flash occurs

while ~kbcheck
```

```
    % Setup the current trial (target location on screen etc)
    tarDir  = dirs(ceil(rand*length(dirs)));
    tarXPos = tarEcc * cos(pi*tarDir/180.0);
    tarYPos = tarEcc * sin(pi*tarDir/180.0);

    % Synchronization flash
    syncRect = [0,0,flashSize,flashSize];  % demensions of the square in
pixels indicating top,left,bottom,right corners
    screen(f,'FillRect',WhiteIndex(0),syncRect);  % makes a white square
in offscreen window 'f'

    % Fixation on
    fRect = [0.5*(scrWidth-fixSize) 0.5*(scrHeight-
fixSize),0.5*(scrWidth+fixSize) 0.5*(scrHeight+fixSize)];  % demensions
of the square in pixels indicating top,left,bottom,right corners
    screen(f,'FillRect',WhiteIndex(0),fRect);  % makes a white square in
offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    if (~usePhotodiode)
        ECM.SoftwareTrigger;  % Use software triggering instead of
photodiode triggering
    end;

    % Synchronization flash off
    screen(f,'FillRect',BlackIndex(0),syncRect);  % makes a white square
in offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    disp('1: Fixation On');
    waitsecs(fix_to_target_delay);  %  Waits 1 second

    % Synchronization flash
    screen(f,'FillRect',WhiteIndex(0),syncRect);  % makes a white square
in offscreen window 'f'

    % Target on
    tRect = [0.5*(scrWidth-tarSize)+tarXPos 0.5*(scrHeight-
tarSize)+tarYPos,0.5*(scrWidth+tarSize)+tarXPos
0.5*(scrHeight+tarSize)+tarYPos];  % demensions of the square in pixels
indicating top,left,bottom,right corners
    screen(f,'FillRect',WhiteIndex(0),tRect);  % makes a white square in
offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    if (~usePhotodiode)
        ECM.SoftwareTrigger;  % Use software triggering instead of
photodiode triggering
    end;
    % Synchronization flash off
    screen(f,'FillRect',BlackIndex(0),syncRect);  % makes a white square
in offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    disp('2: Target On');
    waitsecs(target_on_duration);  %  Waits 1 second

    % Synchronization flash
```

```
    screen(f,'FillRect',WhiteIndex(0),syncRect);  % makes a white square
in offscreen window 'f'

    % Target off & reward
    screen(f,'FillRect',BlackIndex(0),tRect);  % delete target in
offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    if (~usePhotodiode)
        ECM.SoftwareTrigger;  % Use software triggering instead of
photodiode triggering
    end;
    % Synchronization flash off
    screen(f,'FillRect',BlackIndex(0),syncRect);  % makes a white square
in offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    disp('3: Target Off & Reward');
    waitsecs(reward_duration);  %  Waits 1 second
    if (ECM.ModuleState==3)
        trialRewarded = true;
        disp('Trial Rewarded !');
        beep;
    else
        trialRewarded = false;
    end;
    ECM.SoftwareTrigger;  % Use software triggering to insure
synchronization between ECM and display

    % Fixation off & inter_trial
    screen(f,'FillRect',BlackIndex(0),fRect);  % makes a white square in
offscreen window 'f'
    screen(window,'WaitBlanking');
    screen('CopyWindow',f,window);  % Copies 'f' onto 'window'
    disp('0: Fixation Off & Inter-Trial');
    pause(inter_trial);   % Pause allows for CTRL+C to be read

    % To re-synchronize trials that went out of sync without the
    % photodiode, send the state machine to state 0 by turning it on/off;
    if (~usePhotodiode)
        ECM.Stop;
        ECM.Run;
    end;

    xy=ECM.GetXYData;    % Data comes in triplets (State, X, Y)
    if (length(xy)>0)

        % Extract State, X and Y in separate vectors
        lx=floor(length(xy)/3);
        xi=1:3:3*lx;
        s=xy(xi);
        x=xy(xi+1);
        y=xy(xi+2);
        disp(sprintf(' Acquired %d XY samples.',lx));
        lt=1000.0*lx/ECM.XYSamplingFrequency;

        % Plot XY data (figure will not be updated while psychophysics
        % toolbox is locking the display)
```

