McsUsbNet.dll for MCS STG devices

Introduction

This DLL provides the .NET interface to MCS devices. This documentation is about the STG200x & STG400x series stimulus generators

System requirements

The DLL can be used with any .NET compatible language.

It requires the Microsoft C runtime libraries to be installed:

- MSVCR100.dll
- MSVCP100.dll

It also requires the USB driver and/or the MC_Card driver to be installed.

The simplest way to achieve this is to install the latest MC_Rack setup.

All examples assume that the Mcs.Usb namespace is loaded:

```
using namespace Mcs.Usb;
```

Include the file McsUsbNet.dll into the references of your project.

STG200x & STG400x Series Stimulus Generators

Introduction

The STG200x & STG400x Series Stimulus Generators have two distinct modes of operation, the Download mode and the Streaming mode.

Download mode

The Download mode is the "classic" mode of operation, as used by the MC Stimulus software. In this mode, one or multiple waveforms are defined in PC memory and downloaded to the STG. The waveforms are stored in STG device onboard memory and can be sent to the analog and sync outputs once or multiple times. The STG can operate independantly from the PC (without computer connection) after the download. Output is triggered either by the front panel start/stop button, the digital trigger inputs or under software control.

In the Download mode, there are up to eight independent triggers available (depending on the device). The user can assign each of the analog outputs and sync (digital) outputs to any of the triggers.

The analog output waveform is stored sample by sample in the STG memory. To reduce memory usage, this data can be compressed: whenever a given output value is to be held for more than one sample period, it has only to be given once. The user can define the number of sample periods for that a pattern should remain active. Compression is done for each channel independantly of the others, thus the algorithm to compress the data is very easy to implement.

A new feature of the Download mode is the segmentation of the STG memory. The onboard memory can be devided into up to 100 segments. Each segment can hold its own waveform pattern. Under software control, the user can switch between the defined segments within milliseconds. Another option is to use the four trigger inputs to select between four predefined segments. This option is accessible from the MC_Stimulus Software as the "Multi-File mode", and can start each of up to four defined waveforms within microseconds. This feature allows a predefinied flexible response (feedback) to recorded data.

UsbNetDll::CStg200xDownloadNet is the class for using the STG in download mode.

Memory Layout and Trigger Setup

The class to be used for the Download mode is UsbNetDll::CStg200xDownloadNet, which is derived from UsbNetDll::CStg200xBasicNet. You can add a poll handler delegate (UsbNetDll::OnStg200xPollStatus) to the constructor UsbNetDll::CStg200xDownloadNet::CStg200xDownloadNet.

The connection to a STG is established by UsbNetDll::CMcsUsbNet::Connect. When this function is called without argument, the first STG found on the USB bus is used. When more than one STG is connected, this function can take for example the serial number of the required STG as argument.

```
CStg200xDownloadNet device = new CStg200xDownloadNet();
device.Connect();
```

To use the Download mode, the memory layout of the STG200x can be set up, if the default is not sufficient. The total amount of memory available in the STG is obtained by the UsbNetDll::CStg200xDownloadNet::GetTotalMemory call. With UsbNetDll::CStg200xDownloadNet::SendSegmentDefine the segment sizes are assigned.

```
uint32_t memory = device.GetTotalMemory();  // obtain total memory available

uint[] segmentmemory = new uint[2];     // each segments has half of total memory
segmentmemory[0] = memory / 2;
segmentmemory[1] = memory / 2;
device.SendSegmentDefine(segmentmemory);// setup the STG
```

Next, for each segment, one has to assign the amount of memory to be used for each channel and sync output. This is done by UsbNetDll::CStg200xDownloadBasicNet::SetCapacity. Its arguments contain a list of memory sizes, with one entry per channel and one entry per sync output. Again, the total memory assigned to the channels and sync outputs must not exceed the memory assigned to the segment.

```
uint32_t nchannels = device.GetNumberOfAnalogChannels();
uint32_t nsync = device.GetNumberOfSyncoutChannels();
uint[] channel_cap = new uint[nchannels];
uint[] syncout_cap = new uint[nsync];
for (int i = 0; i < 2; i++)                             // for each segment
{
    device.SendSegmentSelect((uint32_t)i);                    // switch to segment
    uint32_t segment_mem = device.GetMemory();                // get memory available in this segment
    for(int j = 0;j < nchannels;j++)
    {
        channel_cap[j] = segment_mem/(nchannels+nsync); // devide memory amount to all channels
    }
    for(int j = 0;j < nsync;j++)
    {
        syncout_cap[j] = segment_mem/(nchannels+nsync); // and all sync outs.
    }

    device.SetCapacity(channel_cap, syncout_cap);       // define memory for current segment
}
```

