fenrir / hugen79

GPIB-USBCDC CONTROLLER

USER MANUAL

Version 1.2
Dec 02, 2021

Table of Contents

Change Log

Nov 27, 2021	Initial version
Nov 29, 2021	Added “debug 0” description
Dec 02, 2021	Added Hugen79 pics and FW update connector pin description

1 Introduction

Fenrir GPIB-USBCDC is an interface bridge between GPIB (HPIB) and USB communication device class. It is a Prologix GPIB-USB adapter clone with EFM8 Universal Bee or C8051F38x microcontroller.
Fenrir GPIB-USBCDC controller converts any computer with a USB port into a GPIB Controller or Device.
In Controller mode, Fenrir GPIB-USBCDC controller can remotely control GPIB enabled instruments such as Oscilloscopes, Logic Analyzers, and Spectrum Analyzers.
In Device mode, Fenrir GPIB-USBCDC controller converts the computer into a GPIB peripheral for downloading data and screen plots from the instrument front panel.
In both modes, Fenrir GPIB-USBCDC controller interprets high level commands received from the host computer and performs the appropriate low-level GPIB protocol handshaking.

2 Installation

Fenrir GPIB-USBCDC controller is recognized as serial port, which will be installed with CDC .inf file. To install it, simply connect the Fenrir GPIB-USBCDC to the PC and it will be automatically discovered and installed by the operating system.

3 Firmware

The official firmware binary is published in github release. To build the firmware by yourself, install Small Device C Compiler - SDCC (testing with ver 3.3.0 #8604), and just "make" at "firmware" directory of the downloaded code. The generated firmware name is `gpib-usbcdc.hex`. The firmware code is published under New BSD License.
Depending on the hardware you are using you will need the original fenrir firmware or hugen79 modded firmware.

Fenrir git URL: https://github.com/fenrir-naru/gpib-usbcdc
Hugen79 git URL: https://github.com/hugen79/gpib-usbcdc

3.1 Write firmware to hardware

Connect a GPIB-USBCDC board and a PC via USB debug adapter (UDA) or compatible one. The minimum required programming connections are summarized in the following table. Then, use Flash Programming Utilities.

Note:
the board may not be recognized by a PC when an UDA is connected via USB hubs. UDA is recommended to connect a PC directly.

Signal	UDA side	Fenrir GPIB-USBCDC board side	Hugen79 GPIB-USBCDC board side
C2D	4th pin	CON3 3rd pin	V.D.C.G. 2nd pin
C2CK	7th pin	CON3 4th pin	V.D.C.G. 3rd pin
GND	3rd pin	CON3 1st pin	V.D.C.G. 4th pin

See Fenrir layout and Hugen79 PCB for more detail on connectors.

4 Host Software

A wide variety of host software may be used to communicate with Fenrir GPIB-USBCDC controller:

Terminal programs – any terminal emulation program such as HyperTerminal, Tera Term Pro, or Minicom can be used to communicate with the controller and instruments connected to it.

Custom applications – any programming language or environment that provides access to serial ports (if using VCP driver) or allows interfacing to DLL (if using D2XX driver) may be used to develop custom applications. Graphical programming environments like National Instruments LabView and Agilent VEE may be used as well.

EZGPIB – an easy to use, programming environment developed by Ulrich Bangert for developing data acquisition applications. Web link to this tool can be found at prologix.biz.

Plotter emulators – plotter emulation applications such as 7470.exe, PrintCapture and Plottergeist can be used to render screen plots downloaded using Fenrir GPIB-USBCDC controller. Details on how to configure these tools are available at prologix.biz.

5 Configuration

Fenrir GPIB-USBCDC controller can be configured using any of the following methods:

Prologix.exe – Prologix.exe is an open source tool developed by John Miles for configuring the controller. Web link to the tool can be found at prologix.biz.