```
        max_xy=max(abs(xy));
        max_xy=max(max_xy,9000);
        h=figure(2);
        set(h,'Color','k');
        % X subplot
        h=subplot(2,1,1);
        %set(h,'Visible','Off');
        cla;
        hold on;
        box on;
        axis([1 lx -max_xy max_xy]);
        h=title('X');
        us=unique(s);
        us=sort(us);
        % Plot each state with a different color
        for i=us
            ix=find(s==i);
            h=plot(ix,x(ix),'-');
            set(h,'Color',state_colors{1+mod(i,length(state_colors))});
            text(ix(1),0.15*max_xy,sprintf('State
%d',i),'Rotation',90,'FontSize',7,'VerticalAlignment','Top');
            h=line([ix(1) ix(1)],[-0.9*max_xy 0.9*max_xy]);
            set(h,'LineStyle',':');
        end;
        h=line([1 lx],[0 0]);   % zero line
        set(h,'LineStyle',':','Color','y','Visible','On');
        hold off;
        % Y subplot
        h=subplot(2,1,2);
        %set(h,'Visible','Off');
        cla;
        hold on;
        box on;
        axis([1 lx -max_xy max_xy]);
        h=title('Y');
        xlabel('time');
        %plot(y,'w-');
        % Plot each state with a different color
        for i=us
            iy=find(s==i);
            h=plot(iy,y(iy),'-');
            set(h,'Color',state_colors{1+mod(i,length(state_colors))});
            text(iy(1),0.15*max_xy,sprintf('State
%d',i),'Rotation',90,'FontSize',7,'VerticalAlignment','Top');
            h=line([iy(1) iy(1)],[-0.9*max_xy 0.9*max_xy]);
            set(h,'LineStyle',':');
            %[i min(iy) max(iy)]
        end;
        h=line([1 lx],[0 0]);   % zero line
        set(h,'LineStyle',':','Color','y','Visible','On');
        hold off;
        drawnow;
        pause(0.1);
        clear xy;
    else
        disp('No data !');
    end;
```

```
end;

