Project RLaB^{plus} (1.0)

libgpib: linux-gpib wrapper library

as seen on

rlabplus.sourceforge.net

© Marijan Koštrun

`kostrun@phys.uconn.edu`

February 24, 2013

Citation Info

If you find RLaB2 Rel. 2 useful, please cite it in your research work as following:

> Marijan Koštrun, Project *rlabplus*, `http://rlabplus.sourceforge.net`, 2004-2013. Ian Searle and coworkers, Project RLaB, `http://rlab.sourceforge.net`, 1992-2001.

Contents

1 libgpib.so: linux-gpib wrapper library 5

1.1 Generic Stuff . . . 6

1.1.1 Organization of shared object libraries in *rlabplus* . . . 6

1.1.2 Installation . . . 6

1.1.3 How to Use the library . . . 6

1.2 Introduction . . . 7

1.3 Control Functions . . . 8

1.3.1 ibdev . . . 8

1.3.2 ibfind . . . 8

1.3.3 ibonl . . . 8

1.3.4 ibclr . . . 9

1.3.5 ibsre . . . 9

1.3.6 ibsic . . . 9

1.3.7 ibln . . . 10

1.4 I/O Functions . . . 11

1.4.1 ibwrt . . . 11

1.4.2 ibrd . . . 11

1.4.3 ibqrd . . . 12

References . . . 17

Chapter 1

libgpib.so: linux-gpib wrapper library

1.1 Generic Stuff

1.1.1 Organization of shared object libraries in *rlabplus*

In *rlabplus* the `gpib` library is located as follows. Say user's name is *John Wayne* and his home directory is `/home/jwayne/`:

```
rlab/
    lib.so/
        gpib/
            rlabplus_libgpib.so
            rlabplus_gpib.c
            rmake
        libgpib.so.r
```

The purpose of the files is as follows:

`rmake`, build script for the library;

`rlabplus_gpib.c`, source code for the library;

`rlabplus_libgpib.so`, shared object library;

`libgpib.so.r`, loader for the library into RLaB.

Please note, in *rlabplus*, `rlab/lib.r` and `rlab/lib.so` are system directories. Typically, the library is downloaded as a tarred archive, which should be unpacked in user's home directory. Together with the library comes a test script, `eg_keithley.r`, which is extracted in `test/gpib`.

Note: Compilation warning. In order for shared library to work properly, it has to be compiled with the same compiler flags the RLaB was built with. The version available for download was compiled with the following flags (`CFLAGS` set in *.bashrc*):

```
-O3 -malign-double -funroll-all-loops -fPIC
```

The same flags are used in building the RLaB available for download from *sourceforge* web site.

1.1.2 Installation

Download the tarred library to your home directory. Untar it. This should extract the library components to proper locations.

1.1.3 How to Use the library

Put the line

```
rfile libgpib.so
```

at the beginning of your script. If the library was installed properly and the compile flags match, the functions provided by the library should be available for use after you run your script the first time.

1.2 Introduction

GPIB is a protocol for communication between a computer and a measuring instrument. The shared object library *gpib.so* provides an user with access to such a measuring device within the *rlabplus* environment. The library is based on the GPIB-Linux project by Hess (2006), and thus requires the successful installation of the later.

In order to use the library, besides the GPIB-Linux supported interface, a compatible measuring instrument is required, as well. Here, we use Keithley 2700 Multimeter from Integra Series (in further text just “Keithley”).

1.3 Control Functions

1.3.1 ibdev

Format: $ud = ibdev(minor, pad, sad/, options/)$

Arguments:

1 *minor*, index of the board;

2 *pad*, primary address of a device;

3 *sad*, secondary address of a device;

4 *options* $=<<$ *timeout*; *send_eoi*; *eos* $>>$, the list the entries of which are

- *timeout*, string, acceptable values are "TNONE", "T10us", "T30us", "T100us", "T300us", "T1ms", "T3ms", "T10ms", "T30ms", "T100ms", "T300ms", "T1s", "T3s", "T10s", "T30s", "T100s", "T300s", "T1000s".
- *send_eoi*, integer. If non-zero then the EOI line is asserted with the last byte sent during writes..
- *eos*, character or integer, specifies the end-of-string (EOS) character or its ASCII code. Non-zero value implies that the reception of EOS character should terminate the reads.
- *eos_mode*, string, acceptable values are "REOS" that enables termination of reads when *eos* is received; "XEOS" that asserts EOI line whenever *eos* character is send during writes; "BIN" that matches the *eos* character using all 8 bites. *eos_mode* is read only if *eos* is provided.

Result: *ud*, an integer, descriptor of a device.

Abstract: *ibdev* initializes the instrument and returns the descriptor *ud* which should be used in consuequent communications with the device.
Default values for the device is what ever is set in */etc/gpib.conf*, with the exception of *eos* = "\n" (new line) and *eos_mode* = "REOS|BIN".

