Project "eStick Control Center" (eCC)

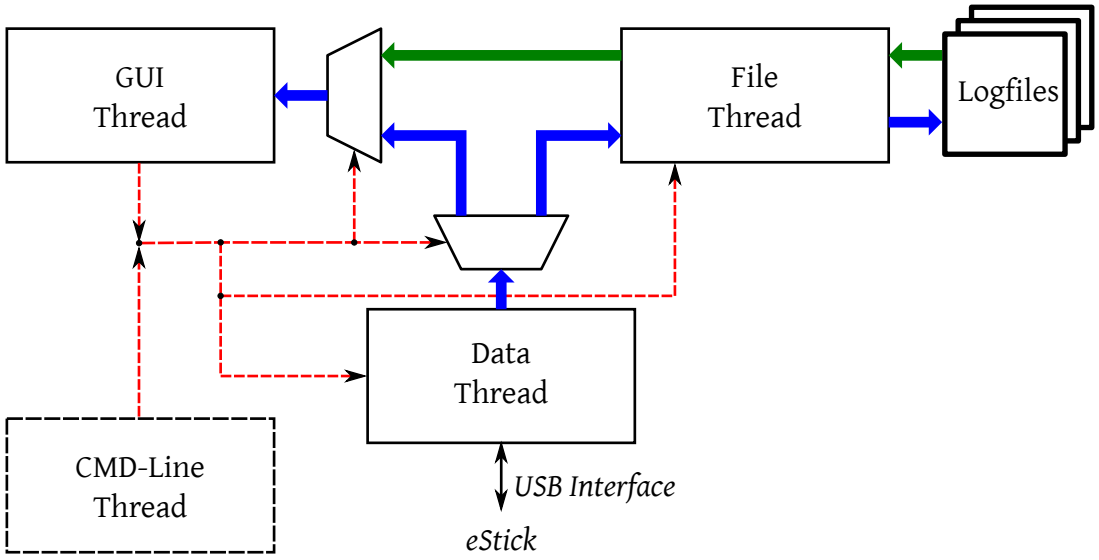
Project Description: The *default project*[1] shall implement a multi-threaded control and data interface for the eStick that communicates with the latter using the libUSB library. A program-image for the eStick as well as sample code for the libUSB library will be provided.
The following block diagram illustrates the multiple threads along with the control and data-flow of the program. In particular, the program consists of the following threads:

(1) GUI Thread

(2) Data Thread

(3) File Thread

(4) CMD Line Thread (optional)



Control information is illustrated with dashed red arrows. Data flow is illustrated with thicker blue and green arrows, respectively. Per default the program is invoked with the graphical user interface (GUI operation). Only when the program is invoked with the command-line option `./eCC -cmd` the optional command-line thread is used in place of the GUI thread (CMD operation). Either thread shall provide mechanisms to control the application; only the GUI thread shall provide additional means to visualize the data.

The entire application shall operate in the following states: `STOP`, `RECORD-USB`, `RECORD-SIMULATION`, and `REPLAY` (GUI operation only).

After program invocation the program shall start in `STOP` mode. Here the user shall be able to configure the interface and the files for recording and replay.

In `RECORD` mode the data from the Data Thread shall be forwarded to the GUI Thread (when available) and the File Thread (blue arrows). The GUI Thread is responsible to present the data to

[1] You are free to define and implement an alternative project.

the user, whereas the File Thread is responsible to store the data in respective log-files. The Data Threads either streams the data/control information from/to the eStick (`RECORD-USB Mode`) or provides a respective data source/sink using random numbers (`RECORD-SIMULATION MODE`). In this mode no data is streamed from the File Thread to the GUI Thread.

In `REPLAY` mode the data shall be read from the log-files and forwarded to the GUI thread (green arrows). In this mode no data is streamed from the Data Thread.

Protocol: For control/data transfers a simple protocol shall be implemented using host-to-eStick and eStick-to-host sequences. Transfers shall be encapsulated within `<` and `>` tags. The following host-to-eStick transfers shall be supported:

- g,#p ... get a value from an I/O port (#p $\mapsto$ b|c|d)
- r,#p ... get a value from the eStick from a defined source (#p $\mapsto$ 1|2|3|4)
- s,#p,#v ... writes the value #v to an I/O port (#p $\mapsto$ b|c|d)
- d,#p,#v ... writes the value #v to a direction register (#p $\mapsto$ b|c|d)
- w,#p,#v ... write a value #v to the eStick to a defined sink (#p $\mapsto$ 1|2|3|4)

Furthermore, the eStick will response to every of these transfers with an eStick-to-host transfer as follows:

- a ... transfer acknowledge in response to `s`, `d` and `w` messages
- e ... when the message was not recognized
- a,#v ... transfer acknowledge aith value #v in response to `g` and `r` messages

Note: All command and data values are transmitted as ASCII values. Data values are integer values in the range 0... 255 transmitted as ASCII values. For example, the value 255 is transmitted as ASCII values '2', '5' and '5'. There are no spaces allowed within the messages.

Example 1 (set's some LEDs on the eStick):

```
<d,c,4>
<a>
<s,c,4>
<a>
<s,b,170>
<a>
```

Example 2 (read PORT B of the eStick – the data value is sent by the eStick):

```
<g,b>
<a,170>
```

Example 3 (the data value is generated by the eStick):

```
<r,1>
<a,42>
```

Implementation Requirements: The program may be implemented in small groups with a clear separation of task responsibilities. Furthermore, some minimal requirements need to be addressed:

- The program shall feature a file menu and a status bar.
- Values obtained with the `r` commands shall be plotted in a 2D xy-plot using the cairo library. You may opt to plot the values for #p in separate plots or in different colors in the same plot.
- The program shall provide an export functionality that allows to export the actual graphical representation as a *.svg file.
- The program shall provide `SAVE`, `SAVE AS` and `OPEN` file menu entries that allow to store/read the data to/from a TAB separated text-file. (see below)
- Try to provide a clean user interface and minimize the number of pop-up windows.

Data Files: The recorded data values (obtained with the `g` or `r` commands) within the data files shall be separated with TABs (`'\t'`) and tagged with a time-stamp provided by the `g_get_monotonic_time()` or an equivalent function. The following example show two possible realizations:

```
% UTC time   r1   r2   r3   r4
1299832264   42   21   16   80
1299832872   41   21   17   79
1299833156   41   21   16   79
1299833644   41   21   17   79
...

% UTC time   ga     gb     gc     gd
1299832264   0x00   0x00   0x04   0x00
1299833156   0x00   0x21   0x04   0x00
...
```

Provided Demonstration Programs: In order to build the host program that demonstrates the communication with an eStick, you need to install the `libusb-dev` package (Linux) or the respective library for MS-Windows (the latter may require some code modifications). The respective library is hosted on http://www.libusb.org. (tested with V0.1)

To build the program enter: `gcc -Wall -lusb -o usb_eStick usb_eStick`

Flashing the eStick: In order to flash the eStick grab the respective `ecc.hex` file and flash it with a suitable flash-tool, e.g. eStickFlashTool, Flip or dfu-programmer. E.g. to flash the eStick perform:

```
dfu-programmer at90usb162 erase
dfu-programmer at90usb162 flash ecc.hex
dfu-programmer at90usb162 start
```