Before the STG can start, the trigger has to be configured. This is done by the UsbNetDll::CStg200xDownloadNet::SetupTrigger call. Its arguments are a list of channelmaps, syncoutmaps and repeats, one for each of the four available triggers. channelmap is a bitmap, each bit representing one of the available channels. To assign channel 1 and syncout 1 to trigger 1 and channel 3 to trigger 2 use:

```
uint32_t TriggerInputs = device.GetNumberOfTriggerInputs();
uint[] channelmap = new uint[TriggerInputs];
uint[] syncoutmap = new uint[TriggerInputs];
uint[] repeat = new uint[TriggerInputs];
for (int i = 0; i < TriggerInputs; i++)
{
    channelmap[i] = 0;
    syncoutmap[i] = 0;
    repeat[i] = 0;
}
// Trigger 0
channelmap[0] = 1; // Channel 1
syncoutmap[0] = 1; // Syncout 1
repeat[0] = 0; // forever

// Trigger 1
channelmap[1] = 4; // Channel 3

device.SetupTrigger(channelmap, syncoutmap, repeat);
```

For the STG400x series you have to set the output mode of the channels. UsbNetDll::CStg200xDownloadNet::SetVoltageMode interprets the values as voltages. UsbNetDll::CStg200xDownloadNet::SetCurrentMode as currents.

```
// Only meaningfull for STG400x
device.SetVoltageMode();
```

For each segment, data can be sent to each of the defined channels and sync outputs using the UsbNetDll::CStg200xDownloadNet::SendChannelData and UsbNetDll::CStg200xDownloadNet::SendSyncData calls. channeldata and syncdata are a list of analog and digital samples as a list of two byte values (unsigned short). Multiple calls to UsbNetDll::CStg200xDownloadNet::SendChannelData and UsbNetDll::CStg200xDownloadNet::SendSyncData to the same channel append data to that channel.

If the Multi-File mode of the STG is enabled using the UsbNetDll::CStg200xDownloadNet::EnableMultiFileMode call, the four trigger inputs are used to switch between four segments. A hardware trigger signal (TTL) on trigger input 1 selects the first segment and starts all pulses in this segment. Thus with the Multi-File mode, one can predefine four stimulus patterns and switch between them without a connection to the PC.

The STG200x series has an analog resolution of 13 bits, thus the analog data contains the information in bits 0 to 12 of each sample. Bits 13 to 15 have to be 0.

```
int DACResolution = device.GetDACResolution();

// Data for Channel 0
{
    device.ClearChannelData(0);

    double factor = 0.1;
    const int l = 1000;
        ushort[] pData = new ushort[l];
        Uint64_t[] tData = new Uint64_t[l];
        for (int i = 0; i < l; i++)
        {
            // calculate Sin-Wave
            double sin = factor * (Math.Pow(2, DACResolution - 1) - 1.0) *
                Math.Sin(2.0 * (double)i * Math.PI / (double)l);

            // calculate sign
            pData[i] = sin >= 0 ? (ushort)sin : (ushort)((int)Math.Abs(sin) +
                (int)Math.Pow(2, DACResolution - 1));

            tData[i] = (Uint64_t)20; // duration in µs
        }
        device.SendChannelData(0, pData, tData);
}

// Data for Channel 3
{
    device.ClearChannelData(2);

    double factor = 0.1;

    const int l = 700;
    // without compression
    ushort[] pData = new ushort[l];
    Uint64_t[] tData = new Uint64_t[l];
    for (int i = 0; i < l; i++)
    {
        // calculate Sin-Wave
        double sin = factor * (Math.Pow(2, DACResolution - 1) - 1.0) *
            Math.Sin(2.0 * (double)i * Math.PI / (double)l);

        // calculate sign
        pData[i] = sin >= 0 ? (ushort)sin : (ushort)((int)Math.Abs(sin) +
            (int)Math.Pow(2, DACResolution - 1));

        tData[i] = (Uint64_t)20; // duration in µs
    }
    device.SendChannelData(2, pData, tData);
}

// Data for Sync 0
{
    device.ClearSyncData(0);
    ushort[] pData = new ushort[1000];
    Uint64_t[] tData = new Uint64_t[1000];
    for (int i = 0; i < 1000; i++)
    {
        pData[i] = (ushort)(i&1);
        tData[i] = 20;
    }
    device.SendSyncData(0, pData, tData);
}
```

Start the trigger by pushing the front button or by software

```
// Start Trigger 1 and 2
device.SendStart(1 + 2); // Trigger 1 und 2
```

see the StgDownloadExampleNet in the example directory.