Terminal program – any terminal emulation program such as HyperTerminal, Tera Term Pro, or Minicom can be used to configure the controller by manually entering appropriate commands (See Commands). Using the terminal program open the virtual COM port created by the USB driver. Serial port parameters such as baud rate, data bits, stop bits and flow control do not matter and may be set to any value. You may want to enable the "Local Echo" feature in the terminal program to view the commands being entered. Please consult the FAQ at prologix.biz, or the program's user manual, for detailed configuration steps for various terminal programs.

Fenrir GPIB-USBCDC Controller stores the latest configuration settings in non-volatile memory. These settings are not address specific. If you have multiple instruments on the GPIB bus that require different configuration settings, you must change the settings before communicating with each instrument.

6 Operating Modes

Fenrir GPIB-USBCDC controller can operate in two modes – CONTROLLER and DEVICE. You can switch between the two modes using `++mode` command (see Commands).

6.1 Controller Mode

In Controller mode, the GPIB-USB Controller acts as the Controller-In-Charge (CIC) on the GPIB bus. When the controller receives a command over the USB port terminated by the USB terminator – **CR** (ASCII 13) or **LF** (ASCII 10) – it addresses the GPIB instrument at the currently specified address (See `++addr` command) to listen, and passes along the received data.

When Read-After-Write feature is enabled (See `++auto` command) Fenrir GPIB-USBCDC Controller will addresses the instrument to talk after sending a command, in order to read its response. All data received from instruments over GPIB is sent to host over USB. Read-After-Write feature simplifies communication with instruments. You send commands and read responses without consideration for low level GPIB protocol details.

When Read-After-Write feature is not enabled Fenrir GPIB-USBCDC controller does not automatically address the instrument to talk. You must use the `++read` command to read data.

Controller mode is used to remotely control instruments and to download screen plots by sending plot commands from a host computer.

6.2 Device Mode

In Device mode, Fenrir GPIB-USBCDC Controller acts as another peripheral on the GPIB bus. In this mode, the controller can act as a GPIB TALKER or GPIB LISTENER only. Since Fenrir GPIB-USBCDC Controller is not the Controller-In-Charge while in this mode, it expects to receive commands from a GPIB controller. When Device mode is enabled Fenrir GPIB-USBCDC controller configures itself as a GPIB Listener.

All data received by the controller over the GPIB port is passed along to the USB port without buffering. All data received from the host over USB is buffered until the GPIB controller addresses Fenrir GPIB-USBCDC controller to talk, at which time the buffered data is passed along to the GPIB port. The controller can buffer only one command. A subsequent command received over USB will overwrite the previously buffered one, if the previous one has not yet been transmitted over GPIB.

Device mode is used to download screen plots from the instrument front panel for rendering using plotter emulation software.

7 Data Transmission

In Controller and Device modes, characters received over USB port are aggregated in an internal buffer and interpreted when a USB termination character – **CR** (ASCII 13) or **LF** (ASCII 10) – is received. If **CR**, **LF**, **ESC** (ASCII 27), or '**+**' (ASCII 43) characters are part of USB data they must be escaped by preceding them with an **ESC** character. All un-escaped **LF**, **CR** and **ESC** and '**+**' characters in USB data are discarded.

As mentioned earlier, an un-escaped **CR** or **LF** acts as the USB terminator. The terminating **CR** or **LF** is removed and GPIB termination characters (specified by `++eos` command) are appended before transmitting data to instruments.

Any USB input that starts with the unescaped "++" character sequence is interpreted as a controller command and not transmitted over GPIB.

When configured to do so – using the `++auto` command or the `++read` command – characters received from instruments are transmitted to host. Unlike while sending data to instruments, no character substitution is performed. The firmware buffers only one character. The USB driver may provide additional buffering. If a device is talking and the buffers (firmware's and driver's) are full, the device continues to be in TALK mode but is stopped from further sending data using GPIB handshaking.

The `++eot_char` command may be used to detect GPIB EOI signal assertion.

7.1 Binary Data Transmission

Fenrir GPIB-USBCDC controller can send and receive binary data to and from GPIB enabled instruments.

