**http://sunburst.usd.edu/~schieber/psyc770/IO32.html**

**Mex-File Plug-in for Fast MATLAB Port I/O Access  
(32-bit Windows 2000, XP, Vista, 7)**

Note: The software described below currently supports 32-bit versions of Windows (2000,/XP/Vista/7).    
A version of this software for 64-bit Windows can be found [here](http://people.usd.edu/%7Eschieber/psyc770/IO64.html).  
A version for running 32-bit MATLAB (e.g., Student version) on 64-bit Windows can be found [here](http://people.usd.edu/%7Eschieber/psyc770/IO32on64.html).

Windows Vista and 7 users should note the **Vista Installation Notes** near the end of this document.

The *inportb()* and *outportb()* functions for accessing hardware I/O ports provided by Cogent 2000 are highly limited in their utility.  To begin with they are very slow - demonstrating latencies of up to 100 msec.  In addition, they can only access hardware ports located in the very lowest reaches of the 64K I/O port address space (i.e., 0x000-0x3FF).  Windows' Plug 'n Play manager usually maps add-in PCI cards to regions much higher in the address space.  As a result, these PCI cards can not be accessed using the built-in Cogent 2000 commands.

In order to accomplish very fast port I/O using a NO COST add-on to MATLAB, we have developed a C++ extension (mex-file) that uses native methods to access low-level hardware.  This mex-file is named **io32.dll**.  It uses a freeware self-installing system driver named **inpout32.dll**. [Note: Self-installation of the driver requires that the MATLAB is run with Administrator privileges.  The driver must have been previously installed in order to support non-Administrator users].

Once these two modules are installed in your MATLAB path, you can use **io32()** to read and write to I/O port locations anywhere in the 64K address space.  A simple benchmark test ([iotimer\_io32b.m](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/iotimer_io32b.m) or [iotimer\_io32.m](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/iotimer_io32.m)) reveals that port I/O latencies of approximately 0.010 msec (i.e., 10 microseconds) can be achieved from within MATLAB using this approach.

To install this expanded capability: download the [**io32.dll**](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/io32.dll) module and move it to a directory in your MATLAB path (e.g., c:\cog2000\Cogent2000v1.28\Toolbox in the case of the USD PSYC 770 standard Cogent 2000 installation specification).  MATLAB 7.5+ should rename io32.dll to **io32.mexw32** to prevent the warning message every time the io32 module is loaded.  Next, download the [**inpout32.dll**](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/inpout32.dll) module and move it to the C:\windows\system32 directory (or any other directory that is on the computer's system PATH).

**io32() Command Usage Summary:**

|  |  |
| --- | --- |
| object = **io32**; | Calling **io32** with no input arguments from the MATLAB command window creates an instance of the **io32** interface object and returns a pointer to its location.  This command must be issued first since the object pointer is a required input argument for all other calls to **io32**.  This **io32** call will not work properly unless a return variable is specified (i.e., 'object' in the example to the left). |
| status = **io32(** object **)**; | Calling **io32()** using one input argument and a single return variable causes akernel-level I/O driver named *hwinterface.sys* to be automatically installed (i.e., no manual driver installation is required).  object is the pointer to a previously created instance of *io32* (see the step performed above); and, status is a variable returned from the function that describes whether the driver installation process was successful (0 = successful).  Subsequent attempts to perform port I/O using *io32()* will fail if a non-zero status value is returned here.  This step must be performed prior to any subsequent attempts to read or write I/O port data.  Auto-installation of the driver requires ADMINISTRATOR privileges.  Once the kernel-level driver is successfully installed (i.e., the first time the **io32** command is run), subsequent users will not require admin privileges. |
| **io32(** object, address, data **)**; | Calling **io32()** with three input parameters allows the user to output data to the specified I/O port address.  object is the pointer to a previously created **io32** object (described above); address specifies the physical address of the destination I/O port (<64K); and, data represents the value (between 0-255) being output to the I/O port. |
| data = **io32(** object, address **)**; | Calling **io32()** using two input arguments and one return variable allows the user to read the contents of the specified I/O port.  object is the pointer to a previously created instance of **io32** (see above), address specifies the location of the I/O port being read; and, data contains the integer-format value returned after reading the I/O port. |

The following MATLAB command snippet demonstrates how to use the **io32()** extension:

%create an instance of the *io32* object  
**ioObj = io32;**  
%  
%initialize the *hwinterface.sys* kernel-level I/O driver  
**status = io32(ioObj);**  
%  
%if status = 0, you are now ready to write and read to a hardware port  
%let's try sending the value=1 to the parallel printer's output port (LPT1)  
**address = hex2dec('378');** %standard LPT1 output port address(0x378) **data\_out=1;** %sample data value **io32(ioObj,address,data\_out);** %output command  
%  
%now, let's read that value back into MATLAB  
**data\_in=io32(ioObj,address);**  
%when finished with the *io32* object it can be discarded via  
%'clear all', 'clear mex', 'clear io32' or 'clear functions' command.

