

Journal of Neurophysiology

**A real-time supervisor system based on trinary logic  
to control experiments with behaving animals and humans**

D.F. Kutz<sup>1</sup>, N. Marzocchi<sup>1</sup>, P. Fattori<sup>1</sup>, S. Cavalcanti<sup>2</sup>, C. Galletti<sup>1</sup>

<sup>1</sup>Dipartimento di Fisiologia Umana e Generale, Università di Bologna,

Piazza Porta San Donato 2, 40126 Bologna, Italy

<sup>2</sup>Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna,

Viale Risorgimento, 2 - 40126 Bologna, Italy

Running title: Real-time supervisor system

FINAL ACCEPTED VERSION

Number of text pages: 37

Number of figures: 11

Number of tables: 4

Corresponding author: D.F. Kutz, address as above

Tel.: +39 051 2091748

Fax: +39 051 251731

Email: [kutz@biocfarm.unibo.it](mailto:kutz@biocfarm.unibo.it)

**Abstract**

A new method is presented based on trinary logic able to check the state of different control variables and synchronously record the physiological data and behavioral data of behaving animals and humans. The basic information structure of the method is a time-interval of defined maximum duration, called time-slice, during which the supervisor system periodically checks the status of a specific subset of input channels. An experimental condition is a sequence of time-slices subsequently executed according to the final status of the previous time-slice. The proposed method implements in its data structure the possibility to branch like an if-else cascade and the possibility to repeat parts of it recursively like the while-loop. Therefore, its data structure contains the most basic control structures of programming languages. The method was implemented using a real-time version of LabVIEW™ programming environment to program and control our experimental set-up. Using this supervision system, we synchronously record 4 analog data channels at 500Hz (including eye-movements) and the time-stamps of up to 6 neurons at 100kHz. The system reacts with a resolution within 1ms to changes of state of digital input channels. The system is set to react to changes in eye-position with a resolution within 4ms. The time-slices, experimental conditions and data are handled by relational databases. This facilitates the construction of new experimental conditions and data analysis. The proposed implementation allows continuous recording without an inter-trial gap for data storage or task management. The implementation can be used to drive electrophysiological experiments of behaving animals and psychophysical studies with human subjects.

## **Introduction**

Electrophysiology research in behaving animals is still a growing field in neuroscience as seen from the ongoing increase in the number of publications each year. The use of sophisticated digital systems (based on microcomputer and related software) in the experimental setup has allowed neural activity to be studied in behaving animals in more and more complex experiments. Nowadays, some research methodologies require the synchronous recording of the bioelectrical activity of different neurons, eye-movements, electromyography, and other external events or data describing the behavior of the animal.

The algorithms described in the literature for neurophysiological experiments with animals or humans (Kling-Petersen and Svensson, 1993; Budai, 1994; Nordstrom et al., 1995; Poindessault et al., 1995; Desimone et al., 2004) have two main problems. The first is due to the limited computer memory size. This forces algorithms to be developed with a maximum recording time limited by the hardware so that experiments must be designed in a trial-by-trial fashion. The second problem is a consequence of the trial-by-trial design of the experiment. When an animal does not react as predicted by the scientist (for instance a monkey does not perform eye-movements as intended) during a trial, the program has to stop and wait until the experimental situation comes back to the origin before starting a new trial. Any information during this time is lost.

To overcome these limitations we developed a new method based on trinary logic able to check the state of different control variables and continuously record electrophysiological data and data describing the animal's behavior. The basic information structure of the method is a time-interval of defined maximum duration, called time-slice, during which the supervisor system periodically checks the status of a specific subset of input channels. The status of a time-slice is based on trinary logic which refers to the three behavior-dependent states: non-fulfillment of required behavior, correct performance of task, or erroneous task performance. An experiment is

seen as a sequence of time-slices, subsequently executed according to the final state of the previous time-slice.

This paper describes the algorithm and its implementation using the real-time version of LabVIEW™ programming environment (National Instruments Corp., Austin (TX), U.S.A.). Since its introduction, the LabVIEW G-language (Kodosky and Dye, 1989) has been one of the most powerful software tools to create easily supervised systems to control experiments. This is demonstrated by numerous examples in the literature, showing the advantages of a graphical software environment in human and animal studies (in vitro: Budai et al., 1993; Kullmann et al., 2004; in vivo: Kling-Petersen and Svensson, 1993; Budai, 1994; Nordstrom et al., 1995; Poindessault et al., 1995; animal behavior: Puppe et al., 1999).

Our algorithm also allows continuous recording in the case of unpredicted animal behavior, and can be used to drive psychophysical studies with normal human subjects or patients. Preliminary data have been presented in abstract form (Marzocchi et al., 2004).

## **Method**

This section is divided into two parts. The first (Theory) describes the new method, introducing the concept of a time-slice and its algorithm and higher level structures based on the time-slice concept. The second part describes the method implementation and the hardware used. In addition, the Appendix contains a more detailed description of the different procedures implemented with the LabVIEW G-language (Kodosky and Dye, 1989).

## **Theory**

### Control of experiments using behaving animals

Experiments using awake, behaving animals are difficult to control because the behavior of an animal is only partially predictable. Although animals like monkeys can be trained for certain tasks (e.g. to perform a saccade towards a target), the induced behavior will be correctly performed with a probability less than 100%. Experiments based on animal behavior must be designed so that they take into account the probabilities of an animal producing behavior in the sequence and timing intended by the scientist. The timing of the task sequence expresses the given probabilities of an animal producing a behavior (for instance the maximum time from a stimulus to execution of a behavior). A supervisor system must be able to wait for the execution of an intended behavior and react immediately when it is produced, or interrupt the task if the animal does not produce the behavior in a given time. Therefore, any element of a task sequence can have three distinct states: ongoing behavior, correct completion of behavior or false completion of behavior. The three states can be expressed by trinary logic.

The problems of controlling an electrophysiological experiment with behaving animals are illustrated by an example from the literature (Fattori et al., 2001). In this experiment a monkey (*Macaca fascicularis*) had to produce straight arm movements from a start button towards a defined target button indicated by an illuminated LED, and then come back to the start button (Fig. 1A). In the meantime, the monkey had to maintain fixation of the reaching target (LED). Bioelectrical

activity of single neurons in the posterior parietal cortex was recorded extracellularly. The computer had to set the LED (off, green, red) to control the buttons (off, pushed), control the position of the eyes and record the event of neural activity as well as eye movements and events of change of the external apparatus (LED, buttons). The sequence of the experiment is the following. The monkey:

1. presses the start button (placed at chest level in front of it)
2. waits for a variable time for a LED in front of it to light up green
3. makes a saccade towards the position of the LED
4. fixates the LED and waits for it to change color (green -> red)
5. releases the start-button as a result of a color change
6. makes an arm movement towards the target button and presses the target button
7. remains with its hand on the target button and waits for the LED to switch off
8. releases the target button as a result of the LED switch-off
9. makes an arm movement back to the start button and presses the start button
10. is rewarded.

The experiment is shown in figure 1B as a time sequence of states of input channels to be controlled (upper part) and output channels to be set (lower part). Advancement in the sequence depends not only on the supervisor system but also in large part on the behavior of the animal. For example, it is impossible either to foresee the exact time an animal needs to release the start button after the LED color change or the time needed to move its arm towards the target button. It is only possible to set some upper limits for these intervals in ranges of physiological behavior. Depending on the animal's behavior, the supervisor system has to decide to go ahead in the experimental sequence or to change requests to the animal as necessary (for example breaking the trial and finishing any type of stimulation).

**Figure 1 about here**

An analysis of the state-graph in figure 1B, will show how a supervisor system can control different behavior requirements. Behavior requirements can be classified as four kinds: to remain in the current behavior (e.g. fixation of a visual target), to reach a behavioral condition (e.g. starting to press a button), to end a behavioral condition (e.g. releasing a button), to avoid producing a behavior (e.g. not making a saccade to a distracter). Examples for the first three behavior requirements are shown in figure 1B in the 1<sup>st</sup>, 4<sup>th</sup>, and 5<sup>th</sup> intervals. At the beginning (interval 1) the supervisor system waits for the monkey to press the start button. Expressed in sense of information, the system waits for the condition “pushed” to be reached. In a subsequent interval (no. 4), when the LED lights up green and the monkey fixates it, the supervisor system checks that the start button remains “pushed”. In the next interval (no. 5) the LED changes color and the system waits for the condition “pushed” of the start button is end. This description, confined for simplicity to the state change of the start button, shows that a pre-defined condition of a control variable can be reached in a time interval, can end during a time interval, or can remain for a time interval.

At the moment in which a pre-defined condition is reached or ends, the time interval finishes and the control program advances in the sequence. In contrast, in cases in which the condition has to remain at a pre-defined value the time interval finishes after its whole execution. Thus, during each interval, here often called time-slice, the input channels (except channel Eye) and output channels remain unchanged, as the state-graph reveals (Fig. 1B). In those cases in which a change in input channel is expected, it is the change itself which provokes the transition from one time-slice to the next. The same is true for the output channels: output channels are changes at the beginning of a time-slice. Any change in an output channel first needs a transition from one time-slice to the next.

The input channel Eye is a special input channel, not only because eye position data are analog and not digital (like the start button), but also because it is not always necessary to control eye movements. In the example experiment of figure 1, eye movements are controlled only during the 3<sup>rd</sup> to 8<sup>th</sup> time intervals.

### The time-slice state

Summarizing the above description, a time-slice can be defined as a time-interval of defined maximum duration ( $T_{\max}$ ) during which neither the input or output signals changed in relation to prefixed values. Four distinct time-slices can be introduced in consideration of the behavior: Remain, Reach, End, and Avoid. The supervisor system periodically checks the time elapsed ( $T$ ) and a time-slice-specific subset of input channels. Each of these input channels is compared to time-slice-specific conditions yielding a state value  $S_{in} = 0, 1$  or  $2$  for each comparison (Table 1). In addition, time  $T$  is assessed compared to the maximum time allowed for that slice ( $T_{\max}$ ) and the  $S_T$  value is assigned. The sum of  $S_{in} + S_T$  determines the time-slice state, and thus progression of the subsequent behavior of the control algorithm (Table 2).