No special action is necessary to receive binary data from instruments. Any binary data received from the instrument is transmitted over USB to PC unmodified, just as with ASCII data. Since binary data from instruments is not usually terminated by CR or LF characters (as is usually the case with ASCII data), you may want to use the ++eot_enable command to detect EOI indicating end of data. See `++eot_enable` command help for more details.

Special care must be taken when sending binary data to instruments. If any of the following characters occur in the binary data -- **CR** (ASCII 13), **LF** (ASCII 10), **ESC** (ASCII 27), '**+**' (ASCII 43) – they must be escaped by preceding them with an **ESC** character.

For example, to send the following (decimal) binary data:

```
00 01 02 13 03 10 04 27 05 43 06
```

it must be escaped as follows:

```
00 01 02 27 13 03 27 10 04 27 27 05 27 43 06
```

Further more, most instruments will get confused if GPIB termination characters, such as CR or LF, are appended to binary data. Use `++eos 3` command to disable such behavior. See `++eos` command help for more details.

8 Commands Prologix compatible

Fenrir GPIB-USBCDC controller provides several commands to configure its behavior. They are explained in detail in the following sections. All commands start with the "++" character sequence.

Commands must be terminated with **CR** or **LF**.

8.1 addr

The addr command is used to configure, or query the GPIB address. Meaning of the GPIB address depends on the operating mode of the controller. In CONTROLLER mode, it refers to the GPIB address of the instrument being controlled. In DEVICE mode, it is the address of the GPIB peripheral that Fenrir GPIB-USBCDC controller is emulating.

An optional secondary address may also be specified. Secondary address must be separated from the primary address by a space character. Valid secondary address values are 96 to 126 (decimal). Secondary address value of 96 corresponds to secondary GPIB address of 0, 97 corresponds to 1, and so on. Specifying secondary address has no effect in DEVICE mode.

If the command is issued with no parameters, the currently configured address (primary, and secondary, if specified) is returned.

SYNTAX: `++addr [<PAD>[<SAD>]]`

where:

PAD *(Primary Address) is a decimal value between 0 and 30.*
SAD *(Secondary Address) is a decimal value between 96 and 126. SAD is optional.*

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLES:

`++addr 5`	– Set primary address to 5
`++addr`	– Query current address
`++addr 9 96`	– Set primary address to 9 and secondary address to 0

NOTE:

Default GPIB address of many HP-GL/2 plotters is 5.

8.2 auto

Fenrir GPIB-USBCDC controller can be configured to automatically address instruments to talk after sending them a command in order to read their response. The feature called, Read-After-Write, saves the user from having to issue read commands repeatedly. This command enabled or disabled the Read-After-Write feature.

In addition, auto command also addresses the instrument at the currently specified address to TALK or LISTEN. `++auto 0` addresses the instrument to LISTEN and `++auto 1` addresses the instrument to TALK.

If the command is issued without any arguments it returns the current state of the readafter-write feature.

SYNTAX: `++auto [0|1]`

MODES AVAILABLE: CONTROLLER

NOTE:

Some instruments generate “`Query Unterminated`” or “`-420`” error if they are addressed to talk after sending a command that does not generate a response (often called nonquery commands). In effect the instrument is saying, I have been asked to talk but I have nothing to say. The error is often benign and may be

ignored. Otherwise, use the `++read` command to read the instrument response. For example:

```
++auto 0        — Turn off read-after-write and address instrument to
listen
SET VOLT 1.0    — Non-query command
*IDN?           — Query command
++read eoi      — Read until EOI asserted by instrument
"HP54201A"      — Response from instrument
```

8.3 clr

This command sends the Selected Device Clear (SDC) message to the currently specified GPIB address. Please consult the programming manual for details on how a particular instrument responds to this message.

SYNTAX: `++clr`

MODES AVAILABLE: CONTROLLER

8.4 eoi

This command enables or disables the assertion of the EOI signal with the last character of any command sent over GPIB port. Some instruments require EOI signal to be asserted in order to properly detect the end of a command.

