Mex-File Plug-in for Fast MATLAB Port I/O (64-bit Windows XP, Vista, 7,8,10)

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

A version of this software for 32-bit Windows can be found here.
A version of this software for running 32-bit MATLAB on 64-bit Windows can be found here.

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 **io64.mexw64**. It uses a freeware self-installing system driver named **inpoutx64.dll**. [Note: Self-installation of the driver requires that the MATLAB runs with Administrator privileges. The driver must have been previously installed in order to support non-Administrator users].

To install this expanded capability: download the **io64.mexw64** module and move it to a directory in your MATLAB path (e.g., c:\cog2000\Cogent2000v1.29\Toolbox in the case of the USD PSYC 770 standard Cogent 2000 64-bit Windows installation). Next, download the **inpoutx64.dll** module and move it to the C:\windows\system32 directory (i.e., This module must reside in the Windows system PATH).

Special Note: Because the inpoutx64.dll was compiled using Visual Studio, the Microsoft Visual C++ 2005 SP1 Redistributable (x64) Package must be installed on your computer. Use the Control Panel to see if it is already installed. If not, the installer application can be downlaoded from Microsoft at http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=18471

io64() Command Usage Summary:

object = **io64**;	Calling **io64** with no input arguments creates a persistent instance of the **io64** interface object and returns a 64-bit handle to its location. This command must be issued first since the object handle is a required input argument for all other calls to **io64**. This **io64** call will not work properly unless a return variable is specified (i.e., 'object' in the example to the left).
status = **io64(** object **)**;	Calling **io64()** using one input argument and a single return variable causes the *inpoutx64.sys* kernel-level I/O driver to be automatically installed (i.e., no manual driver installation is required). object is the handle to a previously created instance of io64 (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 io64() 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.
io64(object, address, data **)**;	Calling **io64()** with three input parameters allows the user to output data to the specified I/O port address. object is the handle to an **io64** 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 = **io64(** object, address **)**;	Calling **io64()** using two input arguments and one return variable

	allows the user to read the contents of the specified I/O port. object is the handle to a previously created instance of **io64** (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 **io64()** extension:

```
%create an instance of the io64 object
ioObj = io64;
%
% initialize the interface to the inpoutx64 system driver
status = io64(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
data_out=1;                        %sample data value
io64(ioObj,address,data_out);   %output command
%
% now, let's read that value back into MATLAB
data_in=io64(ioObj,address);
%
% when finished with the io64 object it can be discarded via
% 'clear all', 'clear mex', 'clear io64' or 'clear functions' command.
```

MATLAB Scripts to Simplify Port I/O

The code examples above reveal that using the **io64()** 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 io64.mexw64, config_io.m, inp.m and outp.m files and move them to a directory in your MATLAB path. In addition, download the **inpoutx64.dll** module and move it to the C:\windows\system32 directory as previously described above.

MATLAB I/O Script Usage:

config_io;	Installs the *inpoutx64* 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 test reveals that I/O using these scripts is significantly slower than calling the **io64()** 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 most experimental psychology applications (such as scanning a button box, etc.). Use direct calls

to **io64()** if your application requires the shortest possible I/O latencies (e.g., updating an analog output stream).

The following MATLAB code snippet demonstrates how to use the new I/O scripts:

```
% initialize access to the inpoutx64 low-level I/O driver
config_io;
% optional step: verify that the inpoutx64 driver was successfully 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);
```

Reaction Time Benchmark/Results

Since our lab uses low-level digital I/O to control stimuli and/or collect human response times with millisecond accuracy requirements, we developed a **io64()** timing benchmark that mimics a classic "reaction time" protocol. This approach involved building a hardware-based ***Reaction Time Simulator*** that produces an output "response" precisely 200 msec after receiving a "stimulus" input from the system under test (see **RT Simulator** for additional details and a hardware schematic diagram).

The temporal latency of the MATLAB **io64()** module was assessed by triggering the black box **RT Simulator** via the PC's Line Printer (LPT) Data Register bit-0 (connector pin 2) and monitoring the arrival time of the "response" from the **RT Simulator** via LPT Status Register bit-4 (connector pin 13). See **Printer Port** for additional details about the PC's legacy printer port I/O interface.

The elapsed time recorded between the stimulus output and the response input given an "ideal system" would always be exactly 200 msec. In order to characterize the latency behavior our real systems we collected "reaction times" for 200 consecutive trials (each separated by an intertrial interval of 200 msec). The statistical results summarizing this test performed on an HP EliteBook Model 8540w equipped with a parallel LPT port on an ExpressPort peripheral card running *MATLAB 2008b* on Windows 7 (x64) are reported in **Table 1** (below). The standard deviation of the latency distribution was very small (only 78 MICROSECONDS) and the range separating the shortest latency from the longest latency was a mere 0.291 msec (291 microseconds). These results clearly indicate that low-level digital I/O implemented via MATLAB **io64()** on Windows-based computers is capable of the temporal precision needed to support the most demanding behavioral research protocols.

	Mean	Standard Deviation	Minimum	Maximum	Range
Windows 7 Enterprise (SP1) HP Elitebook 8540w; 8 GB RAM Intel Core i7 Q720 @ 1.60 GHz PCI-ExpressCard IEEE 1284 Parallel Port	200.191	0.078	200.009	200.300	0.291

Table 1.
Results of **io64()** benchmark latency tests using external 200 msec Reaction Time Simulator

(Statistics based on 200 consecutive trials. All times reported in milliseconds)

Windows Vista/7/8/10 Installation Notes (64-bit)

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 7.7-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 inpoutx64.dll file to the C:\WINDOWS\SYSTEM32 directory.
4. Download the io64.mexw64, config_io.m, inp.m and 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 value 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 *freeware* INPOUTX64 driver for 64-bit Windows XP/Vista/7 can be found here.
Special thanks to Phil Gibbons (www.highrez.co.uk) for providing the signed 64-bit version of the inpoutx64.sys kernel-level driver.
Versions of this software for 32-bit Windows systems can be found here

Last revised: 10 July 2018

Professor Schieber's Home Page - Previous Page