**Table 1 about here**

**Table 2 about here**

Example: In time-slice 1 (Fig. 1) the monkey is expected to press the start button, therefore the supervisor system waits for a change in the digital input channel monitoring the start button. As long as the button is not pressed and time elapsed is shorter than the time allowed for time-slice 1, digital input status ( $S_{in}$ ) and time status ( $S_T$ ) values are both 0 and the supervisor system stays in time-slice 1. If the monkey presses the start button in time, the digital input status ( $S_{in}$ ) becomes 1 and the supervisor system proceeds to time-slice 2. Finally, if the maximum time expires without the start button being pressed, the time status ( $S_T$ ) becomes 2 and the time-slice finishes incorrectly, causing the supervisor system to restart the experiment.

An important feature of the time-slice concept is that during a time-slice only one input channel is monitored for a change while the other input channels are either ignored or are expected to maintain their status without a change. An example for the latter case is time-slice 3 when the



LED in front of the monkey is changed to green and the monkey is expected to make a saccade towards the LED while keeping the start button pressed. In this time-slice the digital input channels of start button and target button are expected to remain unchanged, whereas the channel Eye reaches a reference value (see table 1). If the task consists of making the saccade and moving the arm from one button to another, it has to be split into two time-slices. In the first time-slice the input from the analog channel monitoring eye-position would be expected to change. Then, in the second time-slice, the execution of the arm movement would be monitored. Because saccades are faster than arm movements, this strategy works quite well.

The condition: an orderly sequence of time-slices

The next step is the construction of an experimental condition based on a sequence of time-slices. In principle, this involves composing an experiment as a sequence of time-slices like the experiment shown in figure 1. In the experiment of figure 1 the supervisor system executes the sequence until the animal behaves as foreseen by the protocol. Problematic is the situation when the animal does not behave as required. To handle this situation a specific time-slice called error-handling slice (eh) has been introduced with the same data-structure as the other time-slices. Its function is to manipulate the experimental situation up to a point at which a new experimental condition can start. Therefore, two pointers must be added to the data-structure of a time slice, one for the advancement following a correct end of time slice (time-slice state = 1) and one following an erroneous end (time-slice state > 1). These pointers will be called true-index and false-index, respectively. For convenience time-slices in condition sequences will be called working slices (ws). An example of a condition based on a sequence of time-slices is shown in figure 2A. The figure shows that the false-index of the error-handling slice points back to itself. This behavior is necessary because any time-slice must have a defined maximum duration, but it cannot be foreseen how much time is required until the experimental situation allows a new start of the experiment. Pointing the

false-index of the error-handling slice back to itself creates a continuous repetition of this time-slice until the true-index condition is achieved (while-loop on a time-slice).

In principle, it is possible to create a specific error-handling slice for each working slice. But, experience of a real implementation showed that it is much more convenient to handle the false-indexes in a way that they point to just one error-handling slice. Note that an error-handling slice is part of a condition. A condition ends with the end of execution of its last working slice or the end of execution of an error-handling slice. To concatenate a sequence of conditions, the true-indexes of any possible last slice (irrespective of whether it is a working slice or an error-handling slice) must point to the first working slice of a condition (see figure 2A). It is worth noting that the true-index pointer of the error-handling slice can be used for different training strategies, i.e. the same condition can be repeated by pointing back the true-index to the first working slice of the same condition (while-loop on a condition) or, as a different strategy, pointing the true-index to the first workings slice of the following condition (like in figure 2A).

The fact that each time-slice has two successors, usually the next working slice and an error-handling slice, shows an important behavior of the algorithm, the possibility of implementing a branch in the execution. This possibility can be used to meet the needs of experimental interest. For example, you may want to study the neural response in a two choice reaction task, like presenting two visual stimuli to an animal and asking it to decide if they are similar or not by making a saccade upward or downward, respectively. Depending on the reaction of the animal the experiment will proceed in one of two different manners. In cases like this, the two pointers (true- and false-index) of the working slice revealing the animal's decision, will point to a working slice. Such a situation is shown in figure 2B. The branch point in the flow of the experiment is at the second working slice (ws\_2 in figure 2B). The experimental condition described in figure 2B contains the time-slice sequence ws\_1, ws\_2, ws\_3, ws\_4, ws\_7 as well as ws\_1, ws\_2, ws\_5, ws\_6, ws\_7. It is worth noting that the two branches converge on one time-slice (ws\_7) before ending the condition. This is not necessary, but it simplifies the handling of the pointers to the first working slice of the next

condition. A problem to solve here is again the defined maximum duration of the time-slice. In the example the branch of the false-index will be followed either if during the previous working slice (ws\_2) the animal made a saccade downward or if the time-slice was timed out. So the first thing to do in the false-index branch is to check if the saccade occurred. This can be done by constructing a working slice (ws\_5 in figure 2B) in which the animal has to fix on the downward position (=> Remain in table 1). If the animal has reached the position the supervisor system proceeds as required otherwise it terminates immediately. In principle, another branch can be added in any branch. So, the program can behave like an if-else cascade.

### **Figure 2 about here**

Task: a sequence of conditions

The task structure will be introduced to complete the description of the experimental protocol. A task is referred to as a set of one or more conditions. For example, if the experimental interest is to study the neural response to saccadic eye movements, it could be desirable to study the neural response to different movement directions and/or amplitudes. Each test to one direction or amplitude will be a different experimental condition. All conditions prepared for an experiment will be grouped in one task (e.g. the eight conditions for studying saccadic eye movements along the directions of the cardinal and oblique axes). No order is imposed on the execution of the conditions. The decision to use an orderly sequence of conditions or a randomized one often depends on the experimental protocol, so it should be possible to present to the animal an orderly sequence as well as a randomized one.

## Implementation

The method is implemented so as to perform a task, like that described by Fattori et al. (2001). Our actual setup (figure 3) allows single- and multi-electrode recordings. Neurons are separated by means of a Schmitt trigger (BAK Electronics Inc., Germantown (MD), U.S.A.) or a Multiple Spike-Detector (Alpha-Omega Eng., Nazareth, Israel) and their time-stamps are recorded by a counter device used for buffered event counting. Eye positions are measured using an infrared oculometer (DR. BOIUS Devices for movement measurements, Karlsruhe, Germany) and recorded by an analog input device of a multifunctional-board. LED and buttons are through a homemade interface controlled by a digital I/O device.

The method is implemented using the LabVIEW G-language (Kodosky and Dye, 1989) on two computers (see figure 3), an ordinary PC running Windows (host-PC) and a dedicated PXI/CompactPC (National Instruments) for real-time control and data acquisition. For a flexible management of time-slices, conditions and tasks, we used a relational database (Microsoft Access™) read out by LabVIEW. The host-PC serves as the interface between the user and the PXI-PC. It programs on the PXI, stores the data acquired by the PXI-PC, and analyzes the acquired data online in the form of peri-stimulus time histograms. The PXI-PC contains the program to control the experiment and acquire analog, digital and counter-data.

### **Figure 3 about here**

#### Hardware

PXI (PCI eXtensions for Instrumentation) is a CompactPCI specification, which combines the high-speed PCI bus with integrated timing and triggering designed specifically for high performance measurements. Deterministic real-time performance is achieved with an embedded controller and LabVIEW real-time environment. PXI boards can be connected together for precise synchronization of functions by means of a real-time system integration bus (RTSI bus). LabVIEW and the PXI components are available from National Instruments Corp. (Austin, Texas, U.S.A.).

In our present system the PXI-PC control module is built into an eight-slot chassis and includes a Pentium III-processor, RAM and a hard disk. We use a multifunction data acquisition device for analog acquisition, a 96-channel Digital-IO device for digital input/output operations and an 8-channel counter/timer device for spike acquisition and internal timing utilities (table 3). We dedicated 2 counters of the Timing IO-device for timing purposes of the PXI-program. One counter produces TTL-pulses at a frequency of 500Hz to command the analog acquisition. Higher frequencies to command an analog acquisition device are possible. The maximum frequency depends mainly on the characteristics of the device. The second counter serves to count the ongoing time and works at a frequency of 100kHz. The other 6 counters are reserved for acquisition of time-stamps of detected spikes or external digital events (for this reason we use the word counter-data rather than spike-data) and work at a frequency of 100kHz. To record more time-stamps additional Timing IO-devices (each with 8 counters) can be added. The devices are wired to the periphery by a homemade interface which serves for output signal adaptation (i. ex. the reward system needs a 12 V command-voltage) or stabilize incoming signals from pressed buttons (using flip-flop circuitry). In our application the RTSI bus is used to trigger and synchronize analog data and spikes acquisition. One of the PXI trigger bus lines is used to start all hardware devices synchronously and another one is used to route a square wave generated by one of the Timing-IO device counters to the scan clock input of the analog data acquisition board. This allows a time synchronized analog data acquisition at constant intervals.

**Table 3 about here**

## Software

The supervisor system is organized in two main modules running on the host-PC and the PXI-PC respectively (for this reason indicated as host-PC program and PXI-PC program). This structure was fixed by the fact that the real-time system runs as a target on which the host-computer downloads applications, and it does not provide a user interface for applications. Data could have been stored on PXI-PC hard disk, however we preferred host-PC because this allows us to perform some on-line analyses of the data, such as trials statistics and neural discharges visualization (raster plots and PSTH).

The front-panel (user interface) of the host-PC program contains all the controls needed by the experimenter, such as task choice, selection of active spike channels, buttons to start and stop the experiment, and some useful indicators, like trial counters, program status, etc. Furthermore the host-PC program communicates by a TCP/IP line with the PXI-PC program, sending user commands and receiving data and other information. Received data are stored on a relational database, allowing easier retrieval of data for on-line and off-line analyses using simple SQL queries. In particular this enables highly flexible on-line analysis of the spike trains: during the acquisition it is possible to change the visualized spike channels, the marker used to align neural activity and the conditions analyzed. This provides the scientist with a good tool to understand recorded neurons behavior to optimize their investigation.