SYNTAX: `++eoi [0|1]`

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLES:

```
++eoi 1    - Enable EOI assertion with last character
++eoi 0    - Disable EOI assertion
++eoi      - Query if EOI assertion is enabled or disabled
```

8.5 eos

This command specifies GPIB termination characters. When data from host is received over USB, all non-escaped LF, CR and ESC characters are removed and GPIB terminators, as specified by this command, are appended before sending the data to instruments. This command does not affect data from instruments received over GPIB port.

If the command is issued with no arguments then the current configuration is returned.

SYNTAX: `++eos [0|1|2|3]`

where:

0 *– CR+LF,*
1 *– CR,*
2 *– LF,*
3 *– None*

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLES:

```
++eos 0    - Append CR+LF to instrument commands
++eos 1    - Append CR to instrument commands
++eos 2    - Append LF to instrument commands
++eos 3    - Do not append anything to instrument commands
++eos      - Query current EOS state
```

8.6 eot_enable

This command enables or disables the appending of a user specified character (see also `++eot_char`) to USB output whenever EOI is detected while reading a character from the GPIB port.

If the command is issued without any argument, the current state of `eot_enable` is returned.

SYNTAX: `++eot_enable [0|1]`

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLES:

```
++eot_enable 1 - Append user defined character when EOI detected
++eot_enable 0 - Do not append character when EOI detected
++eot_enable   - Query current eot_enable state
```

8.7 eot_char

This command specifies the character to be appended to USB output when `eot_enable` is set to 1 and EOI is detected.

If the command is issued without any argument, the currently specified character is returned.

SYNTAX: `++eot_char [<char>]`

where:

<char> *is a decimal value less than 256*

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLES:

`++eot_char 42` - Append * (ASCII 42) when EOI is detected
`++eot_char` - Query currently configured eot_char

8.8 ifc

This command asserts GPIB IFC signal for 150 microseconds making Prologix GPIBUSB controller the Controller-In-Charge on the GPIB bus.

SYNTAX: `++ifc`

MODES AVAILABLE: CONTROLER

8.9 llo

This command disables front panel operation of the currently addressed instrument.

SYNTAX: `++llo`

MODES AVAILABLE: CONTROLLER

8.10 loc

This command enables front panel operation of the currently addressed instrument.

SYNTAX: `++loc`

MODES AVAILABLE: CONTROLLER

8.11 lon

This command configures the GPIB-USB controller to listen to all traffic on the GPIB bus, irrespective of the currently specified address. This configuration is also known as “listen-only” mode. In this mode, the controller can only receive, but cannot send any data.

SYNTAX: `++lon [0|1]`

MODES AVAILABLE: DEVICE

EXAMPLES:

`++lon 1` - Enable “listen-only” mode
`++lon 0` - Disable “listen-only” mode
`++lon` - Query “listen-only” mode

8.12 mode

This command configures the Fenrir GPIB-USBCDC controller to be a CONTROLLER or DEVICE.

If the command is issued without any arguments, the current mode is returned.

SYNTAX: `++mode [0|1]`

where:

1 *– CONTROLLER,*
0 *– DEVICE*

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLES:

`++mode 1` - Switch to CONTROLLER mode
`++mode 0` - Switch to DEVICE mode
`++mode` - Query current mode

8.13 read_tmo_ms

This command specifies the timeout value, in milliseconds, to be used in the `++read` command and `++spoll` command. Timeout may be set to any value between 1 and 3000 milliseconds.

SYNTAX: `++read_tmo_ms <time>`

where:

<time> is decimal value between 1 and 3000

MODES AVAILABLE: CONTROLLER

8.14 rst

This command performs a power-on reset of the controller. The process takes about 5 seconds. All input received over USB during this time are ignored.

