

g.tec medical engineering GmbH  
4521 Schiedlberg, Sierningstrasse 14, Austria  
Tel.: (43)-7251-22240-12  
Fax: (43)-7251-22240-39  
[office@gtec.at](mailto:office@gtec.at), <http://www.gtec.at>

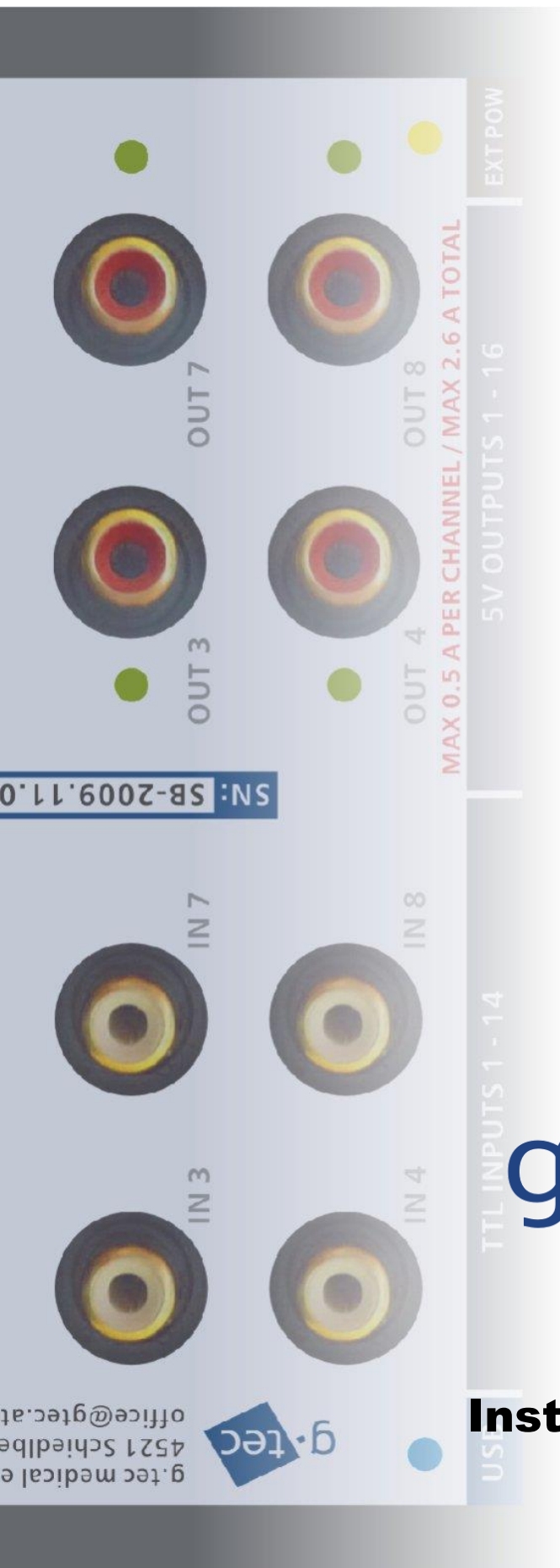


# g·STIMbox

USB STIMULATOR DIGITAL I/O BOX

## Instructions for use V1.16.00

Copyright 2016 g.tec medical engineering GmbH



## How to contact g.tec:



++43-7251-22240-12

**Phone**



++43-7251-22240-39

**Fax**



g.tec medical engineering GmbH  
Sierningstrasse 14, 4521 Schiedlberg, Austria

**Mail**



<http://www.gtec.at>

**Web**



[office@gtec.at](mailto:office@gtec.at)

**E-mail**

AT/CA01/RC000989-00

**ÖBIG Reg. numbers**

AT/CA01/M0003062-00

11-467

**UMDNS code**

# Contents

<b>1</b>	<b>To the Reader .....</b>	<b>5</b>
<b>2</b>	<b>Release Notes .....</b>	<b>6</b>
<b>3</b>	<b>Related Products .....</b>	<b>7</b>
<b>4</b>	<b>Conventions .....</b>	<b>8</b>
<b>5</b>	<b>Installation and Configuration .....</b>	<b>9</b>
5.1	Hardware and Software Requirements .....	10
5.1.1	Hardware Requirements .....	10
5.1.2	Software Requirements .....	10
5.2	Installation of g.STIMbox .....	11
5.3	Determining the COM-port of the g.STIMbox .....	12
5.4	Files on your Computer .....	13
<b>6</b>	<b>Introduction to g.STIMbox .....</b>	<b>14</b>
<b>7</b>	<b>Explanation of switches, connectors and LEDs .....</b>	<b>17</b>
7.1	Sockets and connectors on the front side .....	17
7.2	Sockets, connectors and LEDs on the top side .....	17
7.3	Marking on the top side .....	18
<b>8</b>	<b>Safe operation of the g.STIMbox .....</b>	<b>19</b>
8.1	Instructions for use .....	19
8.2	Switching off and storage of g.STIMbox .....	19
8.3	Using the g.STIMbox with Simulink .....	20
8.3.1	Quickstart .....	20
8.3.2	Demos .....	25
8.3.3	Block Reference .....	32
8.4	Using the g.STIMbox with the MATLAB-API .....	35
8.4.1	Demos .....	35
8.4.2	Function Reference .....	41
8.4.3	Error Handling .....	57
8.5	Using the g.STIMbox with the C-API .....	58
8.5.1	Usage .....	58
8.5.2	Demos .....	60
8.5.3	Function Reference .....	61
8.5.4	Error codes .....	79
8.6	gSTIMboxEPmaster .....	80
<b>9</b>	<b>General notes .....</b>	<b>81</b>
	Transportation and storage conditions .....	81
	Location details .....	81
	Waste disposal details .....	81
	Cleaning .....	81

<b>10</b>	<b>Technical specifications.....</b>	<b>82</b>
10.1	g.STIMbox.....	82
<b>11</b>	<b>PIN assignment .....</b>	<b>83</b>

# 1 To the Reader

Welcome to the medical and electrical engineering world of g.tec!

Discover the only professional biomedical signal processing platform under MATLAB and Simulink. Your ingenuity finds the appropriate tools in the g.tec elements and systems.

Choose and combine flexibly the elements for biosignal amplification, signal processing and stimulation to perform even real-time feedback.

Our team is prepared to find the better solution for your needs.

Take advantage of our experience!

Dr. Christoph Guger

Dr. Guenter Edlinger

## **Researcher and Developer**

Reduce development time for sophisticated real-time applications from month to hours.

Integrate g.tec's open platform seamlessly into your processing system.

g.tec's rapid prototyping environment encourages your creativity.

## **Scientist**

Open new research fields with amazing feedback experiments.

Process your EEG/ECG/EMG/EOG data with g.tec's biosignal analyzing tools.

Concentrate on your core problems when relying on g.tec's new software features like ICA, AAR or online Hjorth's source derivation.

## **Study design and data analysis**

You are planning an experimental study in the field of brain or life sciences? We can offer consultation in experimental planning, hardware and software selection and can even do the measurements for you. If you have already collected EEG/ECG/EMG/EOG, g.tec can analyze the data, can do feature extraction and can prepare the results ready for publication.

## 2 Release Notes

Release notes bring to your attention new features of, and changes to, the g.STIMbox software when upgrading to a newer version.

### *New features*

None.

### *Changes*

- Supports Windows 10, 64-bit; Windows 7 and 32-bit support retired.
- Support of MATLAB 2015a

### **3 Related Products**

g.tec provides several biosignal analysis elements that are especially relevant to the kinds of tasks you perform with the g.STIMbox.

For more detailed information on any of our elements, up-dates or new extensions please visit our homepage [www.gtec.at](http://www.gtec.at) or just send us an email to [office@gtec.at](mailto:office@gtec.at).

## 4 Conventions

Item	Format	Example
MATLAB code	Courier	to start Simulink, type simulink
String variables	<i>Courier italics</i>	set(P_C,'PropertyName',...)
Menu items	<b>Boldface</b>	Select <b>Save</b> from the <b>File</b> menu



## 5 Installation and Configuration

This chapter includes the following sections:

[Hardware and Software Requirements](#)

[Installation from a CD](#)

[Determining the COM-port the g.STIMbox is connected to](#)

[Files on your Computer](#)

## 5.1 Hardware and Software Requirements

### 5.1.1 HARDWARE REQUIREMENTS

g.STIMbox software requires a PC compatible desktop or notebook workstation running Microsoft Windows.

The table below lists optimal settings:

Hardware	Properties
CPU	Pentium working at 2000 MHz
Harddisk	20-30 GB
RAM	1024 MB
USB 2.0 high speed port	One free USB port for each g.STIMbox

### 5.1.2 SOFTWARE REQUIREMENTS

The g.STIMbox requires the installation of the g.STIMbox driver which is shipped with the device and MATLAB. Make sure that the MATLAB installation works correctly before installing the g.USBamp software. Depending on your Windows operating system administrator rights might be necessary for the installation.

**NOTE:** It is highly recommended to turn off the User Account Control of Windows 7 operating system when using the g.STIMbox.

Software	Version
g.STIMbox driver	1.16.00
MATLAB*	R2015a
Simulink*	R2015a
Microsoft Visual C++ Redistributable Package	2015
Microsoft Visual Studio (for C-API programming)	2015
Windows	Windows 10 Pro English Threshold 2 Win32 or Win64
PDF-Reader	Adobe Acrobat Reader DC 2015

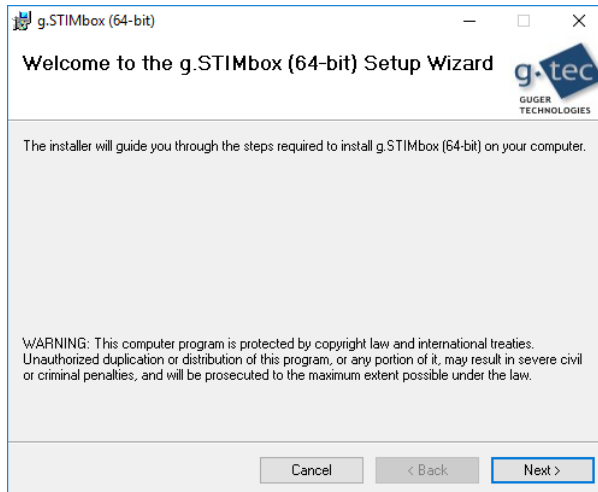
\* Products of The MathWorks Inc., Natick, USA

## 5.2 Installation of g.STIMbox

**NOTE:** Do not connect g.STIMbox hardware before finishing the software installation!

The installation of g.STIMbox software consists of two steps:

1. Insert the g.tec product CD into the CD-drive and change to the g.STIMbox directory of your CD-drive and double-click the `setup.exe` file. This will open the installation welcome dialog.



Then just follow the instructions on the screen.

During this installation the driver software for g.STIMbox hardware will be installed.

2. To set the MATLAB path to the g.STIMbox folder click **Set Path** in the **Home** tab of the command ribbon under the **Environment** group.

In the **Set Path** dialog click **Add with Subfolders...** and browse to the folder

`C:\Program Files\gtec\gSTIMbox`

and click **OK**.

Then click again **Add with Subfolders...** and browse to the folder

`C:\Users\<username>\gtec\gSTIMbox`

and click **OK**.

Click the **Save** button in the **Set Path** dialog and close it with the **Close** button. This finishes g.STIMbox installation.

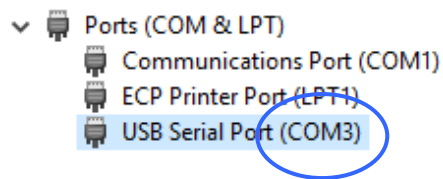
Now connect g.STIMbox to the PC or notebook and the driver software will be installed automatically.

### 5.3 Determining the COM-port of the g.STIMbox

To find out the COM-port the g.STIMbox is connected to perform the following steps

**Step 1:** Connect the Mini-USB cable connector to the USB socket of g.STIMbox and the PC USB connector of the cable to a USB connector of the PC.

**Step 2:** Open the Windows device manager to check which COM-port is used. This port number is needed for operating the g.STIMbox.



The correct operation of g.STIMbox is indicated by a green LED on the top side. If the LED is off, please check the USB connection.

## 5.4 Files on your Computer

**g.STIMbox files** - are stored under

```
C:\Program Files\gtec\gSTIMbox
```

and its subdirectories. Also example scripts and programs are situated there. Please refer to the subsequent Sections of this document for further information.

**Example Simulink models** are stored under

```
C:\Users\<username>\gtec\gSTIMbox\Examples
```

Please refer to the subsequent Sections of this document for further information.

**g.STIMbox.dll**

This file is required as interface between the g.STIMbox and the computer. It is located in the System folder.

**gSTIMboxEPmaster.exe**

This is an application which can be used for delivering stimuli for measurements of evoked potentials (EP). Stored under

```
C:\Program Files\gtec\gSTIMbox
```

**Help files** - are stored under

```
C:\Program Files\gtec\gSTIMbox\Help
```

## 6 Introduction to g.STIMbox

g.STIMbox is a stimulator digital I/O box with USB 2.0 technology. The device provides 14 digital inputs and 16 digital outputs, which are connected via two 26 pin Sub-D connectors. 8 digital inputs and 8 digital outputs can be used via Cinch connectors at the top of the box. g.STIMbox can be connected directly to a PC or notebook with an USB connector without any additional data acquisition device needed.



*g.STIMbox*

## Highlights

- Digital inputs and outputs are controlled with precise timing from a computer
- Digital outputs can produce very stable frequencies
- Direct control with a PC/notebook via USB
- Fully USB-powered (max. 200 mA overall output current)
- External +5 V DC power supply connection supported (max. 2.6 A overall output current)
- 14 digital Inputs (TTL)
  - 14 inputs via 26 pin Sub-D male connector
  - 8 inputs via Cinch connectors
- 16 digital outputs (TTL)
  - All 16 outputs with FET amplifier stage (max. 500 mA per channel, max. 2.6 A overall)
  - 16 outputs via 26 pin Sub-D female connector
  - 8 outputs via Cinch connectors with Status-LEDs

### **g.STIMbox consists of the following items:**

1 g.STIMbox – USB stimulation box

1 USB cable

1 power supply 5V

**ACCESSOIRES (for separate purchase):**



*g.SSVEPbox*

The g.SSVEPbox can be used for SSVEP BCI systems with four different classes.