screen('CloseAll')  % closes window and all offscreen windows

ECM.XYAcquire = 0;
ECM.Stop;    % Stop the state machine if for some reason it hasn't been
stopped
ECM.PortOpen = 0; % Close comm port
close(1);
```

A number of other programming examples are located in the Matlab subdirectory of the ECM program folder.

2.5 *SCHEDULED MAINTENANCE*

The Gain and Calibration of the analog inputs, described in section 2.3 Functional Checkout of this manual, should be performed on a yearly basis. If any discrepancies are found, please contact Technical Services at (207) 666-8190.

2.6 *REFERENCE*

The following is a categorical list of the ECM ActiveX Properties, Methods. And Events. Please reference the online help of the ECM program group for more in-depth information.

General Properties

Timing

StateMachineClock	Sets/Retrieves to frequency (in Hz) at which the state machine operates.
UseMilliseconds	Sets/Retrieves the measurement unit (milliseconds or state machine clocks) that is used for all the ECM timing parameters.

Communications

CommPort	Sets/Retrieves the USB port number, or USB device ID, to connect to.
PortOpen	Sets/Retrieves the status (open/closed) of the USB communications between PC and ECM.

Status

ModuleState	Retrieves the state of the ECM module. Requires that **NotifyPC** property is set to **TRUE** in order for the PC to be aware of which state ECM has entered while running. This is different from the state that is selected (by setting the **State** property) for editing its properties. This is a read-only property.

State Selection & Counts

NumStates	Sets/Retrieves the total number of states defined for the state machine.
State	Sets/Retrieves to which state is currently selected for editing. All state-specific properties apply for the currently selected state.

Analog Inputs & Data Acquisition

VoltageCalibration	Sets/Retrieves the voltage calibration of both analog inputs, in volts per ADC level.
XYAcquire	Sets/Retrieves whether analog data acquisition is enabled. This is a global flag. For turning on and off the data acquisition in a specific state, use **XYDataAcquire** property.
XInputCalibration	Sets/Retrieves the calibration constant of the X analog input. This constant takes values near 1.0, and is meant to balance the X and Y inputs, so that they provide identical values. For changing the global voltage calibration, use **VoltageCalibration** property.
XInputOffset	Sets/Retrieves the input offset of the X analog input.
XYSamplingFrequency	Sets/Retrieves the ADC sampling frequency of the analog inputs.
XYViewInterval	Sets/Retrieves the interval at which the marker position is updated on screen. If data acquisition is turned on, the marker position is updated every time a new XY data packet arrives, but if it's turned off, it is updated every *n*=**XYViewInterval** ms or state machine clocks, depending on the value of the **UseMilliseconds** property.
YInputCalibration	Sets/Retrieves the calibration constant of the Y analog input. This constant takes values near 1.0, and is meant to balance the X and Y inputs, so that they provide identical values. For changing the global voltage calibration, use **VoltageCalibration** property.
YInputOffset	Sets/Retrieves the input offset of the Y analog input.

Serial Interface and Serial Data Acquisition

TouchAcquire	Sets/Retrieves whether touch screen or mouse position data acquisition is globally enabled. This is a global flag. For turning on and off the data acquisition in a specific state, use **TouchDataAcquire** property.
TouchCalibration	Sets/Retrieves the calibration constant of the touch panel or serial mouse, in desired position units (meters, inches, degrees of visual angle etc) per step.
TouchDevice	Sets/Retrieves the device type (serial mouse, different touch screen controllers) connected to ECM's serial port.
TouchSamplingFrequency	Sets/Retrieves the frequency (in Hz) at which the touch screen or mouse coordinates inputs are sampled.
TouchTraceNumPoints	Sets/Retrieves the number of position samples displayed as a trailing trace to the position marker, so that the trajectory in XY space of the behavioral variable can be better visualized.
TouchViewInterval	Sets/Retrieves the interval at which the marker position is updated on screen in states where the touch screen / serial mouse position data acquisition is turned off. The interval can be specified in milliseconds or number of state machine clocks, depending on the value of the **UseMilliseconds** property.
TouchXCalibration	Sets/Retrieves the calibration constant of the X coordinate reported by touch screens or mice connected to ECM's serial port. For changing the global touch screen calibration, use **TouchCalibration** property.
TouchXDisplayRange	Sets/Retrieves the range of values for the horizontal position (X axis) reported by the touch screen or mouse connected to ECM's serial port, that are displayed in control's window.
TouchXOffset	Sets/Retrieves the offset value of the touch screen or mouse X coordinates, that is substracted from the input.
TouchYCalibration	Sets/Retrieves the calibration constant of the Y coordinate reported by touch screens or mice connected to ECM's serial port. For changing the global touch screen calibration, use **TouchCalibration** property
TouchYDisplayRange	Sets/Retrieves the range of values for the vertical position (Y) reported by the touch screen or mouse connected to ECM's serial port, that are displayed in control's window.
TouchYOffset	Sets/Retrieves the offset value of the touch screen or mouse Y coordinates, that is substracted from the input.