The host-PC drives the PXI-PC program, which has three main duties: controlling the advancement of the experimental sequence, acquiring data and transmitting them to the host-PC. The control loop, executed with a real-time priority, represents the implementation of the time-slice algorithm. It handles one time-slice at a time. First, it sets the output values, and then continuously reads the actual values of the input signals (digital lines, analog channels and elapsed time), comparing them with the intended values to calculate the state value. Based on this result the control loop continues to check the input signals or proceeds with another time-slice as described in

the theory section. Further information on implementation details and a description of some emblematic procedures can be found in the Appendix.

#### Using relational databases for data and tasks

As previously mentioned, to achieve a flexible management of information regarding task structures and acquired data we used relational databases based on the Structured Query Language (SQL), readable with different DBMS such as LabVIEW Database Toolkit using the Open DataBase Connectivity (ODBC).

The database containing information on the structure of time-slices, conditions and tasks and their linkage includes five tables joined as shown in figure 4A. Any task/condition/time-slice is identified by a unique reference number called ID and is described by a record in the corresponding descriptor table, containing as fields all the general parameters necessary for its definition. The *Time-Slice Descriptor* includes the ID of the time-slice, its name and description, a set of output values and the subset of input to be monitored and their awaited values. The *Task Descriptor* record includes only its ID, its name and a brief description string. The remaining two tables are used to define condition and task structures. Each record in the *Condition Structure* Table associates a time-slice with a condition, states the time-slice position in the sequence and specifies the true-index and the false-index. Therefore by querying this table pre-defined time-slices can be assembled into any kind of experimental sequence considering all the records with the same ID Condition. Similarly different conditions can be assembled into a task by means of the *Task Structure* Table.

The database for acquired data includes four tables as shown in figure 4B. The main table called *Recordings* associates to each experimental recording an identification number (ID recording) used to reference the entrances of the other tables, and specifies the path of the binary file containing analog data. This kind of storage for analog data was chosen to enhance the time to write data to hard disk. The second table in the database is for chronological data, i.e. information about time-slices execution (state value, start-time, end-time). The remaining two tables are used for

spike acquisition and contain the list of units recorded on selected spike channels and the spike time-stamps respectively. The advantage of storing spike data and chronological data in a database is not only for off-line analyses but it also allows on-line analyses during the experiment. Reading the data base for different criteria like different conditions of a task, different behavioral events of a condition or different counter-number (=> different neurons) allows the incoming data to be separated for different response characteristics of different neurons and to adapt the query procedure as a result, thereby allowing us to resynchronize histogram and raster-plots to new events during the recording.

**Figure 4 about here**



## **Results**

We tested the time accuracy of the supervisor system during synchronous recording, forcing the system to react as fast as possible to changes in digital input values. The recording of changes of digital input is the most critical one because it depends on the reaction time of the control algorithm. We used a pulse generator (BAK Electronics Inc., Germantown (MD), U.S.A.) to generate square pulse waves with equal duration of on- and off-phase (duty-cycle 50%). The waves were recorded as a continuous signal by the analog device, the rising edges of each wave as time-stamps by the counter device (simulating spike data), and each change to high or low value by the digital device (simulating behavioral data). We constructed a condition with only two time-slices in which the first time-slice waited for the change to high value of the digital input line and the second one waited for the opposite change. So each change from off- to on-phase of the pulse and vice versa triggered a decision by the supervisor system with advancement in the time-slice series and entry of a new record in the chronological data table of the database.

We calculated the accuracy of synchronous detection between data from the counter device (simulated spikes) and the entries for the first time-slice of each condition in the table of chronological data (behavioral events) both triggered by the same event (the rising edge of a square wave). This means that we can estimate the accuracy with which behavioral data can be correlated with spike data. The results (Fig. 5) show that the system reacted with an error of synchronous detection less than 1ms when the duration of a time-slice was 40ms or longer. Forcing the system to react faster increased the error. The maximum error of synchronous detection of all measurements was not longer than 4ms. Forcing the system to react faster than any 4ms the supervisor system broke down. This dead time of the supervisor system was due to the fact that the inner while-loop of the control loop contains a sequence structure with 4 elements (see Appendix). Executing all 4 sequence frames, 4ms time was needed. During the execution of longer time-slices ( $\geq 40$ ms), when the supervisor system needed to take a decision, the supervisor system was executing the third

sequence frame of the control loop containing the time-slice control algorithm (see fig. A4 in the Appendix). The loop contained in this sequence frame was performed with a precision of 1ms. Taking into account the fact that the fastest behavior a monkey can produce is an expressed saccade (latency > 50ms), we can be sure that any relevant behavior of an animal can be correlated with spike data with accuracy below 1ms.

We also estimated the error of synchronous detection between the analog device (eye position or EMG) vs. digital device and between analog device vs. counter device. The error was 0.73ms +/- 0.71 and 1.16ms +/- 0.58, respectively. Taking into account that the sample frequency of analog data was 500Hz, the supervisor system recorded data with a sufficient precision to correlate analog data with neural or behavioral data.

### **Figure 5 about here**

Figure 6 shows the continuous recording of neural activity, eye position, and behavioral data for more than 15 minutes in which the animal had to perform a delayed-reach task similar to that described in figure 1. The neurons discharged with varying frequencies during the whole recording. The neuron shown in the second line of the raster plot discharged throughout the recording, the one in the first line discharged less often, and the neuron in the third line discharged rarely. The animal performed the task during the first three minutes and paused for the following 7 minutes, with two exceptions. The supervisor system continued to record data, and during the pauses it waited for the moment when the animal decided to restart the task. Continuous recording was possible thanks to the qualities of the algorithm. The first time-slice of the condition was a Reach time-slice set to wait for the animal to press the start button with a duration up to  $T_{\max} = 5\text{s}$ . If after this duration the animal has not pressed the start button, the time-slice finished with an error and the supervisor system proceeded with the error-handling slice. The error-handling slice waited for no button to be pressed, which was immediately fulfilled, because the animal did not pressed any button. Therefore,

the supervisor system restarted the condition with its first time-slice waiting for the animal to press the start button for another 5 seconds, and so on.

It is worth noting that the recording of eye-position signals and time-stamps of detected spikes are independent from the time needed by the supervisor system to make a decision. The continuous recording allows us to correlate neural response with behavior during the performance of a task but also to correlate neural response recorded during the inter-trial phase, independently from its duration.

**Figure 6 about here**

## **Discussion**

This paper describes the design and implementation of a new algorithm to control experiments with awake, behaving animals. The algorithm was implemented by LabVIEW on an embedded PC fitted with a real-time operating system (RTOS). This implementation represents a good comparison between high performance and ease of use and it allows neural data, eye movement, electromyography, and behavioral data to be recorded synchronously and continuously with an alignment error of less than 1ms. The data recording is not done in a trial oriented manner and it is independent of the animal's behavior. Data and tasks are stored in relational databases. The algorithm does not need an inter-trial interval for data storage or task management. The use of relational databases allows online-analysis during continuous recording. The algorithm could be used to drive psychophysical experiments in which human subjects have to react to stimuli, performing eye movements, hand movements, etc.

The basic information structure of the algorithm is a time-interval of defined maximum duration, called a time-slice. The status of a time-slice is based on trinary logic which takes into account three possible behavioral states: non-fulfillment of task-required behavior, correct performance of task, or erroneous task performance. The proposed data structure to implement the algorithm allows branching of the condition (if-else cascade) as well as looping of parts of a condition (while-control). Therefore, the proposed data structure itself contains the most basic control structures of programming languages. This advantage simplifies the construction of complex tasks, because the construction of cascades or loops is done by the concatenation of time-slices. In addition it should be noted that the construction of these control structures does not need knowledge of specific program languages.

The algorithm described is not the first to be published in the literature. Poindessault et al. (1995) gave first complete program for stimulation and data acquisition using LabVIEW. This

program was designed to perform stimulus control and data acquisition and off-line analysis of visually driven neurons in frog. The main difference of between our program and that of Poindessault can be summarized in two points: 1) their program is constructed for experiments with animals which do not have to produce any kind of behavior, 2) using the classical (non real-time) LabVIEW environment (LabVIEW 2 on a Macintosh II computer), the program was not suited to real-time needs. Any node of the program was executed about 1/60s in round robin fashion. Therefore, real-time application had needed to be design around the hardware (Poindessault et al., 1995).

Recently, Kullmann et al. (2004) described a new control and recording program using LabVIEW-RT. The program called “G-clamp” is an implementation of the dynamic clamp method for whole cell voltage clamp studies. The authors divided their program into two parts, one to be executed in real-time on a PXI-PC and a second part, executed on a Windows-PC, used as a User-Interface. This program is designed for a fixed duration of trial/sweep, and therefore it is suitable for experimental studies different from those recording behaving subjects. It stimulates a cell and records data for a defined time, without the need to react on an intended behavior. Another difference lies in the possibility to extent the program. Kullmann et al. have foreseen the implementation of new virtual ion-channels as an addition to the software in the real-time cycle. The experimental task (application of pulses of defined duration) remains unchanged. In contrast, our program is designed to construct as many tasks as possible with the hardware environment.

Another program to control neurophysiological experiments with awake, behaving animals is CORTEX, developed by Robert Desimone in the early 1980’s and re-elaborated by many of his colleagues in the following years. We will refer here to the VCortex Ver. 2.0 User-Manual (Desimone et al., 2004). This program is developed in the C-language for a Windows-PC. CORTEX is mainly a program designed to study neural responses to visual stimulation. It is a trial

oriented supervisor system in which a defined experimental condition is tested during one trial. The basic concept in organizing the experiment is called condition and it works similarly to our concept of condition. In CORTEX a condition is divided into screens of images (TEST0 to TEST15). A TEST# contains a defined set of visual stimulation during a defined period. This is a concept similar to the output part of our time-slice notion. A difference between the two algorithms is the fact that a CORTEX condition cannot contain more than 16 TEST items.