1.3.2 ibfind

Format: $ud = ibfind(s)$

Arguments:

1 *s*, character descriptor of a device, as given in `/etc/gpib.conf`.

Result: *ud*, an integer, descriptor of a device.

Abstract: *ibfind* initializes the instrument and returns the descriptor *ud* which should be used in consuequent communications with the device. If using *ibfind* then the details of the communications should be set in the file `/etc/gpib.conf` rather than through *ibdev*.

1.3.3 ibonl

Format: $i = ibonl(ud/, on/)$

Arguments:

1 *ud*, an integer, descriptor of a device.

2 *on*, an integer, if given than its value should be unity.

Result: *i*, the status of the device/board as reported by *ibsta* byte.

Abstract: *ibonl* without the second argument releases the resources associated with the device the descriptor of which is *ud*. If the second argument is given than the values for that device are set to defaults (either from the file or as set by *ibdev*).

1.3.4 ibclr

Format: $i = ibclr(ud)$

Arguments:

1 *ud*, an integer, descriptor of a device.

Result: *i*, the status of the device/board as reported by *ibsta* byte.

Abstract: *ibclr* clears the device.

1.3.5 ibsre

Format: $i = ibsre(ud/, on/)$

Arguments:

1 *ud*, an integer, descriptor of a device.

2 *on*, an integer, if given than its value should be unity.

Result: *i*, the status of the device/board as reported by *ibsta* byte.

Abstract: If *on* is not given than the REN (Remote Enable) line is unasserted, if it is given and is unity than the REN line is asserted. The board has to be a system controller.

1.3.6 ibsic

Format: $i = ibsic(ud)$

Arguments:

1 *ud*, an integer, descriptor of a device.

Result: *i*, the status of the device/board as reported by *ibsta* byte.

Abstract: *ibsic* performs the interface clear, i.e., it resets the GPIB bus. The board *ud* becomes controller-in-charge.

1.3.7 ibln

Format: $i = ibln(ud, pad, sad)$

Arguments:

1 *ud*, an integer, descriptor of the board;

2 *pad*, primary address of the device being queried;

3 *sad*, secondary address of the device being queried.

Result: i, an integer.

Abstract: *ibln* checks whether the device with primary and secondary address *pad* and *sad* is present or not. In the former case i is nonzero, otherwise it is set to zero.

1.4 I/O Functions

1.4.1 ibwrt

Format: $i = ibwrt(ud, cmds)$

Arguments:

1 *ud*, an integer, descriptor of a device;

2 *cmds*, a string vector, a sequence of commands passed to the device *ud*.

Result: *i*, the status of the device/board as reported by *ibsta* byte, following writing of each command line from *cmds*.

Abstract: *ibwrt* writes the character information contained in the string vector *cmds* to the device *ud*.

1.4.2 ibrd

Format: $c = ibrd(ud/, nbytes/)$

Arguments:

1 *ud*, integer, descriptor of a device;

2 *nbytes*, integer (optional), number of bytes to be read from the device *ud*.

Result: *c*, integer array of length *nbytes*; or string if *nbytes* was not provided.

Abstract: Function receives the response of the device *ud*. If *nbytes* is given then these are returned in an integer row vector. Conversely, if *nbytes* is omitted than the response of the device *ud* is returned in a single string, where the transfer of the response terminates when the first '\0' is encountered.

Note: The functions *char* and *ascii* can be used to efficiently manipulate the response of the instrument.

Note: One could concievably use character read (*nbytes* omitted) for processing a response of an instrument to a query pertaining to its settings, e.g., for oscilloscope Tektronix TDS210 (*ud*3) to find out its horizontal scale,

```
>> ibwrt(ud3, "HOR:MAI:SCA?"); ibrd(ud3)
5.0E-9

>> ibwrt(ud3, "HOR:MAI:SCA?"); s=ibrd(ud3,10)
 matrix columns 1 thru 6
          53            46            48            69            45            57

 matrix columns 7 thru 10
          10             0             0             0
>> char(s)
5.0E-9
```

In both cases, a postprocessing of the reply is necessary. Here, the function `strtod()` suffices to get the numeric value.

1.4.3 ibqrd

Format: $c = ibqrd(ud, /nbytes/, query, /, n/)$

Arguments:

1 *ud*, integer, descriptor of a device;

2 *nbytes*, integer (optional), number of bytes to be read from the device *ud*, in a single query;

3 *query*, string vector, a sequence of commands to be passed to the device *ud*;

4 *n*, integer (optional), number of times the device will be queried.

Result: *c*, integer matrix of size *n* rows and *nbytes* columns containing the reply of the instrument to query; or, vector of *n* string entries, where each string represents a reply of the device to the query.

Abstract: *ibqrd* performs querried read of the device *ud* where the repetition is performed at the lower level.

Note: The purpose of this command is to increase the speed of querrying the device as doing it through script may not be fast enough for some applications.

