TECH NOTE :: ClipX API for Linux

Version: 2019-09-09
Author: Michael Guckes, Florian Schopp, Roland Siepmann
Status: HBM: Public

Brief description

This is an instruction for using the ClipXApi.so file for including ClipX in self-made program environments in Linux-based systems. The library uses a TCP/IP connection to the port 55000 which causes that no other connection to this port can be established during a measurement. However, the web server can be used normally, because it uses another port. The ClipX API is based on a communication to the object directory of ClipX and uses its internal fifo storage. This causes a limitation of the maximal possible measurement rate of 1kHz (minimal rate 1Hz).

Note: Make sure to use the latest version of the ClipX API.

https://www.hbm.com/de/7077/clipx-praeziser-leicht-integrierbarer-messverstaerker/

Integration

In principle, the ClipX API can be integrated in each project. Since the programming was done in C ++, the integration in the same language environment is the easiest. Both for the integration in C, and also for the integration in C#, this technote contains an example.

Functions

ClipX is implemented as a C++ class. So, each ClipX device can be represented as a separate instance of this class. The class has the following methods:

```
class ClipX
{
public:
    ClipX_API ClipX();
    ClipX_API int Connect(const char* IP);
    ClipX_API int Disconnect();
    ClipX_API int SDORead(int idx, int subidx, char* res, int size);
    ClipX_API int SDOWrite(int idx, int subidx, const char* val);
    ClipX_API int startMeasurement();
    ClipX_API int AvailableLines();
    ClipX_API int ReadNextLine(double* MVLine);
    ClipX_API int ReadNextBlock(int maxreads, double* time, double* value1, double* value2, double* value3, double* value4, double* value5, double* value6);
    ClipX_API int stopMeasurement();
    ClipX_API bool GetOverFlowFlag();
    ClipX_API bool isConnected();
    ClipX_API void ClearBuffer();
    ClipX_API ~ClipX();
    bool keepAlive = true;
```

Connect

The ‘Connect’ method creates a socket and establishes the connection. The IP address is entered as input parameter. If the connection was successful, the starts a keep alive thread (more in chapter ‘KeepAlive’) method returns 0. Otherwise it returns a value below 0.

Disconnect

The ‘Disconnect’ method disconnects from the ClipX device and closes the socket. If this procedure is successful, it returns 0.

SDORead

The ‘SDORead’ method makes it possible to read from the object directory of ClipX. The inputs are the index and subindex, a char array, which stores the result, and the size, which should be read (typically: sizeof(res)). The method returns the amount of read bytes.

Note: Please refer to the ClipX manual for further information concerning the object directory and the available indices and objects.

SDOWrite

The ‘SDOWrite’ method allows to write to the object directory of ClipX. The inputs are the index, subindex and the value (Char/String) of the attribute, that should be written. If the process was successful, the method returns 0, otherwise it returns -1.

Note: Please refer to the ClipX manual for further information concerning the object directory and the available indices and objects.

startMeasurement

The ‘startMeasurement’ method initializes the measurement. Consequently, the keep alive thread is finished, the filling of the fifo storage is started and after that, the measurement thread is started, which reads all entries of the fifo every 100ms and stores them in a buffer/struct.

AvailableLines

The 'AvailableLines' method returns the amount of non-read measurement lines. This is the amount of lines, which are already read from the fifo, but not read from the internal buffer via the 'ReadNextLine' method.

ReadNextLine

The 'ReadNextLine' method reads the next available line of measurement values from the internal buffer/struct. The input parameter is expected to be a size 7 double array, which stores first the timestamp and then the six signals. As a return value, this method returns the number of lines still available.

Note: By calling the 'startMeasurement' method, the current time is read from ClipX. The time stamp of the individual measurement values consists of this time and the added tick, which is delivered by the fifo.

ReadNextBlock

The 'ReadNextBlock' method reads an amount of measurement values (predefined by the user with the variable 'maxreads') from the internal buffer/struct. The input parameter is expected to be a double pointer of each of the 7 signals (6 signals + 1 time). As a return value, this method returns the number of lines still available. If less measurement lines than 'maxreads' are available, all available measurement lines are read.

Note: By calling the 'startMeasurement' method, the current time is read from ClipX. The time stamp of the individual measurement values consists of this time and the added tick, which is delivered by the fifo.

stopMeasurement

The 'stopMeasurement' method de-initializes the measurement, ends the measurement thread and starts the keep alive thread again. If the measurement has successfully ended, this method returns 0. Is no device was connected or no measurement has been started before calling this method, it returns -1.

GetOverFlowFlag

The 'GetOverFlowFlag' method returns a Boolean, which contains the information, if some values got lost by not reading them. This overflow only concerns information about the internal buffer overflow and not about the fifo.

Note: Information about the overflow of the fifo storage can be taken from the fifo control flags (object directory).

isConnected

The 'isConnected' method returns a boolean, which signalizes if there is currently a connection to a ClipX established.

ClearBuffer

The ClearBuffer method sets the read pointer on the position of the write pointer. This causes that the next values to be read, are the ones, that are last written. The values is between are not deleted but they are skipped.

KeepAlive

The KeepAlive method is a private method, which is not available for the user. It is used internally to avoid a timeout of the connection between PC and ClipX. For this reason, each 10 seconds a KeepAliveMessage is send to the ClipX. The KeepAlive thread is startet when the measurement starts (method: startMeasurement) and ends automatically when ClipX is disconnected (method: Disconnect).

ClipX_Interface for C implementation

For C implementation the .dll contains a ClipX interface. It provides a handle which allows you to access to the full scope of functions of ClipX in C.

```
typedef struct sClipX * MHandle;
ClipX_API MHandle __stdcall ClipX_Connect(const char *);
ClipX_API void __stdcall ClipX_SDORead(MHandle m, int idx, int subidx, char* val, int size);
ClipX_API void __stdcall ClipX_SDOWrite(MHandle m, int idx, int subidx, char* val);
ClipX_API int __stdcall ClipX_startMeasurement(MHandle m);
ClipX_API int __stdcall ClipX_AvailibleLines(MHandle m);
ClipX_API int __stdcall ClipX_ReadNextLine(MHandle m,double* MVLine);
ClipX_API int __stdcall ClipX_stopMeasurement(MHandle m);
ClipX_API void __stdcall ClipX_Disconnect(MHandle m);
```

Calling methods is just the same as in C++, except for the handle input at each method.

Example

Both, the ‘testmain.cpp’ and the ‘testmainint.cpp’ are examples for an integration of the ClipX in a project. The first one uses directly the class ClipX, whereas the second one uses the C interface.

```
#include "ClipX.h"
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

int main(){
    ClipX  dev = ClipX();
    char buf[100];
    dev.Connect("172.21.64.181");
    dev.SDOWrite(0x4428, 8, "100");
    dev.SDORead(0x4428, 8, buf, 100);
    printf("Fifo Fill Rate: %s \n", buf);
    dev.startMeasurement();
    printf("Measurement Started");
    for (int i = 0; i < 2; i++) {
        printf("Lines: %i", dev.AvailableLines());
                //std::cout << dev.AvailibleLines() ;
                double values[7];
                do {
                    if (dev.ReadNextLine(values) > 0)
                        printf("Time: \t %f \t %f \t %f \t %f \t %f \t %f \t %f \n", values[0], values[1], values[2], values[3], values[4], values[5], values[6]);
                } while (dev.AvailableLines() > 0);
                sleep(1);
            }
    dev.stopMeasurement();
    dev.Disconnect();
}
```

Disclaimer

These examples are for illustrative purposes only. They cannot be used as the basis for any warranty or liability claims.