*g.STIMbox push button*

*g.STIMbox single led*



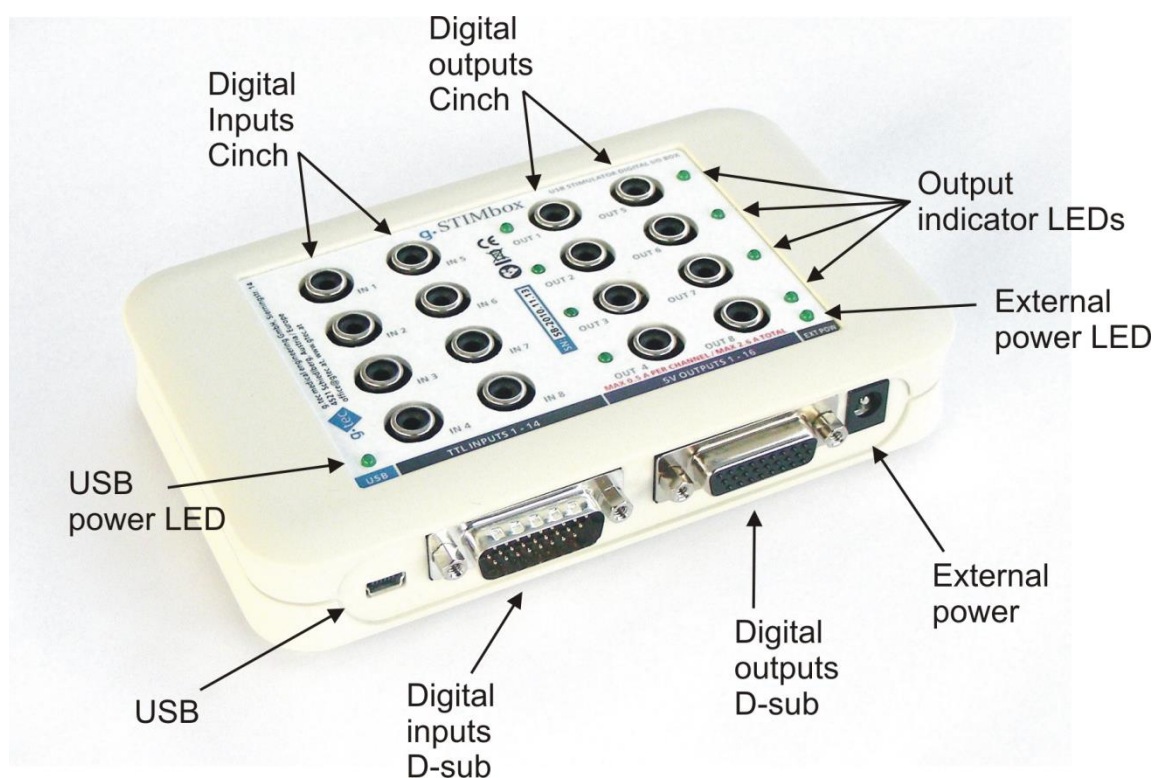
## 7 Explanation of switches, connectors and LEDs

### 7.1 Sockets and connectors on the front side

g.STIMbox has one USB socket, one 26-pin Sub-D connector (14 digital inputs, Vcc and GND), one 26-pin Sub-D socket (16 digital outputs, Vcc and GND) and a +5 V socket for an external power supply.

### 7.2 Sockets, connectors and LEDs on the top side

g.STIMbox has one USB power LED, 8 Cinch sockets for digital inputs, 8 Cinch sockets for digital outputs, 8 output indicator LEDs and a power LED for external power supply.



*g.STIMbox sockets, connectors and LEDs*

### 7.3 Marking on the top side



CE mark



do not dispose with domestic waste



attention to instructions

g.tec medical engineering GmbH, Sierningstr. 14  
4521 Schiedlberg, Austria / Europe  
office@gtec.at, www.gtec.at

manufacturer address

**SN: SB-2009.11.05**

Serial number in the format SB-Year.Month.Number

## **8 Safe operation of the g.STIMbox**

### **8.1 Instructions for use**

#### **Attention**

- the device is **not** protected against water
- it is not allowed to use other power supplies than the USB port or the genuine external +5 V DC-power supply unit.

#### **Service and repair**

For safety, performance and reliability of the device, the manufacturer will be responsible if:

- a) service, repair and changes are performed by the manufacturer only.
- b) the device is used according to the instruction for use.

#### **The intended environment of use**

The device **must not** be used in dangerous conditions such as wet rooms or explosive environments.

The relative humidity must be between 25 % and 95 %.

#### **Properties of PC or notebook**

The PC or notebook used together with g.STIMbox must have a USB 2.0 connector and must operate under a Windows operating system (Windows Vista).

### **8.2 Switching off and storage of g.STIMbox**

To switch off g.STIMbox and to store the device correctly, disconnect the USB cable and/or external power supply unit.

## 8.3 Using the g.STIMbox with Simulink

### 8.3.1 QUICKSTART

#### Running g.STIMbox

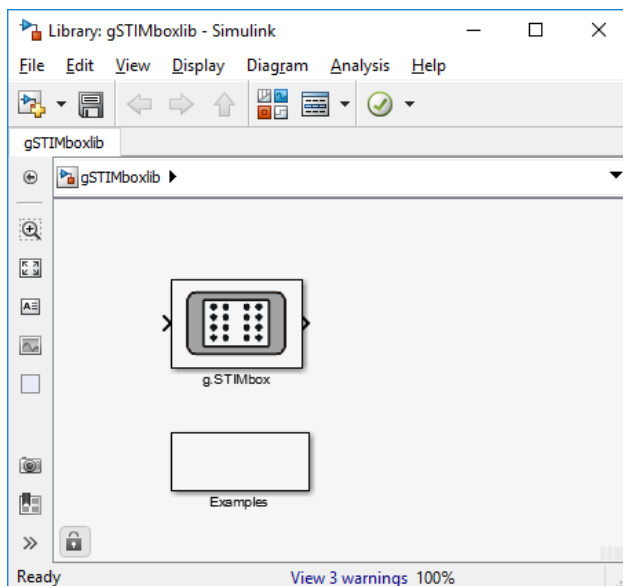
Connect output port 1 of the g.STIMbox to its input port 1 with a cinch connection cable.



Connect your g.STIMbox with the USB-cable to the computer.

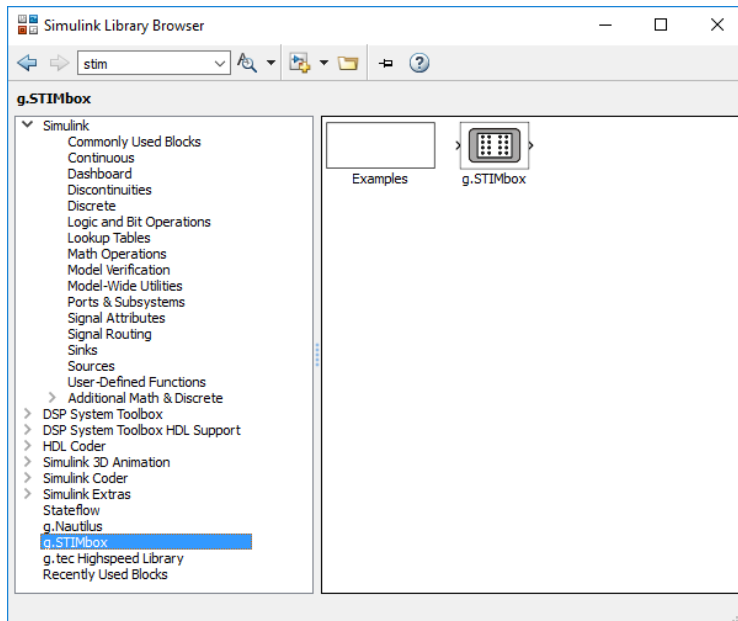
To open the g.STIMbox block library type under the MATLAB command window

```
gSTIMboxlib
```



To start the **Simulink Library Browser** in MATLAB click the button **Simulink Library** in the **Home** command ribbon under the **Simulink** group or type `simulink` in the command window.

The **Simulink Library Browser** gives access to all Simulink based blocksets. Scroll down to **g.STIMbox** to show the respective processing blockset.

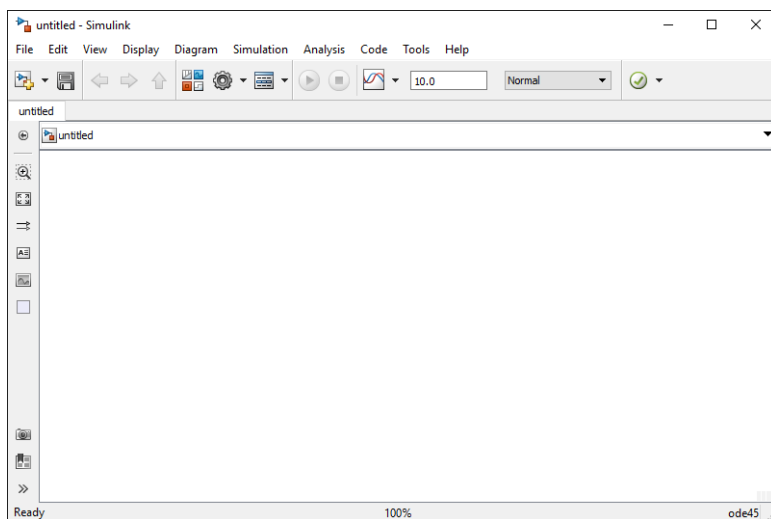


## Creating a Simulink Model

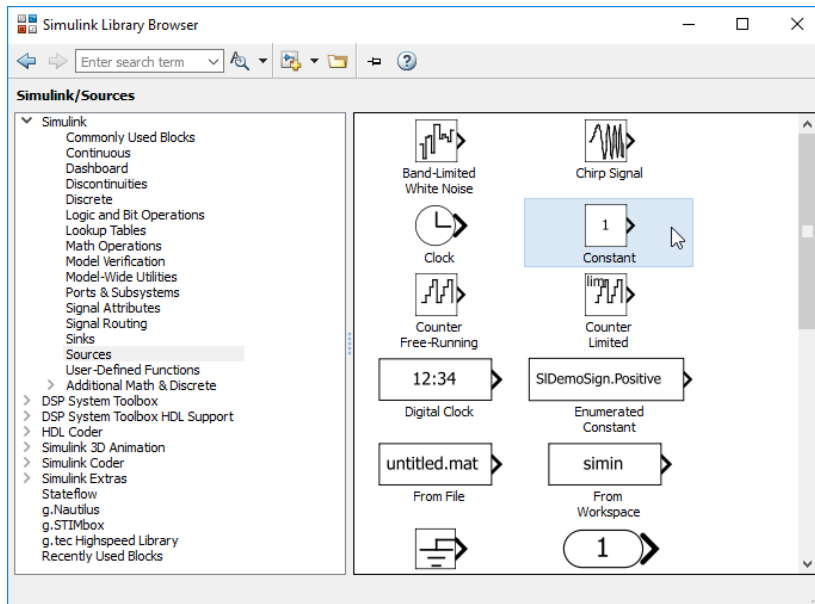
To create a new Simulink model, click on the **Create a new model** icon in the **Simulink Library Browser**.



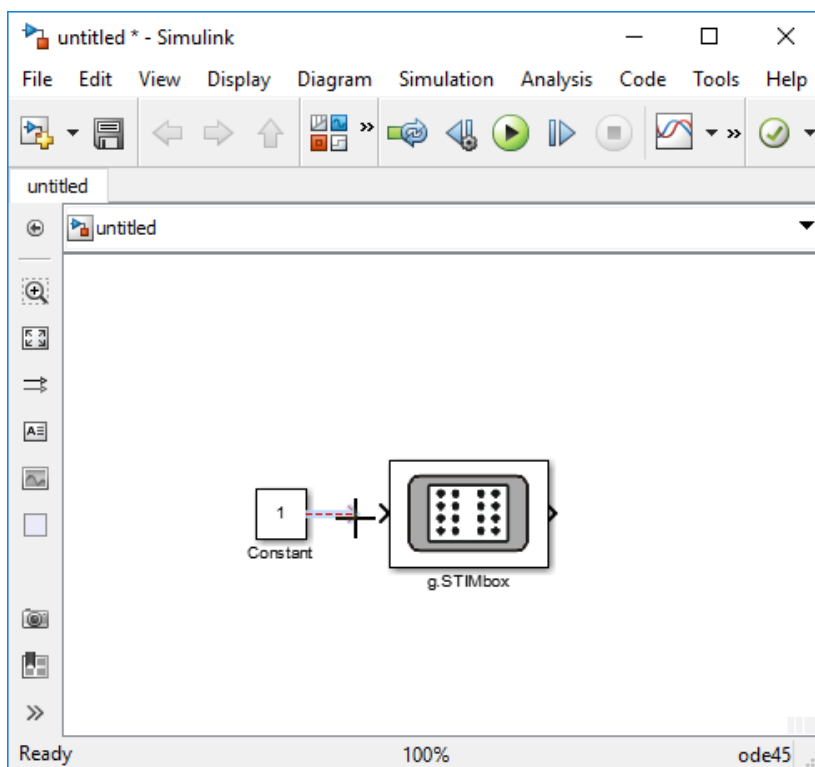
A new and empty model window opens with the Simulink control menu.



Go to the **Simulink Library Browser** and select the **Constant** from the **Sources** folder. This block is a source block and is used to produce a constant output signal. Drag and drop the block into your empty Simulink model.



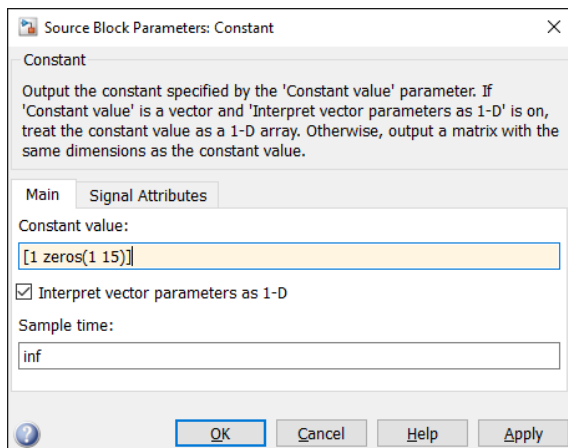
Select the **g.STIMbox** block from the **g.STIMbox** blockset and put it also into the new model. Connect the output of the **Signal Generator** and the input of the **g.STIMbox** block by holding down the mouse button and dragging the line from one block to the second one.