Example: Communication with Keithley instrument to retrieve the measured values (VDC) at maximal rate.

```
//
// eg_keithley.r: short the terminals for voltage measurement
//
rfile libgpib.so

// initialize gpib devices
if (!exist(ud1))
{
  ud1 = ibdev(0,0,0); // gpib main board, here gpib-usb-hs, set to 0
}
if (!exist(ud2))
{
  ud2 = ibdev(0,7,0); // set keithley's gpib-address to 7
}

// number of readings
N = 100;

// data list for plotting
mydata = <<>>;

// use for storing the numerical values
x = zeros(N,2);

// initialize keithley:
//  1:  SYST:PRES         'Continuous measurement mode (INIT:CONT ON)'
//  2:  FUNC 'VOLT:DC'    'Select DCV function'
//  3:  VOLT:DC:NPLC 0.1  'Set maximum read rate (FAST)'
mycode = [ "SYST:PRES", "FUNC 'VOLT:DC'", "VOLT:DC:NPLC 0.1"];
ibwrt(ud2,mycode);

//
// read N data points through internal loop
//
r = ibqrd(ud2,["SENS:DATA:FRES?"],N);


# To above query Keithley replies with
#   -1.10276960E-06VDC,+6606.482SECS,+66586RDNG#
# this is to be extracted into a matrix with columns
# [time, voltage]
datarange = [1:strindex(r[1],"VDC")-1];
timerange = [strindex(r[1],"VDC")+4:strindex(r[1],"SECS")-1];

for (i in 1:r.nr)
```

```
{
  x[i;1] = strtod( substr(r[i], timerange) );
  x[i;2] = 1e6 * strtod( substr(r[i], datarange) );
}
mydata.a = x;

//
// single query reading within rlab loop
//
for (i in 1:N)
{
  r = ibqrd(ud2,["SENS:DATA:FRES?"]);
  x[i;1] = strtod( substr(r, timerange) );
  x[i;2] = 1e6 * strtod( substr(r, datarange) );
}
mydata.b = x;

//
// read/write query within rlab loop
//
for (i in 1:N)
{
  ibwrt(ud2, "SENS:DATA:FRES?");
  r = ibrd(ud2);
  x[i;1] = strtod( substr(r, timerange) );
  x[i;2] = 1e6 * strtod( substr(r, datarange) );
}
mydata.c = x;


// plot
xlabel ( "Instrument Time (sec)" );
ylabel ( "Instrument Reading (\gmV)" );
plegend( ["N queries", "1 query", "ibwrt/ibrd"] );
plot   ( mydata );

// reading times difference internal/external
ta = last(mydata.a)[1] - mydata.a[1;1];
tb = last(mydata.b)[1] - mydata.b[1;1];
tc = last(mydata.c)[1] - mydata.c[1;1];
printf("Times for %g measuremets\n", N);
printf("\tN queries : %g sec\n", ta);
printf("\t1 query   : %g sec\n", tb);
printf("\tibwrt/ibrd: %g sec\n", tc);
```

Example: Communication with Textronic TDS-210. This requires binary manipulation functions as implemented in RLaB using functions *ascii*, *char*, and internal integer representation of the values for which logical operators act bitwise.

```
//
// eg_tds210.r: ch1 terminal on probe compensation
//
rfile libgpib.so

// initialize gpib devices
if (!exist(ud1))
{
  ud1 = ibdev(0,0,0); // init gpib main board, here gpib-usb-hs, pid,sid = 0,0
}
if (!exist(ud3))
{
  ud3 = ibdev(0,6,0); // init tds210, pid,sid = 0,6
}

//
// get the responses of the device
//
reply = [];
for (i in 1:255)
{
  r = ibask(ud3, i);
  if (isempty(r)){ continue; }
  reply = [reply; i, r];
}
printf("Configuration bits for the device tds210 (ud3):\n");
reply

//
// basic commnunication to set-up waveform parameters
//
ibclr( ud3 );   // clear the osciloscope
mycode = [ ...
    "DATA:SOURCE CH1"; ...
    "DATA:ENCDG RIBINARY;WIDTH 1"; ...
    "HORIZONTAL:RECORDLENGTH 1000"; ...
    "DATA:START 1"; ...
    "DATA:STOP 1000"; ...
    "HEADER OFF"; ...
    "ACQUIRE:STATE RUN"; ...
    "CURVE?" ];
ibwrt ( ud3, mycode );

//
```

```
//
//
c = ibrd( ud3, 1 );   // read first byte, '#'
c = ibrd( ud3, 1 );   // read second byte, the length of a string containing the length
n = strtod( c );      // convert it to number
L = ibrd( ud3, n );   // read string containing the number of bytes to transfer
N = strtod ( L );     // convert the string to the number
s = ibrd( ud3, N );   // read in N bytes, each representing a data point
x = [1:N]';
y = ascii(s)' +0;
plot( [x,y] );
```

References

Hess, F. M. (2006). *Linux-GPIB 3.2.06 Documentation.* (`http://linux-gpib.sourceforge.net`.)