The trial oriented design of CORTEX has some disadvantages: 1) it needs at least a gap of 500ms between each trial to store and analyze data; 2) the on-line analysis takes into account only the last stored trial, so during recordings, histograms cannot be resynchronized to a new event; 3) the maximum number of data for each type of data is set to 15000. Although this number is large, it constrains the user to create conditions which are not too long. For example, recording eye positions (2 channels) at a sample frequency of 1kHz limits the maximum trial duration to 7.5s. The design of our program allows continuous recording without an inter-trial gap (see fig. 6) and we can theoretically store an infinite amount of data. In addition, the storage of behavioral as well as spike data in a relational database allows us to resynchronize histograms and raster-plots to different events during the recording. CORTEX contains a large set of pre-defined functions for visual stimulation. As our program was written with the intention to produce visual as well as other kinds of stimulation by hardware outside the supervisor system, the supervisor system controls only the start and end of different stimulations and the responses evoked by them. CORTEX contains also the possibility to branch the execution or create loops of different parts, but this can be done only of the level of the CSS-program language and not in the data structure. If one wants to change parts of a task, it is necessary to rewrite the program.

In conclusion, our program does not contain any limitation in the length of a condition. It synchronously records the signals of 4 analog channels (i.e. eye-position signals, electromyography etc.), the time-stamps of several single neurons simultaneously (by means of single- and multi-

electrode recording and multi-spike detection), sets digital output channels, and reacts immediately on changes of selected digital or analog channels. In our present system, we reserved 6 counters for acquisition of time-stamps of detected spikes. This number can be scaled by installing additional Timing IO-devices. The time-slice based construction of experimental conditions allows an easy construction of different experimental protocols.

## **Appendix**

The following is a description of some emblematic procedures to explain the implementation of the time-slice algorithm addressing questions linked to the LabVIEW G-language and its main structures like the while-loop, case-structure, sequence-structure, event-case structure, etc. Three types of synchronization structures of the LabVIEW environment play a key role in our program: queue, notifier and occurrence. For a detailed explanation of these structures see <http://www.ni.com/labview>. A Queue is a buffer in which data are stored and the first data stored in the queue are the first data read from the queue (FIFO-method). The amount of data which can be stored on a queue is virtually infinite and depends on the size of RAM on the PXI-PC or on the host-PC. A notifier is a single element buffer. When a new value is written to a notifier the old one is overwritten, reading a value from a notifier does not cancel the value in a notifier. In LabVIEW, queues and notifiers can be of any data type including complex structures (like cluster) and polymorphic ones. An occurrence is a synchronization structure equivalent to a classical Boolean flag. It serves to start or stop synchronously different, separated parts of the program without forcing LabVIEW to poll on a global variable.

### The host-PC program

The main program running on the host-PC contains two while-loops working in parallel as main structures. The first one, containing an event-structure, controls all input elements of the front-panel (fig. A1-A) and sends the commands to the PXI-PC. The while-loop interfaces the user with the program running on PXI-PC. One example of an event-case is shown in figure A1-B. This event-case is executed when the “Load Task” button of the front panel is pressed. It allows the user to select one of the tasks. When the event-case is executed it reads for the specified task the sequence of time-slices, converts them to a string and sends it with an identification letter ahead (here “t”) through a TCP-connection to the PXI-PC. The PXI-PC confirms the reception of the



time-slice sequence sending back the identification letter. The event-case structure is set to have no timeout. In this way it reacts immediately to any detected event on the front panel.

The second main structure of the host program, again a loop structure, contains a set of parallel working while-loops to read the transmitted data from the PXI-PC and store them in a file or database on the host-PC. An example is shown in figure A2 containing the two loops to read spike data and store them in the database. The first loop (fig. A2-A) contains three read procedures. The first one reads from a TCP-line the number of data to be transmitted, the second one reads the counter-number (that corresponds to the spike channel) to which the data belong, and the third one reads the data and converts them into an array. All three pieces of information are stored in a cluster and saved on a queue for spike data. The queue is needed as a buffer to mediate between the timing of data reception and the timing to write data to hard disk.

The second loop (fig. A2-A) reads the data from its queue (here that of spike data) and saves them in the database using the ODBC-Toolkit of LabVIEW. The other two kinds of data (analog and chronological data) are received and stored in the same manner, with one difference: as explained in the implementation section the analog data are stored in a binary file not in the database and the database contains just the reference to the binary file.

**Figure A1 about here**

**Figure A2 about here**

The PXI-PC program

The main program running on the PXI-PC is based on a while-loop for the communication and six subVIs running in parallel in real-time (fig. A3). There are a couple of subVI's for acquisition and transmission of analog data (analog acq, tx analog data) and spike data (spike acq, tx spike data). The other two subVI's are the control sub-VI controlling the experiment (real-time ctrl) and a subVI transmitting the chronological data in relation to the advancement of the time-slice

sequence (tx chron data). The task control subVI has time-critical real-time priority, and is set to work with a time-accuracy of 1ms. The other subVI have lower real-time priorities (see table A1).

The procedure of the main program is divided into sequence-frames. In the first frame, the counter, which triggers the analog acquisition, starts then pauses and waits for its activation during the recording sessions. This “warm-up round” of the counter ensures a synchronous start at the beginning of each recording. The next frame contains the creation of the data queues (“analog”, “time\_slices”, “spikes”), the control queue “exit”, the notifier “eyes”, and the start occurrence. Their references are passed to the acquisition and transmission subVI’s in the third frame (shown in fig. A3). This frame also contains the main while-loop, which receives the task from the host-PC, starts the control and acquisition routines, and stops them. The start signal for each recording is given by means of the start-occurrence and the stop signal is given by the queue “exit”. The queue “exit” contains Boolean value. The presence of a value on the queue indicates to the other subVIs to stop the acquisition or the transmission process. Depending on the value on the queue, the subVIs waits for a new start signal (false-value) or terminates definitively (true-value).

**Table A1 about here**

**Figure A3 about here**

The control loop

The control loop (fig A4) is the core of the time-slice algorithm implementation. In fact it handles one time-slice at a time e.g. it reads the actual state of the input channels and compares them to the intended values until it finds a result-value corresponding either to a correct end or to an erroneous end of the time-slice. Then, based on this value it starts the following time-slice.

The control loop has the following construction going from outside to inside (fig A4): an outer while-loop, a case structure (not shown), an intermediate while-loop, and a sequence structure containing in frame #2 the inner while-loop. The outer while loop cycles around all tasks applied during a recording session. The intermediate while loop cycles around all time-slices of a task and

the inner while-loop cycles around all the evaluations of input channel values during a time-slice. The main VI passes to this subVI the references of the start occurrence, the queues “task”, “time\_slices”, “exit”, the notifier “eyes”, and the reference of the static TCP-line. When started, the subVI first configures the input and output boards of the PXI-DIO96 device and one clock of the TIO-device, which serves for time-measurement of the time-slices, and waits at the start occurrence (for the start procedure see the description for all other subVIs later on). When the start-signal arrives, the subVI sends the synchronization sequence to the TIO-device to start counters and clock, takes one element from the “task”-queue (an array of time-slices elements) and enters the intermediate while-loop. The intermediate while-loop contains a sequence structure with the following sequence-frames:

#0: reads the actual value of the clock at the start of the time-slice

#1: reads the actual time-slice element of the task-array, calculates the time when the time-slice finishes, and writes the digital output values to the PXI-DIO96 device.

#2: contains the innermost while-loop timing with 1ms (minimum loop timing possible in LabVIEW 7.0), reads out the actual state of the digital input-channels, the position of the eyes (passed by the notifier “eyes”), and the actual time, compares the values with the intended values and calculates the result value as described above, stops when the state value is  $> 0$ .

#3: reads the finishing time of the time-slice, writes the number of condition, the time-slice index, the state value of the time-slice, and the states of the digital input-channels in a cluster, which is set on the “time-slice” queue, these values will be transmitted to the host-PC and constitute the chronological data. Finally the index of the next time-slice is calculated in relation to the state value:  $\text{actual-index} + \text{true-index}$ , if state value = 1 or  $\text{actual-index} + \text{false-index}$ , if state value  $> 1$ .

When the program has passed the sequence structure, the intermediate loop tests whether the task has terminated correctly and sends the stop-sequence for all subVIs by the “exit”-queue. The outer loop terminates depending on the value taken from the “exit”-queue.

**Figure A4 about here**

PXI data-acquisition and transmission

The acquisition of data and their transmission is described using the example of analog data acquisition (fig. A5). The acquisition of the counter data is solved in a similar manner. The subVIs are started immediately as the references of the queues, occurrence, and notifier are passed to them. As soon as the acquisition VI is started (fig. A5-A), the MIO-device is configured for the number of channels to acquire (here four channels) and an external scan-clock is assigned. The scan signal is sent on one of the RTSI-bus lines from the dedicated counter. The MIO-device is started and paused immediately, so it is armed to be used. The reference number of the MIO-device and the references of the queues, notifier, and occurrence are passed to a while-loop containing a case structure. The case structure is guided by the start occurrence in the following way. The occurrence is set to have no timeout and the program waits on the occurrence until the start signal arrives. Immediately after the start signal, the program passes into the case structure, starts the MIO-device and begins to read out the buffer assigned to the MIO-device. The buffer read procedure is organized in an inner while loop which is timed to be executed every 4 ms. In each loop at least 2 samples are read from the device buffer and sent on the data queue (“analog”). The values of first two channels, containing the data of eye-position, are stored also in a notifier (“eyes”). The number of data points remaining on the device-buffer is stored in a shift register and used to extend the number of data to read in the next cycle of the inner loop. After reading the device buffer, it checks whether an element on the “exit” queue is present. If not, it begins with a new read operation, otherwise it stops the inner while loop, immediately sets the MIO-device to pause and takes one element from the queue “exit”. In

relation to the value from the queue, as explained previously, the program finishes the outer while loop and terminates the subVI or goes back to wait a new start signal on the start-occurrence.