Double click on the Constant block to open its parameters dialog. In the field Constant Value enter

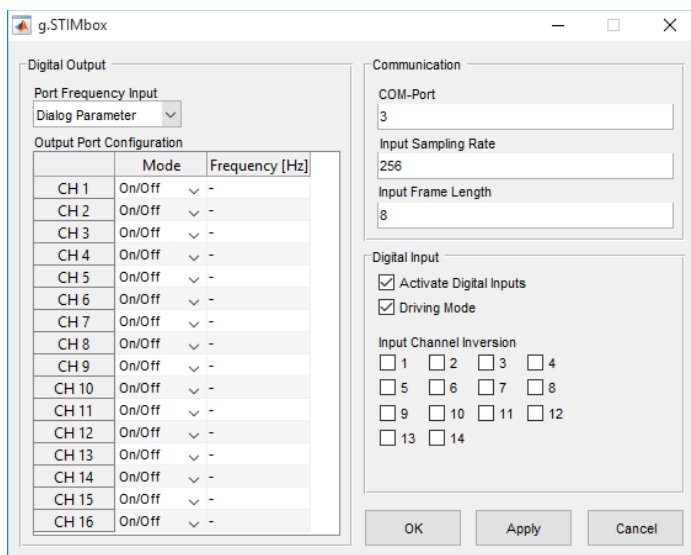
```
[1 zeros(1,15)]
```

This means that the Constant block will output a vector of length 16 with the first member being 1 and the rest 0.

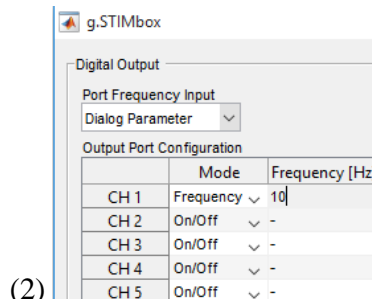
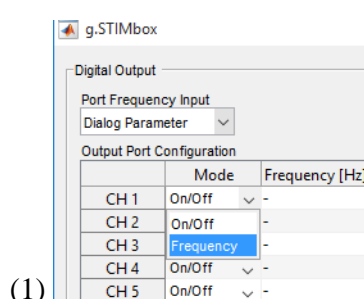


Click **OK** to apply your change.

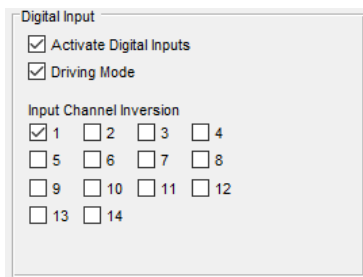
Double-click on the g.STIMbox block to open its parameters dialog. The following window appears:



In the **COM-Port** text field enter the COM port your g.STIMbox is connected to. Set the **Mode** of Output port 1 to **Frequency** (1) and set a frequency of 10 Hz (2).



In the Digital Input pane switch on the channel inversion for input port 1.

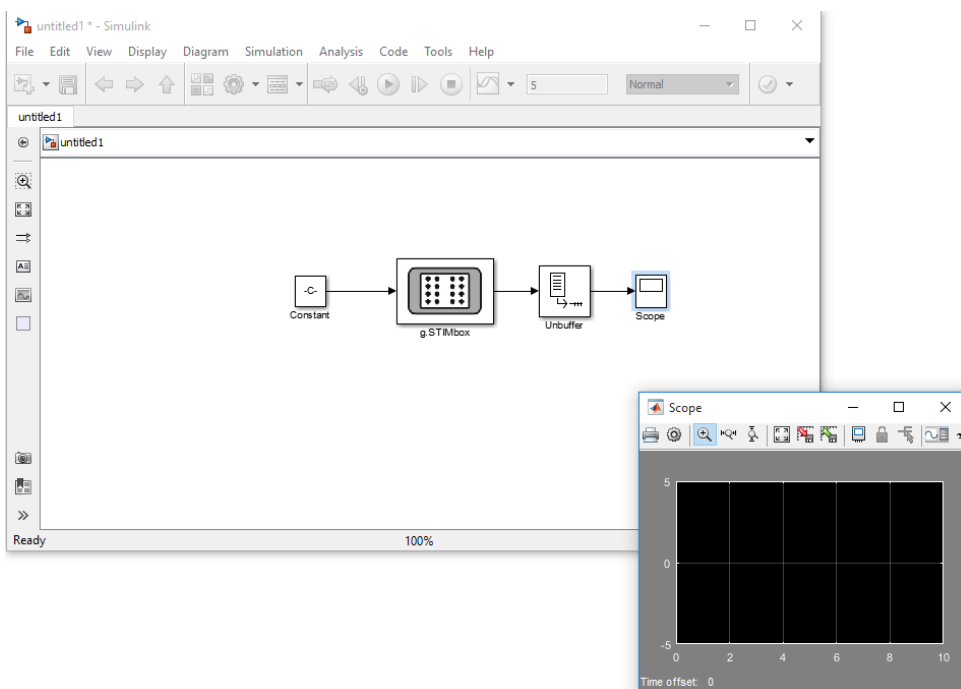


Click **OK** to apply your changes.

Drag an **Unbuffer** block from the **DSP System Toolbox/Signal Management/Buffers** blockset into your model and connect it to the output of the **g.STIMbox** block.

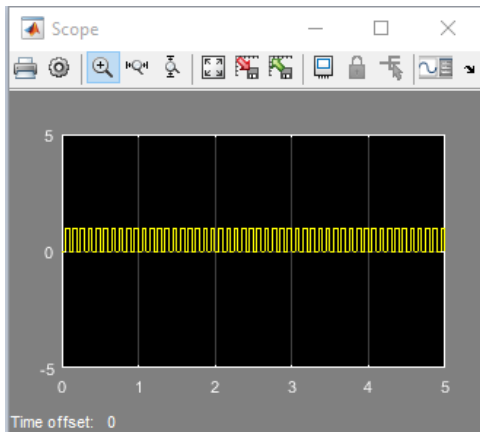
Drag a **Scope** block from the **Simulink/Sinks** blockset into the model. Connect it to the output port of the **Unbuffer** block. Open it by double-clicking it.

Set the simulation time of your model to 5. This has the effect that Simulink will stop the simulation after five seconds by itself.



To start the simulation, click on **Start/Run** under the **Simulation** menu. The Simulink model executes, sets the output port 1 of the **g.STIMbox** to 10Hz and records it back at input port 1.





After the model stopped the **Scope** shows the recorded input signal. It is a rectangle signal with a pulse length of 1/10s (10Hz) recorded for 5 seconds.

### 8.3.2 DEMOS

There are two Simulink models included in the software for the g.STIMbox as examples.

Before running the examples make sure that the g.STIMbox is connected to one of the USB-ports of the computer and the operating system has the driver for the virtual COM port installed correctly (see Section 5.2).

The examples can be opened by entering the following commands into the MATLAB command line:

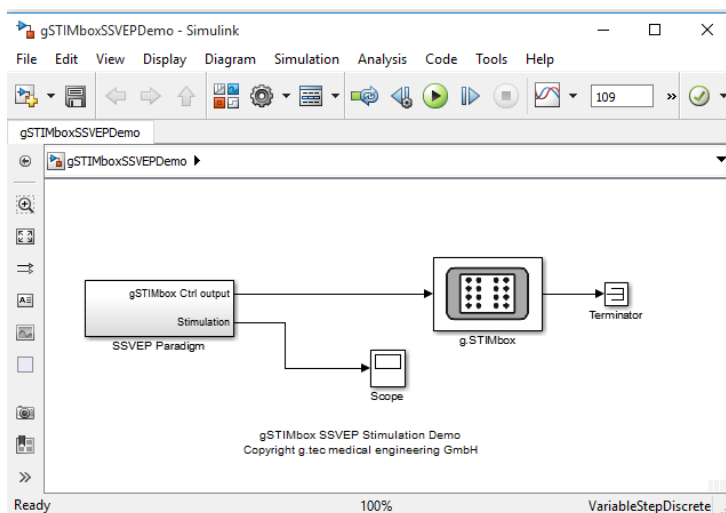
```
open gSTIMboxSSVEPDemo
or
open gSTIMboxDigInDemo
or
open gSTIMboxVIBROTACTILEP300Demo
```

### 8.3.2.1 gSTIMbox SSVEP Stimulation

For this example you need the g.SSVEPbox, which is available for purchase separately. It has to be connected to the g.STIMbox like it is shown in the picture below.

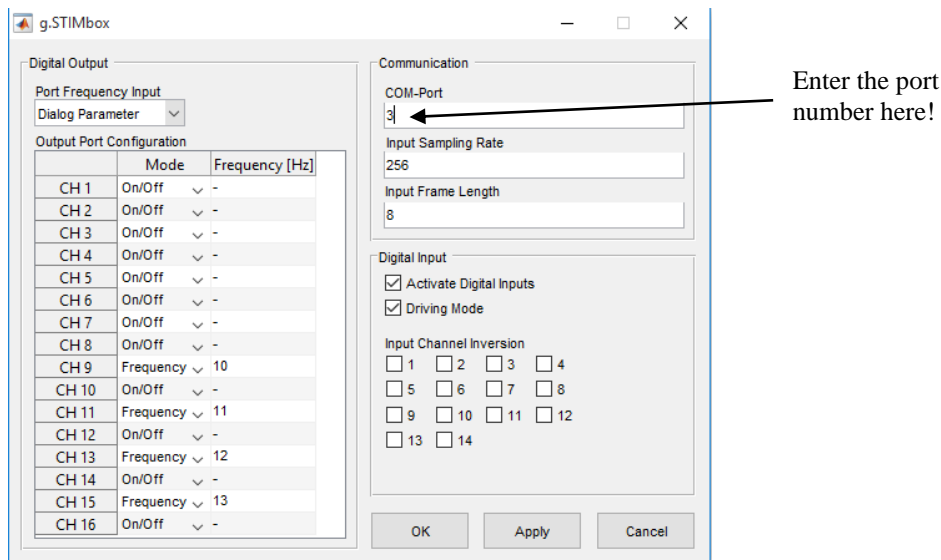


After you enter `open gSTIMboxSSVEPDemo` into the MATLAB command line the Simulink model depicted below opens.



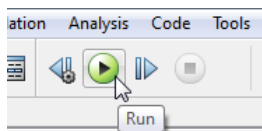
The model contains a paradigm for SSVEP experiments (block **SSVEP Paradigm**) which causes the stimulation LEDs to flicker for seven seconds and turns them off for three seconds. The green arrow LEDs on the g.SSVEPbox are used as indication for the test subject where to look.

Double-click the g.STIMbox block to open the **parameters dialog** and enter the port number your g.STIMbox is assigned to before you start the model.

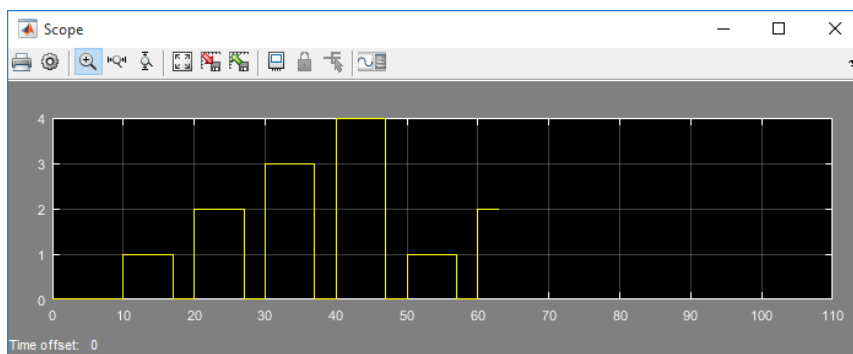


Please do not change other settings in this dialog box within the example models. After you entered the COM-port, click **OK**.

Now you can start the model by clicking on the **Play** symbol.



During the simulation you can look at the state of the paradigm with the included Scope. After a double click it opens and shows the progress of the paradigm.



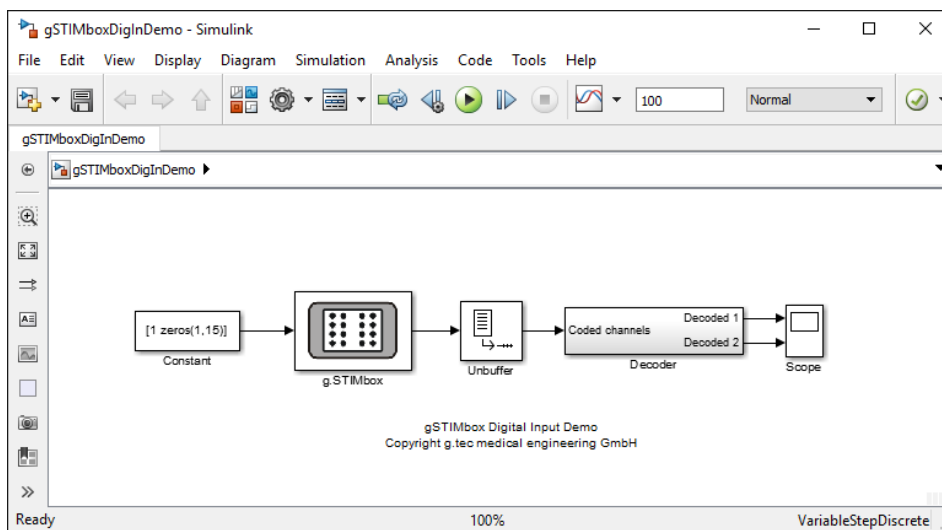
In this screenshot of the Scope taken during a stimulation it can be seen that the paradigm started its first stimulation 10 seconds after the start of the simulation with the first stimulation frequency. The current time is about 63 seconds and the current stimulation index is 2.

### 8.3.2.2 gSTIMbox Digital Input Demo

For this example you need a conventional Cinch cable and a g.STIMbox push button, which is available for purchase separately. The parts have to be connected to the g.STIMbox like it is shown in the picture below.

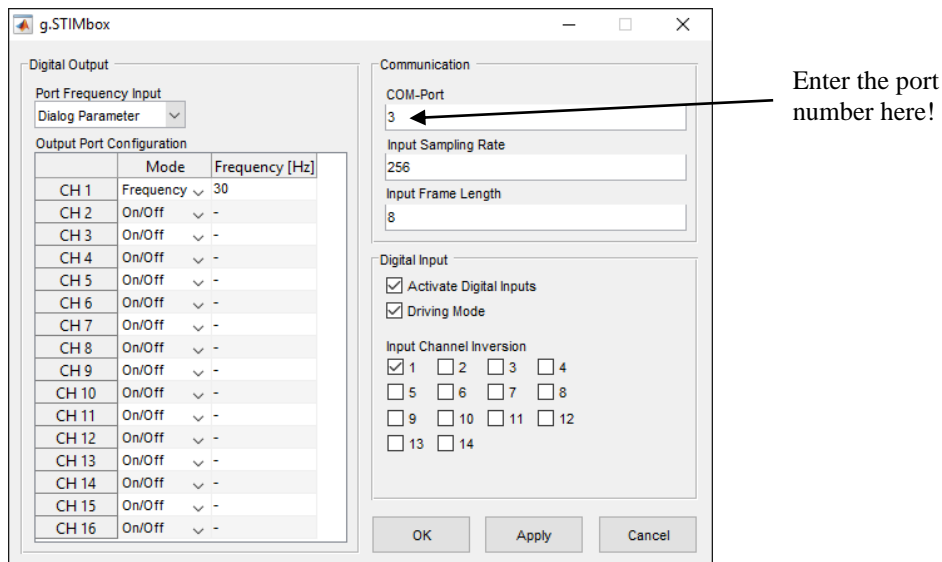


After you enter `open gSTIMboxDigInDemo` into the MATLAB command line the Simulink model depicted below opens.