Generated on Fri Jan 9 2015 13:58:50 for McsUsbNet.dll for STG by dox *1.7.6.1*

- Main Page

- Namespaces
- Classes

- McsUsbNet.dll for MCS STG devices
- STG200x & STG400x Series Stimulus Generators

Streaming mode

The other mode of operation is the Streaming mode. Here the analog output is sent to the STG device in "real time". The PC has to be connected to the STG all the time. The data that is sent to the analog output is downloaded from the PC to the STG on the fly.

The Streaming mode is useful for applications where flexible feedback is needed as well for applications where very long waveforms which are not repeated (such as white noise) are used.

The Streaming mode works by use of two ring buffers which hold data. One is in PC memory and managed by the DLL, and one is in on-board STG memory. Data is transfered from PC memory to the STG via the USB bus in time slices of one millisecond.

The user can define both the size of the ring buffer in DLL memory and in the STG memory. Once the Streaming mode is started, the STG request data from the PC. The data rate from PC to STG is variable and controlled by the STG. The STG request data from the PC at a rate to keep its internal ringbuffer at about half full.

It is the responsibility of the user to keep the ring buffer in the memory of the PC filled, so the DLL can supply sufficient data to the STG. To do so, the Windows DLL allows to define a "callback" function which is called whenever new data is needed, or more precise, as soon as the ring buffer in the memory of the PC falls below the user defined threshold.

Small buffers have the advantage of a low latency between data generation in the callback funtion and its output as a analog signal from the STG. However for low latency to work, the user-written callback function has to be fast and to produce a steady flow of data.

In the Streaming mode, all triggers are available as well. Each of the eight analog and sync outputs can be assigned to one of the triggers.

The output rate is user defined with a maximum of 50 kHz

UsbNetDll::CStg200xStreamingNet is the class for using the STG in streaming mode.

Memory Layout and Trigger Setup

With the constructor for UsbNetDll::CStg200xStreamingNet::CStg200xStreamingNet, the name of the callback function for the data handler is provided. The data handler function is called automatically, whenever the STG needs new data. This data is first written to a ring buffer in the memory of the PC. The size for this ring buffer is defined as first argument in the constructor. The user provided delegate gets the trigger number which needs new data as argument

```
CStg200xStreamingNet device = new CStg200xStreamingNet(10000, dataHandler, errorHandler);
```

The callback funtion, which is defined in the constructor, is called whenever the STG needs new data for a trigger, or more precise, whenever the ring buffer in PC memory falls below the defined threshold.

The user can query the amount of space available for queuing by use of the UsbNetDll::CStg200xStreamingNet::GetDataQueueSpace call. Its return value is the number of samples that can be send to the STG.

User code is required to fill an array analog and sync out data, sample by sample for up to the maximum number of samples as obtained by UsbNetDll::CStg200xStreamingNet::GetDataQueueSpace or UsbNetDll::CStg200xStreamingNet::GetSyncoutQueueSpace.

The values for the analog outputs are 16 bits signed integers. The lower bits are trunctated according to the resolution of the

STG. This behaviour is different to the behaviour in download mode.

Note: Compression as described in the download mode can NOT be used for the streaming mode.

The new data is sent to the STG by using the UsbNetDll::CStg200xStreamingNet::EnqueueData call.

```
void dataHandler(uint32_t trigger)
{
    double factor = 1;

    if (trigger == 0) // Callback for Trigger 1
    {
        {// Handle Channel 1
            uint32_t channel = 0;
            for (; ; )
            {
                uint32_t space = device.GetDataQueueSpace(channel);
                if (space < 1000)
                    break;

                short[] data = new short[1000];
                for (int i = 0; i < 1000; i++)
                {
                    // Calc Sin-Wave (16 bits) lower bits will be removed according resolution
                    double sin = factor * (Math.Pow(2, 16 - 1) - 1.0) *
                        Math.Sin(2.0 * (double)i * Math.PI / (double)1000);

                    data[i] = (short)sin;
                }
                uint32_t enqueued = device.EnqueueData(channel, data);
            }
        }
        {// Handle Channel 3
            uint32_t channel = 2;
            for (; ; )
            {
                uint32_t space = device.GetDataQueueSpace(channel);
                if (space < 700)
                    break;
                short[] data = new short[700];
                for (int i = 0; i < 700; i++)
                {
                    // Calc Sin-Wave (16 bits) lower bits will be removed according resolution
                    double sin = factor * (Math.Pow(2, 16 - 1) - 1.0) *
                        Math.Sin(2.0 * (double)i * Math.PI / (double)700);

                    data[i] = (short)sin;
                }
                uint32_t enqueued = device.EnqueueData(channel, data);
            }
        }
        {// Handle Syncout 1
            uint32_t channel = 0;
            for (; ; )
            {
                uint32_t space = device.GetSyncoutQueueSpace(channel);
                if (space < 1000)
                    break;
                ushort[] data = new ushort[1000];
                for (int i = 0; i < 1000; i++)
                    data[i] = (ushort)(i & 1);
                uint32_t enqueued = device.EnqueueSyncout(channel, data);
            }
        }
    }
}

void errorHandler()
{
}
```

The connection to a STG is established by UsbNetDll::CMcsUsbNet::Connect. When this function is called without argument, the first STG found on the USB bus is used. When more than one STG is connected, this function can take the serial number of the required STG as an argument.

```
device.Connect();
```

With enabling or disabling the continous mode it can be selected how the STG handles an "out of data" situation.