UARTSettings	Sets/Retrieves the settings (baud rate, number of data bits, parity, stop bits) of ECM's asynchonous serial port (UART).

Control Appearance

BackColor	Sets/Retrieves the background color of the ActiveX control.
ForeColor	Sets/Retrieves the foreground color of the ActiveX control.
MarkerSize	Sets/Retrieves the size of the marker showing the current position (for instance eye position) for the signal at the analog inputs.
XDisplayRange	Sets/Retrieves the range of horizontal (X) voltages that is displayed in control's window.
XYTraceNumPoints	Sets/Retrieves the number of ADC samples displayed as a trailing trace to the position marker, so that the trajectory in XY space of the behavioral variable can be better visualized.
TouchTraceNumPoints	Sets/Retrieves the number of touch screen or mouse position samples displayed as a trailing trace to the position marker, so that the trajectory in XY space of the behavioral variable can be better visualized.
YDisplayRange	Sets/Retrieves the range of vertical (Y) voltages that is displayed in control's window.

State-Specific Properties

Timing

MinDuration	Sets/Retrieves the minimum duration the state machine stays in a state, even if a transition trigger has occured. The transition is queued when it occurs, then is performed at the end of the specified time interval. The duration is expressed as the number of state machine clocks.

Notifications

NotifyPC	Sets/Retrieves whether ECM is going to notify PC when it has entered the current state.

Digital Inputs

DigitalInputsActive	Sets/Retrieves which of the 16 digital input lines can trigger a transition to a different state.
DigitalInputsActiveLow	Sets/Retrieves which of the active digital inputs trigger a transition to another state when reaching a low level.
DigitalInputsNextState	Sets/Retrieves to which state a transition is made when one of the digital inputs changes state.
Event1InputActive	Sets/Retrieves whether Event 1 input can trigger a transition to a different state.
Event1InputActiveLow	Sets/Retrieves whether Event 1 input triggers a transition to another state when it has a low level.
Event1InputNextState	Sets/Retrieves to which state a transition is made when Event 1 occurs.
Event2InputActive	Sets/Retrieves whether Event 2 input can trigger a transition to a different state.
Event2InputNextState	Sets/Retrieves to which state a transition is made when Event 2 occurs.
Event3InputActive	Sets/Retrieves whether Event 3 input can trigger a transition to a different state.
Event3InputActiveLow	Sets/Retrieves whether Event 3 input triggers a transition to another state when it has a low level.
Event3InputNextState	Sets/Retrieves to which state a transition is made when Event 3 occurs.
Event4InputActive	Sets/Retrieves whether Event 4 input can trigger a transition to a different state.
Event4InputNextState	Sets/Retrieves to which state a transition is made when Event 4 occurs.

Software Input

SoftwareEventInputActive	Sets/Retrieves whether Software Event variable can trigger a transition to a different state.
SoftwareEventNextState	Sets/Retrieves to which state a transition is made when Software Event occurs.

Analog Inputs

XLowerBound	Sets/Retrieves the lower bound of the window used for checking the values of the X analog input.
XUpperBound	Sets/Retrieves the upper bound of the window used for checking the values of the X analog input.
XYMonitorActive	Sets/Retrieves whether the virtual device that checks the X and Y inputs against a predefined window can trigger a transition to a different state.
XYMonitorActiveInbound	Sets/Retrieves whether the process that monitors the analog position can trigger a transition when it detects the position to be inside/outside the bounding rectangle defined by **XLowerBound**, **XUpperBound**, **YLowerBound**, and **YUpperBound** properties.
XYMonitorNextState	Sets/Retrieves to which state a transition is made when the XY monitor detects the input voltages to be out-of-bounds.
XYMonitorSource	Sets/Retrieves what channel the XY monitor uses: XY analog inputs or the "virtual input" that is obtained by differentiating in real time the analog inputs. For instance, if X and Y are position inputs, then the virtual inputs obtained by differentiation represent the velocity.
YLowerBound	Sets/Retrieves the lower bound of the window used for checking the values of the Y analog input.
YUpperBound	Sets/Retrieves the upper bound of the window used for checking the values of the Y analog input.

Serial Interface

TouchMonitorActive	Sets/Retrieves whether the process that monitors the touch or mouse position is active and can trigger a transition to a different state when it detects the position to be inside/outside (depending on the **TouchMonitorActiveInbound** property) the bounding rectangle defined by **TouchXLowerBound**, **TouchXUpperBound**, **TouchYLowerBound**, and **TouchYUpperBound** properties.
TouchMonitorActiveInbound	Sets/Retrieves whether the process that monitors the touch screen or mouse position can trigger a transition when it detects the position to be inside/outside the bounding rectangle defined by **TouchXLowerBound**, **TouchXUpperBound**, **TouchYLowerBound**, and **TouchYUpperBound** properties.
TouchMonitorNextState	Sets/Retrieves to which state a transition is made when the touch screen or mouse position is detected to be out-of-bounds.
TouchMonitorSource	Sets/Retrieves what channel the touch scren or serial mouse position

	monitor uses: position values or the values obtained by differentiating in real time the position.

Timers

GlobalTimer1Active	Sets/Retrieves whether the built-in global timer 1 is running and can trigger a transition to a different state when its value matches the **GlobalTimer1Match** value.