The transmission procedure of the data is shown in figure A5-B. The algorithm, similar to the acquisition algorithm, is based on a construction of an outer while-loop and a case structure containing the inner while-loop. The reference of the start-occurrence, of the “analog”- and “exit”-queue are passed to this subVI as well as the reference of a static TCP-line for communication with the host-PC. After the start of outer while-loop and the reception of start signal on the start occurrence, the program passes to the case-structure. It first opens a dedicated TCP-line for transmission of the analog data, and then passes to the inner while-loop, where it takes first one element from the queue, converts it to a string and sends the string to the host-PC through the dedicated TCP-line. The inner loop is stopped when an element is found on the “exit”-queue and no more data elements are stored on the data queue. If there are data still stored on the queue, the inner loop continues to send the data. This makes sure that all acquired data are sent to the host-PC. After finishing the inner while-loop, the subVI sends a character to the host (here “a”) on the static TCP-line to inform that the transmission of analog data is terminated. The inner while loop is timed to be executed every 4 ms. We decided to use this short interval to ensure a high transmission rate for analog data, because of their large amount. The outer while-loop is finished in relation to the value taken from the “exit”-queue.

**Figure A5 about here**

**Acknowledgements**

We thank PH Kullmann for useful feedback on the manuscript. This work was supported by grants from Ministero dell'Istruzione, dell' Università e della Ricerca (MIUR), Italy.

## **References**

1. **Budai D, Kehl LJ, Poliac GI and Wilcox GL.** An iconographic program for computer-controlled whole-cell voltage clamp experiments. *J Neurosci Methods* 48: 65-74., 1993.
2. **Budai D.** A computer-controlled system for post-stimulus time histogram and wind-up studies. *J Neurosci Methods* 51: 205-211., 1994.
3. **Desimone R, White TM, Norden-Krichmar TM, Olsen J and Boulden E.** CORTEX - A program for data acquisition and experimental control (5 ed.): <http://www.cortex.salk.edu/>, 2004.
4. **Fattori P, Gamberini M, Kutz DF and Galletti C.** 'Arm-reaching' neurons in the parietal area V6A of the macaque monkey. *Eur J Neurosci* 13: 2309-2313, 2001.
5. **Kling-Petersen T and Svensson K.** A simple computer-based method for performing and analyzing intracranial self-stimulation experiments in rats. *J Neurosci Methods* 47: 215-225., 1993.
6. **Kodosky J and Dye R.** Programming with pictures. *Computer Language Magazine* 1: 61-69, 1989.
7. **Kullmann PH, Wheeler DW, Beacom J and Horn JP.** Implementation of a fast 16-Bit dynamic clamp using LabVIEW-RT. *J Neurophysiol* 91: 542-554, 2004.
8. **Marzocchi N, Cavalcanti S, Galletti C, Fattori P and Kutz DF.** A new real-time supervisor system for data acquisition analysis and control of visuomotor task experiments. *FENS Abstr.* 2: A052.013, 2004.
9. **Nordstrom MA, Mapletoft EA and Miles TS.** Spike-train acquisition, analysis and real-time experimental control using a graphical programming language (LabView). *J Neurosci Methods* 62: 93-102, 1995.
10. **Poindessault JP, Beauquin C and Gaillard F.** Stimulation, data acquisition, spikes detection and time/rate analysis with a graphical programming system: an application to vision studies. *J Neurosci Methods* 59: 225, 1995.
11. **Puppe B, Schon PC and Wendland K.** Monitoring of piglets' open field activity and choice behaviour during the replay of maternal vocalization: a comparison between Observer and PID technique. *Lab Anim* 33: 215-220, 1999.

## **Figure legends**

Figure 1: Schematic representation of the task used in (Fattori et al., 2001).

A) Schematic drawing of the experimental setup. B) The upper part shows the states of input controls to be checked during the task. The input-controls Start Button and Touch Button are digital controls, which are checked during the total duration of the task. The input control Eye is a special analog control of two channels (x, y) controlling eye position, which is checked only during a specific time interval ( $ts_3 - ts_8$ ). The lower part shows output channels (Reward, LED) and their states (on/off and red/green/off, respectively) during the task. Vertical dashed lines define the start and end of a time-slice. The time-slices are numbered below.

Figure 2: A) Schematic drawing of the construction of a condition: each time-slice contains in its data structure the information of the output-channels, input-controls (not shown) and a true- and false-index. The supervisor system executes a time-slice while the time-slice status is =0. If the time-slice status becomes =1 the supervisor system advances in the sequence to the next time-slice indicated by the true-index (T). If the time-slice status becomes >1 the supervisor system will execute the time-slice indicated by the false-index (F). The true-indexes are pointing to the next working-slice ( $ws_1, ws_2, ws_n$ ), the false-indexes are pointing to the error-handling slice ( $eh_1$ ). Note that the false-index of the error-handling slice is pointing back to itself.

B) Schematic drawing of the branching of a condition: the condition branches after the second working-slice ( $ws_2$ ). Its true-index points to the sequence  $ws_3, ws_4$  and its false-index points to the sequence  $ws_5, ws_6$ . Both branches converge to the last working slice.

Figure 3: Schematic drawing representing the hardware and software structure of the implemented real-time supervisor system.



Figure 4: Hierarchy and relations of tables in the relational databases

A) The tables Task Descriptor and Task Structure describes a Task, the tables Condition Descriptor and Condition Structure describe each Condition of a Task, the table Time-Slice Descriptor describes each time-slice of a Condition.

B) Acquired data are stored in four tables. Table Recordings associate the general data of each recording with the other tables. Table chronological data contains information on time-slices execution. The remaining two tables are used for acquisition of spike time-stamps of different spikes detected by means of single- and multi-spike detection.

Figure 5: Results of the performance test

The graph shows the difference of time-stamps in ms recorded with the counter device and as chronological data by the supervisor program. The abscissa gives the duration of the time-slices and the ordinate the differences. Each point represents the median and S.D. of the test.

Figure 6: Continuous recording

The graph shows the continuous recording of three neurons, eye-position signals and behavioral events for more than 15min of recording time. Sixteen rows comprise the graph, each row demonstrates the data for up to 1min and the running minute is indicated on the left side. The first three lines show the raster-plot of the simultaneously recorded neurons, the following two lines show the horizontal and vertical eye-position, respectively. In the last line, a thick line indicates the intervals in which the animal performed the task. Thin vertical lines show start and end of trials.

**Figure legends of figures in the appendix**

Figure A1: A) Front panel of the Host-PC program with different parameters to be set by the user.

B) “Load Task” event structure to transmit the task values to the PXI-PC. For details see text.

Figure A2: Program on Host-PC to receive and store spike data

A) While-loop to receive data from the PXI-PC.

B) While-loop to store the received data in the database. For details see text.

Figure A3: LabVIEW diagram of the PXI-PC main program

Only the second and third frames of the program are shown.

Figure A4: LabVIEW diagram of the supervisor program on PXI-PC

The supervisor program with the third frame of the sequence structure containing the innermost while-loop.

Figure A5: Analog data recording and sending programs on PXI-PC

A) While loop to record analog data.

B) While loop to transmit recorded data to the Host-PC. For details see text.

## Tables

Table 1 Time-slice state coding.

The state is coded for input signals ( $S_{in}$ ) and for time ( $S_T$ ).  $A_{in}$  are the analog inputs,  $A_{ref}$  are the reference values of the inputs,  $d_{max}$  is the maximum difference acceptable.  $D_{in}$  are the current values of digital inputs,  $D_{mask}$  sets on the digital inputs to be controlled and  $D_{set}$  settles the awaited value for the settled digital inputs.  $T$  is the elapsed time.  $T_{max}$  is the acknowledged maximum length of time-slice. Time-slice state is computed as  $S_{in} + S_T$ .

T, F = True, False. For description of behaviors Remain, Reach, End, and Avoid see text.

		Intended Behavior							
		Remain		Reach		End		Avoid	
		T	F	T	F	T	F	T	F
$S_{in}$	$ A_{in} - A_{ref}  < d_{max}$ $D_{in} \& D_{mask} == D_{set}$	0	2	1	0	0	1	2	0
$S_T$	$T > T_{max}$	1	0	2	0	2	0	1	0

Table 2: The behavior of the control algorithm  
 For definition of  $S_{in}$ ,  $S_T$ ,  $T$ , and  $T_{max}$  see table 1.

Time-slice state $(S_{in} + S_T)$	Motive / Cause	Algorithm outcome
0	no change in any selected input channel has occurred and $T$ has not exceeded $T_{max}$	go ahead with the same time-slice
1	an intended change in an input channel or time-slice finishes without any change of the selected input channels as intended (correct end of the time-slice)	go ahead with the next time slice of the experiment
>1	an unintended change in selected input channel or time-slice finishes without any intended change of a selected input channel (erroneous end of a time-slice)	go ahead with a time-slice different from the foreseen one

Table 3 Hardware specifications

The specifications regard our implementation. Other boards can be also used depending on the experimental requirements. Instead of the LabVIEW Professional Version also the Full Version can be used. LabVIEW and the PXI components are available from National Instruments Corp. (Austin, Texas, U.S.A.)

