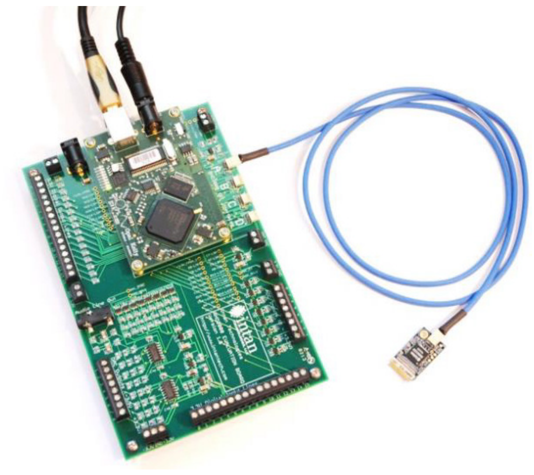
RHD2000

MATLAB Toolbox Getting Started Guide



26 January 2015; updated 27 October 2016

This document provides basic information about installing the RHD2000 MATLAB Toolbox, running its examples, and programming with it. **MATLAB 2012 or later** is required for using this toolbox.

Installation

The RHD2000 MATLAB Toolbox comes as a self-extracting executable installer (.EXE file), which you should run. It installs several dependencies, if needed, and installs the toolbox and examples. By default, it installs the toolbox and examples in the **<matlabroot>\toolbox\local\RHD2000 Matlab Toolbox** directory, but that can be changed during the install process.

Whichever directory the toolbox is installed in should be added to the MATLAB path. Typically this happens automatically as part of the installation, which can be tested by:

```
help rhd2000
```

If this command returns "rhd2000 not found," you should manually add it to your path:

addpath(***the path where you installed it***);

```
savepath;
```

Setup

The RHD2000 MATLAB Toolbox consists of a Windows Dynamic Link Library (DLL) and several .m files that call into it. Using a DLL in MATLAB requires that mex be configured. If you haven't already done this (e.g., if you've never used mex before), you'll need to issue the command:

```
mex -setup cpp
```

and choose your C++ compiler. If the `cpp` option returns an error, omit it. See MATLAB's documentation for more information.

Additionally, you should have installed the RHD2000 USB Interface Board drivers, and should be able to run the RHD2000 interface software.

Getting started in MATLAB

Let's start with a simple example of setting the LEDs on the USB interface board. Plug in an RHD2000 Evaluation Board and issue the following commands:

```
driver = rhd2000.Driver();
board = driver.create_board();
board.LEDs = [1 0 1 0 1 0 1 0];
```

The first line creates an RHD2000 Driver object. All programs that control an RHD2000 board will contain this line; the Driver object loads the DLL and makes it available to other classes in the toolbox. For more information on the Driver, type `doc rhd2000.Driver`.

The second line connects to the board. The 'board' variable contains the connection to the specific board you're controlling, and all board- and chip-level commands will go through it. For more information on the Board, type `doc rhd2000.Board`.

The third line sets the LEDs. The LEDs are controlled by the 'LEDs' property of 'board.' The 'LEDs' property is an array of length 8, where each element is 0 or 1. The setting above turns every second LED on. To turn them all off again, issue the command:

```
board.LEDs = zeros(1,8);
```

Now let's look at the RHD2000 chips attached to the board. Type:

```
board.Chips
```

and you'll get a result something like this:

```
ans =
    rhd2132
    none
    none
    none
    none
    none
    none
    none
```

In this case, a single RHD2132 chip is attached to Port A. (Specifically, to Port A, MISO 1.) Your results may vary, depending on what is attached to the board. For more information, type `doc rhd2000.Board.Chips`.

The 'board' object has various properties and methods to control the RHD2000 evaluation board and attached chips. To look a little more deeply, type:

```
board
```

You'll get something like:

```
ans =
  Board with properties:
            handle: 1
         DacManual: 0
    DigitalOutputs: [NaN NaN NaN NaN NaN NaN NaN NaN 0 0 0 0 0 0 0 0]
              LEDs: [8x1 double]
```

```
          SamplingRate: [1x1 rhd2000.SamplingRate]
                 Chips: [8x1 rhd2000.Chip]
              SaveFile: [1x1 rhd2000.savefile.SaveFile]
         DigitalInputs: [16x1 int32]
               FIFOLag: 0
    FIFOPercentageFull: 0
```

You can learn more about using the board object to control a board by typing `doc rhd2000.Board`.

One final note: as long as you have the 'board' object around, you have a live connection to the RHD2000 evaluation board, and as long as you have the 'driver' object around, you have the library loaded. You can disconnect these both by typing:

```
clear
```

For more information

Start by reading the documentation of `rhd2000.Driver` and `rhd2000.Board`. Click on links, especially ones labeled "For more information, see." This will provide you with an overview of the capabilities and examples of using individual functions to control the board.

Examples

Examples are installed at **<matlabroot>\toolbox\local\RHD2000 Matlab Toolbox\examples**, or at the other path that you chose during the install process. We recommend that you look through the examples, run them, and study their code to learn how to use the toolbox. Here is a list of examples and their descriptions:

NAME	DESCRIPTION
read_one_data_block	Example of reading data. Demonstrates the basics of reading and the data types available.
read_continuously	GUIDE example GUI for reading from a board.
episodic_recording	GUIDE example GUI for reading from a board. Saving is controlled by Digital Input 0.
read_optimized	Non-GUIDE example, showing how to acquire and save data quickly.
real_time_analysis	Example of real time analysis of data.
save_file_options	Example of configuring save files
set_default_parameters	Example of setting parameters
dac_manual	Example of using DAC Manual to drive an analog output.
measure_all_impedances	Example of measuring impedance
two_boards	GUIDE example GUI for reading from two boards.

Troubleshooting

The RHD2000 Matlab Toolbox produces error messages (i.e., uses the MATLAB `error` command), when things go wrong. The text of the error message, especially the topmost text, will typically tell you what went wrong (e.g., you've specified an invalid parameter).

One common error message is

```
The requested system device cannot be found.
```

This means the system can't find the board. There are a couple of reasons for this:

1. No board is connected and powered.
2. Something else is already connected to the board.

The former is easy to fix; plug in a board and try again. The latter can happen:

- If you already have a board object (in which case, you may want to use it, or to `clear` that variable before continuing).
- If you have some other object that has retained a reference to the board (e.g., a DataBlock object). You should `clear` that object before continuing.
- If some other program (e.g., the RHD2000 Interface) is open, using the board.

When you `run_continuously`, it is crucially important that you process data from the board fast enough that the FIFO queue doesn't fill up. The documentation and examples have details of how to do this, and how to confirm you're keeping up with the board.

Additional Help

For any feedback or additional information not covered by the examples or documentation, please email **info@intantech.com**.