**MATLAB Scripts to Simplify Port I/O**

The code examples above reveal that using the **io32()** extensions is a bit complex.  In an attempt to reduce this complexity, a set of MATLAB scripts has been developed to simplify I/O programming.

In order to have access to these scripts: download the [io32.dll](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/io32.dll), [config\_io.m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/config_io.m), [inp.m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/inp.m) and [outp.m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/outp.m) files and move them to a directory in your MATLAB path. In addition, download the [**inpout32.dll**](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/inpout32.dll) module and move it to the C:\windows\system32 directory (or, C:\WINNT\system32 for Windows 2000 users) as previously described above.

**MATLAB I/O Script Usage:**

|  |  |
| --- | --- |
| **config\_io;** | Installs the kernel-level driver required to access low-level hardware.  This command must be given prior to any attempts to use the custom **inp()** or **outp()** scripts. |
| **outp( address, byte );** | This function writes the 8-bit value passed in the variable named byte to the I/O port specified by address. |
| **byte = inp( address );** | This function read the I/O port location specified by address and returns the result of that operation. |

A simple benchmark ([iotimer\_inp.m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/iotimer_inp.m)) reveals that I/O using these scripts is somewhat slower than calling the **io32()** object directly (as demonstrated above).  Instead of being able to read a port with a latency of approximately 10 microseconds, using the **inp()** script yields a latency of approximately 40 microseconds. This is fast enough for many experimental psychology applications (such as scanning a button box, etc.).  Use direct calls to **io32()**, as described above, if your application requires the shortest possible I/O latencies.

The following MATLAB code snippet demonstrates how to use the m-file I/O scripts:

% install and/or initialize the *kernel-level* I/O driver  
**config\_io;**  
% optional step: verify that the driver was successfully installed/initialized  
**global cogent;  
if( cogent.io.status ~= 0 )  
   error('inp/outp installation failed');  
end**  
%write a value to the default LPT1 printer output port (at 0x378)  
**address = hex2dec('378');  
byte = 99;  
outp(address,byte);**  
%read back the value written to the printer port above  
**datum=inp(address);**

**Windows Vista and 7 Installation Notes**

Although our lab does not yet have much experience with Windows Vista/7, we were able to successfully install the software described above using the procedure described below (using MATLAB 7.5.0-R2007b and MATLAB 7.7-R2008b):

|  |  |
| --- | --- |
| 1. | Log in as a user with Administrator privileges. |
| 2. | Disable UAC (User Account Control).  An easy way to do this in Windows Vista is to: Start-Run-MSCONFIG. Select the Tools tab, scroll down to the option for "Disable UAC" and select it. Next, press the "Launch" button. You must then RESTART the system for this change to take effect. |
| 3. | Download and copy the [inpout32.dll](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/inpout32.dll) file to the C:\WINDOWS\SYSTEM32 directory. |
| 4. | Download the [io32.dll](http://sunburst.usd.edu/%7Eschieber/psyc770/misc/io32.dll), [config\_io.m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/config_io.m), [inp.m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/inp.m) and [outp,m](http://sunburst.usd.edu/%7Eschieber/psyc770/m-files/outp.m) files to a working directory of your choice. This directory will be added to your MATLAB path in step-6 below. |
| 5. | Start MATLAB in "Run as Administrator" mode (Right-click icon and select "Run as Administrator"). |
| 6. | Add the directory containing the downloaded m-files to your MATLAB path via the File|Set Path|Add with Subfiles... menu command. |
| 7. | Run "config\_io" from the MATLAB command window.  If there's no error message at this point, you've successfully installed the software. |
| 8. | Optional: If you need to re-enable UAC (User Account Control), follow the instructions in step-2 but select "Enable UAC" instead of "Disable UAC". |

**Parsing Individual Bits within an I/O Byte**

When one reads an I/O port one is usually interested in the status of a single bit among the 8-bits returned by a call to **inp(address)**. MATLAB provides a number of functions to deal with data on a 'bitwise' basis.  For example, the following lines of code show how to test the status of a single input line using the **bitget()** function:

% Read current value of an input port at the specified address  
% Note that the value returned by inp(address) is coerced into an 8-bit format using **uint8**  
**response = uint8( inp(address) );**  
% Take some action if the least-significant-bit is currently at logical-0 level  
**if (bitget( response,1) == 0)  
   display('Input is active')  
end**

See also: bitset(), bitand(), bitor(), bitxor() for additional bitwise operators

Additional information about the INPOUT32 (*hwinterface.sys*) driver for Windows can be found [here](http://www.logix4u.net/inpout_theory.htm).  
  
Information about **IOReadWrite**, a Java class that uses the *UserPort.sys* driver to perform port I/O, can be found [here](http://sunburst.usd.edu/%7Eschieber/psyc770/IOReadWrite.html).  
An improved version of **IOReadWrite** based upon the inpout32.dll driver can be found [here](http://sunburst.usd.edu/%7Eschieber/psyc770/IOReadWrite2.html).

**Last revised: 3 May 2010**

[Professor Schieber's Home Page](http://www.usd.edu/%7Eschieber) - [Previous Page](javascript:history.back())