<b>Component</b>	<b>Part</b>	<b>Main Characteristics</b>
Host computer	2.80 GHz Pentium IV with 1GB RAM, 40 GB HD and 10/100 Ethernet card	The Ethernet connection between host and embedded controller can run through a LAN or directly between the two machines.
	Windows 2000	
	LabVIEW 7.0 Express Professional Development System	
	LabVIEW Real-Time Module	
	LabVIEW Database Connectivity Toolset	
	Microsoft Access	
Embedded Controller	PXI-8176 Real-Time Embedded Controller	1.20 GHz Pentium III with 128 MB RAM, 15 GB HD, integrated 10/100 BaseTX Ethernet with real-time operating system and LabVIEW Real-Time Engine installed.
	PXI-1000B	General Purpose PXI Chassis (8 Slot, DC Powered)
	PXI-6070E	Multifunction I/O Card with 16 Analog Inputs (1.25 MS/s, 12-bit), 2 analog Outputs, 8 digital I/O lines, two 24-bit counters, analog triggering
	PXI-6602	8-channel, 32-bit up/down counter/timer module
	PXI-6508	Digital I/O Card with 96 static digital I/O lines in 8-bit ports

**Tables in the appendix**

Table A1 Priorities of the real-time subVIs

<b>Name</b>	<b>Priority</b>
Task_ctrl.vi	time critical priority (highest)
Analog_acq.vi	normal priority
Spikes_acq.vi	normal priority
Tx_chronological_data.vi	normal priority
Tx_analog_data.vi	normal priority
Tx_spikes_data.vi	normal priority

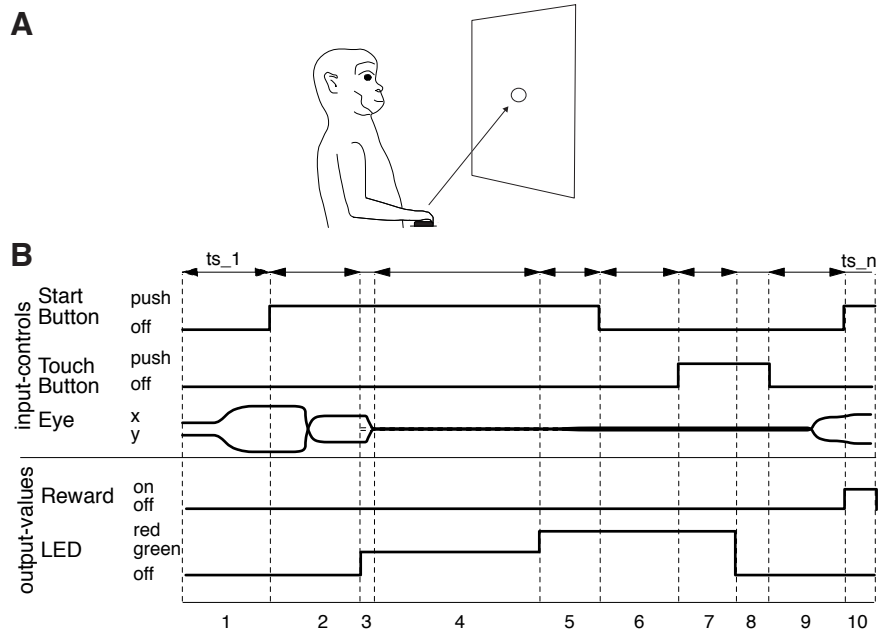


figure 1

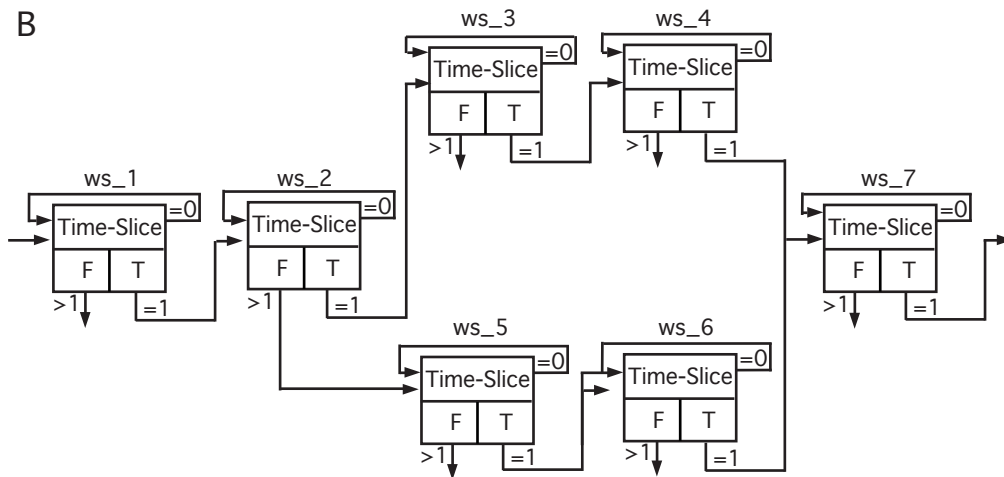
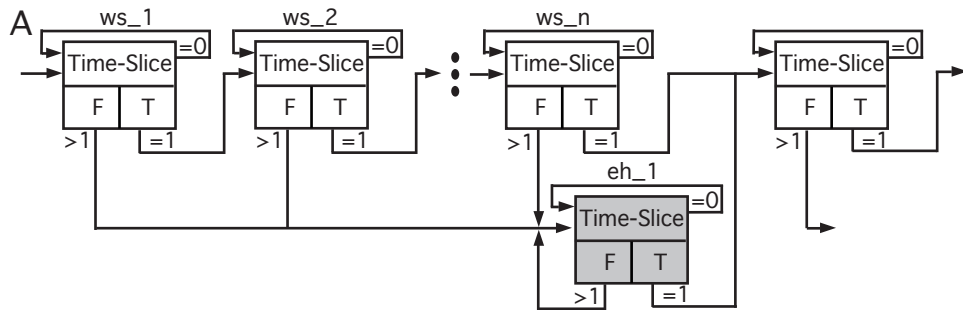


figure 2



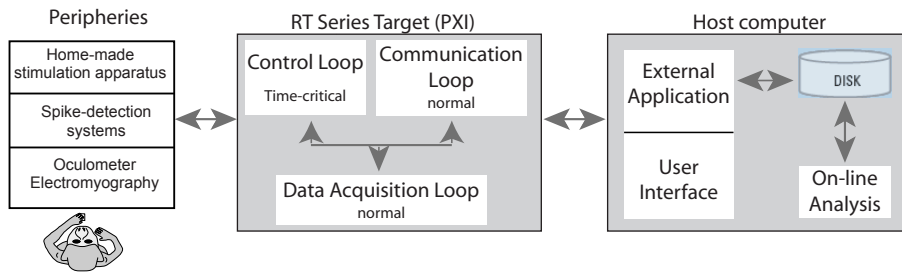


figure 3

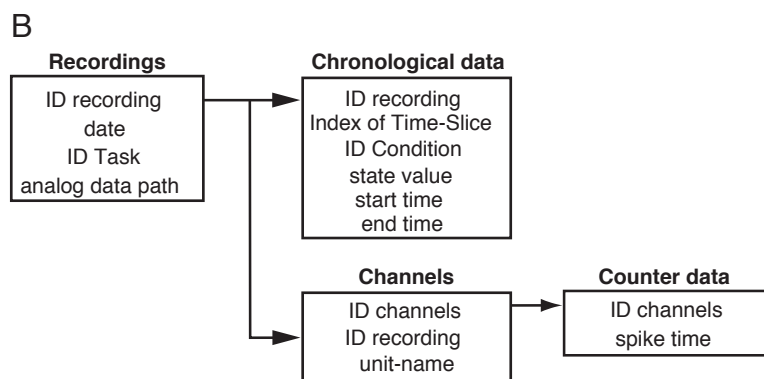
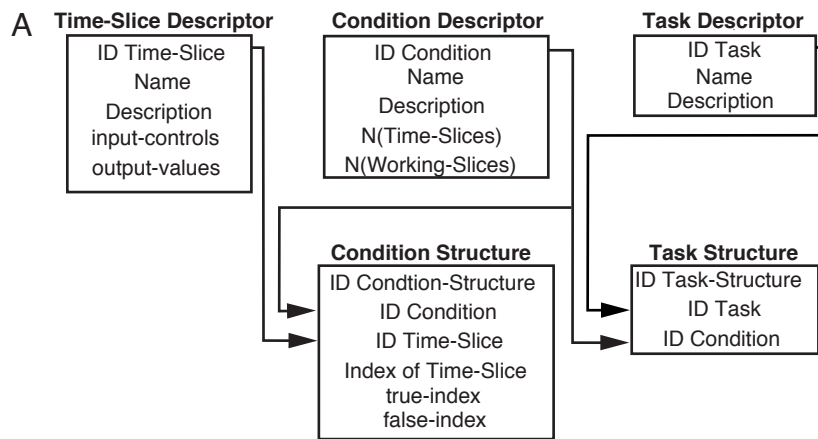