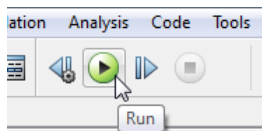
This Simulink example sets the first output of the g.STIMbox such, that it toggles it with 30 Hz. Due to the hardware connection from output 1 to input 1 this signal is re-recorded with the g.STIMbox. The second input of the device records the switches performed with the g.STIMbox push button. The block **Decoder** decodes the signal coming from the g.STIMbox to separate signals. These signals are then displayed with the **Scope**.

Double-click on the g.STIMbox block to open the **parameters dialog** and enter the port number your g.STIMbox is assigned to before you start the model.

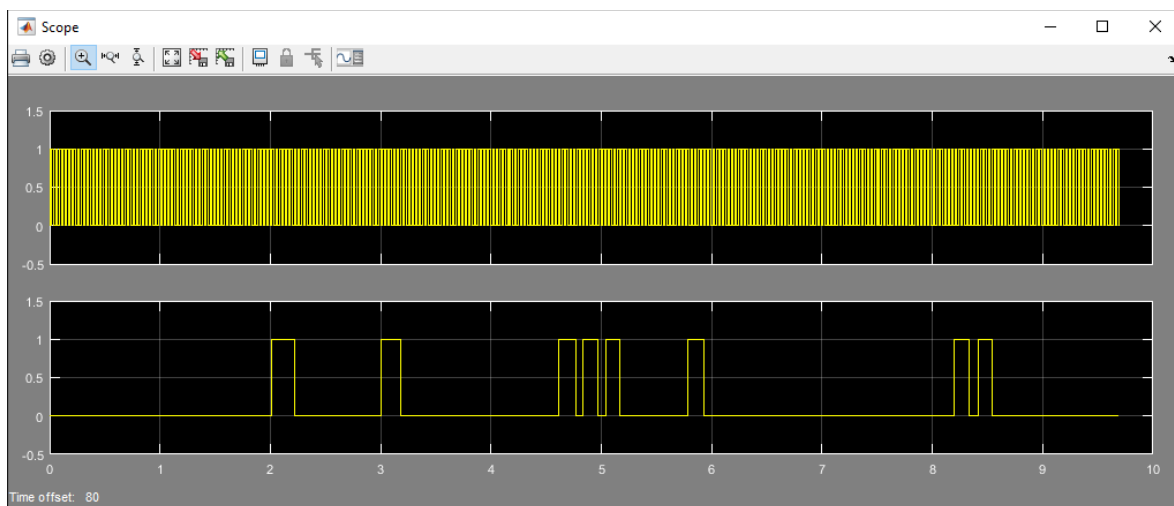


Please do not change other settings in this dialog box within the example models. After you entered the COM-port, click **OK**.

Now you can start the model by clicking on the **Play** symbol.



During the simulation you can inspect the signals recorded by the g.STIMbox with the Scope. After a double click it opens and shows two signals.



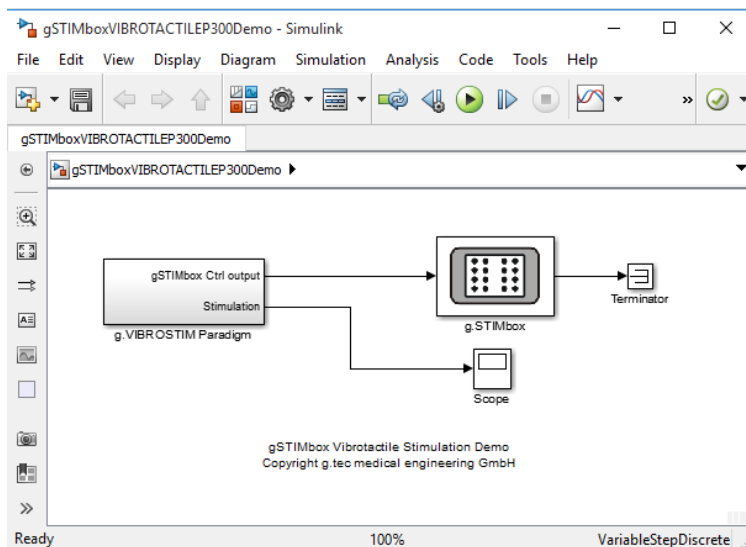
In this screenshot of the Scope taken during a simulation in the first channel the toggled signal coming from output 1 can be seen. The second channel shows the button presses of the user.

### 8.3.2.3 gSTIMbox Vibrotactile P300 Demo

For this example you need two g.VIBROSTIM vibrotactile stimulators, which are available for purchase separately. They have to be connected into the output ports OUT 1 and OUT 2 of the g.STIMbox.

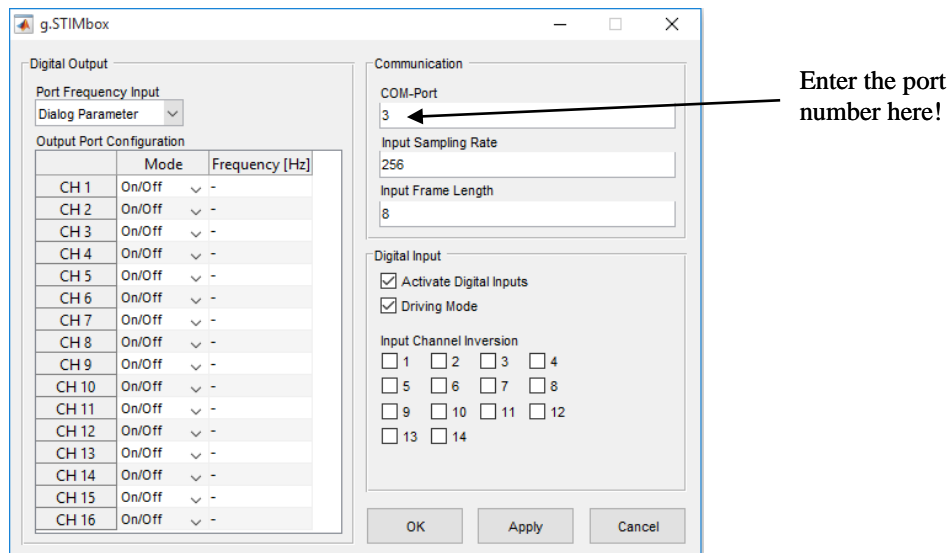


After you enter `open gSTIMboxVIBROTACTILEP300Demo` into the MATLAB command line the Simulink model depicted below opens.



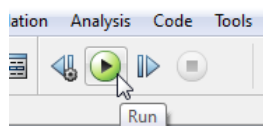
The model contains a paradigm for a tactile P300 experiment. It uses an oddball paradigm in which low-probability deviant items are mixed with high-probability standard items. The standard stimulus is delivered by the first factor, which is plugged into OUT 1. The deviant stimulus is delivered by the second factor, which is plugged into OUT 2. The stimulus time, as well as the off-time is set to 125ms.

Double-click on the g.STIMbox block to open the **parameters dialog** and enter the port number your g.STIMbox is assigned to before you start the model.

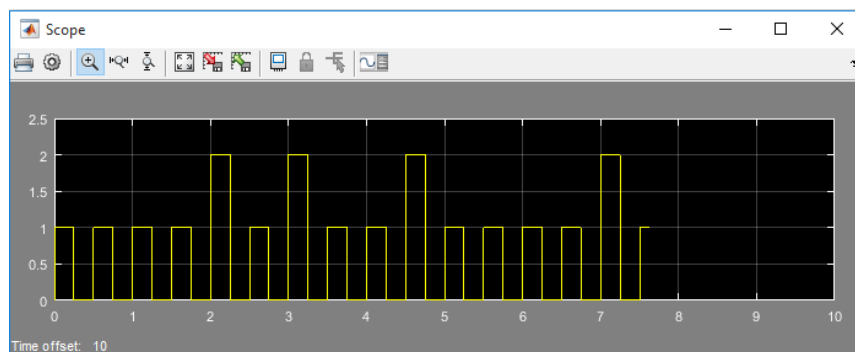


Please do not change other settings in this dialog box within the example models. After you entered the COM-port, click **OK**.

Now you can start the model by clicking on the **Play** symbol.



During the simulation you can look at the state of the paradigm with the included Scope. After a double click it opens and shows the progress of the paradigm.



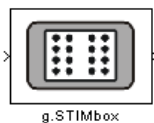
In this screenshot of the Scope taken during the stimulation, the single stimuli can be seen. A value of one is a standard stimulus, a value of two represents a deviant stimulus.

### 8.3.3 BLOCK REFERENCE

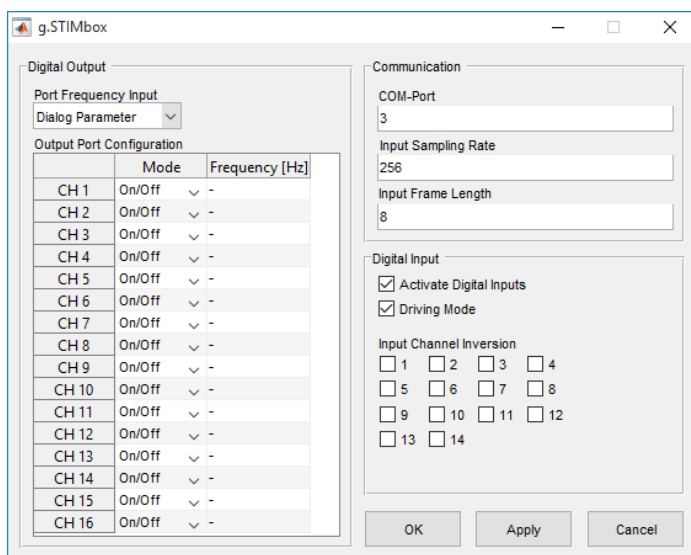
The block library of the g.STIMbox contains one block. It has one input and one output port by default.

The input port accepts vectors of the length 16 containing ones or zeros to activate or deactivate the 16 digital output ports of the hardware box.

The output port delivers data recorded from the digital inputs of the hardware box to Simulink. It outputs vectors of the length <frame length>, settable in the parameter dialog of the g.STIMbox (see below).

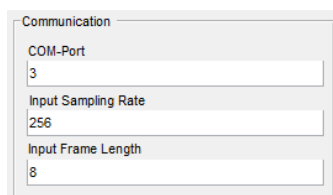


A double-click onto this block in a Simulink model opens its parameter dialog.



The parameter dialog of the g.STIMbox consists of three sections: **Communication**, **Digital Output**, and **Digital Input**.

#### Communication



#### COM-Port

Here the COM-Port the g.STIMbox is connected to has to be inserted. To find out the correct port see Section 4.2.



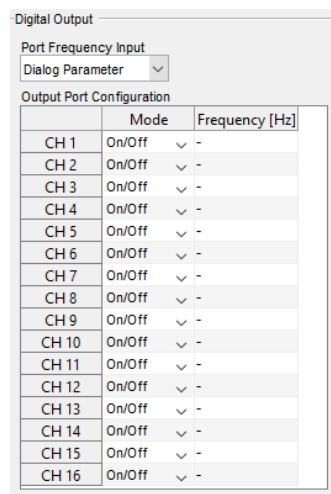
## Input Sampling Rate

Here the Sampling Rate of the digital input monitoring can be set. Valid numbers are from 16 to 300 Hz.

## Input Frame Length

Here the frame length of the data packets the g.STIMbox sends to the computer can be set. The output port of the g.STIMbox block outputs vectors of this length when the model is running and **Input Monitoring** is activated.

## Digital Output



	Mode	Frequency [Hz]
CH 1	On/Off	-
CH 2	On/Off	-
CH 3	On/Off	-
CH 4	On/Off	-
CH 5	On/Off	-
CH 6	On/Off	-
CH 7	On/Off	-
CH 8	On/Off	-
CH 9	On/Off	-
CH 10	On/Off	-
CH 11	On/Off	-
CH 12	On/Off	-
CH 13	On/Off	-
CH 14	On/Off	-
CH 15	On/Off	-
CH 16	On/Off	-

## Output Port Configuration

In this table the modes of the hardware output ports of the g.STIMbox can be specified in the column **Mode**. Valid modes are **On/Off** and **Frequency**.

If the **Mode** switch of a port  $n$  is set to **On/Off** a 1 at index  $n$  of the vector connected to the input port of the g.STIMbox block causes output  $n$  to be set to high and a 0 to be set to low.

If the **Mode** switch of a port  $n$  is set to **Frequency** a 1 at index  $n$  of the vector connected to the input port of the g.STIMbox block causes output  $n$  to toggle between high and low in the frequency specified (see below).

If the **Mode** switch of a port is set to **Frequency** and the **Port Frequency Input** setting (see below) is set to **Dialog Parameter** the frequency the port should present can be set in the column **Frequency [Hz]** of the table.

## Port Frequency Input

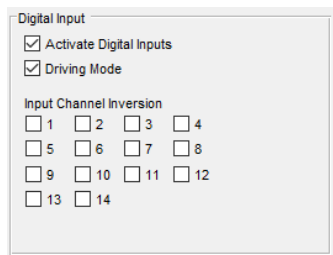
Here you can choose if you want to specify the frequencies the g.STIMbox delivers at its hardware outputs in the parameter dialog (**Dialog Parameter**) or via a second input port of the g.STIMbox block (**Input Port**).

If **Input Port** is selected, the g.STIMbox block presents a second input port where vectors of the length 16 can be connected which contain the frequencies to present.

If **Dialog Parameter** is selected the frequencies can be entered in the second column of the table under **Frequency [Hz]** if the **Mode** switch of the port is set to **Frequency**.

Valid frequencies are from 1 to 50 Hz with 1 Hz steps.

## Digital Input



Digital Input

☒ Activate Digital Inputs

☒ Driving Mode

Input Channel Inversion

<input type="checkbox"/> 1	<input type="checkbox"/> 2	<input type="checkbox"/> 3	<input type="checkbox"/> 4
<input type="checkbox"/> 5	<input type="checkbox"/> 6	<input type="checkbox"/> 7	<input type="checkbox"/> 8
<input type="checkbox"/> 9	<input type="checkbox"/> 10	<input type="checkbox"/> 11	<input type="checkbox"/> 12
<input type="checkbox"/> 13	<input type="checkbox"/> 14		

### Activate Digital Inputs

With this checkbox the input recording function of the g.STIMbox can be switched on and off. If the checkbox is not checked, the g.STIMbox block has no output port. If it is checked, the block has an output port which delivers the recorded data.