SYNTAX: `++rst`

MODES AVAILABLE: CONTROLLER, DEVICE

8.15 spoll

This command performs a serial poll of the instrument at the specified address. If no address is specified then this command serial polls the currently addressed instrument (as set by a previous `++addr` command). This command uses the time-out value specified by the `++read_tmo_ms` command.

SYNTAX: `++spoll [<PAD>[<SAD>]]`

where:

***PAD** (Primary Address) is a decimal value between 0 and 30.*
***SAD** (Secondary Address) is a decimal value between 96 and 126. SAD is optional.*

MODES AVAILABLE: CONTROLLER

EXAMPLE:

`++spoll 5` - Serial poll instrument at primary address 5
`++spoll 9 96` - Serial poll instrument at primary address 9, secondary address 0
`++spoll` - Serial poll currently addressed instrument

8.16 srq

This command returns the current state of the GPIB SRQ signal. The command returns ‘1’ is SRQ signal is asserted (low) and ‘0’ if the signal is not asserted (high).

SYNTAX: `++srq`

MODES AVAILABLE: CONTROLLER

8.17 status

The `++status command` is used to specify the device status byte to be returned when serial polled by a GPIB controller. If the RQS bit (bit #6) of the status byte is set then the SRQ signal is asserted (low). After a serial poll,

SRQ line is de-asserted and status byte is set to 0. Status byte is initialized to 0 on power up. SRQ is also de-asserted and status byte is cleared if DEVICE CLEAR (DCL) message, or SELECTED DEVICE CLEAR (SDC) message, is received from the GPIB controller.

If the command is issued without any arguments it returns the currently specified status byte.

SYNTAX: `++status [0-255]`

MODES AVAILABLE: DEVICE

EXAMPLE:

`++status 48` - Specify serial poll status byte as 48. Since bit #6 is set, this command will assert SRQ.

`++status` - Query current serial poll status byte.

8.18 trg

This command issues Group Execute Trigger GPIB command to devices at the specified addresses. Up to 15 addresses maybe specified. Addresses must be separated by spaces. If no address is specified then Group Execute Trigger command is issued to the currently addressed instrument (as set by a previous `++addr` command). Refer to the programming manual for a specific instrument’s response to Group Execute Trigger command.

SYNTAX: `++trg [<PAD1> [<SAD1>] <PAD2> [SAD2] … <PAD15> [<SAD15>]]`

MODES AVAILABLE: CONTROLLER

8.19 ver

This command returns the version string of the Fenrir GPIB-USBCDC controller.

SYNTAX: `++ver`

MODES AVAILABLE: CONTROLLER, DEVICE

8.20 help

This command prints a brief summary of all available commands.

SYNTAX: `++help`

MODES AVAILABLE: CONTROLLER, DEVICE

9 Commands different from Prologix

There is a small set of commands with a little different implementation from Prologix one or some new commands not available in Prologix USB Controller.

9.1 read

This command can be used to read data from an instrument until:

- EOI is detected or timeout expires, or
- Timeout expires

Timeout is set using the `++read_tmo_ms` command and applies to inter-character delay, i.e., the delay since the last character was read. Timeout is not be confused with the total time for which data is read.

SYNTAX: `++read [eoi]`

MODES AVAILABLE: CONTROLLER

EXAMPLES:

```
++read          - Read until timeout
++read eoi      - Read until EOI detected or timeout
```

9.2 savecfg

This command saves the current values of all configuration parameters in the internal EPROM.

SYNTAX: ++savecfg

MODES AVAILABLE: CONTROLLER, DEVICE

9.3 debug

This command enables the debug output for each command issued.

If the command is issued without any arguments it returns the currently debug level.