GlobalTimer1Match	Sets/Retrieves the value that GlobalTimer1 must take in order for a transition to a different state to be triggered. The duration can be specified in milliseconds or number of state machine clocks, depending on the value of the **UseMilliseconds** property.
GlobalTimer1MatchNextState	Sets/Retrieves to which state a transition is made when the GlobalTimer1 value matches the GlobalTimer1Match property.
GlobalTimer1Reset	Sets/Retrieves whether the built-in global timer 1 is going to be reset upon entering the current state.
GlobalTimer2Active	Sets/Retrieves whether the built-in global timer 1 is running and can trigger a transition to a different state when its value matches the **GlobalTimer2Match** value.
GlobalTimer2Match	Sets/Retrieves the value that GlobalTimer2 must take in order for a transition to a different state to be triggered. The duration can be specified in milliseconds or number of state machine clocks, depending on the value of the **UseMilliseconds** property.
GlobalTimer2MatchNextState	Sets/Retrieves to which state a transition is made when the GlobalTimer2 value matches the GlobalTimer2Match property.
GlobalTimer2Reset	Sets/Retrieves whether the built-in global timer 1 is going to be reset upon entering the current state.
TimerActive	Sets/Retrieves whether the timer starts/keeps running when the state machine enters the current state.
TimerDuration	Sets/Retrieves whether the duration, in state machine clocks, after which the timer triggers a transition to another state.
TimerExpiredNextState	Sets/Retrieves to which state a transition is made when the timer counter expires.
TimerKeepsCountingFromPreviousState	Sets/Retrieves whether the timer keeps counting from previous state, or it is restarted at the beginning of the current state.

Digital Outputs

DigitalOutputs	Sets/Retrieves the value of the 16 digital output lines.

DigitalOutputsActive	Sets/Retrieves the state (driven / high-impedance) of the 16 digital output lines.
ESGTrigger	Sets/retrieves whether ESGTrigger is on or off for the current state.
ESGTriggerDuration	Sets/retrieves the duration ESGTrigger output stays high during the current state. At the end of the specified interval, the ESGTrigger output is reset.
RecordGate	Sets/Retrieves whether Trial/Record Trigger/Gate output is on or off for the current state.
RewardTrigger	Sets/Retrieves whether Reward Trigger output is on or off for the current state.
RewardTriggerDuration	Sets/retrieves the duration RewardTrigger output stays high during the current state. At the end of the specified interval, the RewardTrigger output is reset.
VSGTrigger	Sets/Retrieves whether Visual Stimulus Generator (VSG) Trigger output is on or off for the current state.
VSGTriggerDuration	Sets/retrieves the duration VSGTrigger output stays high during the current state. At the end of the specified interval, the VSGTrigger output is reset.

Data Acquisition

TouchDataAcquire	Sets/Retrieves whether analog data acquisition is on or off for the current state.
XYDataAcquire	Sets/Retrieves whether analog data acquisition is on or off for the current state.

Methods

AboutBox	Displays information about the software and hardware.
ClearState	Reset all variables of the current state to their defaults.
GetDigitalInputsNextState	Retrieves the state to which a transition is made when the specified line of the 16-bit digital input triggers a transition. Each digital line can trigger a transition to a different state.
GetGlobalTimers	This method is invoked to retrieve at any time the value of the global timers. Since these values are not available to the PC in real time, the application posts a request to ECM and returns immediately (no value provided). When ECM replies to the request, a **GlobalTimers** event is fired, and the values of the timers are returned as arguments of the event.
GetXYData	Retrieves analog XY data from the ECM module.
GetTouchData	Retrieves XY touch screen or mouse position data from the ECM module.

Reset	Performs a software reset of the module.
Run	Starts the state machine.
SetXYLoopbackData	Sets loopback XY data used for development/testing stage of the experiment. Data simulates analog inputs.
SetDigitalInputsNextState	Sets to which state a transition is made when the specified line of the 16-bit digital input triggers a transition. Each digital line can trigger a transition to a different state.
SoftwareTrigger	Triggers a transition to another state, if module is in a state in which **SoftwareEventInputActive** property is set to **true** at the time the commmand was issued.
Stop	Stops the state machine and brings it in the initial state.
UARTWrite	Writes a character (byte) to the ECM asynchronous serial port (UART) .
UpdateState	Updates the information about the current state on the ECM. Any modification of the state properties is local on the PC, and is sent to ECM when **UpdateState** command has been issued.
UpgradeFirmware	Upgrades the DSP firmware.

Events

StateChanged	Event that is fired when ECM has notified PC that a transition to a new state has been made. Occurs only when **NotifyPC** is set to **true** in the new state.
DigInputChanged	Event that is fired when ECM has notified PC that a transition to a new state has been made. Occurs only when **NotifyPC** is set to **true** in the new state.
GlobalTimers	GlobalTimers event is fired following a request by the application for the global timers values (by invoking **GetGlobalTimers**).
TouchData	Event that is fired when new behavioral data (XY) has been received from ECM.
XYData	Event that is fired when new behavioral data (XY) has been received from ECM.