### Driving Mode

With this checkbox the g.STIMbox can be set whether to drive the model with its hardware clock or not. If this box is checked the routine for getting the data from the g.STIMbox will wait until a new frame of data is available. If it is unchecked the routine will return a frame filled with the last data sample in the case that no new data frame is available.

### Input Channel Inversion

With this function certain input ports of the g.STIMbox can be inverted. In normal mode (without inversion) the input port is low when a possibly connected switch is open and high when it is closed. For signal generators with active low output for example the input of the g.STIMbox can be inverted in order to give an active high signal.

## 8.4 Using the g.STIMbox with the MATLAB-API

### 8.4.1 DEMOS

There are two MATLAB m-file scripts included in the software for the g.STIMbox as examples.

Before running the scripts make sure that the g.STIMbox is connected to one of the USB-ports of the computer and the operating system has the driver for the virtual COM port installed correctly (see Section 5.2).

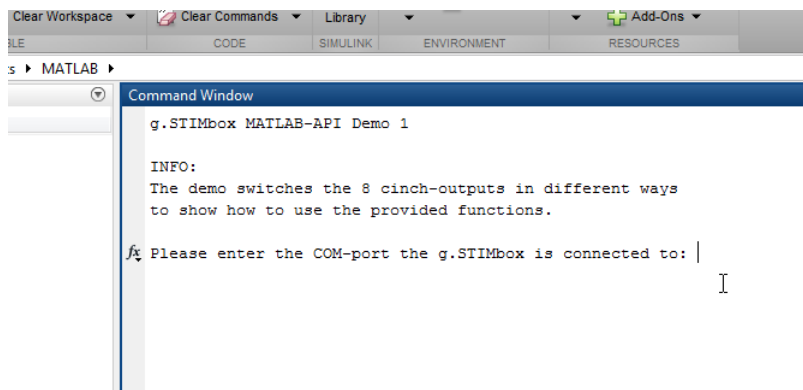
To run the scripts type in the MATLAB command prompt:

gSTIMboxDemo1

or

gSTIMboxDemo2

The scripts output some information about the demo and request to input the COM port the g.STIMbox is connected to.



Enter the COM port and press Enter.

The script starts to run now and controls the inputs and outputs of the g.STIMbox.

**Note:** For demo 2 a g.STIMbox push button is needed. See Section 8.4.1.2.

#### 8.4.1.1 MATLAB-API DEMO 1

The first demo of the g.STIMbox MATLAB-API controls outputs 1 to 8 of the g.STIMbox.

#### **gSTIMboxDemo1.m**

```
% gSTIMbox MATLAB-API Demo 1

function gSTIMboxDemo1()

clc

disp('g.STIMbox MATLAB-API Demo 1');

disp(' ');

disp('INFO:');
```

```

disp('The demo switches the 8 cinch-outputs in different ways');
disp('to show how to use the provided functions.');
```

disp(' ');

```

prompt = 'Please enter the COM-port the g.STIMbox is connected to: ';
com = input(prompt);

% initialize the g.STIMbox with a sampling rate of 256Hz, 8 Samples frame
% length and driving mode
handle = gSTIMboxinit(com, 256, 8, 1);

disp(' ');

disp('g.STIMbox MATLAB-API Demo 1 is running...');

disp(' ');

% get version information and print it
disp(['Hardware Version: ' sprintf('%.2f', gSTIMboxgetHWVersion(handle))]);
disp(['API Version: ' sprintf('%.2f', gSTIMboxgetDriverVersion())]);
disp(['COM Port: ' num2str(com)]);

disp(' ');

disp('--> setting ports to frequencies 1, 2, 3, 4, 5, 6, 7, and 8 Hz.');
```

```

% reset the g.STIMbox
gSTIMboxreset(handle);
% set the frequencies
gSTIMboxsetFrequency(handle, [1:8], [1:8]);
% set the port modes
gSTIMboxsetMode(handle, [1:8], ones(1,8));
% set the port states
gSTIMboxsetPortState(handle, [1:8], ones(1,8));
% wait for 20 seconds
pause(20)

% reset the g.STIMbox
gSTIMboxreset(handle);
disp('--> setting ports to computer-controlled implementation 1');
```

```

% switch the output ports of the g.STIMbox in a specific pattern
for j=1:20
    for i=1:8
        gSTIMboxsetPortState(handle, i, 1);
        pause(0.1);
    end

    for i=1:8
        gSTIMboxsetPortState(handle, i, 0);
        pause(0.1);
    end
end

% reset the g.STIMbox
gSTIMboxreset(handle);
disp('--> setting ports to computer-controlled implementation 2');
```

```

% switch the output ports of the g.STIMbox in a specific pattern
for j=1:20
    for i=1:8
        if i==1
            gSTIMboxsetPortState(handle, i, 1);
        else
            gSTIMboxsetPortState(handle, [i i-1], [1 0]);
        end
        pause(0.05);
    end

    for i=8:-1:1
        if i==8
            gSTIMboxsetPortState(handle, i, 1);
        else
            gSTIMboxsetPortState(handle, [i i+1], [1 0]);
        end
        pause(0.05);
    end
end

% reset and close the g.STIMbox and exit
gSTIMboxreset(handle);
gSTIMboxclose(handle);

disp(' ');
disp('*** Program finished ***');

```

#### 8.4.1.2 MATLAB-API DEMO 2

In the second Demo of the g.STIMbox MATLAB-API four outputs of the box are controlled by the MATLAB script whereas other four outputs are controlled by a g.STIMbox push button connected to input port 1 of the g.STIMbox. The following picture shows the hardware setup.



For this Demo two .m files are required. gSTIMboxDemo2.m is the main file which controls the output of the g.STIMbox. The second file contains the function which is called via a MATLAB timer object periodically and is managing the input of the g.STIMbox.

## **gSTIMboxDemo2.m**

```
% gSTIMbox MATLAB-API Demo 2

function gSTIMboxDemo2()

clc

disp('g.STIMbox MATLAB-API Demo 2');

disp(' ');

disp('INFO:');
disp('Output ports 1-4 are controlled by the PC whereas output ports 5-8');
disp('can be controlled by attaching and using a switch at input port 1');
disp('of the g.STIMbox.');
```

disp(' ');

prompt = 'Please enter the COM-port the g.STIMbox is connected to: ';  
com = input(prompt);

% initialize the g.STIMbox with a sampling rate of 256Hz, 8 Samples frame  
% length and driving mode  
handle = gSTIMboxinit(com, 256, 8, 1);

disp(' ');

disp('g.STIMbox MATLAB-API Demo 2 is running...');

disp(' ');

% get version information and print it  
disp(['Hardware Version: ' sprintf('%.2f', gSTIMboxgetHWVersion(handle))]);  
disp(['API Version: ' sprintf('%.2f', gSTIMboxgetDriverVersion())]);  
disp(['COM Port: ' num2str(com)]);

disp(' ');

% preparing UserData for timer function  
userData.handle = handle;  
userData.framelength = gSTIMboxgetInputFrameLength(handle);  
userData.lastSample = 0;  
userData.ports = 5:8;  
userData.portIdx = 1;  
userData.inputdata = [];

% preparing timer function  
t = timer;  
set(t, 'TimerFcn', @gSTIMboxDemo2InputThread)  
set(t, 'Period', 0.001)  
set(t, 'ExecutionMode', 'fixedSpacing')  
set(t, 'TasksToExecute', Inf)  
set(t, 'UserData', userData);

% start input monitoring and timer function  
gSTIMboxstartInputMonitoring(handle);

```

start(t)

% switch the output ports 1:4 of the g.STIMbox in a specific pattern
for j=1:20
    for i=1:4
        gSTIMboxsetPortState(handle, i, 1);
        pause(0.1);
    end

    for i=1:4
        gSTIMboxsetPortState(handle, i, 0);
        pause(0.1);
    end
end

% stop timer
stop(t);

% stop input monitoring, reset and close g.STIMbox
gSTIMboxstopInputMonitoring(handle);
gSTIMboxreset(handle);
gSTIMboxclose(handle);

% plot recorded data
userData = get(t, 'UserData');

plot(bitand(userData.inputdata, 1)');
set(gca, 'YLim', [-0.2 1.2])
set(gca, 'YTick', [0 1])
xlabel('Time (s)');
ylabel('gSTIMbox Input 1');

inputlength = length(userData.inputdata);
divider = 0;
ticks = 100;
while ticks > 10
    divider = divider + 128;
    ticks = inputlength/divider;
end

set(gca, 'XTick', 0:divider:(floor(ticks)*divider));
set(gca, 'XTickLabel', 0:(divider/256):((floor(ticks)*divider)/256));

disp(' ');

disp('*** Program finished ***');

end

```

## **gSTIMboxDemo2InputThread.m**

```
function gSTIMboxDemo2InputThread(obj, event)

% get the UserData structure
userData = get(obj, 'UserData');

% get input data from the g.STIMbox
[inputdata overunderflow] = gSTIMboxgetInputs(userData.handle);

% concatenate current input data with previously collected data
userData.inputdata = [userData.inputdata inputdata'];

for i=1:userData.frameLength
    % detect rising edges (sample i-1 == 0 and i == 1)
    if bitand(userData.lastSample, 1) == 0 && ...
        bitand(inputdata(i), 1) == 1
        disp('*');

        % set the next output to 'on'
        states = zeros(1,4);
        states(userData.portIdx) = 1;

        gSTIMboxsetPortState(userData.handle, userData.ports, states);

        userData.portIdx = userData.portIdx + 1;
        if userData.portIdx > 4
            userData.portIdx = 1;
        end
    end
    userData.lastSample = inputdata(i);
end

set(obj, 'UserData', userData);
end
```

You can use these scripts as templates for your own scripts for realizing paradigms.



## 8.4.2 FUNCTION REFERENCE

[gSTIMboxreset](#)  
[gSTIMboxsetFrequency](#)  
[gSTIMboxsetInputInversion](#)  
[gSTIMboxsetMode](#)  
[gSTIMboxinit](#)  
[gSTIMboxsetPortState](#)  
[gSTIMboxstartInputMonitoring](#)  
[gSTIMboxstopInputMonitoring](#)  
[gSTIMboxclose](#)  
[gSTIMboxgetConnStatus](#)  
[gSTIMboxgetDriverVersion](#)  
[gSTIMboxgetHWVersion](#)  
[gSTIMboxgetInputFrameLength](#)  
[gSTIMboxgetInputs](#)  
[gSTIMboxgetInputSampleRate](#)

**NOTE:** To allow customized error handling procedures it is recommended to surround calls to the g.STIMbox MATLAB-API by try/catch statements in order to allow a proper error handling! See Section 8.4.3.

#### **8.4.2.1 gSTIMboxreset**

Call it with: `gSTIMboxreset(handle);`

**DESCRIPTION:** The function resets a g.STIMbox represented by handle opened with the `gSTIMboxinit` function to its initial state.

**INPUT:**

handle...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxreset(handle);  
gSTIMboxclose(handle);
```

#### 8.4.2.2 *gSTIMboxsetFrequency*

Call it with: `gSTIMboxsetFrequency(handle, ports, frequencies);`

**DESCRIPTION:** This function sets certain output ports of the `g.STIMbox` to certain frequencies. The box will toggle the outputs high and low in this frequency if the modes of these ports are set to 1 with the function `gSTIMboxsetMode` and the states of the ports are set to 1 with the function `gSTIMboxsetPortState`.

**INPUT:**

`handle`...The handle representing the `g.STIMbox` (generated with the function `gSTIMboxinit`).

`ports`...An array containing the output ports of the `g.STIMbox` to which the frequencies given in the next parameter will be assigned.

`frequencies`...An array containing the frequencies in Hz.

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxsetFrequency(handle, [1 2 3 4], [10 11 12 13]);  
gSTIMboxclose(handle);
```

In this example the frequencies 10Hz, 11Hz, 12Hz, and 13Hz are assigned to the output ports 1, 2, 3, and 4 of the `g.STIMbox` represented by `handle`. Afterwards the handle is closed again with `gSTIMboxclose`.

### 8.4.2.3 *gSTIMboxsetInputInversion*

Call it with: `gSTIMboxsetInputInversion(handle, inversionVector);`

**DESCRIPTION:** With this function certain input ports of the g.STIMbox can be inverted. In normal mode (without inversion) the input port is low when a possibly connected switch is open and high when it is closed. For signal generators with active low output for example the input of the g.STIMbox can be inverted in order to give an active high signal.

**INPUT:**

`handle...`The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

`inversionVector...`An array containing the input ports of the g.STIMbox to be inverted. An empty array (`[]`) means that no input port will be inverted.

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxsetInputInversion(handle, [5 7 9]);  
gSTIMboxclose(handle);
```

In this example the input ports 5, 7, and 9 of the g.STIMbox represented by `handle` will be inverted. Afterwards the handle is closed again with `gSTIMboxclose`.

#### 8.4.2.4 *gSTIMboxsetMode*

Call it with: `gSTIMboxsetMode(handle, ports, modes);`

**DESCRIPTION:** This function sets certain output ports of the `g.STIMbox` either to be in on/off mode (0) or in frequency mode (1). In both modes a port STATE of 0 means that the output port of the `g.STIMbox` is switched off. A port state of 1 in mode 0 means that the output port is switched on continuously whereas in mode 1 it means that the `g.STIMbox` toggles the output port in the frequency specified with the function `gSTIMboxsetFrequency`. The port states can be set with the function `gSTIMboxsetPortState`.

**INPUT:**

`handle`...The handle representing the `g.STIMbox` (generated with the function `gSTIMboxinit`).

`ports`...An array containing the output ports of the `g.STIMbox` to which the modes given in the next parameter will be assigned.

`modes`...An array containing the modes for the output ports.

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxsetMode(handle, [1 2 3 4], [0 1 0 1]);  
gSTIMboxclose(handle);
```

In this example the output ports 1 and 3 of the `g.STIMbox` represented by `handle` are set to mode 0 (on/off mode) and output ports 2 and 4 are set to mode 1 (frequency mode). Given, that the states of the ports 1 to 4 are set to 1 (on) ports 1 and 3 will be switched on and ports 2 and 4 will flicker in a given frequency. Afterwards the handle is closed again with `gSTIMboxclose`.

