Application Note: Digital Input

The purpose of this application note is to describe usage of the network-based digital input function with the RCB-LVDS module.

Introduction

The RCB-LVDS module is often used in systems in which synchronization or time stamps with respect to other system components is necessary. In wired systems, it has been customary to directly wire additional digital indicators into the data acquisition module. This presented no great burden to the experimenter, since a few more wires or cables were of little consequence.

The new degree of freedom available with the RCB-LVDS in which the test subject is no longer tethered by data acquisition cabling also presents a new difficulty: synchronization with other system components. It is still possible, of course, for the RCB-LVDS module to potentially accept auxiliary wired inputs to the RHD2000, but doing so erases the purpose of using the wireless data acquisition.

This application note describes the network-based digital input function available with the RCB-LVDS module that can be used to help synchronize data acquisition with other system components.

Other methods of time synchronization are under development. In particular we are experimenting with NTP time servers.

Use Case: Synchronization with External Devices

Consider the system in Fig. 1, consisting of a test subject, and a source of external sensory stimulation. The objective of the experiment is to collect electrophysiological data from the test subject in response to the sensory stimulation.

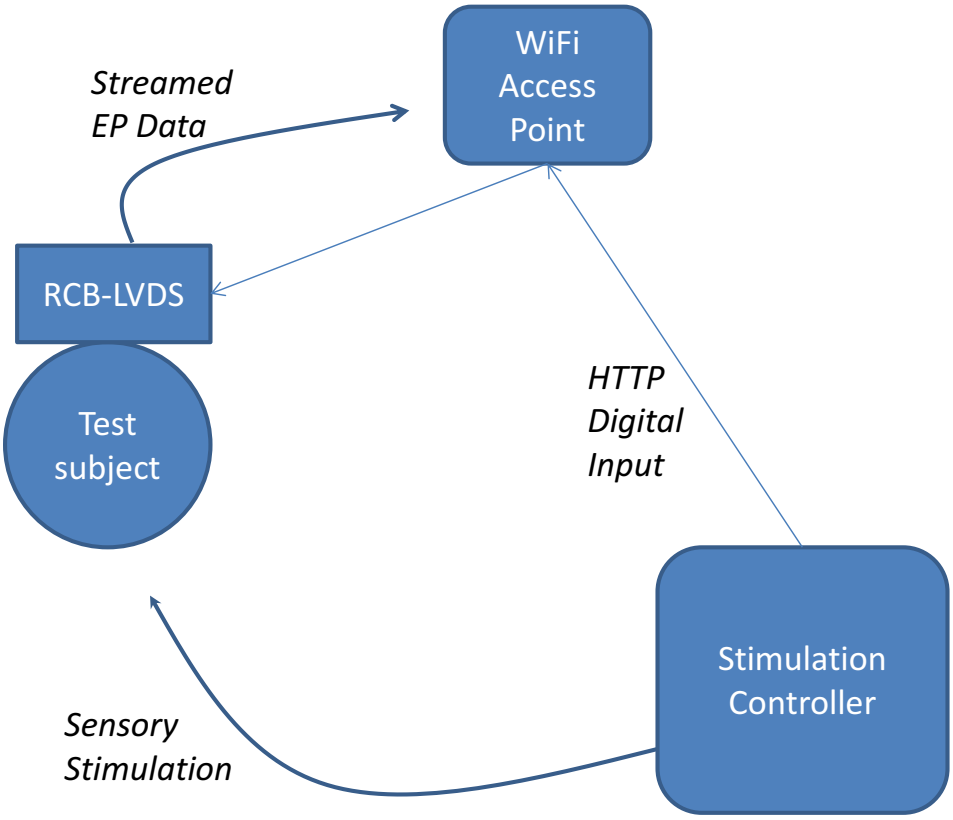

Fig. 1. Example test setup with external sensory stimulation.

The sensory stimulation is under the control of a computer. The experiment management software on the control computer controls the stimulation sequence and pattern. When the stimulation changes, the fact can be registered in the acquisition data stream using the Digital Input function of the RCB-LVDS module.

There are sixteen (16) digital input bits that are under user control. The user can manipulate any single or combination of bits with each HTTP post command. The digital input bit states appear in the GUI, and in the recorded data. In this example, we will choose digital input number 5 to indicate a change in stimulus.

The digital input manipulation commands are documented in the RCB-LVDS API. We use Set and Clear bit commands.

Assume that the RCB-LVDS module IP number is 192.168.0.93.

Pseudo code might look like this:

Control Stimulus "on"
Set Digital In 5
Control Stimulus "off"
Clear Digital In 5

In Matlab® the command would look like this:

```
% Control the Stimulus "on" with a Matlab Command

% Set Digital In 5
[s, status] = urlread('http://192.168.1.93', 'post', ...
        {'__SL_P_UDI', 'S5'});

% Control the Stimulus "off" with a Matlab Command

% Clear Digital In 5
[s, status] = urlread('http://192.168.1.93', 'post', ...
        {'__SL_P_UDI', 'C5'});
```

In Qt the command might look like this:

```
// RCB-LVDS Control Object
Http *controlHttp = new Http;

// Tell controlHttp object the IP address of our RCB-LVDS module
controlHttp->setRcblvdsIP("192.168.0.93");

// Control the Stimulus "on"
…
// Set Digital In 5
controlHttp->setDigInBit(5);

// Control the Stimulus "off"
…
// Clear Digital In 5

controlHttp->clearDigInBit(5);
```

The Qt example code is part of the Qt GUI. It will be released Open Source when finished.