SYNTAX: `++debug [<level>]`

where:

<level> *is a bitmask in decimal format of:*

0 (000000000) disable debug
1 (000000001) enable serial command echo
2 (000000010) enable GPIB echo
4 (000000100) verbose output

MODES AVAILABLE: CONTROLLER, DEVICE

EXAMPLE:

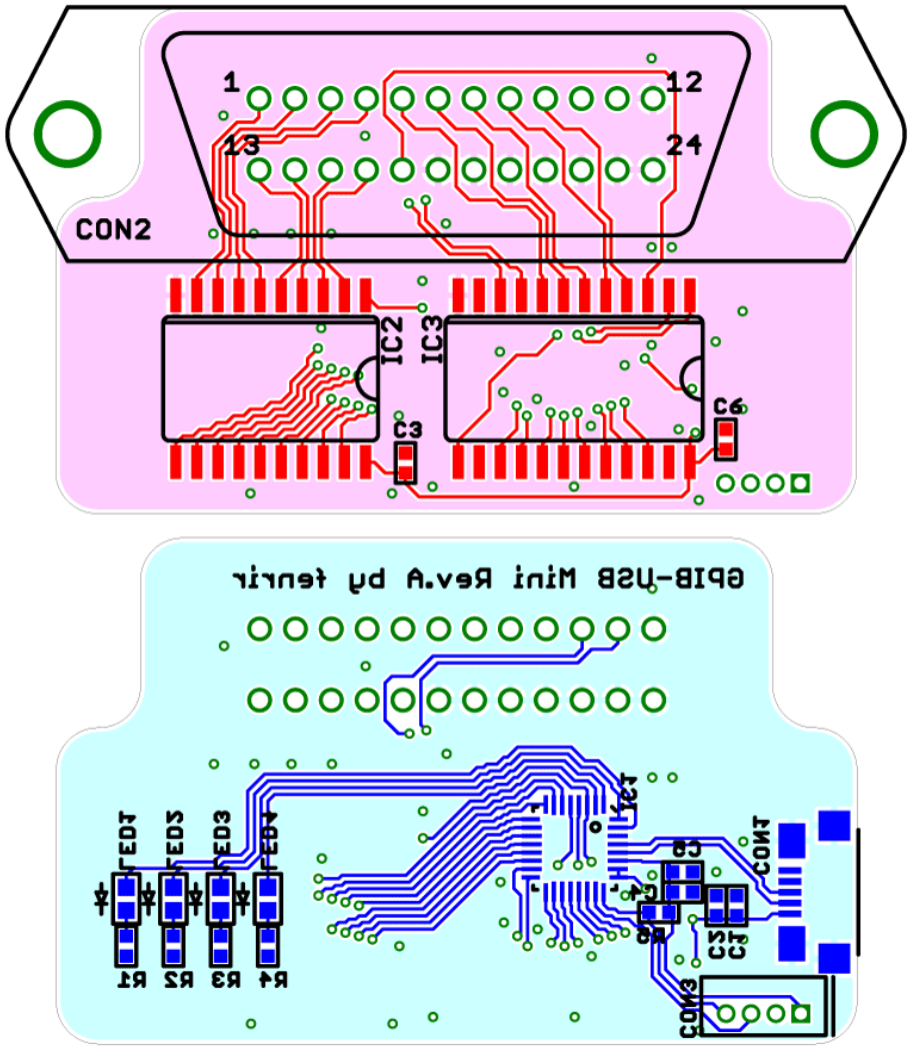
`++debug 0` - Disables debug
`++debug 1` - Enables echo of serial commands issued
`++debug 2` - Enables GPIB echo
`++debug 3` - Enables echo of serial commands and GPIB
`++debug 4` - Enables verbose output
`++debug 5` - Enables verbose output and echo of serial commands
`++debug 6` - Enables verbose output and GPIB echo
`++debug 7` - Enables verbose output and echo of serial commands and GPIB
`++debug` - Query current debug level