#### **8.4.2.5 gSTIMboxinit**

Call it with:

```
handle = gSTIMboxinit(comPort, samplingrate, framelength,  
drivingMode);
```

**DESCRIPTION:** This function initializes the communication between the computer and the g.STIMbox.

**INPUT:**

comPort...The number of the virtual COM port the g.STIMbox is connected to.

samplingrate...The sampling rate in Hz the g.STIMbox will digitize its input ports.

framelength...The length of the frames (in samples) with digitized input data the g.STIMbox will send to the computer.

drivingMode...Can be 0 or 1. 0 means that the command gSTIMboxgetInputs will not block the process. 1 means that the mentioned command blocks the process until new data from the g.STIMbox is available.

**OUTPUT:**

handle...The handle is needed for all commands concerning the g.STIMbox. It must be stored in the workspace.

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxclose(handle);
```

In this example the g.STIMbox is initialized with COM-Port 10, a sampling rate of 256 Hz, a frame length of 8 samples and the driving mode on. Afterwards the handle is closed again with gSTIMboxclose.

#### 8.4.2.6 **gSTIMboxsetPortState**

Call it with: `gSTIMboxsetPortState(handle, ports, states);`

**DESCRIPTION:** This function sets the states of certain output ports of the g.STIMbox either to 1 (on) or 0 (off). For ports which are in mode 0 (on/off mode) this simply means that the port is switched on or off. For ports which are in mode 1 (frequency mode) it means that the frequency specified with the function `gSTIMboxsetFrequency` is either generated or the port is switched off. The port modes can be set with the function `gSTIMboxsetMode`.

##### **INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

`ports`...An array containing the output ports of the g.STIMbox to which the states given in the next parameter will be assigned.

`states`...An array containing the states for the output ports.

##### **OUTPUT:**

none

##### **EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);
gSTIMboxsetPortState(handle, [1 2 3 4], [0 1 1 1]);
gSTIMboxclose(handle);
```

In this example the output port 1 of the g.STIMbox represented by `handle` is set to 0 (off) and output ports 2 to 4 are set to 1 (on). Depending on the ports modes output ports 2 to 4 will either flicker in a certain frequency or be switched on. Afterwards the handle is closed again with `gSTIMboxclose`.

#### **8.4.2.7 gSTIMboxstartInputMonitoring**

Call it with: `gSTIMboxstartInputMonitoring(handle);`

**DESCRIPTION:** With this function the input monitoring of the g.STIMbox, represented by `handle` is started. After the call of this function the input can be received using the function `gSTIMboxgetInputs`.

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxstartInputMonitoring(handle);  
gSTIMboxclose(handle);
```



#### **8.4.2.8 gSTIMboxstopInputMonitoring**

Call it with: `gSTIMboxstopInputMonitoring(handle);`

**DESCRIPTION:** With this function the input monitoring of the g.STIMbox, represented by `handle` is stopped. The g.STIMbox stops to send data to the computer and the function `gSTIMboxgetInputs` should not be called after the call to this function.

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxstopInputMonitoring(handle);  
gSTIMboxclose(handle);
```

#### **8.4.2.9 gSTIMboxclose**

Call it with: `gSTIMboxclose(handle);`

**DESCRIPTION:** The function closes a `handle` to a `g.STIMbox` opened with the `gSTIMboxinit` function. After the call of this function a communication with the `g.STIMbox` is not possible anymore. It can be disconnected from the computer.

**INPUT:**

`handle...`The handle representing the `g.STIMbox` (generated with the function `gSTIMboxinit`).

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
gSTIMboxclose(handle);
```

#### **8.4.2.10 gSTIMboxgetConnStatus**

Call it with: `connectionState = gSTIMboxgetConnStatus(handle);`

**DESCRIPTION:** The function returns the state of the connection to a g.STIMbox for a given `handle` created by the function `gSTIMboxinit`.

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

`connectionState` which is 1 in case if the connection is working and 0 in case if the connection does not work or the handle was closed with the function `gSTIMboxclose`.

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
connectionState = gSTIMboxgetConnStatus(handle);  
gSTIMboxclose(handle);
```

#### **8.4.2.11 gSTIMboxgetDriverVersion**

Call it with: `driverVersion = gSTIMboxgetDriverVersion();`

**DESCRIPTION:** The function returns the version of the g.STIMbox driver installed on the system.

**INPUT:**  
none

**OUTPUT:**  
driverVersion...The version number of the installed driver for the g.STIMbox.

**EXAMPLE:**  
`handle = gSTIMboxinit(comport, 256, 8, 1);`  
`driverVersion = gSTIMboxgetDriverVersion();`  
`gSTIMboxclose(handle);`

#### **8.4.2.12 gSTIMboxgetHWVersion**

Call it with: `HWVersion = gSTIMboxgetHWVersion(handle);`

**DESCRIPTION:** The function returns the version of the firmware running on the microcontroller of the g.STIMbox represented by `handle`.

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

`HWVersion`...The version number of the firmware of the g.STIMbox.

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
HWVersion = gSTIMboxgetHWVersion(handle);  
gSTIMboxclose(handle);
```

#### **8.4.2.13 gSTIMboxgetInputFramelength**

Call it with: `inputFramelength = gSTIMboxgetInputFramelength(handle);`

**DESCRIPTION:** The function returns the length of the frames which are transmitted from the g.STIMbox, represented by `handle` to the computer when the input monitoring is activated (with the function `gSTIMboxstartInputMonitoring`).

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

`none`

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);
inputFramelength = gSTIMboxgetInputFramelength(handle);
gSTIMboxclose(handle);
```

#### **8.4.2.14 gSTIMboxgetInputs**

Call it with: `[input, overUnderflow] = gSTIMboxgetInputs(handle);`

**DESCRIPTION:** The function returns digitized input data of the g.STIMbox, represented by `handle` when the input monitoring is activated (after the call of the function `gSTIMboxstartInputMonitoring`). The length of the return value `input` is determined by the frame length specified in the method `gSTIMboxinit`.

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

`input`...A vector of the length of a frame (set in the `gSTIMboxinit` method) containing input data. The `framelength` can be determined by using the function `gSTIMboxgetInputFramelength`.

`overUnderflow`... This variable contains information about an empty data-receive buffer or a buffer overflow. It is 1 in case of a data receive buffer overflow, 2 in the case of an empty receive buffer and 0 otherwise.

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);
gSTIMboxstartInputMonitoring(handle);
[input, overUnderflow] = gSTIMboxgetInputs(handle);
gSTIMboxstopInputMonitoring(handle);
gSTIMboxclose(handle);
```

#### **8.4.2.15 gSTIMboxgetInputSamplerate**

Call it with: `inputSamplerate = gSTIMboxgetInputSamplerate(handle);`

**DESCRIPTION:** The function returns the sampling rate in Hz at which the g.STIMbox, represented by `handle` digitizes its input ports once the input monitoring is activated (with the function `gSTIMboxstartInputMonitoring`).

**INPUT:**

`handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

none

**EXAMPLE:**

```
handle = gSTIMboxinit(comport, 256, 8, 1);  
inputSamplerate = gSTIMboxgetInputSamplerate(handle);  
gSTIMboxclose(handle);
```



### 8.4.3 ERROR HANDLING

In order to allow customized error handling procedures it is recommended to surround calls to functions of the g.STIMbox MATLAB-API with try/catch statements. In this way it is possible to handle the error, to access the error ID and to display an error message to the user. The example code below shows a way how to achieve that.

```
try
    handle = gSTIMboxinit(-1, 256, 8, 1);
catch ME
    disp(['Error identifier: ' ME.identifier]);
    disp(['Error message: ' ME.message]);
    return;
end
```

This code produces the following output:

```
Error identifier: gSTIMbox:semanticError
Error message: Error: Input argument 1 must be a positive
natural number!
```

The following table gives an overview of the error identifiers of the g.STIMbox MATLAB-API

Error ID	Meaning
gSTIMbox:returnValueMismatch	The function was called with too much output parameters.
gSTIMbox:parameterMismatch	The function was called with too few or too many input parameters.
gSTIMbox:datatypeMismatch	The datatype of one or more input parameters doesn't fit.
gSTIMbox:dimensionMismatch	The dimension of one or more input parameters doesn't fit.
gSTIMbox:semanticError	A semantic error occurred.
gSTIMbox:memoryAllocationError	The function was unable to allocate memory.
gSTIMbox:portIndexOutOfRange	A port index was out of the valid range of 1:16.
gSTIMbox:invalidMode	An invalid mode value was given. Valid values are 0 and 1.
gSTIMbox:invalidState	An invalid port state value was given. Valid values are 0 and 1.
gSTIMbox:invalidFrequency	An invalid frequency was given. Valid values are from 1 to 50.
gSTIMbox:emptyInputError	An input parameter of the function was empty.
gSTIMbox:C_API_ERROR_XXX	An error in the underlying C-API occurred. Please refer for the error code (marked with xxx) to the table of C-API error codes in Section 8.5.4.

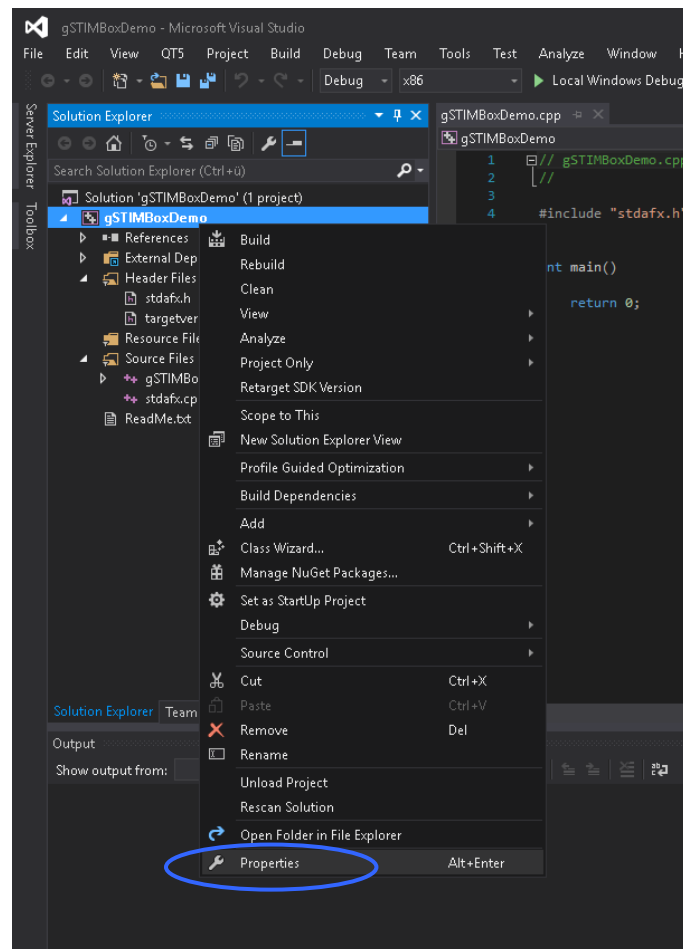
## 8.5 Using the g.STIMbox with the C-API

### 8.5.1 USAGE

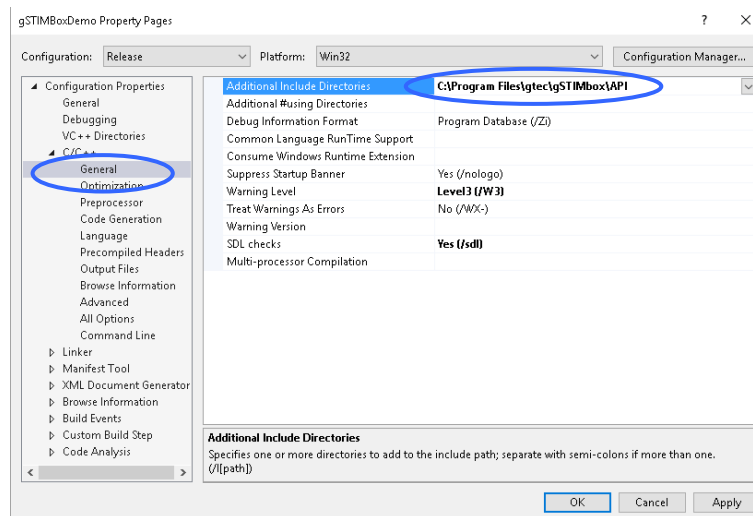
The g.STIMbox comes with a C-API which allows the user to control the box from a C application.

There are three important files, namely **gSTIMbox.lib**, **gSTIMbox.dll** and **gSTIMbox.h** which have to be used when developing C applications which make use of the g.STIMbox.

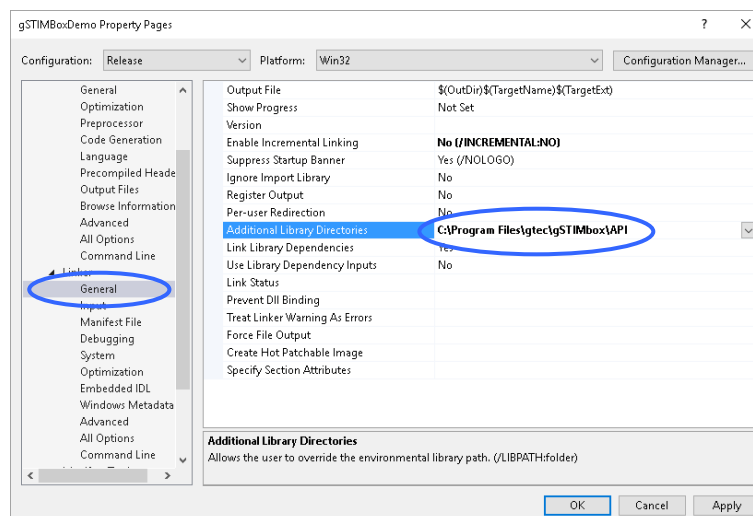
To use the API in Microsoft Visual Studio open your project-properties.



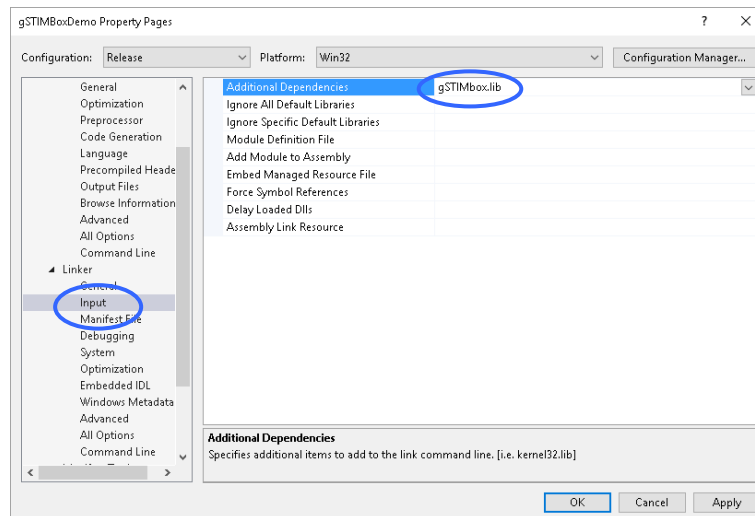
Then select **General** under **C/C++** and add the path where the *gSTIMbox.h* file resides to the **Additional Include Directories**.



Then select **General** under **Linker** and add the path to the **Additional Library Directories**.



Select **Input** under **Linker** and add the filename of the library to the **Additional Dependencies**.



Finally include the header file in your source-code:

```
#include "gSTIMbox.h"
```

**Note:** To deploy applications which make use of the g.STIMbox, use the installer of the g.STIMbox package which comes on CD and install it on the target computer.

### 8.5.2 DEMOS

Two demo applications are coming on CD with the C-API.

Simply call it from your terminal with

```
gSTIMboxDemo1
```

or

```
gSTIMboxDemo2
```

respectively.

You can also call the demo directly by double-clicking the **gSTIMboxDemo1.exe** or **gSTIMboxDemo2.exe**. The demos give a short description of what is implemented and ask for the port number the gSTIMbox is connected to. Insert the number and press Enter and the demos will start. The sources of the two small demo programs are also included to show how to use the API-functions. The names of the sourcefiles are **gSTIMboxDemo1.cpp** and **gSTIMboxDemo2.cpp** respectively.

### 8.5.3 FUNCTION REFERENCE

[gSTIMboxinit](#)  
[gSTIMboxreset](#)  
[gSTIMboxsetMode](#)  
[gSTIMboxsetPortState](#)  
[gSTIMboxsetFrequency](#)  
[gSTIMboxclose](#)  
[gSTIMboxgetHWVersion](#)  
[gSTIMboxgetDriverVersion](#)  
[gSTIMboxgetConnStatus](#)  
[gSTIMboxgetInputs](#)  
[gSTIMboxstartInputMonitoring](#)  
[gSTIMboxstopInputMonitoring](#)  
[gSTIMboxsetInputInversion](#)  
[gSTIMboxgetInputSampleRate](#)  
[gSTIMboxgetInputFrameLength](#)  
[gSTIMboxgetErrorMessage](#)

### 8.5.3.1 gSTIMboxinit

```
int gSTIMboxinit(  
    HANDLE *handle,  
    int comPort,  
    int samplingrate,  
    int framelength,  
    int drivingMode  
);
```

Call it with:

```
int error = gSTIMboxinit(&handle, comPort, samplingrate, framelength, drivingMode);
```

**DESCRIPTION:** This function initializes the communication between the computer and the g.STIMbox.

**INPUT:**

HANDLE \*handle...The address of the HANDLE variable.

int comPort...The number of the virtual COM port the g.STIMbox is connected to.

int samplingrate...The sampling rate in Hz the g.STIMbox will digitize its input ports with.

int framelength...The length of the frames (in samples) with digitized input data the g.STIMbox will send to the computer.

int drivingMode...Can be 0 or 1. 0 means that the command gSTIMboxgetInputs will not block the process. 1 means that the mentioned command blocks the process until new data from the g.STIMbox is available.

**OUTPUT:**

HANDLE \*handle...The address of the handle is stored to the memory segment the pointer points to. The handle is needed for all commands concerning the g.STIMbox. It must be kept.

int error...In case the function succeeds the error value is 0. Otherwise it will be >0.

Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;  
int comport = <yourCOMport>;  
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);  
if(error){ /* deal with error */ }  
error = gSTIMboxclose(handle);  
if(error){ /* deal with error */ }
```

In this example the g.STIMbox is initialized with COM-Port 10, a sampling rate of 256 Hz, a frame length of 8 samples and the driving mode on. Afterwards the handle is closed again with gSTIMboxclose.

### 8.5.3.2 *gSTIMboxreset*

```
int gSTIMboxreset(HANDLE handle);
```

Call it with:

```
int error = gSTIMboxreset(handle);
```

**DESCRIPTION:** The function resets a g.STIMbox represented by handle opened with the `gSTIMboxinit` function to its initial state.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxreset(handle);
if(error){ /* deal with error */ }
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

### 8.5.3.3 gSTIMboxsetMode

```
int gSTIMboxsetMode(HANDLE handle, int nrPorts, int port[], int mode[]);
```

Call it with:

```
int error = gSTIMboxsetMode(handle, nrPorts, port, mode);
```

**DESCRIPTION:** This function sets certain output ports of the g.STIMbox either to be in on/off mode (0) or in frequency mode (1). In both modes a port STATE of 0 means that the output port of the g.STIMbox is switched off. A port state of 1 in mode 0 means that the output port is switched on continuously whereas in mode 1 it means that the g.STIMbox toggles the output port in the frequency specified with the function gSTIMboxsetFrequency. The port states can be set with the function gSTIMboxsetPortState.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

int nrPorts...An integer describing the length of the following input arrays.

int ports[]...An array containing the output ports of the g.STIMbox to which the modes given in the next parameter will be assigned.

int modes[]...An array containing the modes for the output ports.

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int nrPorts = 4;
int port[] = {0, 1, 2, 3};
int mode[] = {0, 1, 0, 1};
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxsetMode(handle, nrPorts, port, mode);
if(error){ /* deal with error */ }
/* ... */
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

In this example the output ports 1 and 3 of the g.STIMbox represented by handle are set to mode 0 (on/off mode) and output ports 2 and 4 are set to mode 1 (frequency mode). Given, that the states of the ports 1 to 4 are set to 1 (on) ports 1 and 3 will be switched on and ports 2 and 4 will flicker in a given frequency. Afterwards the handle is closed again with gSTIMboxclose. Note that the port indexing in C starts with 0 (i.e. Hardware output port 1 has index 0)!



#### 8.5.3.4 gSTIMboxsetPortState

```
int gSTIMboxsetPortState(HANDLE handle, int nrPorts, int port[], int state[]);
```

Call it with:

```
int error = gSTIMboxsetPortState(handle, nrPorts, port, state);
```

**DESCRIPTION:** This function sets the states of certain output ports of the g.STIMbox either to 1 (on) or 0 (off). For ports which are in mode 0 (on/off mode) this simply means that the port is switched on or off. For ports which are in mode 1 (frequency mode) it means that the frequency specified with the function gSTIMboxsetFrequency is either generated or the port is switched off. The port modes can be set with the function gSTIMboxsetMode.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

int nrPorts...An integer describing the length of the following input arrays.

int port[]...An array containing the output ports of the g.STIMbox to which the states given in the next parameter will be assigned.

int state[]...An array containing the states for the output ports.

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int nrPorts = 4;
int port[] = {0, 1, 2, 3};
int state[] = {0, 1, 1, 1};
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxsetPortState(handle, nrPorts, port, state);
if(error){ /* deal with error */ }
/* ... */
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

In this example the output port 1 of the g.STIMbox represented by handle is set to 0 (off) and output ports 2 to 4 are set to 1 (on). Depending on their ports modes output ports 2 to 4 will either flicker in a certain frequency or be switched on. Afterwards the handle is closed again with gSTIMboxclose. Note that the port indexing in C starts with 0 (i.e. Hardware output port 1 has index 0)!

### 8.5.3.5 *gSTIMboxsetFrequency*

```
int gSTIMboxsetFrequency(HANDLE handle, int nrPorts, int port[], double freq[]);
```

Call it with:

```
int error = gSTIMboxsetFrequency(handle, nrPorts, port, freq);
```

**DESCRIPTION:** This function sets certain output ports of the g.STIMbox to certain frequencies. The box will toggle the outputs high and low in this frequency if the modes of these ports are set to 1 with the function `gSTIMboxsetMode` and the states of the ports are set to 1 with the function `gSTIMboxsetPortState`.

**INPUT:**

`HANDLE handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

`int nrPorts`...An integer describing the length of the following input arrays.

`int port[]`...An array containing the output ports of the g.STIMbox to which the frequencies given in the next parameter will be assigned.

`int freq[]`...An array containing the frequencies in Hz.

**OUTPUT:**

`int error`...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int nrPorts = 4;
int port[] = {0, 1, 2, 3};
double freq[] = {10, 11, 12, 13};
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxsetFrequency(handle, nrPorts, port, freq);
if(error){ /* deal with error */ }
/* ... */
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

In this example the frequencies 10Hz, 11Hz, 12Hz, and 13Hz are assigned to the output ports 1, 2, 3, and 4 of the g.STIMbox represented by `handle`. Afterwards the handle is closed again with `gSTIMboxclose`. Note that the port indexing in C starts with 0 (i.e. Hardware output port 1 has index 0)!

### 8.5.3.6 *gSTIMboxclose*

```
int gSTIMboxclose(HANDLE handle);
```

Call it with:

```
int error = gSTIMboxclose(handle);
```

**DESCRIPTION:** The function closes a handle to a g.STIMbox opened with the gSTIMboxinit function. After the call of this function a communication with the g.STIMbox is not possible anymore. It can be disconnected from the computer.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;  
int comport = <yourCOMport>;  
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);  
if(error){ /* deal with error */ }  
error = gSTIMboxclose(handle);  
if(error){ /* deal with error */ }
```

### 8.5.3.7 *gSTIMboxgetHWVersion*

```
int gSTIMboxgetHWVersion(HANDLE handle, float &HWVersion);
```

Call it with:

```
int error = gSTIMboxgetHWVersion(handle, HWversion);
```

**DESCRIPTION:** The function returns the version of the firmware running on the microcontroller of the g.STIMbox represented by handle.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

float &HWVersion...The reference to a float variable where the version number should be stored.

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0.

Please refer to the error codes in Section 8.5.4.

float &HWVersion...The reference to the variable the version number of the firmware of the g.STIMbox is stored.

**EXAMPLE:**

```
HANDLE handle;
float HWVersion = 0.0;
int comport = <yourCOMport>;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxgetHWVersion(handle, HWVersion);
if(error){ /* deal with error */ }
printf_s("Hardware version: %.2f\n", HWVersion);
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

### 8.5.3.8 *gSTIMboxgetDriverVersion*

```
int gSTIMboxgetDriverVersion(double &driverVersion);
```

Call it with:

```
int error = gSTIMboxgetDriverVersion(driverVersion);
```

**DESCRIPTION:** The function returns the version of the g.STIMbox driver installed on the system.

**INPUT:**

double &driverVersion...The reference to a double variable where the version number of the installed driver of the g.STIMbox should be stored.

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

double &driverVersion...The reference to the variable the version number of the installed driver of the g.STIMbox is stored.

**EXAMPLE:**

```
HANDLE handle;
double driverVersion = 0.0;
int comport = <yourCOMport>;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxgetDriverVersion (driverVersion);
if(error){ /* deal with error */ }
printf_s("Driver version: %.2f\n", driverVersion);
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

### 8.5.3.9 *gSTIMboxgetConnStatus*

```
int gSTIMboxgetConnStatus(HANDLE handle, int &state);
```

Call it with:

```
int error = gSTIMboxgetConnStatus(handle, state);
```

**DESCRIPTION:** The function returns the state of the connection to a g.STIMbox for a given handle created by the function gSTIMboxinit.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

int &state... The reference to an integer variable where the connection state to the g.STIMbox should be stored.

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0.

Please refer to the error codes in Section 8.5.4.

int &state... The reference to the variable where the connection state to the g.STIMbox is stored. It is 1 in case if the connection is working and 0 in case if the connection does not work or the handle was closed with the function gSTIMboxclose.

**EXAMPLE:**

```
HANDLE handle;
int state = 0;
int comport = <yourCOMport>;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }
error = gSTIMboxgetConnStatus(handle, state);
if(error){ /* deal with error */ }
printf_s("Connection state: %d\n", state);
error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

### 8.5.3.10 gSTIMboxgetInputs

```
int gSTIMboxgetInputs(HANDLE handle, int *buffer, int &overUnderflow);
```

Call it with:

```
int error = gSTIMboxgetInputs(handle, buffer, overUnderflow);
```

**DESCRIPTION:** The function returns digitized input data of the g.STIMbox, represented by handle when the input monitoring is activated (after the call of the function gSTIMboxstartInputMonitoring. The length of the return value input is determined by the frame length specified in the method gSTIMboxinit.

**INPUT:**

HANDLE handle...The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

int \*buffer...A pointer to a preallocated segment of storage where the input data should be stored.

int &overUnderflow...A reference to a variable where information about a data receive-buffer overflow or an empty data receive-buffer should be stored.

**OUTPUT:**

int error...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

int \*buffer...A vector of the length of a frame (set in the gSTIMboxinit method) containing input data. The framelength can be determined by using the function gSTIMboxgetInputFramelength.

int &overUnderflow... The reference to the variable where the information about an empty data-receive buffer or a buffer overflow is stored. It is 1 in case of a data receive buffer overflow, 2 in the case of an empty receive buffer and 0 otherwise.

**EXAMPLE:**

```
HANDLE handle;
int framelength = 0;
int overUnderflow = 0;
int comport = <yourCOMport>;

int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }

error = gSTIMboxgetInputFramelength(handle, framelength);
if(error){ /* deal with error */ }

int *buffer = new int[framelength];

error = gSTIMboxstartInputMonitoring(handle);
if(error){ /* deal with error */ }
```

```

for(int i=0; i<10; i++)
{
    error = gSTIMboxgetInputs(handle, buffer, overUnderflow);
    if(error){ /* deal with error */ }
    /* ... */
}

error = gSTIMboxstopInputMonitoring(handle);
if(error){ /* deal with error */ }

delete[] buffer;

error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }

```



#### **8.5.3.11 gSTIMboxstartInputMonitoring**

```
int gSTIMboxstartInputMonitoring(HANDLE handle);
```

Call it with:

```
int error = gSTIMboxstartInputMonitoring(handle);
```

**DESCRIPTION:** With this function the input monitoring of the g.STIMbox, represented by `handle` is started. After the call of this function the input can be received using the function `gSTIMboxgetInputs`.

**INPUT:**

`HANDLE handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

`int error`...In case the function succeeds the error value is 0. Otherwise it will be  $>0$ . Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

→ see example of the function `gSTIMboxgetInputs`.

### **8.5.3.12 gSTIMboxstopInputMonitoring**

```
int gSTIMboxstopInputMonitoring(HANDLE handle);
```

Call it with:

```
int error = gSTIMboxstopInputMonitoring(handle);
```

**DESCRIPTION:** With this function the input monitoring of the g.STIMbox, represented by `handle` is stopped. The g.STIMbox stops to send data to the computer and the function `gSTIMboxgetInputs` should not be called after the call to this function.

**INPUT:**

`HANDLE handle`...The handle representing the g.STIMbox (generated with the function `gSTIMboxinit`).