figure 4

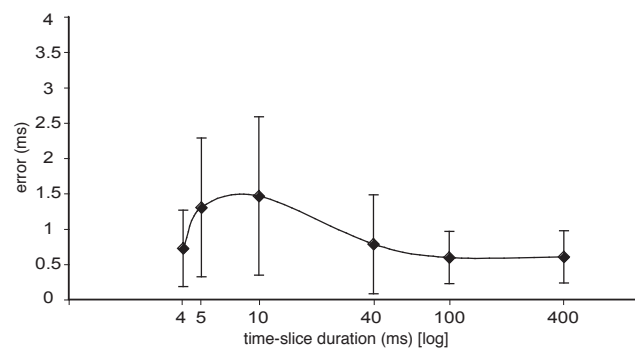


figure 5

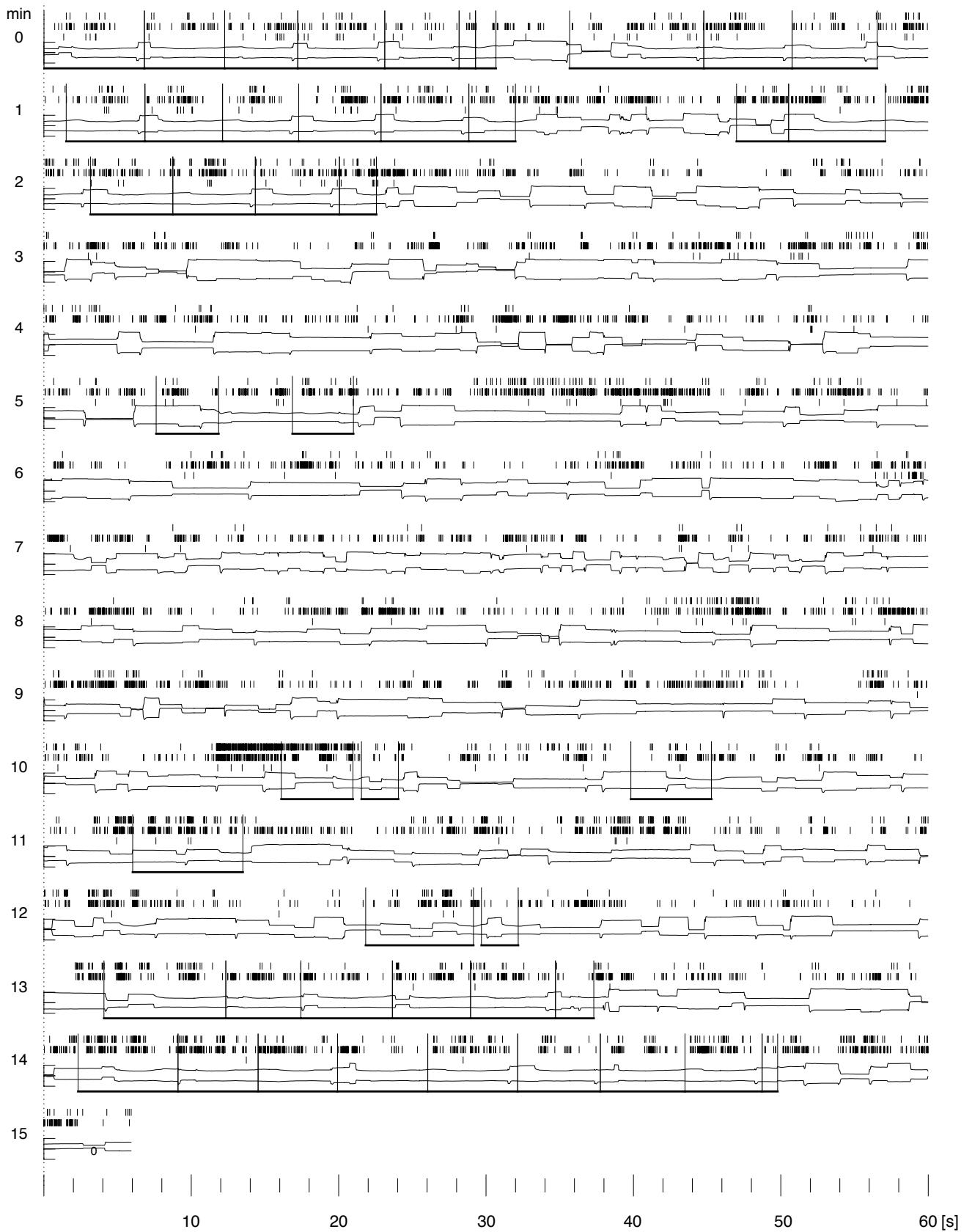
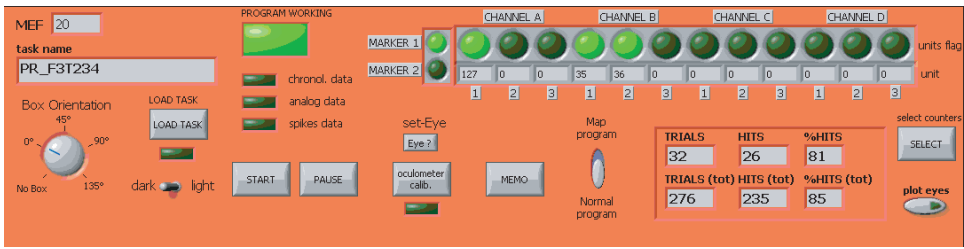


figure 6

A



B

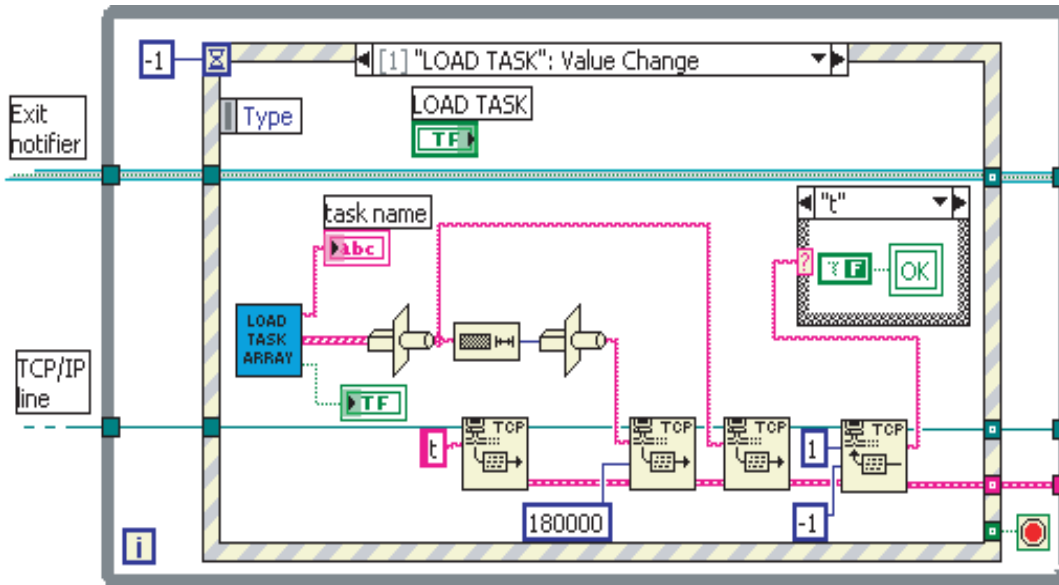
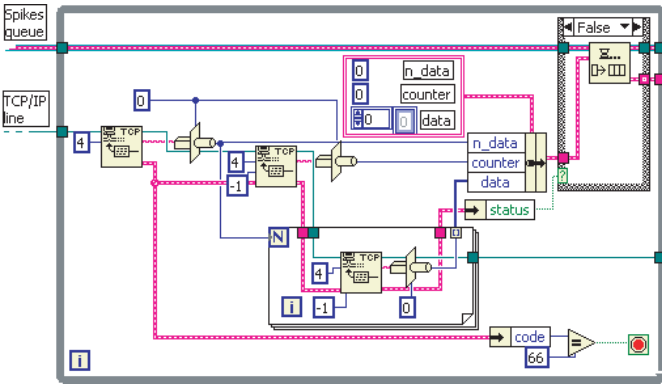