10 Specifications and Technical details

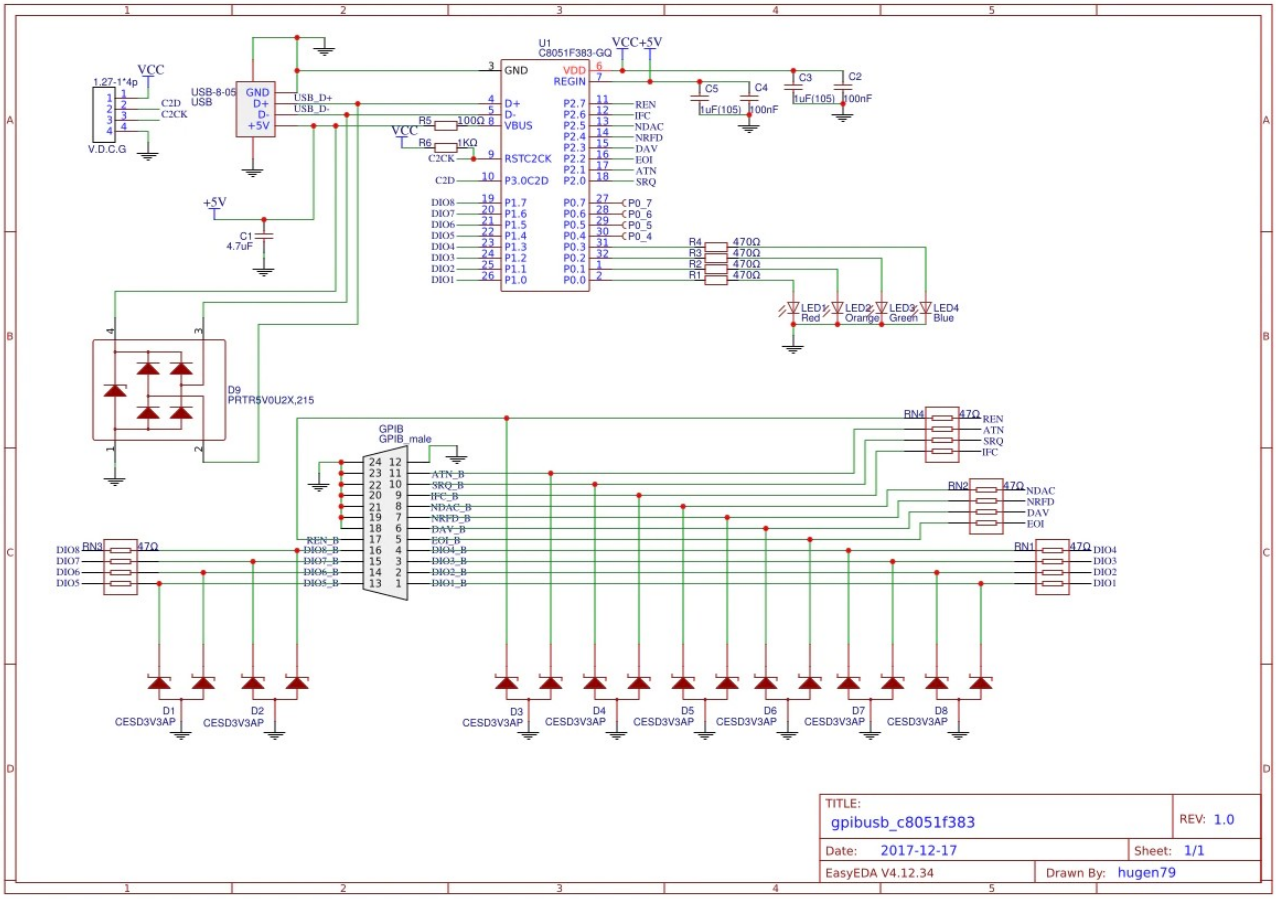
Supported OS: Microsoft Windows (7 or later), Mac OS X, Linux (kernel 2.6 or later), FreeBSD

Supported Standards: IEEE 488.1, IEEE 488.2, USB 1.1, USB 2.0
GPIB commands not supported: PARALLEL POLL, PASS CONTROL
Power: USB bus powered, +5V, 100 mA (max)
Supported USB hubs: Self-powered and Bus-powered hubs
Indicators: TALK, LISTEN, CONTOLLER MODE, ACTIVITY