**OUTPUT:**

`int error`...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

→ see example of the function `gSTIMboxgetInputs`.

### 8.5.3.13 *gSTIMboxsetInputInversion*

```
int gSTIMboxsetInputInversion(HANDLE com, int inversion);
```

Call it with:

```
int error = gSTIMboxsetInputInversion(handle, inversion);
```

**DESCRIPTION:** With this function certain input ports of the g.STIMbox can be inverted. In normal mode (without inversion) the input port is low when a possibly connected switch is open and high when it is closed. For signal generators with active low output for example the input of the g.STIMbox can be inverted in order to give an active high signal.

**INPUT:**

**HANDLE handle...**The handle representing the g.STIMbox (generated with the function gSTIMboxinit).

**int inversion...**An integer with the bits corresponding to the input ports of the g.STIMbox to be inverted set. Zero (0) means that no input port will be inverted.

**OUTPUT:**

**int error...**In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }

int inversion = ( (1<<4) | (1<<6) | (1<<8) );
error = gSTIMboxsetInputInversion(handle, inversion);
if(error){ /* deal with error */ }

error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

In this example the input ports 5, 7, and 9 of the g.STIMbox represented by handle will be inverted. Afterwards the handle is closed again with gSTIMboxclose. Note that the port indexing in C starts with 0 (i.e. Hardware output port 1 has index 0)!

#### 8.5.3.14 *gSTIMboxgetInputSamplerate*

```
int gSTIMboxgetInputSamplerate(HANDLE com, int &inputSamplerate);
```

Call it with:

```
int error = gSTIMboxgetInputSamplerate(handle, inputSamplerate);
```

**DESCRIPTION:** The function returns the sampling rate in Hz at which the g.STIMbox, represented by *handle* digitizes its input ports once the input monitoring is activated (with the function *gSTIMboxstartInputMonitoring*).

**INPUT:**

*HANDLE handle*...The handle representing the g.STIMbox (generated with the function *gSTIMboxinit*).

*int &inputSamplerate*... The reference to an integer variable where the input sampling rate of the g.STIMbox should be stored.

**OUTPUT:**

*int error*...In case the function succeeds the error value is 0. Otherwise it will be >0.

Please refer to the error codes in Section 8.5.4.

*int &inputSamplerate*... The reference to an integer variable where the input sampling rate of the g.STIMbox is stored.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int inputSamplerate = 0;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }

error = gSTIMboxgetInputSamplerate(handle, inputSamplerate);
if(error){ /* deal with error */ }
printf("Input Sampling Rate: %d\n", inputSamplerate);

error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

### 8.5.3.15 *gSTIMboxgetInputFramelength*

```
int gSTIMboxgetInputFramelength(HANDLE com, int &inputFramelength);
```

Call it with:

```
int error = gSTIMboxgetInputFramelength(handle, inputFramelength);
```

**DESCRIPTION:** The function returns the length of the frames which are transmitted from the g.STIMbox, represented by *handle* to the computer when the input monitoring is activated (with the function *gSTIMboxstartInputMonitoring*).

**INPUT:**

*HANDLE handle*...The handle representing the g.STIMbox (generated with the function *gSTIMboxinit*).

*int &inputFramelength*... The reference to an integer variable where the input framelength of the g.STIMbox should be stored.

**OUTPUT:**

*int error*...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.

*int &inputFramelength*... The reference to an integer variable where the input framelength of the g.STIMbox is stored.

**EXAMPLE:**

```
HANDLE handle;
int comport = <yourCOMport>;
int inputFramelength = 0;
int error = gSTIMboxinit(&handle, comport, 256, 8, 1);
if(error){ /* deal with error */ }

error = gSTIMboxgetInputFramelength(handle, inputFramelength);
if(error){ /* deal with error */ }
printf("Input Framelength: %d\n", inputFramelength);

error = gSTIMboxclose(handle);
if(error){ /* deal with error */ }
```

### 8.5.3.16 gSTIMboxgetErrorMessage

```
int gSTIMboxgetErrorMessage(int error, int messagelength, char *errormessage);
```

Call it with:

```
int error1 = gSTIMboxgetErrorMessage(error, messagelength, errormessage);
```

**DESCRIPTION:** The function translates a g.STIMbox C-API error code into a human readable string.

**INPUT:**

int error...The error code returned by another g.STIMbox C-API function.  
int messagelength...An integer describing the length of the following preallocated char\* which is the destination for the human readable translation of the error code.  
char \*errormessage...A preallocated segment of memory which is the destination for the human readable translation of the error code given.

**OUTPUT:**

int error1...In case the function succeeds the error value is 0. Otherwise it will be >0. Please refer to the error codes in Section 8.5.4.  
char \*errormessage...After the function returns without an error the translated message to the given error error will be stored in this segment of memory.

**EXAMPLE:**

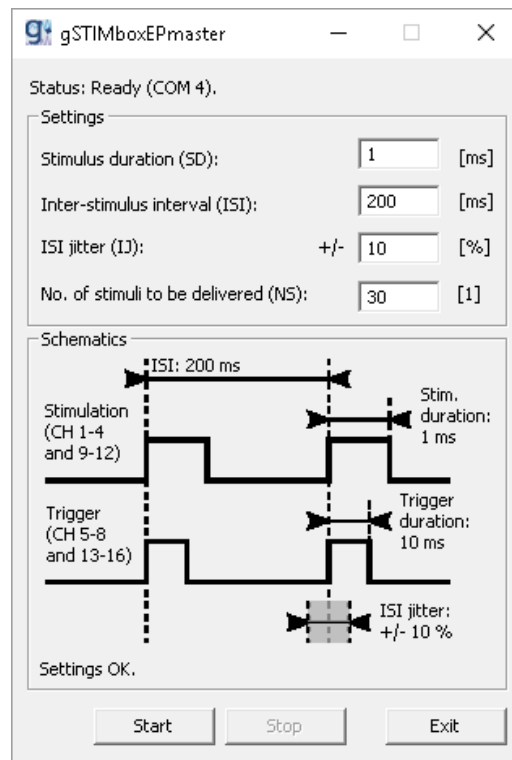
```
HANDLE handle;
int error = gSTIMboxinit(&handle, -1, 256, 8, 1);
if(error) {
    int messagelength = 50;
    char* errormessage = new char[messagelength];
    if(!errormessage)
    {
        printf_s("Unable to retrieve error message!\n");
        return 7;
    }
    int error1 = gSTIMboxgetErrorMessage(error, messagelength,
errormessage);
    if(error1)
    {
        printf_s("Unable to retrieve error message!\n");
        delete[] errormessage;
        return error1;
    }
    else
    {
        printf_s("Error: %s\n", errormessage);
        delete[] errormessage;
        return error;
    }
}
/* ... */
```

#### 8.5.4 ERROR CODES

The following table gives an overview of the error codes returned by the functions of the g.STIMbox C-API.

Error Code	Meaning
0	Success.
1	COM-Port out of range.
2	Error opening COM-Port.
3	Sampling rate out of range.
4	Frame length out of range.
5	Frame length greater than sampling rate.
6	Error sending command.
7	Memory allocation error.
8	Maximum number of connected devices reached.
9	Invalid handle.
10	Null pointer exception.
11	Port count out of range.
12	Invalid mode specified.
13	Invalid port number specified.
14	Invalid port state specified.
15	Frequency out of range.
16	Error retrieving error message.
17	Error retrieving software version info.

## 8.6 gSTIMboxEPmaster



With this small application it is possible to deliver stimuli which are suited for EP measurement.

The application searches for a connected device automatically. After a device is found the Status message switches to: Status: Ready (COM XX)

The XX stands for the COM port the g.STIMbox is connected to.

The editable paradigm parameters are:

- Stimulus duration  
The length of a stimulus
- Inter-stimulus interval (ISI)  
The time between two stimuli onsets
- ISI jitter (IJ)  
Randomized variation for the ISI
- No. of stimuli to be delivered (NS)  
Number of stimulus repetitions

The application will check the entered settings instantaneously and give according information in the lower part of the Schematics section. If everything is OK the message will be: Settings OK.

A click on the **Start** button starts the stimulus delivery.

The stimuli are delivered through ports 1-4 and 9-12.

Triggers with a constant length of 10ms are delivered through outputs 5-8 and 13-16.

A click on the **Stop** button stops the delivery again.



## **9 General notes**

### **TRANSPORTATION AND STORAGE CONDITIONS**

The device can be stored at temperatures between –20 to +60 degrees Celsius. The relative humidity must be between 25 % and 95 %. Wait before usage of the device till condensed water disappeared (wait at least 1h in a heated room).

### **LOCATION DETAILS**

Do not use the device near a heating system or directly in the sun. The maximal temperature of the environment must not be above 40 ° Celsius.

### **WASTE DISPOSAL DETAILS**

Bring the device to a recycling center or sent it back to the manufacturer.

### **CLEANING**

You can clean the device carefully with a damp cloth. Liquid must not enter the g.STIMbox.

## 10 Technical specifications

### 10.1 g.STIMbox

Model	g.STIMbox
Type	USB stimulation digital I/O box
Rated power consumption (max.)	
USB powered	5 V x 200 mA = 1W
externally powered	5 V x 2.6 A = 13W
Rated DC voltage	5 V
Rated current of fuse	200 mA resettable fuse for USB power connection 2,6 A, resettable fuse for external 5 V power supply 0,5 A, resettable fuse for each of the 16 digital outputs
Produced	2010
Producer	g.tec medical engineering GmbH Sierningstrasse 14 4521 Schiedlberg Austria <a href="http://www.gtec.at">www.gtec.at</a>
Maximum voltages at the following sockets	
USB	5 V DC
DIGITAL I/O	5 V DC
POWER SUPPLY	5 V DC

#### Digital inputs

TTL inputs	Sensitivity:	5 V
------------	--------------	-----

#### Digital outputs

TTL outputs	Sensitivity:	5 V
-------------	--------------	-----

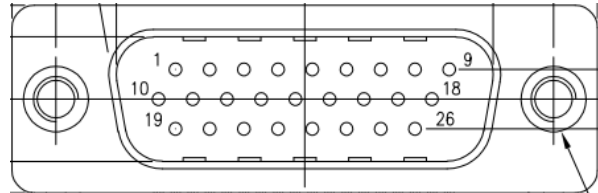
## 11 PIN assignment

### g.STIMbox

#### *Pin-assignment for the 26 pin Sub-D plug*

##### **INPUTS**

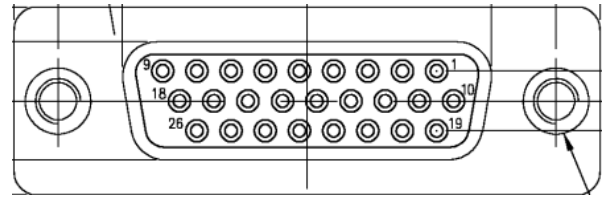
Pin 1	digital input 1
Pin 2	digital input 2
Pin 3	digital input 3
Pin 4	digital input 4
Pin 5	digital input 5
Pin 6	digital input 6
Pin 7	digital input 7
Pin 8	digital input 8
Pin 9	digital input 9
Pin 10	digital input 10
Pin 11	digital input 11
Pin 12	digital input 12
Pin 13	digital input 13
Pin 14	digital input 14
Pin 15	NC
Pin 16	NC
Pin 17	NC
Pin 18	+5 V
Pin 19	GND
Pin 20	NC
Pin 21	NC
Pin 22	NC
Pin 23	NC
Pin 24	NC
Pin 25	NC
Pin 26	NC



***Pin-assignment for the 26 pin Sub-D socket***

***OUTPUTS***

Pin 1 digital output 1  
Pin 2 digital output 2  
Pin 3 digital output 3  
Pin 4 digital output 4  
Pin 5 digital output 5  
Pin 6 digital output 6  
Pin 7 digital output 7  
Pin 8 digital output 8  
Pin 9 digital output 9  
Pin 10 digital output 10  
Pin 11 digital output 11  
Pin 12 digital output 12  
Pin 13 digital output 13  
Pin 14 digital output 14  
Pin 15 digital output 15  
Pin 16 digital output 16  
Pin 17 NC  
Pin 18 +5 V  
Pin 19 GND  
Pin 20 NC  
Pin 21 NC  
Pin 22 NC  
Pin 23 NC  
Pin 24 NC  
Pin 25 NC  
Pin 26 NC

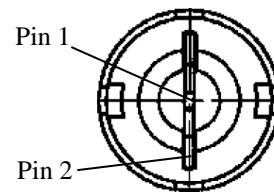


### ***Cinch sockets***

Pin-assignment for the external cinch sockets

Pin 1 digital input/output

Pin 2 GND

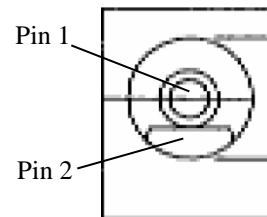


### ***Power supply unit cable***

Pin-assignment for the external power socket

Pin 1 +5 V

Pin 2 GND



### ***USB socket***

Pin-assignment for the 5 pin Mini-USB

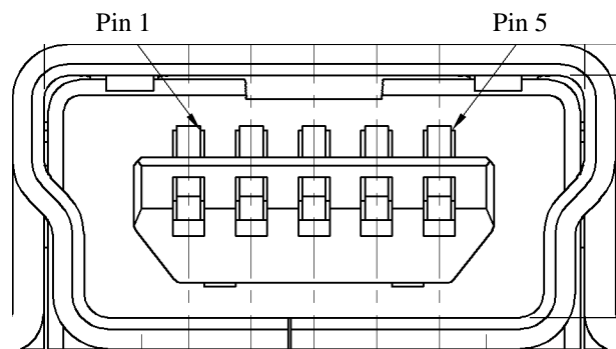
Pin 1 +Vbus

Pin 2 D -

Pin 3 D +

Pin 4 ID

Pin 5 GND



## **g.SSVEPbox**

### ***Pin-assignment for the 26 pin Sub-D plug***

#### ***INPUTS***

- Pin 1 stimulation LED left
- Pin 2 indicator LED left
- Pin 3 stimulation LED forward
- Pin 4 indicator LED forward
- Pin 5 stimulation LED right
- Pin 6 indicator LED right
- Pin 7 stimulation LED backward
- Pin 8 indicator LED right
- Pin 9 NC
- Pin 10 NC
- Pin 11 NC
- Pin 12 NC
- Pin 13 NC
- Pin 14 NC
- Pin 15 NC
- Pin 16 NC
- Pin 17 NC
- Pin 18 +5 V
- Pin 19 GND
- Pin 20 NC
- Pin 21 NC
- Pin 22 NC
- Pin 23 NC
- Pin 24 NC
- Pin 25 NC
- Pin 26 NC