figure A1

**A**



**B**

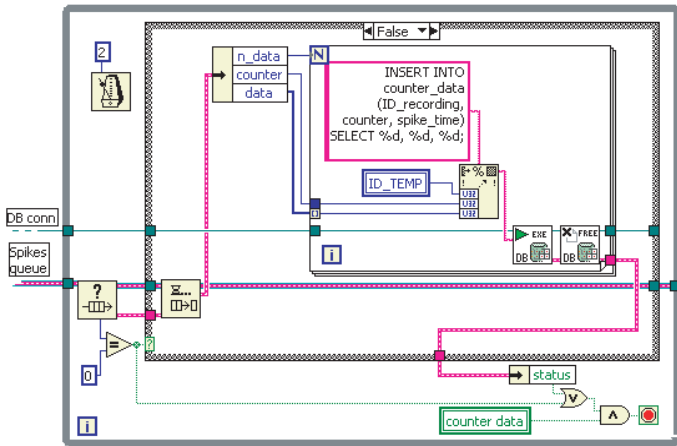


figure A2

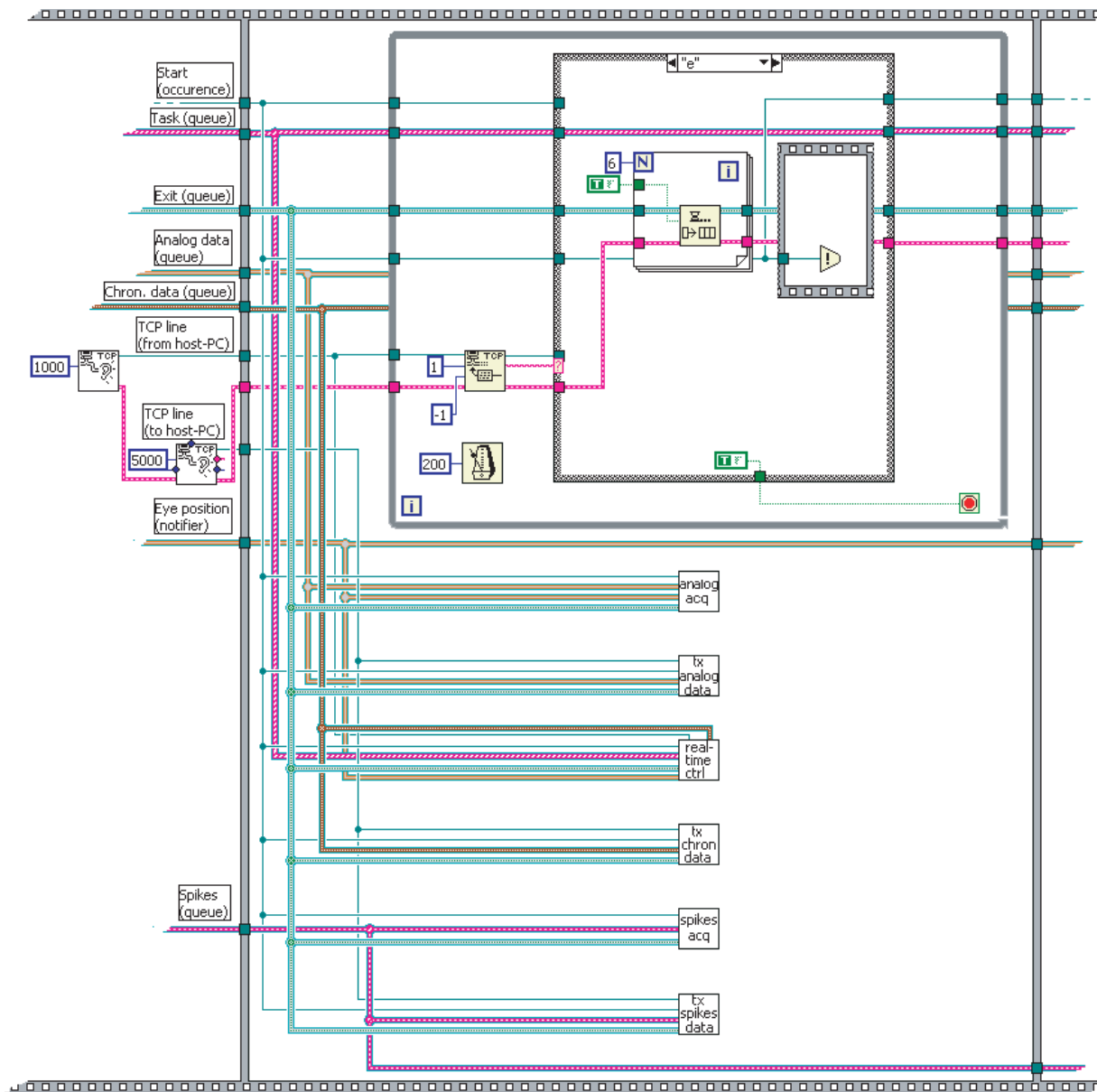


figure A3

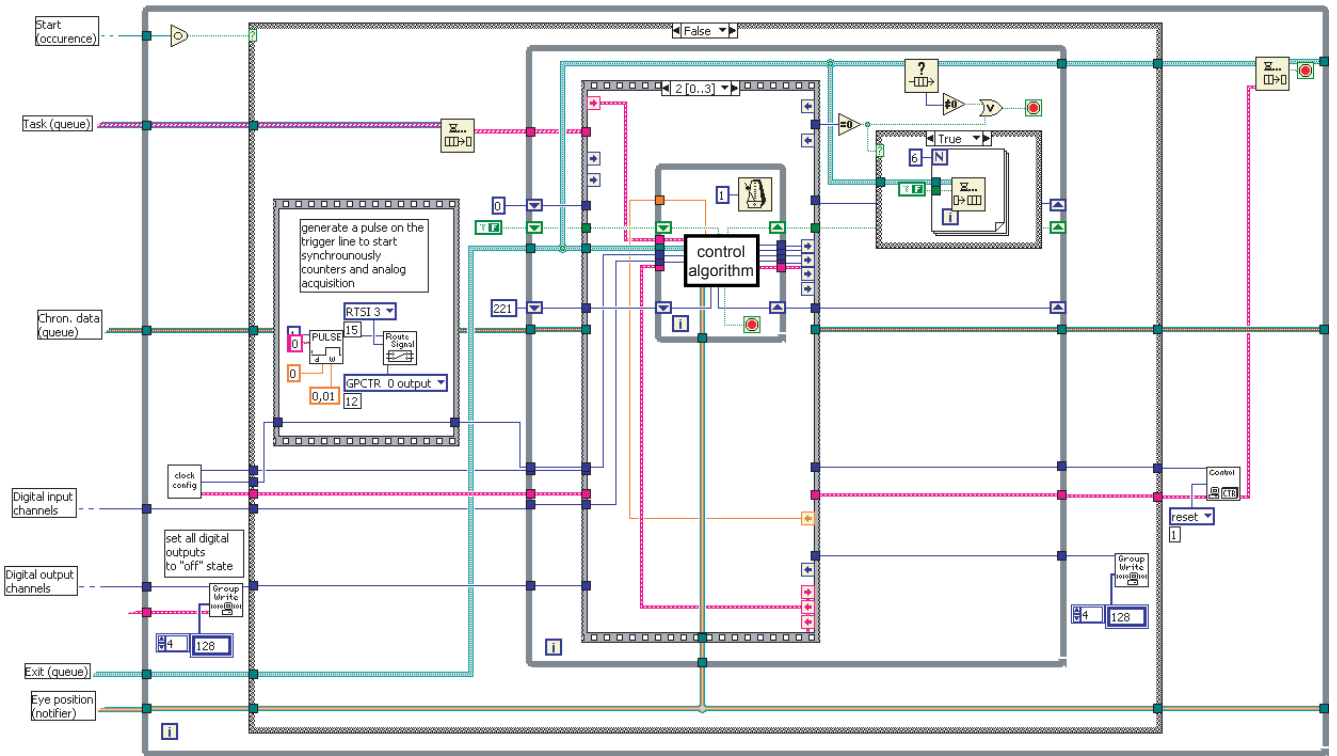


figure A4



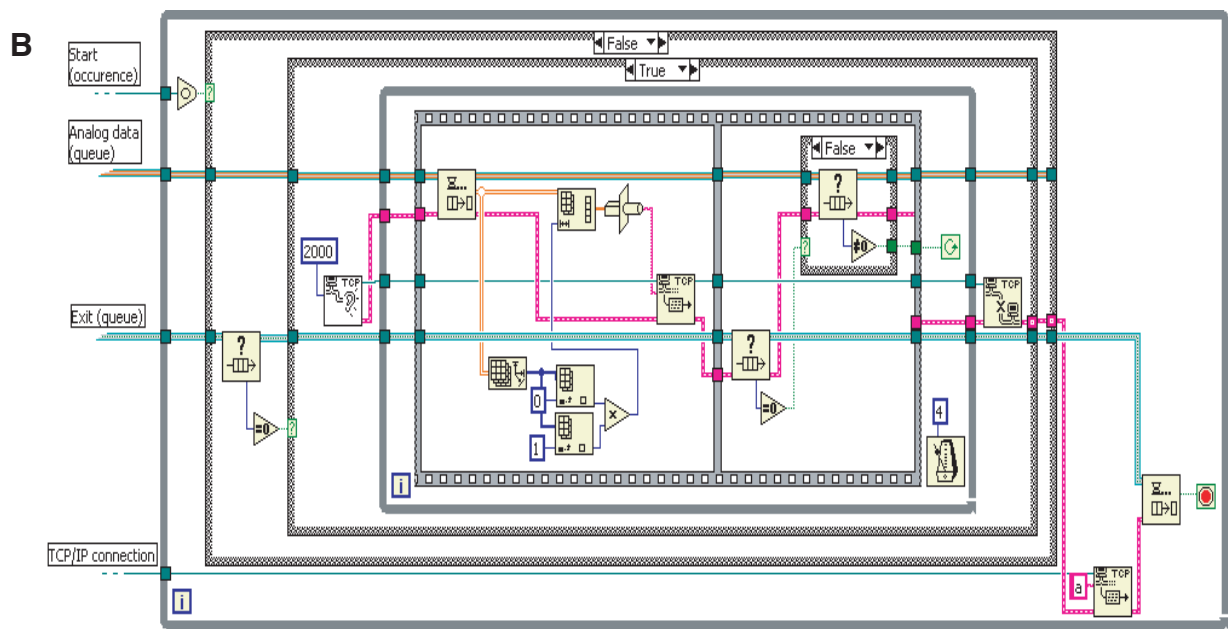
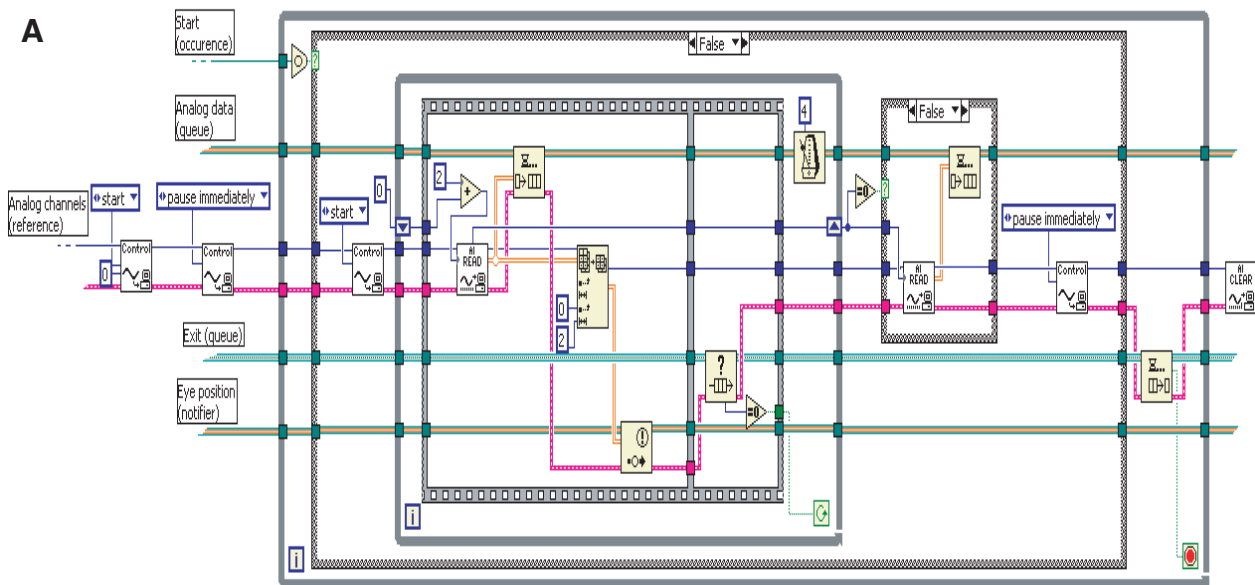


figure A5