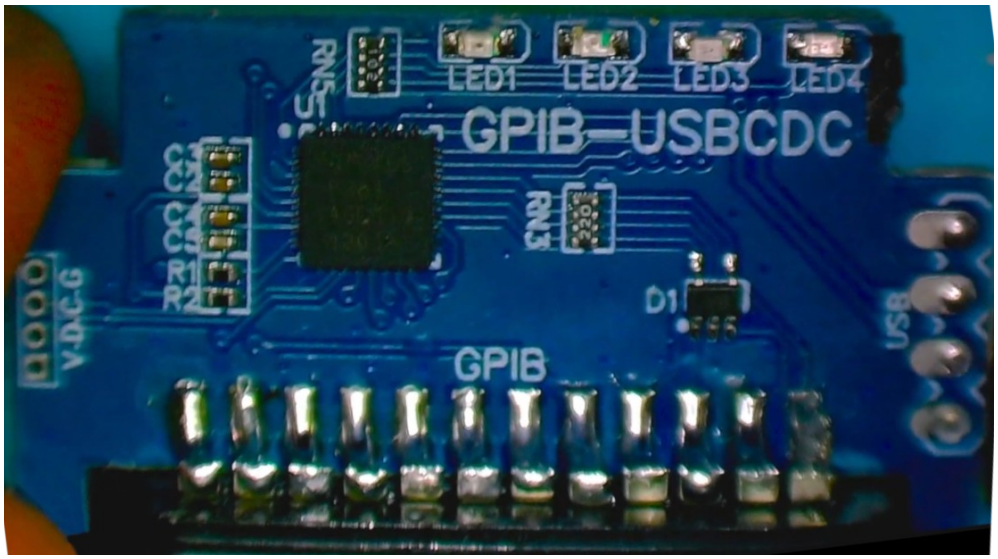
10.1 Fenrir layout

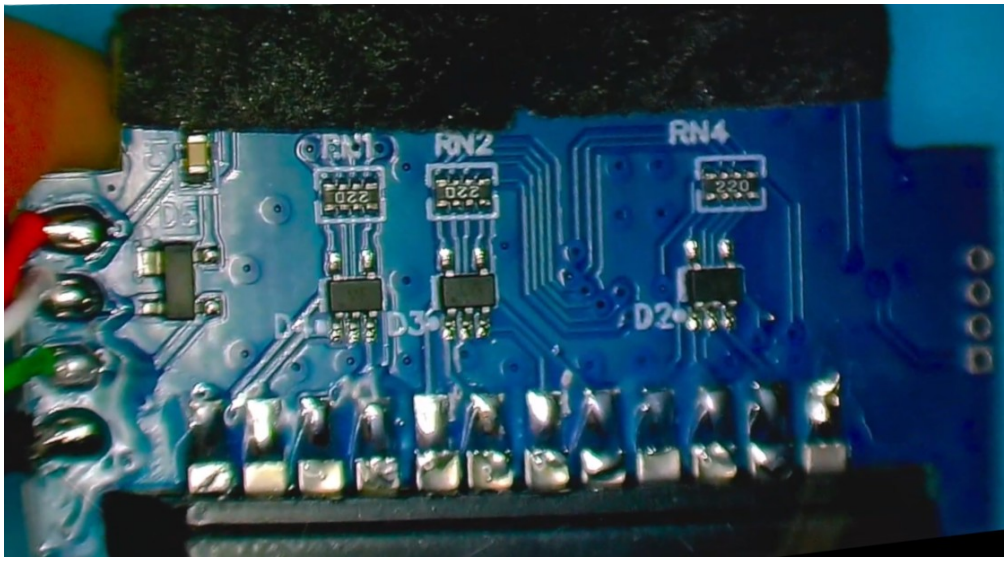


10.2 Hugen79 schematics



10.3 Hugen79 PCB




RN1
RN2
RN4
220
220
220
D3
D2