Use Case: Multiple Stimulation Modes

Suppose the experiment involves two or more stimulation modes or phases. Multiple Digital Input bits can be used to indicate these phase changes. For example, two bits can be used to indicate four separate states. Two new methods could be implemented in http.cpp:

```
// Set some bits without affecting others
// mask is hexadecimal
void Http::setDigInBits(int mask)
{
    qDebug() << "Setting Bits " << chan << postToken("__SL_P_UDI", "O" +
QString::number(mask,16));
// NOTE: The command is capital letter “o”, as in “OR”, not a zero!

}

// Clear some bits without affecting others
// mask is hexadecimal
void Http::clrDigInBits(int mask)
{
    qDebug() << "Clearing Bits " << chan << postToken("__SL_P_UDI", "A" +
QString::number(~chan,16));

}
```

Another way this can be used is the case in which two or more separate stimuli are generated by separate means. In this case, each stimulator can control a separate Digital Input bit without interfering with the others.

Implementation Details

There are two parameters of interest to the user regarding the digital input bits: resolution, and latency. **Resolution** is the effective sample rate of the digital input bits, and is always less than the system sample rate. **Latency** is the time lag between when the http POST command is issued from the user's control application to when it actually appears in the data stream. These concepts are discussed separately.

Resolution

The state of the Digital Input bits is reported once for each data packet from the RCB-LVDS. Thus, the time resolution is variable depending on number of enabled channels and sample rate. Regardless of the time resolution, the GUI produces digital input data files with the same time grid as the analog channels, the state of the Digital Input being held constant until a change is registered. Listed in the table below are several examples, and the associated time resolution of the Digital Input bits.

Table I. Digital Input sample rates (Resolution) for various system sample rates and number of enabled channels.

Sample Rate	Number of Enabled Channels	Packet Time, Resolution of Digital Inputs
20 kHz	32	1 ms
15 kHz	32	1.5 ms
10 kHz	32	2.2 ms
30 kHz	16	1.2 ms
25 kHz	16	1.5 ms
20 kHz	16	2.0 ms
10 kHz	16	3.8 ms

Because of the time resolution, multiple commands to set, clear, or toggle a Digital Input within a single packet period will not be reflected to the output, and only the last state will appear.

Latency

When an http POST command is issued by the user's application, it must traverse the network layers in the user's computer, routed through the network nodes, and finally arrive at the RCB-LVDS. The real time delay from application to RCB-LVDS is called **latency**. This latency depends heavily on the amount of traffic on the network, the quality of the network devices (routers, etc.), and other factors. These inherent delays in the control computer and network are sometimes variable. Testing with a specific experiment configuration is recommended. Latencies of one to two milliseconds have been observed, up to as much as several tens of milliseconds.

Conclusion

We have shown the usage of the network-based digital input function with the RCB-LVDS module to wirelessly coordinate time stamps with respect to other system components. This application note describes the network-based digital input function available with the RCB-LVDS module that can be used to help synchronize data acquisition with other system components. Considerable flexibility is provided to the user, and the user is referred to the API for more information.

Appendix

http.cpp from GUI (fragment)

The following code snippet is from the RCB-LVDS GUI. See the code base for the complete class. The RCB-LVDS GUI is open-source.

```
/*
 * http.cpp
 *
 * This class handles http communication with RCB-LVDS module.
 *
 */

#include "http.h"
#include <QObject>
#include <QUrl>
#include <QUrlQuery>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QThread>
#include <QHostInfo>
#include <QNetworkInterface>
#include <QList>
#include <QProcess>
#include <QTimer>

#include <QVBoxLayout>
#include <QPushButton>
#include <QDialogButtonBox>
#include <QDialog>
#include <QLineEdit>

#include <QMessageBox>

#include <vector>
using namespace std;


void Http::setRcblvdsIP(const QString &s)
{
    if (rcblvdsIP != s)
    {
        rcblvdsIP = s;
        qDebug() << "rcblvdsIP changed to :" << rcblvdsIP << endl;

        isConnectedToModule = 0;    // reset ping counter because of new IP

        emit triggerPings(isConnectedToModule);
    }
}
```

```
Http::Http(QObject *parent) : QObject(parent)
{

    networkRequest.setHeader(QNetworkRequest::ContentTypeHeader,
                                "application/x-www-form-urlencoded");
    networkManager = new QNetworkAccessManager(this);

    connect(networkManager, SIGNAL(finished(QNetworkReply*)), &eventLoop,
SLOT(quit()));

    qDebug() << "HttpPost object created";

    /*
     * Find all the active local network interfaces for user to choose from.
     */

    QList<QHostAddress> list = QNetworkInterface::allAddresses();

    qDebug() << list;

    isConnectedToModule = 0;
    emit triggerPings(isConnectedToModule);


    // Set up periodic ping of module to know connectivity
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(pingRcblvdsIP()));
    timer->start(5000);
}



void Http::setDigInBit(int chan)
{
    qDebug() << "Setting Bit " << chan << postToken("__SL_P_UDI", "S" +
QString::number(chan,10));

}

void Http::clrDigInBit(int chan)
{
    qDebug() << "Clearing Bit " << chan << postToken("__SL_P_UDI", "C" +
QString::number(chan,10));

}



QString Http::postToken(const QString &token, const QString &value)
// Post a Token to the rcblvds
{

    if (isConnectedToModule)
```

```
    {

        // Setup the webservice url
        QUrl serviceUrl = QUrl("http://" +rcblvdsIP);
        QByteArray postData;
        QNetworkReply *reply;
        QUrl params;
        QUrlQuery query;
        query.addQueryItem(token, value);

        params.setQuery(query);

        postData = params.toEncoded(QUrl::RemoveFragment);

        // Check to see if previous call has finished
        if (eventLoop.isRunning())
            eventLoop.exit(0);

        reply = networkManager->post(networkRequest,postData);
        return reply->readAll();
    }
    else
        return "No Module";


}
```