When UsbNetDll::CStg200xStreamingNet::EnableContinousMode is used, the STG does not stop when it runs out of data, but it keeps running and sends a zero voltage to its outputs.

When UsbNetDll::CStg200xStreamingNet::DisableContinousMode is used, the STG stops when it runs out of data. It has to be retriggered to resume the output.

```
device.EnableContinousMode();
```

UsbNetDll::CStg200xStreamingNet::SetOutputRate is used to set the sampling rate.

```
device.SetOutputRate(50000);
```

To use the Streaming mode, the memory layout of the STG has to be set up. To total amount of memory available in the STG is obtained by the UsbNetDll::CStg200xStreamingNet::GetTotalMemory call.

This memory can be assigned to four ring buffers (one per trigger) which buffer the data received from the PC via USB cable. This is done with the CStg200xStreaming::SetCapacity call. The total amount of memory must not exceed the total memory size as obtained by UsbNetDll::CStg200xStreamingNet::GetTotalMemory.

This internal ring buffer is crucial for proper operation of the Streaming mode. The size of the ring buffer determines the latency of the Streaming mode. The firmware of the STG requests data from the PC in order to keep the ring buffer about half full. Thus the average latency is:

```
latency = (ringbuffersize in bytes/4) / output rate
```

If the ring buffer size is too big, the latency of the STG might be too long. If the ring buffer size is too low, an overflow or underflow of data in the STG ringbuffer might occur, resulting in data jumps of the output signals or the "out of data" situation described erlier.

The following example divides the total memory equally amoung the four triggers:

```
uint32_t dwMemory = device.GetTotalMemory();            // obtain total memory available

uint32_t ntrigger = device.GetNumberOfTriggerInputs();  // obtain number of triggers in this STG
uint[] stg_triggercapacity = new uint[ntrigger];
for (int i = 0; i < ntrigger; i++)
    stg_triggercapacity[i] = dwMemory / ntrigger;
device.SetCapacity(stg_triggercapacity);            // setup the STG
```

or fixed memory sizes:

```
uint32_t ntrigger = device.GetNumberOfTriggerInputs();  // obtain number of triggers in this STG
uuint[] stg_triggercapacity = new uint[ntrigger];
for(int i = 0;i < ntrigger;i++)
    stg_triggercapacity[i] = 50000;
device.SetCapacity(stg_triggercapacity);
```

Before the STG can start, the trigger has to be configured. This is done by the UsbNetDll::CStg200xStreamingNet::SetupTrigger call. Its arguments are a list of channelmaps, syncoutmaps, digoutmap, autostart and callback_threshold, with one entry for each of the available triggers. channelmap is a bitmap, each bit representing one of the available channels. To assign channel 1 and 3 and syncout 1 to trigger 1 use:

```
uint32_t ntrigger = device.GetNumberOfTriggerInputs();  // obtain number of triggers in this STG
uint[] channelmap = new uint[ntrigger];
uint[] syncoutmap = new uint[ntrigger];
```

```
uint[] digoutmap = new uint[ntrigger];
uint[] autostart = new uint[ntrigger];
uint[] callback_threshold = new uint[ntrigger];
for (int i = 0; i < ntrigger; i++)
{
    channelmap[i] = 0;
    syncoutmap[i] = 0;
    digoutmap[i] = 0;
    autostart[i] = 0;
    callback_threshold[i] = 0;
}

channelmap[0] = 0x1 + 0x4; // Channel 1 und Channel 3 to Trigger 1
syncoutmap[0] = 0x1; // Syncout 1 to Trigger 1
autostart[0] = 1;
callback_threshold[0] = 50; // 50% of buffer size

device.SetupTrigger(channelmap, syncoutmap, digoutmap, autostart, callback_threshold);

device.StartLoop();
System.Threading.Thread.Sleep(1000); // Give StartLoop some time
```

Start Trigger by pushing the front button or by Software

```
device.SendStart(1);
```

see the StgStreamingExampleNet in the example directory.

Generated on Fri Jan 9 2015 13:58:50 for McsUsbNet.dll for STG by dox *1.7.6.1*